

George F. Luger



# Inteligência **Artificial**

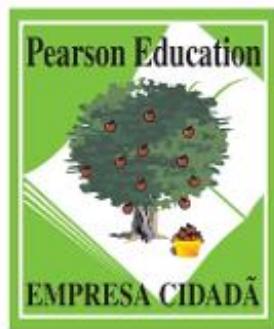
*6<sup>a</sup> edição*





# Inteligência Artificial

*6<sup>a</sup> edição*





# Inteligência Artificial

6<sup>a</sup> edição

George F. Luger  
*University of New Mexico*

Tradução:

Daniel Vieira

Revisão técnica:

Andréa Iabradi Tavares

*Doutora (2004) em Ciência da Computação  
pela Universidade Federal de Minas Gerais.*

PEARSON

**abdr**  
Respeite o direito autoral

©2014 by Pearson Education do Brasil Ltda.

© 2009 by Pearson Education, Inc.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação sem prévia autorização por escrito da Pearson Education do Brasil.

DIRETOR EDITORIAL E DE CONTEÚDO	Roger Trimer
GERENTE EDITORIAL	Kelly Tavares
SUPERVISORA DE PRODUÇÃO EDITORIAL	Silvana Alonso
COORDINADORA DE PRODUÇÃO GRÁFICA	Tatiane Romano
COORDENADOR DE PRODUÇÃO EDITORIAL	Sérgio Nascimento
EDITOR DE AQUISIÇÕES	Vinícius Souza
EDITORA DE TEXTO	Sabrina Leventeinas
EDITORES ASSISTENTES	Luiz Salla e Marcos Guimarães
PREPARAÇÃO	Beatriz Garcia
REVISÃO	Raura Ikeda
CAPA	Pedro Gentile
PROJETO GRÁFICO E DIAGRAMAÇÃO	Casa de Ideias

**Dados Internacionais de Catalogação na Publicação (CIP)**

(Câmara Brasileira do Livro, SP, Brasil)

Luger, George F.  
Inteligência artificial / George F. Luger; tradução Daniel Vieira;  
revisão técnica Andréa Iabudi Tavares. – 6. ed. – São Paulo:  
Pearson Education do Brasil, 2013.

Titulo original: Artificial intelligence  
Bibliografia.  
ISBN 978-85-8143-550-3

1. Inteligência artificial – Aspectos sociais 2. Inteligência artificial  
– Inovações tecnológicas – Aspectos sociais I. Tavares, Andréa Iabudi.  
II. Título.

13-08713

CDD-000.6

Índice para catálogo sistemático:

1. Inteligência artificial 006.3

2013

Direitos exclusivos para a língua portuguesa cedidos à

Pearson Education do Brasil Ltda.,

uma empresa do grupo Pearson Education

Rua Nelson Francisco, 26

CEP 02712-100 – São Paulo – SP – Brasil

Fone: 11 2178-8686 – Fax: 11 2178-8688

vendas@pearson.com

Para a minha esposa, Kathleen, e nossos filhos Sarah, David e Peter.

*Si quid est in me ingenii, judices . . .*  
Cicero, Pro Archia Poeta

GFL



# Sumário

Prefácio	XI
----------	----

## **PARTE I Inteligência artificial: raízes e escopo**

Inteligência artificial: uma tentativa de definição .....	1
<b>CAPÍTULO 1 IA: história e aplicações</b>	<b>3</b>
1.1 Do Éden ao ENIAC: posicionamentos em relação à inteligência, ao conhecimento e à astúcia humana .....	3
1.2 Uma visão geral das áreas de aplicação da IA.....	16
1.3 Inteligência artificial — um resumo.....	25
1.4 Epílogo e referências .....	26
1.5 Exercícios .....	27

## **PARTE II Inteligência artificial como representação e busca**

Introdução à representação e busca .....	29
<b>CAPÍTULO 2 Cálculo de predicados</b>	<b>38</b>
2.0 Introdução .....	38
2.1 Cálculo proposicional (opcional) .....	38
2.2 Cálculo de predicados .....	42
2.3 Usando regras de inferência para produzir expressões do cálculo de predicados.....	52
2.4 Aplicação: um consultor financeiro baseado em lógica .....	60
2.5 Epílogo e referências .....	63
2.6 Exercícios .....	64

<b>CAPÍTULO 3 Estruturas e estratégias para busca em espaço de estados</b>	<b>66</b>
3.0 Introdução .....	66
3.1 Estruturas para busca em espaço de estados.....	69
3.2 Estratégias para busca em espaço de estados .....	78
3.3 Usando o espaço de estados para representar o raciocínio com o cálculo proposicional e de predicados .....	89
3.4 Epílogo e referências .....	100
3.5 Exercícios .....	101

<b>CAPÍTULO 4 Busca heurística</b>	<b>103</b>
4.0 Introdução .....	103
4.1 Subida de encosta e programação dinâmica .....	106
4.2 Algoritmo da busca pela melhor escolha .....	112
4.3 Admissibilidade, monotonicidade e grau de informação .....	121
4.4 Usando heurísticas em jogos .....	125

4.5 Aspectos de complexidade.....	132
4.6 Epílogo e referências.....	135
4.7 Exercícios .....	135

**CAPÍTULO 5 Métodos estocásticos** 138

5.0 Introdução .....	138
5.1 Elementos da contagem (opcional).....	140
5.2 Elementos de teoria da probabilidade .....	142
5.3 Teorema de Bayes .....	151
5.4 Aplicações da metodologia estocástica.....	154
5.5 Epílogo e referências.....	158
5.6 Exercícios .....	159

**CAPÍTULO 6 Construção de algoritmos de controle para busca em espaço de estados** 161

6.0 Introdução .....	161
6.1 Busca baseada em recursão (opcional).....	162
6.2 Sistemas de produção.....	167
6.3 Arquitetura de quadro-negro na solução de problemas.....	181
6.4 Epílogo e referências.....	183
6.5 Exercícios .....	184

---

**PARTE III Capturando inteligência: o desafio da IA**

Representação e inteligência.....	185
-----------------------------------	-----

**CAPÍTULO 7 Representação do conhecimento** 189

7.0 Questões da representação do conhecimento .....	189
7.1 Uma breve história dos esquemas representacionais de IA .....	190
7.2 Grafos conceituais: uma linguagem de rede .....	207
7.3 Alternativas para representações e ontologias.....	216
7.4 Solução de problemas distribuída e baseada em agentes.....	221
7.5 Epílogo e referências.....	225
7.6 Exercícios .....	228

**CAPÍTULO 8 Solução de problemas por método forte** 231

8.0 Introdução .....	231
8.1 Visão geral da tecnologia de sistemas especialistas .....	232
8.2 Sistemas especialistas baseados em regras.....	239
8.3 Sistemas baseados em modelo, em casos e híbridos .....	248
8.4 Planejamento .....	262
8.5 Epílogo e referências.....	274
8.6 Exercícios .....	276

<b>CAPÍTULO 9 Raciocinando em situações incertas</b>	<b>278</b>
9.0 Introdução .....	278
9.1 Inferência abdutiva baseada em lógica.....	279
9.2 Abdução: alternativas à lógica .....	291
9.3 Abordagem estocástica para a incerteza.....	302
9.4 Epílogo e referências .....	315
9.5 Exercícios .....	317
 <b>PARTE IV Aprendizado de máquina</b>	
Métodos simbólico, conexionista, genético e estocástico para o aprendizado de máquina .....	319
<b>CAPÍTULO 10 Aprendizado de máquina: simbólico</b>	<b>321</b>
10.0 Introdução .....	321
10.1 Um arcabouço para o aprendizado simbólico .....	323
10.2 Busca em espaço de versões .....	328
10.3 Algoritmo ID3 para indução de árvores de decisão.....	339
10.4 Viés indutivo e capacidade de aprendizado .....	346
10.5 Conhecimento e aprendizado .....	350
10.6 Aprendizado não supervisionado.....	358
10.7 Aprendizado por reforço .....	366
10.8 Epílogo e referências .....	371
10.9 Exercícios .....	372
<b>CAPÍTULO 11 Aprendizado de máquina: conexionista</b>	<b>375</b>
11.0 Introdução.....	375
11.1 Fundamentos das redes conexionistas.....	377
11.2 Aprendizado do perceptron.....	379
11.3 Aprendizado por retropropagação.....	386
11.4 Aprendizado competitivo .....	392
11.5 Aprendizado hebbiano por coincidência .....	400
11.6 Redes de atratores ou “memórias” .....	409
11.7 Epílogo e referências .....	417
11.8 Exercícios .....	417
<b>CAPÍTULO 12 Aprendizado de máquina: genético e emergente</b>	<b>419</b>
12.0 Modelos de aprendizado social e emergente.....	419
12.1 Algoritmo genético .....	421
12.2 Sistemas classificadores e programação genética .....	429
12.3 Vida artificial e aprendizado social .....	438
12.4 Epílogo e referências .....	447
12.5 Exercícios .....	448

<b>CAPÍTULO 13 Aprendizado de máquina: probabilístico</b>	<b>450</b>
13.0 Modelos estocásticos e dinâmicos do aprendizado.....	450
13.1 Modelos ocultos de Markov (MOMs) .....	451
13.2 Redes Bayesianas dinâmicas e aprendizado .....	459
13.3 Extensões estocásticas ao aprendizado por reforço .....	467
13.4 Epílogo e referências .....	471
13.5 Exercícios.....	472
 <b>PARTE V Tópicos avançados para a solução de problemas por IA</b>	
Raciocínio automatizado e linguagem natural.....	475
<b>CAPÍTULO 14 Raciocínio automatizado</b>	<b>477</b>
14.0 Introdução aos métodos fracos para a prova de teoremas.....	477
14.1 Resolvedor Geral de Problemas e as tabelas de diferenças.....	478
14.2 Prova de teoremas por resolução.....	483
14.3 Prolog e raciocínio automatizado.....	499
14.4 Outras questões em raciocínio automatizado .....	504
14.5 Epílogo e referências .....	509
14.6 Exercícios.....	510
<b>CAPÍTULO 15 Compreensão da linguagem natural</b>	<b>512</b>
15.0 O problema da compreensão da linguagem natural.....	512
15.1 Desconstruindo a linguagem: uma análise .....	515
15.2 Sintaxe.....	517
15.3 Analisadores e semântica da rede de transição.....	523
15.4 Ferramentas estocásticas para análise de linguagem.....	537
15.5 Aplicações da linguagem natural.....	544
15.6 Epílogo e referências .....	551
15.7 Exercícios .....	552
 <b>PARTE VI Epílogo</b>	
Reflexões sobre a natureza da inteligência.....	555
<b>CAPÍTULO 16 Inteligência artificial como investigação empírica</b>	<b>557</b>
16.0 Introdução .....	557
16.1 Inteligência artificial: uma definição revisada.....	559
16.2 A ciência dos sistemas inteligentes .....	569
16.3 IA: Questões atuais e direções futuras .....	577
16.4 Epílogo e referências .....	581
<b>Referências</b>	<b>583</b>
<b>Índice remissivo</b>	<b>605</b>

# Prefácio

*O que precisamos aprender a fazer  
aprendemos fazendo...*

—ARISTÓTELES, *Ethics*

## Bem-vindos à sexta edição!

Tive o prazer de ser convidado a produzir a sexta edição do meu livro sobre inteligência artificial. É um elogio às edições anteriores, iniciadas há mais de vinte anos, que nossa abordagem da IA tenha sido tão valorizada. Também é interessante notar que, com o surgimento de novos desenvolvimentos no campo, podemos apresentar muita coisa a cada nova edição. Agradecemos aos nossos muitos leitores, colegas e alunos por manter nossos tópicos relevantes e nossa apresentação atualizada.

Muitas seções das edições anteriores resistiram muito bem, incluindo a apresentação sobre lógica, algoritmos de busca, representação do conhecimento, sistemas de produção, aprendizado de máquina e, na Sala Virtual, as técnicas de programação desenvolvidas em Lisp, Prolog e, com esta edição, Java. Estas permanecem fundamentais à prática de inteligência artificial, e uma constante nesta nova edição.

Este livro permanece acessível. Introduzimos técnicas de representação vitais, incluindo lógica, semântica e redes conexionistas, modelos gráficos e muito mais. Nossos algoritmos de busca são apresentados de modo claro, primeiro em pseudocódigo e depois na Sala Virtual, muitos deles implementados em Prolog, Lisp e/ou Java. Esperamos que os alunos motivados possam utilizar nossas implementações básicas e estendê-las para aplicações novas e empolgantes.

Criamos, para a sexta edição, um novo capítulo sobre aprendizado de máquina baseado em métodos estocásticos (Capítulo 13). Achamos que a tecnologia estocástica tem um impacto cada vez maior sobre a IA, especialmente em áreas como raciocínio diagnóstico e prognóstico, análise de linguagem natural, robótica e aprendizado de máquina. Para dar suporte a essas tecnologias emergentes, expandimos a apresentação do teorema de Bayes, de modelos de Markov, redes Bayesianas de crença e modelos gráficos relacionados. Nossa expansão inclui um maior uso de máquinas probabilísticas de estado finito, modelos ocultos de Markov e programação dinâmica com o analisador de Earley, além de implementação do algoritmo de Viterbi. Outros tópicos, como computação emergente, ontologias, algoritmos para analisadores sintáticos probabilísticos, que foram tratados de forma superficial nas edições anteriores, tornaram-se suficientemente importantes para merecerem uma discussão mais completa. As mudanças na sexta edição refletem questões de pesquisa emergentes em inteligência artificial e são a evidência da vitalidade contínua em nosso campo.

À medida que o escopo de nosso projeto de IA crescia, éramos sustentados pelo apoio de nosso editor, e de revisores, amigos, colegas e, principalmente, de nossos leitores, que deram ao nosso trabalho uma vida tão longa e produtiva. Continuamos entusiasmados com a oportunidade de escrever que nos foi concedida: cientistas raramente são encorajados a ir além de seus próprios e particulares interesses de pesquisa e traçar trajetórias maiores no campo que escolheram. Nossos leitores nos pediram para fazer exatamente isso. Somos gratos a eles por essa oportunidade. Também nos sentimos encorajados ao ver que as edições anteriores foram usadas em comunidades de IA do mundo inteiro e traduzidas para diversos idiomas, incluindo o alemão, o polonês, o português, o russo e dois dialetos chineses!

Embora a inteligência artificial, como a maioria das disciplinas da engenharia, deva se justificar ao mundo do comércio oferecendo soluções para problemas práticos, entramos no campo da IA pelos mesmos motivos que muitos de nossos colegas e estudantes o fizeram: queremos entender e explorar os mecanismos da mente que possibilitam pensamento e ação inteligentes. Rejeitamos a noção um tanto provinciana de que a inteligência é uma

capacidade exclusiva dos humanos, e acreditamos que podemos efetivamente investigar o espaço de inteligências possíveis projetando e avaliando artefatos inteligentes. Embora o curso de nossas carreiras não tenha nos dado motivo para mudar esses comprometimentos, chegamos a uma apreciação maior pelo escopo, pela complexidade e pela audácia desse empreendimento. No prefácio de nossas edições anteriores, esboçamos três afirmações que acreditamos terem distinguido nosso método de ensino da inteligência artificial. Portanto, é razoável, ao escrever o prefácio desta edição, retornar a esses temas e ver como eles perduraram enquanto nosso campo cresceu.

A primeira dessas metas foi *unificar os diversos ramos da IA por meio de uma discussão detalhada de suas bases teóricas*. No momento em que adotamos essa meta pela primeira vez, parecia que o problema principal estava em harmonizar os pesquisadores que enfatizavam a afirmação e a análise cuidadosa de teorias formais de inteligências (os *organizados*) com aqueles que acreditavam que a própria inteligência era algum tipo de grande golpe que poderia ser mais bem tratado de uma maneira específica e orientada por aplicação (os *bagunceiros*). Essa dicotomia provou ser simples demais.

Na IA contemporânea, os debates entre os organizados e os bagunceiros deram lugar a dezenas de outros debates entre proponentes de sistemas de símbolos físicos e alunos de redes neurais, entre os lógicos e os projetistas de formas de vida artificiais que evoluem de uma maneira mais ilógica, entre arquitetos de sistemas especialistas e os adeptos do raciocínio baseado em casos e, finalmente, entre aqueles que acreditam que a inteligência artificial já foi alcançada e aqueles que acreditam que ela nunca acontecerá. Nossa imagem original da IA como ciência na fronteira em que os foras da lei, os exploradores e outros sonhadores eram lentamente domados pelas disciplinas do formalismo e do empirismo deu lugar a uma metáfora diferente: aquela de uma cidade grande, caótica mas quase totalmente pacífica, onde vizinhanças burguesas obedientes tiravam sua vitalidade de distritos diversos, caóticos, boêmios. Com o passar dos anos que nos dedicamos às diferentes edições deste livro, uma imagem convincente da arquitetura da inteligência começou a surgir a partir da estrutura, da arte e da indústria dessa cidade.

A inteligência é muito complexa para ser descrita por uma única teoria; em vez disso, os pesquisadores vêm construindo uma hierarquia de teorias que a caracteriza em vários níveis de abstração. Nos níveis mais baixos dessa hierarquia, as redes neurais, os algoritmos genéticos e as outras formas de computação emergentes nos permitiram entender os processos de adaptação, percepção, incorporação e interação com o mundo físico que precisa fundamentar qualquer forma de atividade inteligente. Por meio de uma resolução ainda parcialmente entendida, essa população caótica de atores cegos e primitivos fazem surgir os padrões mais arrojados da inferência lógica. Trabalhando nesse nível mais alto, os lógicos têm se baseado no dom de Aristóteles, traçando os contornos de dedução, abdução, indução, manutenção da verdade e inúmeros outros modos e maneiras de raciocinar. Em níveis de abstração ainda mais altos, os projetistas de sistemas de diagnóstico, os agentes inteligentes e os programas de reconhecimento da linguagem natural chegaram a reconhecer o papel dos processos sociais na criação, na transmissão e na sustentação do conhecimento.

Nesse ponto da iniciativa de IA, parece que os extremos do racionalismo e do empirismo só levaram a resultados limitados. Os dois extremos sofrem de aplicabilidade e generalização limitadas. O autor toma uma terceira visão, aquela do condicionamento do empirista — redes semânticas, roteiros, arquiteturas de subsunção — e das ideias claras e precisas do racionalista — o cálculo de predicados, as lógicas não monotônicas, o raciocínio automatizado — sugere um terceiro ponto de vista, o Bayesiano. A experiência das invariâncias relacionais condiciona as expectativas dos agentes inteligentes, e aprender essas invariâncias, por sua vez, influencia as expectativas futuras. Como filósofos, somos encarregados de criticar a validade epistemológica da iniciativa de IA. Para essa tarefa, no Capítulo 16, discutimos o projeto racionalista, o dilema empirista e propomos uma reconciliação construtivista baseada na teoria Bayesiana. Nesta sexta edição, tocamos em todos esses níveis na apresentação da iniciativa de IA.

O segundo comprometimento que fizemos nas edições anteriores foi com a posição central dos formalismos representativos avançados e com as técnicas de busca na metodologia de IA. Talvez esse seja o aspecto mais controverso de nossas edições anteriores e de grande parte do trabalho inicial sobre IA, com muitos pesquisadores em computação emergente questionando se o raciocínio simbólico e a semântica referencial exercem qualquer papel na inteligência. Embora a ideia de representação como atribuição de nomes a coisas tenha sido contestada pela representação implícita fornecida pelos padrões emergentes de uma rede neural ou uma vida artificial, acreditamos que um conhecimento da representação e da busca continue sendo essencial para qualquer profissional sério em inteligência artificial. Também acreditamos que nossa visão geral das tradições históricas e dos precursores da IA no Capítulo 1 contém componentes críticos do treinamento em IA. Além disso, essas são ferramentas valiosas

simas para a análise dos aspectos da IA não simbólica como o poder expressivo de uma rede neural ou a progressão da adaptação de prováveis soluções de problemas de um algoritmo genético. Comparações, contrastes e uma crítica da IA moderna são oferecidos no Capítulo 16.

Nosso terceiro comprometimento foi feito no início do ciclo de vida deste livro: inserir a inteligência artificial no contexto da ciência empírica. No espírito da *Turing Award Lecture*, de Newell e Simon (1976), citamos de uma edição mais antiga:

... a IA não é uma estranha aberração da tradição científica, mas ... parte de uma busca geral por conhecimento (e compreensão) a respeito da própria inteligência. Além disso, nossas ferramentas de programação de IA, juntamente com a metodologia de programação exploratória... são ideais para explorar um ambiente. Nossas ferramentas nos dão meios tanto para entendimento quanto para perguntas. Estamos aqui para apreciar e conhecer fenômenos construtivamente, ou seja, pela aproximação progressiva.

Assim, vemos cada projeto e cada programa como um experimento com a natureza: propomos uma representação, geramos um algoritmo de busca e questionamos a adequação de nossa caracterização para levar em conta parte do fenômeno da inteligência. E o mundo natural dá uma resposta à nossa pesquisa. Nosso experimento pode ser desmontado, revisado, estendido e executado novamente. Nosso modelo pode ser refinado, e nossa compreensão, estendida.

## **Novo na sexta edição**

A maior mudança da sexta edição é a extensão dos métodos estocásticos de IA. Para conseguir isso, revisamos a Seção 9.3 e acrescentamos um novo Capítulo (13), introduzindo o aprendizado de máquina baseado em probabilidade. Nossa apresentação das ferramentas de IA estocásticas e sua aplicação ao aprendizado e à linguagem natural agora é mais abrangente.

Nas bases da teoria da probabilidade na teoria dos conjuntos e na contagem, desenvolvemos as noções de probabilidades, variáveis aleatórias e independência. Apresentamos e usamos o teorema de Bayes primeiro com um sintoma e uma doença, e depois em sua forma geral completa. Examinamos as hipóteses que dão suporte ao uso de Bayes e depois apresentamos o método *Bayes ingênuo*. Apresentamos exemplos de raciocínio estocástico, incluindo a análise dos fenômenos da linguagem e o algoritmo de Viterbi. Também introduzimos a ideia de independência condicional que leva às redes Bayesianas de crença, as RBCs, no Capítulo 9.

No Capítulo 13, introduzimos os modelos ocultos de Markov, MOMs, e mostramos seu uso em diversos exemplos. Também apresentamos diversas variantes do MOM, incluindo MOMs autorregressivos e hierárquicos. Apresentamos as redes Bayesianas dinâmicas, as RBDs, e demonstramos seu uso. Discutimos o aprendizado de parâmetro e estrutura, apresentamos o algoritmo de maximização de esperança e demonstramos seu uso com a propagação de crença em ciclos. Por fim, apresentamos os processos de decisão de Markov, os PDMs, e o processo de decisão de Markov parcialmente observável, PDMPO, no contexto de uma extensão da apresentação anterior do aprendizado por reforço.

Incluímos vários outros exemplos de *máquinas probabilísticas de estado finito* e *reconhecedores probabilísticos*, além do uso da *programação dinâmica*, especialmente com medidas estocásticas (o *algoritmo de Viterbi*). Acrescentamos um analisador estocástico do idioma inglês (com base no trabalho de Mark Steedman na University of Edinburgh), além do uso da programação dinâmica com o analisador de Earley.

Tomamos a decisão importante de remover do livro os capítulos sobre Prolog e Lisp. Parte do motivo disso é que estes ocupavam muito espaço. Também acumulamos uma série de algoritmos de IA escritos em Java. Quando acrescentamos o novo Capítulo 13 sobre métodos estocásticos para aprendizado de máquina, resolvemos que o livro estava ficando muito grande e desajeitado. Assim, a sexta edição é menor que a quinta edição original em mais de 150 páginas, e os algoritmos de IA em Prolog, Lisp e Java serão lançados na Sala Virtual. Desde nossos primeiros dias em IA, sempre achamos que a maneira de entender o poder (e as limitações) dos algoritmos de IA é cons-

trutivamente — ou seja, montando-os! Encorajamos nossa geração atual de leitores a fazer exatamente isso: checar os materiais da Sala Virtual para montar e experimentar diretamente os algoritmos apresentados.

Por fim, fizemos a atualização normal de referências e materiais que uma nova edição garante. No Capítulo 16 revisado, retornamos às questões mais profundas sobre a natureza da inteligência e a possibilidade de criar máquinas inteligentes.

## Sexta edição: o conteúdo

**O Capítulo 1 introduz a inteligência artificial**, começando com uma breve história das tentativas de entender a mente e a inteligência na filosofia, na psicologia e em outras áreas de pesquisa. Em um sentido importante, a IA é uma ciência antiga, com suas raízes vindas pelo menos de Aristóteles. Uma apreciação de seu passado é essencial para se conhecer as questões tratadas na pesquisa moderna. Também apresentamos uma visão geral de algumas das áreas de aplicação importantes em IA. Nossa objetivo no Capítulo 1 é oferecer uma base e uma motivação para a teoria e para as aplicações que se seguem.

**Os capítulos 2, 3, 4, 5 e 6 (Parte II) introduzem as ferramentas de pesquisa para a solução de problemas de IA.** Entre eles estão, no Capítulo 2, o cálculo de predicados apresentado tanto como um sistema matemático quanto como uma linguagem de representação para descrever os recursos essenciais de um problema. A busca e os algoritmos e estruturas de dados usados para implementar a busca são introduzidos no Capítulo 3, para organizar a exploração de situações de problema. No Capítulo 4, discutimos o papel essencial da heurística no foco e na restrição da solução de problemas de busca. No Capítulo 5, introduzimos a metodologia estocástica, tecnologia importante para raciocínio em situações de incerteza. No Capítulo 6, apresentamos uma série de arquiteturas de software, incluindo o quadro-negro e os sistemas de produção para a implementação desses algoritmos de busca.

**Os capítulos 7, 8 e 9 compõem a Parte III: representações para IA, solução de problemas com uso intensivo de conhecimento e raciocínio em situações mutáveis e ambíguas.** No Capítulo 7, apresentamos a história em evolução dos esquemas de representação da IA. Começamos com uma discussão de redes baseadas em associação e estendemos esse modelo para incluir teoria, quadros e roteiros de dependência conceitual. Depois, apresentamos um exame profundo de um formalismo em particular, grafos conceituais, enfatizando as questões epistemológicas envolvidas na representação do conhecimento e mostrando como essas questões são tratadas em uma linguagem de representação moderna. Expandindo esse formalismo no Capítulo 14, mostramos como os grafos conceituais podem ser usados para implementar um *front-end* de bancos de dados em linguagem natural. Concluímos o Capítulo 7 com métodos mais modernos para representação, incluindo *Copycat* e arquiteturas orientadas a agentes.

O Capítulo 8 apresenta o sistema especialista baseado em regra, juntamente com o raciocínio baseado em casos e baseado em modelos, incluindo exemplos do programa espacial da NASA. Essas técnicas para a solução de problemas são apresentadas como uma evolução natural do material na Parte II: usa um sistema de produção de expressões de cálculo de predicados para coordenar uma busca em grafo. Terminamos com uma análise dos pontos fortes e fracos de cada um desses métodos para a solução de problemas com uso intensivo de conhecimento.

O Capítulo 9 apresenta modelos para raciocínio com incerteza, bem como o uso de informações não confiáveis. Introduzimos os modelos Bayesianos, redes de crença, Dempster-Shafer, modelos causais e a álgebra da certeza de Stanford para o raciocínio em situações incertas. O Capítulo 9 também contém algoritmos para manutenção da verdade, raciocínio com modelos mínimos, abdução baseada em lógica e o algoritmo da árvore de clique para as redes Bayesianas de crença.

**A Parte IV, que contém os capítulos 10 a 13, é uma apresentação ampla de problemas no aprendizado de máquina.** No Capítulo 10, oferecemos uma visão detalhada dos algoritmos para aprendizado baseado em símbolos, uma área de pesquisa bastante produtiva, que gera uma série de problemas diferentes e técnicas de solução. Esses algoritmos de aprendizado variam em seus objetivos, nos dados de treinamento considerados, em suas estratégias de aprendizado e nas representações do conhecimento que eles empregam. O aprendizado baseado em símbolos inclui indução, aprendizado de conceito, busca no espaço de versão e ID3. O papel do viés indutivo é considerado, generalizações a partir de padrões de dados, além do uso efetivo do conhecimento para aprender a partir de um exemplo único no aprendizado baseado em explicação. O aprendizado por categoria, ou o agrupamento

conceptual, é apresentado com o aprendizado não supervisionado. O aprendizado por reforço, ou a capacidade de integrar a realimentação a partir do ambiente para uma política para tomar novas decisões, encerra o capítulo.

No Capítulo 11, apresentamos as redes neurais, constantemente chamadas de modelos subsimbólicos ou conexionistas de aprendizado. Em uma rede neural, as informações são implícitas na organização e nos pesos de um conjunto de processadores conectados, e o aprendizado envolve uma reorganização e a modificação do peso geral dos nós e da estrutura do sistema. Apresentamos uma série de arquiteturas conexionistas, incluindo o Perceptron, a retropropagação e a contrapropagação. Demonstramos os modelos de Kohonen, Grossberg e Hebbian. Apresentamos o aprendizado associativo, bem como modelos atratores, incluindo exemplos de redes de Hopfield.

Algoritmos genéticos e métodos evolucionários de aprendizado são introduzidos no Capítulo 12. Sob o ponto de vista considerado, o aprendizado é moldado como um processo emergente e adaptativo. Depois de vários exemplos de soluções de problemas baseadas em algoritmos genéticos, introduzimos a aplicação de técnicas genéticas para resolvidores de problemas mais gerais. Estes incluem sistemas classificadores e programação genética. Depois, descrevemos o aprendizado baseado em sociedade, com exemplos de vida artificial, denominada pesquisa *a-life*. Concluímos o capítulo com um exemplo de computação emergente da pesquisa no Santa Fe Institute.

O Capítulo 13 apresenta métodos estocásticos para o aprendizado de máquina. Começamos com uma definição dos modelos ocultos de Markov e depois apresentamos diversas variações importantes, incluindo o MOM autorregressivo e hierárquico. Depois, apresentamos as redes Bayesianas dinâmicas, uma generalização do MOM, e também a capacidade de rastrear sistemas ao longo do tempo. Essas técnicas são úteis para a modelagem das mudanças em ambientes complexos, como é exigido para o raciocínio diagnóstico e prognóstico. Por fim, acrescentamos um componente probabilístico para aprendizado por reforço introduzido inicialmente no Capítulo 10. Isso inclui a apresentação do processo de decisão de Markov (ou PDM) e o processo de decisão de Markov parcialmente observável (ou PDMPO).

A Parte V, que contém os capítulos 14 e 15, apresenta o raciocínio automatizado e o entendimento da linguagem natural. A prova de teorema, geralmente conhecida como raciocínio automatizado, é uma das áreas mais antigas da pesquisa em IA. No Capítulo 14, discutimos os primeiros programas nessa área, incluindo o *Logic Theorist* e o *General Problem Solver*. O foco principal do capítulo são os procedimentos de prova de resolução binária, especialmente resolução por refutações. A inferência mais avançada com hiper-resolução e paramodulação também é apresentada. Por fim, descrevemos o interpretador Prolog como uma cláusula de Horn e um sistema de inferência baseado em resolução, e vemos a computação Prolog como um exemplo do paradigma lógico da programação.

O Capítulo 15 apresenta o entendimento da linguagem natural. Nosso método tradicional para o entendimento da linguagem, exemplificado por muitas das estruturas semânticas apresentadas no Capítulo 7, é complementado com o método estocástico. Entre elas estão os modelos de Markov, as árvores CART, a análise CHART (o algoritmo de Earley), o agrupamento de informações mútuas e a análise baseada em estatística. O capítulo termina com exemplos em que são aplicadas técnicas de linguagem natural à geração de consulta de bancos de dados, um sistema de resumo de texto, bem como o uso do aprendizado de máquina para generalizar os resultados extraídos da WWW.

Por fim, o Capítulo 16 serve como um epílogo para o livro. Ele trata da questão da possibilidade de uma ciência de sistemas inteligentes, e considera os desafios contemporâneos da IA; ele discute as limitações atuais da IA e projeta seu futuro empolgante.

## Usando este livro

A inteligência artificial é um campo vasto e, consequentemente, este é um livro grande. Embora seja necessário mais do que um único semestre para abordar todo o material oferecido, planejamos este livro de modo que diversos caminhos possam ser tomados. Selecionando subconjuntos do material, usamos este livro para cursos de único semestre e de um ano (dois semestres).

Supomos que a maioria dos alunos tenha feito cursos introdutórios em matemática discreta, incluindo cálculo de predicados, teoria de conjunto, contagem e teoria dos grafos. Se isso não for verdade, o professor deverá gastar mais tempo nesses conceitos nas seções “opcionais” no início dos capítulos introdutórios (2.1, 3.1 e 5.1). Também supo-

mos que os alunos fizeram cursos de estruturas de dados, incluindo árvores, grafos e busca recursiva, uso de pilhas, filas e filas de prioridade. Se não fizeram, então devem gastar mais tempo nas seções iniciais dos capítulos 3, 4 e 6.

Em um curso de um trimestre ou de um semestre, passamos rapidamente pelas duas partes iniciais do livro. Com essa preparação, os alunos são capazes de apreciar o material na Parte III. Depois, consideramos o código Prolog, List ou Java na Sala Virtual do livro e pedimos que os estudantes criem muitas das técnicas de representação e de busca da segunda parte do livro. Como alternativa, uma das linguagens, Prolog, por exemplo, pode ser introduzida mais cedo no curso e ser usada para testar as estruturas de dados e técnicas de busca à medida que forem encontradas. Acreditamos que os metainterpretadores apresentados nos materiais de linguagem são muito úteis na criação de resolvedores de problemas baseados em regras e outros com uso intenso de conhecimento. Prolog, Lisp e Java são excelentes ferramentas para a criação de sistemas de entendimento e aprendizado da linguagem natural; essas arquiteturas são apresentadas nas partes II e III, e há exemplos delas na Sala Virtual do livro.

Em um curso de dois semestres ou três trimestres, podemos abordar as áreas de aplicação das partes IV e V, especialmente os capítulos sobre aprendizado de máquina, com detalhes apropriados. Também esperamos um projeto de programação muito mais detalhado dos alunos. Também achamos que é muito importante, no segundo semestre, que os alunos revejam muitas das principais fontes da literatura sobre IA. É essencial que os alunos vejam tanto onde nos encontramos na evolução do empreendimento de IA quanto como chegamos aqui, apreciando as promessas futuras da inteligência artificial. Usamos material da Web para essa finalidade ou selecionamos um conjunto compilado de leituras, como *Computation and Intelligence* (Luger, 1995).

Os algoritmos do nosso livro são descritos usando um pseudocódigo tipo Pascal. Essa notação usa as estruturas de controle da linguagem Pascal juntamente com descrições em linguagem natural de testes e operações. Acrescentamos duas construções úteis às estruturas de controle Pascal. A primeira é uma instrução case modificada que, em vez de comparar o valor de uma variável com rótulos constantes, como no Pascal padrão, permite que cada item seja rotulado com um teste lógico qualquer. O case avalia esses testes em ordem, até que um deles seja verdadeiro, e depois realiza a ação associada; todas as outras ações são ignoradas. Aqueles que estão acostumados com Lisp notarão que isso tem a mesma semântica da instrução cond da linguagem Lisp.

O outro acréscimo à nossa linguagem de pseudocódigo é uma instrução return, que recebe um argumento e pode aparecer em qualquer lugar dentro de um procedimento ou função. Quando o return é encontrado, ele faz com que o programa saia imediatamente da função, retornando seu argumento como resultado. Fora essas modificações, usamos a estrutura da linguagem Pascal, contando com as descrições em linguagem natural, para esclarecer os algoritmos.

## Agradecimentos

---

Embora eu seja o único autor da sexta edição, este livro sempre foi o produto de meus esforços como professor de ciência da computação, psicologia e linguística na University of New Mexico, juntamente com meus colegas de faculdade e profissionais, alunos de pós-graduação e amigos. A sexta edição também é produto de muitos leitores que me remeteram comentários, correções e sugestões. O livro continuará dessa forma, refletindo o esforço *comunitário*; consequentemente, continuarei a usar os pronomes *nós* e *nossa* ao apresentar o material.

Agradeço a Bill Stubblefield, coautor das três primeiras edições, por mais de vinte anos de contribuições, mas, ainda mais importante do que isso, por sua amizade. Também agradeço aos muitos revisores que ajudaram a desenvolver esta e outras edições anteriores. Entre eles estão Dennis Bahler, Leonardo Bottaci, Skona Brittain, Philip Chan, Peter Collingwood, Mehdi Dastani, John Donald, Sarah Douglas, Christophe Giraud-Carrier, Andrew Kosoresow, Terran Lane, Chris Malcolm, Ray Mooney, Marek Perkowski, Barak Pearlmutter, Dan Pless, Bruce Porter, Stuart Shapiro, Julian Richardson, Jude Shavlik, John Sheppard, Carl Stern, Leon van der Torre, Marco Valtorta e Bob Veroff. Também agradeço pelas inúmeras sugestões e comentários enviados diretamente por e-mail pelos leitores. Por fim, agradeço a Chris Malcolm, Brendan McGonnigle e Akasha Tang, que revisaram o Capítulo 16.

Dos nossos colegas na UNM, agradecemos a Dan Pless, Nikita Sakhnenko, Roshan Rammohan e Chayan Chakrabarti por seu papel importante no desenvolvimento do material dos capítulos 5, 9 e 13; a Joseph Lewis por seus esforços nos capítulos 9 e 16; a Carl Stern por sua ajuda no desenvolvimento do Capítulo 11, sobre aprendizado cone-

xionista; a Bob Veroff por sua análise do material sobre raciocínio automatizado no Capítulo 14; e a Jared Saia, Stan Lee e Paul dePalma por ajudar com os métodos estocásticos de entendimento da linguagem natural, no Capítulo 15.

Agradecemos à Academic Press pela permissão para reimprimir grande parte do material do Capítulo 11; este apareceu inicialmente no livro *Cognitive Science: The Science of Intelligent Systems* (Luger, 1994). Por fim, agradecemos a mais de duas décadas de alunos que usaram as várias versões deste livro e do software na UNM por sua ajuda na expansão de nossos horizontes, além da remoção de erros genéricos e de digitação.

Agradecemos aos nossos muitos amigos na Benjamin-Cummings, Addison-Wesley-Longman e Pearson Education por seu apoio e encorajamento na realização da tarefa de escrita de nossas seis edições, especialmente a Alan Apt na ajuda com a primeira edição; a Lisa Moller e Mary Tudor por sua ajuda na segunda; a Victoria Henderson, Louise Wilson e Karen Mosman por sua assistência na terceira; a Keith Mansfield, Karen Sutherland e Anita Atkinson pelo apoio prestado na quarta; a Keith Mansfield, Owen Knight, Anita Atkinson e Mary Lince por sua ajuda na quinta edição; e a Simon Plumtree, Matt Goldstein, Joe Vetere e Sarah Milmore por sua ajuda na sexta. Linda Cicarella, da University of New Mexico, ajudou a preparar muitas das figuras.

Agradecemos a Thomas Barrow, artista reconhecido internacionalmente e professor (emérito) de artes da University of New Mexico, que criou os fotogramas para este livro.

A inteligência artificial é uma disciplina entusiasmante e recompensadora; que você possa aproveitar seu estudo enquanto aprecia o poder e os desafios desse estudo.

George Luger  
1º de janeiro de 2008  
Albuquerque

## Agradecimentos – edição brasileira

Agradecemos a todos os profissionais que trabalharam na edição deste livro, em especial aos professores Luciano Antônio Digiampietri (USP Leste), Paulo André Lima de Castro (ITA), Andréa Iabradi Tavares (UFOP), Cristiano Pitangui (UFVJM) e Rogério Martins Gomes (CEFET/MG) pela rigorosa avaliação do material.



### Site de apoio do livro

Na Sala Virtual deste livro (<[sv.pearson.com.br](http://sv.pearson.com.br)>), professores e estudantes podem acessar os seguintes materiais adicionais 24 horas:

**Para professores:**

- Apresentações em PowerPoint;
- Manual de soluções (em inglês).

*Esse material é de uso exclusivo para professores e está protegido por senha. Para ter acesso a ele, os professores que adotam o livro devem entrar em contato com seu representante Pearson ou enviar e-mail para [universitarios@pearson.com](mailto:universitarios@pearson.com).*

**Para estudantes:**

- Manual complementar de Lisp, Prolog e Java.



# *In inteligência artificial: raízes e escopo*

*Parafraseando Sancho Pança, tudo deve ter um inicio; e esse inicio deve estar ligado a algo que já existiu antes. Para os hindus, o mundo é sustentado por um elefante, mas o elefante se encontra apoiado em cima de uma tartaruga. Deve-se humildemente admitir que a invenção não consiste em se criar a partir do nada, mas sim a partir do caos; em primeiro lugar, deve-se dispor dos materiais necessários...*

—MARY SHELLEY, *Frankenstein*

## **In inteligência artificial: uma tentativa de definição**

*A inteligência artificial (IA) pode ser definida como o ramo da ciência da computação que se ocupa da automação do comportamento inteligente.* Essa definição é particularmente apropriada a este livro porque enfatiza nossa convicção de que a IA faz parte da ciência da computação e que, desse modo, deve ser baseada em princípios teóricos e aplicados sólidos nesse campo. Esses princípios incluem as estruturas de dados usadas na representação do conhecimento, os algoritmos necessários para aplicar esse conhecimento e as linguagens e técnicas de programação usadas em sua implementação.

Entretanto, essa definição sofre com o fato de que a própria inteligência não é muito bem definida ou compreendida. Embora a maioria das pessoas esteja certa de que reconhece o comportamento inteligente quando o vê, não é certo que alguém possa chegar perto de definir a inteligência de um modo que seria específico o suficiente para ajudar na avaliação de um programa de computador supostamente inteligente, enquanto ainda captura a vitalidade e a complexidade da mente humana.

Como resultado da assombrosa tarefa de criar uma inteligência genérica, os pesquisadores da IA normalmente assumem o papel dos engenheiros ao moldarem artefatos inteligentes específicos. Estes geralmente vêm na forma de ferramentas de diagnóstico, prognóstico ou visualização, que permitem que seus usuários humanos realizem tarefas complexas. Alguns exemplos dessas ferramentas incluem os modelos ocultos de Markov para a compreensão da linguagem natural, sistemas de raciocínio automatizado para provar novos teoremas na matemática, redes Bayesianas dinâmicas para rastrear sinais através de redes corticais e visualização de padrões de dados de expressão de gene, conforme vistos nas aplicações da Seção 1.2.

O problema de *definir* o campo inteiro da inteligência artificial é semelhante ao de definir a própria inteligência: ela é uma única faculdade ou é apenas um nome para uma coleção de capacidades distintas e não relacionadas? Até que ponto a inteligência é aprendida e não existe desde o nascimento? O que acontece exatamente quando ocorre o aprendizado? O que é criatividade? O que é intuição? A inteligência pode ser deduzida do comportamento observável ou ela requer evidências de um mecanismo interno em particular? Como o conhecimento é representado

no tecido nervoso de um ser humano e que lições isso nos traz para o projeto de máquinas inteligentes? O que é autopercepção? Que papel ela desempenha na inteligência? Além disso, o conhecimento sobre a inteligência humana é necessário para construir um programa inteligente, ou uma técnica estritamente de “engenharia” é suficiente para tratar do problema? É possível conseguir inteligência em um computador, ou uma entidade inteligente requer a riqueza de sensações e experiências que só poderiam ser encontradas em uma existência biológica?

Essas são perguntas não respondidas que têm ajudado a modelar os problemas e as metodologias de solução que constituem o núcleo da IA moderna. De fato, parte da atração da inteligência artificial é que ela oferece uma ferramenta única e poderosa para explorar exatamente essas perguntas. A IA oferece um meio e um banco de ensaio para teorias da inteligência: essas teorias podem ser indicadas na linguagem de programação de computadores e podem, consequentemente, ser testadas e verificadas pela execução desses programas em um computador real.

Por esses motivos, nossa definição inicial de inteligência artificial é um pouco ambígua. No máximo, ela só tem levado a mais perguntas e à noção paradoxal de um campo de estudos cujas principais metas incluem sua própria definição. No entanto, essa dificuldade em chegar a uma definição exata de IA é totalmente compreensível. A inteligência artificial ainda é uma disciplina jovem, e sua estrutura, suas considerações e seus métodos não estão definidos tão claramente quanto aqueles de uma ciência mais madura, como a física.

A inteligência artificial sempre esteve mais preocupada com a expansão das capacidades da ciência da computação do que com a definição de seus limites. Manter essa exploração baseada em princípios teóricos sólidos é um dos desafios que os pesquisadores de IA, em geral, e este livro, em particular, enfrentam.

Em virtude do seu escopo e da sua ambição, a inteligência artificial não tem uma definição simples. Até o momento, simplesmente a definimos como *a coleção de problemas e metodologias estudada pelos pesquisadores de inteligência artificial*. Essa definição pode parecer tola e sem sentido, mas ela reforça um argumento importante: a inteligência artificial, como toda ciência, é um empreendimento humano e talvez seja mais bem entendida nesse contexto.

Há várias razões para que qualquer ciência, incluindo a IA, se preocupe com um certo conjunto de problemas e desenvolva um conjunto de técnicas específicas para enfrentar esses problemas. No Capítulo 1, uma breve história da inteligência artificial e das pessoas e hipóteses que a têm modelado explicará por que certos grupos de perguntas chegaram a dominar esse campo e por que os métodos discutidos neste livro foram escolhidos para a sua solução.

# IA: história e aplicações

*Por natureza, todos os homens desejam conhecer...*

—ARISTÓTELES, frase inicial de *Metaphysics*

*Ouvi o resto, e ainda mais admirareis o valor das artes e indústrias que dei aos mortais. Antes de mim — e este foi meu maior benefício —, quando atacados por qualquer enfermidade, nenhum socorro para eles havia, quer em alimento, quer em poções, bálsamos ou medicamentos: eles pereciam. Hoje, graças às salutares composições que lhes ensinei, todos os males são curáveis.*

*Fui eu quem tornou visíveis aos olhos dos homens os sinais flamejantes do céu que havia antes do escurecer. Chega deste assunto. Abaixo da terra, as bênçãos escondidas do homem, cobre, ferro, prata e ouro — haverá alguém para alegar que as descobriu antes de mim? Ninguém, com certeza, que esteja disposto a dizer a verdade sobre isso. Uma breve frase explicará a história: toda a arte que os mortais possuem vem de Prometeu.*

—ÉSQUILO, *Prometheus Bound*

## 1.1 Do Éden ao ENIAC: posicionamentos em relação à inteligência, ao conhecimento e à astúcia humana

Prometeu fala dos frutos de sua transgressão contra os deuses do Olimpo: sua finalidade não foi simplesmente roubar o fogo para a raça humana, mas também iluminar a humanidade através do dom da inteligência ou intelecto: a *mente racional*. Essa inteligência forma a base para toda a tecnologia humana e, em última instância, toda a civilização humana. O trabalho de Ésquilo, o dramaturgo grego clássico, ilustra uma consciência profunda e antiga do extraordinário poder do conhecimento. A inteligência artificial, em seu interesse muito direto no dom de Prometeu, tem sido aplicada a todas as áreas de seu legado — medicina, psicologia, biologia, astronomia, geologia — e a muitas áreas de empreendimento científico que nem Ésquilo poderia ter imaginado.

Embora a ação de Prometeu livrasse a humanidade da doença da ignorância, ela também lhe rendeu a ira de Zeus. Indignado por esse roubo do conhecimento que anteriormente pertencia apenas aos deuses do Olimpo, Zeus ordenou que Prometeu fosse acorrentado a uma rocha bruta para sofrer as devastações dos elementos pela eternidade. A noção de que os esforços humanos para ganhar conhecimento constituem uma transgressão contra as leis de Deus ou da natureza está profundamente arraigada ao pensamento ocidental. Ela é a base da história do Éden e aparece nos livros de Dante e Milton. Tanto Shakespeare quanto as antigas tragédias gregas representaram a ambição intelectual como causa de desastre. A crença de que o desejo por conhecimento por fim deve levar ao

desastre persistiu por toda a história, perdurando pela Renascença, pela Era do Iluminismo e até mesmo pelos avanços científicos e filosóficos dos séculos XIX e XX. Assim, não devemos ficar surpresos com o fato de a inteligência artificial inspirar tanta controvérsia em círculos acadêmicos e populares.

Na realidade, em vez de dissipar esse antigo medo das consequências da ambição intelectual, a tecnologia moderna apenas fez essas consequências parecerem mais prováveis, até mesmo iminentes. As lendas de Prometeu, de Eva e de Fausto foram recontadas na linguagem da sociedade tecnológica. Na introdução de *Frankenstein*, com o interessante subtítulo de *O moderno Prometeu*, Mary Shelley escreve:

Muitas e longas foram as conversas entre Lord Byron e Shelley, às quais fui uma ouvinte devota e silenciosa. Durante uma delas, diversas doutrinas filosóficas foram discutidas, e entre outras, a natureza do princípio da vida, e se havia qualquer probabilidade de que sequer fosse descoberta e comunicada. Elas falavam dos experimentos do Dr. Darwin (não me refiro ao que o doutor realmente fez ou disse que fez, mas, conforme meu propósito, no que se falava que ele teria feito), que preservaram um pedaço de vidro até que, por algum meio extraordinário, ele começou a se mover voluntariamente. Afinal, não é dessa forma que a vida devia ser criada. Talvez um corpo fosse reanimado; o galvanismo deu sinal disso: talvez as partes componentes de uma criatura possam ser fabricadas, reunidas e dotadas de calor vital (Butler, 1998).

Mary Shelley nos mostra em que grau avanços científicos como a obra de Darwin e a descoberta da eletricidade têm convencido até mesmo leigos de que o funcionamento da natureza não era segredo divino, mas poderia ser desmembrado e compreendido sistematicamente. O monstro de Frankenstein não é produto de encantamentos xamanistas ou transações inaudíveis com o submundo: é montado de componentes separadamente “fabricados” e infundidos com a força vital da eletricidade. Embora a ciência do século XIX fosse inadequada para estabelecer o objetivo de compreender e criar um agente totalmente inteligente, ela afirmou a noção de que os mistérios da vida e do intelecto poderiam ser trazidos à luz da análise científica.

### 1.1.1 Uma breve história dos fundamentos da IA

Quando Mary Shelley finalmente, e talvez irrevogavelmente, uniu a ciência moderna ao mito de Prometeu, os fundamentos filosóficos do trabalho moderno em inteligência artificial já haviam sido desenvolvidos durante milhares de anos. Embora as questões morais e culturais levantadas pela inteligência artificial sejam interessantes e importantes, nossa introdução se preocupa mais com a herança intelectual da IA. O ponto de partida lógico para essa história é o gênio Aristóteles ou, como a *Divina Comédia* de Dante se refere a ele, “o mestre dos que sabem”. Aristóteles reuniu as descobertas, as maravilhas e os temores da antiga tradição grega com a análise cuidadosa e o pensamento disciplinado que se tornaram o padrão para a ciência mais moderna.

Para Aristóteles, o aspecto mais fascinante da natureza era a mudança. Em *Física*, ele definiu sua “filosofia da natureza” como o “estudo das coisas que mudam”. Ele distinguiu a *matéria* e a *forma* das coisas: uma escultura é modelada a partir do *material* bronze e tem a *forma* de um ser humano. A mudança ocorre quando o bronze é moldado em uma nova forma. A distinção matéria/forma fornece uma base filosófica para noções modernas como computação simbólica e abstração de dados. Na computação (até mesmo com números), manipulamos padrões que são as formas de material eletromagnético, com as mudanças de forma desses padrões representando aspectos do processo de solução. A abstração da forma a partir do meio de sua representação não apenas permite que essas formas sejam manipuladas computacionalmente, mas também oferece a promessa de uma teoria de estruturas de dados, núcleo da moderna ciência da computação. Ela também dá suporte à criação de uma inteligência “artificial”.

Em sua obra *Metafísica*, começando com as palavras “Todos os homens, por natureza, desejam conhecer”, Aristóteles desenvolveu uma ciência de coisas que nunca mudam, incluindo sua cosmologia e sua teologia. Mais relevante à inteligência artificial, porém, foi a epistemologia de Aristóteles, ou a análise de como os humanos “conhecem” seu mundo, discutida em sua obra *Lógica*. Aristóteles se referia à lógica como o “instrumento” (*organon*), porque percebeu que o estudo do próprio pensamento era a base de todo o conhecimento. Em *Lógica*, ele investiga se certas proposições podem ser “verdadeiras” porque estão relacionadas com outras coisas que são

sabidamente “verdadeiras”. Assim, se sabemos que “todos os homens são mortais” e que “Sócrates é um homem”, então podemos concluir que “Sócrates é mortal”. Esse argumento é um exemplo do que Aristóteles se referia como um silogismo, usando a forma dedutiva *modus ponens*. Embora a axiomatização formal do raciocínio precisasse de outros dois mil anos para o seu amadurecimento completo nos trabalhos de Gottlob Frege, Bertrand Russell, Kurt Gödel, Alan Turing, Alfred Tarski e outros, as suas raízes remontam a Aristóteles.

O pensamento do Renascimento, construído sobre a tradição grega, iniciou a evolução de um modo diferente e poderoso de pensar a respeito da humanidade e sua relação com o mundo natural. A ciência começou a substituir o misticismo como um meio de entender a natureza. Os relógios e, eventualmente, horários de fábricas, sobrepujaram os ritmos da natureza para milhares de moradores urbanos. A maioria das ciências sociais e físicas tem sua origem na noção de que processos, quer sejam eles naturais quer sejam artificiais, poderiam ser analisados e entendidos matematicamente. Em particular, os cientistas e filósofos perceberam que o próprio pensamento, o modo como o conhecimento era representado e manipulado na mente humana, era um tema difícil, porém essencial para o estudo científico.

Talvez o principal evento no desenvolvimento da visão do mundo moderno foi a revolução de Copérnico, a substituição do antigo modelo do universo centralizado na Terra com a ideia de que a Terra e outros planetas estão na realidade em órbitas em torno do Sol. Depois de séculos de uma ordem “óbvia”, em que a explicação científica da natureza do cosmos foi condizente com os ensinos da religião e do bom senso, um modelo drasticamente diferente e não tão óbvio foi proposto para explicar os movimentos dos corpos celestes. Talvez, pela primeira vez, nossas ideias sobre o mundo foram vistas como fundamentalmente distintas de sua aparência. Essa separação entre a mente humana e sua realidade ao redor, entre as ideias sobre as coisas e as próprias coisas, é essencial para o estudo moderno da mente e de sua organização. Essa brecha foi ampliada pelos escritos de Galileu, cujas observações científicas contradisseram ainda mais as verdades “óbvias” sobre o mundo natural e cujo desenvolvimento da matemática como uma ferramenta para descrever esse mundo enfatizaram a distinção entre o mundo e nossas ideias a respeito dele. É dessa brecha que surgiu a noção moderna de mente: a introspecção se tornou um motivo comum na literatura, os filósofos começaram a estudar a epistemologia e a matemática, e a aplicação sistemática do método científico competiu com os cinco sentidos como ferramenta para o conhecimento do mundo.

Em 1620, *Novum Organum*, de Francis Bacon, ofereceu um conjunto de técnicas de busca para essa metodologia científica emergente. Com base na ideia Aristotélica e Platônica de que a “forma” de uma entidade era equivalente à soma de suas “características” necessárias e suficientes, Bacon articulou um algoritmo para determinar a essência de uma entidade. Primeiro, ele criava uma coleção organizada de todas as ocorrências da entidade, enumerando as características de cada uma em uma tabela. Depois, ele coletava uma lista semelhante de ocorrências negativas da entidade, focalizando especialmente instâncias próximas da entidade, ou seja, aquelas que se desviavam da “forma” da entidade por características únicas. Depois, Bacon tenta — essa etapa não fica totalmente clara — criar uma lista sistemática de todas as características essenciais à entidade, ou seja, aquelas que são comuns a todas as ocorrências positivas da entidade e que faltam nas ocorrências negativas.

É interessante ver uma forma da abordagem de Francis Bacon para o aprendizado do conceito refletida nos algoritmos de IA modernos em “Busca em espaço de versões”, na Seção 10.2. Uma extensão dos algoritmos de Bacon também fazia parte de um programa de IA para o aprendizado por descoberta, adequadamente denominado *Bacon* (Langley et al., 1981). Esse programa foi capaz de induzir muitas leis físicas a partir de coleções de dados relacionados aos fenômenos. Também é interessante observar que a questão de se um algoritmo de uso geral era possível para produzir provas científicas aguardou os desafios do matemático do princípio do século XX Hilbert (sua obra *Entscheidungsproblem*) e a resposta do gênio moderno Alan Turing (sua *Máquina de Turing* e provas de *computabilidade* e do *problema da parada*); ver Davis et al. (1976).

Embora a primeira máquina de calcular, o ábaco, tenha sido criada pelos chineses no século XXVI a.C., a maior mecanização dos processos algébricos esperou as habilidades dos europeus do século XVII. Em 1614, o matemático escocês, John Napier, criou logaritmos, as transformações matemáticas que permitiram que a multiplicação e o uso de expoentes fossem reduzidos à adição e à multiplicação. Napier também criou o dispositivo chamado “ossos de Napier”, que foi usado para representar valores de estouro para operações aritméticas. O dispositivo foi mais tarde usado por Wilhelm Schickard (1592-1635), um matemático e sacerdote alemão de Tübingen, que em 1623 inventou um *Relógio Calculador* para realizar adição e subtração. Essa máquina registrava o estouro de seus cálculos pelo soar de um relógio.

Outra máquina de cálculo famosa foi a *Pascaline* que Blaise Pascal, filósofo e matemático francês, criou em 1642. Embora os mecanismos de Schickard e Pascal fossem limitados à adição e à subtração — incluindo estouros e empréstimos —, eles mostraram que processos que, segundo se imaginava, exigiam pensamento e habilidade humana poderiam ser totalmente automatizados. Conforme Pascal citou posteriormente em *Pensées* (Pensamentos) (1670), “A máquina aritmética produz efeitos que chegam mais perto do pensamento que todas as ações dos animais”.

Os sucessos de Pascal com máquinas de cálculo inspirou Gottfried Wilhelm von Leibniz, em 1694, a concluir uma máquina funcional que se tornou conhecida como a *Roda de Leibniz*. Ela integrava um tambor móvel e uma manivela para impulsionar as rodas e os cilindros que realizavam as operações mais complexas de multiplicação e divisão. Leibniz também era fascinado pela possibilidade de uma lógica automatizada para provas de proposições. Retornando ao algoritmo de especificação de entidade de Bacon, onde conceitos eram caracterizados como a coleção de suas características necessárias e suficientes, Leibniz imaginou uma máquina que poderia usar com essas características para produzir conclusões logicamente corretas. Leibniz (1887) também idealizou uma máquina que refletia ideias modernas de inferência dedutiva e prova, pela qual a produção de conhecimento científico poderia se tornar automatizada, um cálculo para raciocínio.

Os séculos XVII e XVIII também viram muita discussão de questões epistemológicas; talvez a mais influente tenha sido o trabalho de René Descartes, uma figura central no desenvolvimento dos conceitos modernos do pensamento e teorias da mente. Em sua obra *Meditações*, Descartes (1680) tentou achar uma base para a realidade puramente por meio da introspecção. Rejeitando sistematicamente a entrada de seus sentidos como não confiáveis, Descartes foi forçado a duvidar até mesmo da existência do mundo físico e foi deixado apenas com a realidade do pensamento; até mesmo sua própria existência teve de ser justificada em termos de pensamento: “*Cogito ergo sum*” (Penso, logo existo). Depois que estabeleceu sua própria existência puramente como uma entidade de pensamento, Descartes deduziu a existência de Deus como um criador essencial e por fim reafirmou a realidade do universo físico como a criação necessária de um Deus generoso.

Podemos fazer duas observações: primeiro, a divisão entre a mente e o mundo físico se tornou tão completa que o processo de pensamento podia ser discutido isoladamente de qualquer entrada sensorial ou questão material específica; segundo, a conexão entre a mente e o mundo físico era tão tênue que exigia a intervenção de um Deus generoso para dar suporte ao conhecimento confiável do mundo físico! Essa visão da dualidade entre a mente e o mundo físico forma a base de todo o pensamento de Descartes, incluindo seu desenvolvimento da geometria analítica. De que outra forma ele poderia ter unificado um ramo da matemática aparentemente tão mundano como a geometria com uma estrutura matemática tão abstrata como a álgebra?

Por que incluímos essa discussão entre mente/corpo em um livro sobre inteligência artificial? Há duas consequências dessa análise que são essenciais para um empreendimento da IA:

1. Ao tentar separar mente e mundo físico, Descartes e outros pensadores relacionados estabeleceram que a estrutura das ideias sobre o mundo não foi necessariamente a mesma que a estrutura de seu tema. Isso está por trás da metodologia da IA, juntamente com os campos da epistemologia, da psicologia, de grande parte da matemática mais avançada e da maior parte da literatura moderna: os processos mentais têm uma existência própria, obedecem às suas próprias leis e podem ser estudados por si próprios.
2. Uma vez que a mente e o corpo são separados, os filósofos acharam necessário encontrar um meio de reconectar os dois, pois a interação entre os planos mental, *res cogitans*, e físico, *res extensa*, de Descartes é essencial para a existência humana.

Embora milhões de palavras tenham sido escritas sobre esse *problema mente-corpo*, e diversas soluções propostas, ninguém explicou com sucesso as interações óbvias entre os estados mentais e as ações físicas, podendo afirmar uma diferença fundamental entre eles. A resposta mais bem aceita para esse problema, e aquela que oferece uma base essencial para o estudo da IA, afirma que a mente e o corpo não são, de modo nenhum, entidades fundamentalmente diferentes. Nessa visão, os processos mentais são realmente alcançados por sistemas físicos como os cérebros (ou computadores). Os processos mentais, como os processos físicos, por fim podem ser caracterizados por meio da matemática formal. Ou então, como reconhecido em *Leviatā*, do filósofo inglês do século XVII Thomas Hobbes (1651), “por raciocínio, quero dizer cálculo”.

### 1.1.2 IA e as tradições racionalista e empirista

Questões de pesquisa modernas em inteligência artificial, como em outras disciplinas científicas, são formadas e evoluem a partir de uma combinação de pressões históricas, sociais e culturais. Duas das pressões mais proeminentes para a evolução da IA são as tradições empirista e racionalista na filosofia.

A tradição racionalista, como vimos na seção anterior, teve um proponente antigo em Platão e foi continuada pelos escritos de Pascal, Descartes e Leibniz. Para o racionalista, o mundo exterior é reconstruído a partir de ideias claras e distintas da matemática. Uma crítica a esse enfoque dualista é o desligamento forçado dos sistemas representativos de seu campo de referência. A questão é se o significado atribuído a uma representação pode ser definido independentemente de suas condições de aplicação. Se o mundo é diferente de nossas crenças sobre ele, nossos conceitos e símbolos criados ainda podem ter significado?

Muitos programas de IA têm muita coisa desse aspecto racionalista. Os primeiros planejadores de robôs, por exemplo, descreviam seu domínio de aplicação ou “mundo” como conjuntos de declarações de cálculo de predicados, e depois um “plano” para ação era criado provendo teoremas sobre esse “mundo” (Fikes et al., 1972; ver também a Seção 8.4). O *Physical Symbol System Hypothesis*, de Newell e Simon (introdução à Parte II e ao Capítulo 16) é visto por muitos como o arquétipo desse enfoque na IA moderna. Duras críticas elegem esse viés racionalista como parte da falha da IA em resolver tarefas complexas, como o entendimento das linguagens humanas (Searle, 1980; Winograd e Flores, 1986; Brooks, 1991a).

Em vez de afirmar como “real” o mundo de ideias claras e distintas, os empiristas continuam a nos lembrar que “nada entra na mente senão por meio dos sentidos”. Essa restrição leva a mais questionamentos de como os seres humanos possivelmente podem perceber conceitos gerais ou as formas puras da caverna de Platão (Platão, 1961). Aristóteles foi um antigo empirista, enfatizando, em *De Anima*, as limitações do sistema perceptivo humano. Empiristas mais modernos, especialmente Hobbes, Locke e Hume, enfatizam que o conhecimento deve ser explicado por meio de uma psicologia introspectiva, porém empírica. Eles distinguem dois tipos de fenômenos mentais: a percepção, de um lado, e o pensamento, a memória e a imaginação, de outro. O filósofo escocês David Hume, por exemplo, distingue *impressões* de *ideias*. As impressões são dinâmicas e vívidas, refletindo a presença e a existência de um objeto externo e não sujeito ao controle voluntário, a *qualia* de Dennett (2005). As ideias, por outro lado, são menos vívidas e detalhadas e mais sujeitas ao controle voluntário da pessoa.

Dada essa distinção entre impressões e ideias, como surge o conhecimento? Para Hobbes, Locke e Hume, o mecanismo explanatório fundamental é a *associação*. Determinadas propriedades perceptivas são associadas por meio da experiência repetitiva. Essa associação repetitiva cria uma disposição na mente para associar as ideias correspondentes, um precursor do enfoque comportamentalista do século XX. Uma propriedade fundamental dessa explicação é apresentada com o ceticismo de Hume. A explicação puramente descritiva de Hume para as origens das ideias não pode, segundo ele, dar suporte à crença na causalidade. Até mesmo o uso da lógica e da indução não pode ser apoiado racionalmente nessa epistemologia empirista radical.

Em *An Inquiry Concerning Human Understanding* (Uma investigação sobre o entendimento humano) (1748), o ceticismo de Hume se estendeu para a análise dos milagres. Embora Hume não tratasse da natureza dos milagres diretamente, ele questionou a crença no miraculoso baseada em testemunhos. Esse ceticismo, é claro, foi visto como uma ameaça direta pelos crentes na Bíblia, bem como por muitos outros perpetuadores de tradições religiosas. O Reverendo Thomas Bayes era um matemático e um ministro religioso. Um de seus trabalhos, denominado *Essay towards Solving a Problem in the Doctrine of Chances* (Ensaio para a solução de um problema na doutrina das probabilidades) (1763), abordou os questionamentos de Hume matematicamente. O teorema de Bayes demonstra formalmente como podemos, por meio do aprendizado das correlações dos efeitos de ações, determinar a probabilidade de suas causas.

A explicação associativa do conhecimento desempenha um papel significativo no desenvolvimento das estruturas e dos programas representativos da IA, por exemplo, na organização de memória com *redes semânticas* e *MOPS* e no trabalho na compreensão de linguagem natural (ver seções 7.0, 7.1 e Capítulo 15). As explicações associativas possuem influências importantes do aprendizado de máquina, especialmente com redes conexionistas (ver seções 10.6, 10.7 e Capítulo 11). O associacionismo também desempenha um papel importante na psicologia

cognitiva, incluindo os *esquemas* de Bartlett e Piaget, bem como na inteira confiança da tradição comportamentalista (Luger, 1994). Por fim, com ferramentas de IA para análise estocástica, incluindo a *rede Bayesiana de crença* (RBC; ou BBN, da sigla em inglês Bayesian belief network) e suas extensões atuais aos sistemas completos de Turing de primeira ordem para modelagem estocástica, as teorias associativas encontraram uma base matemática sólida e um poder de expressão maduro. As ferramentas Bayesianas são importantes para a pesquisa, incluindo diagnósticos, aprendizado de máquina e compreensão de linguagem natural (ver capítulos 5 e 13).

Immanuel Kant, filósofo alemão treinado na tradição racionalista, foi fortemente influenciado pelos escritos de Hume. Como resultado, ele começou a síntese moderna dessas duas tradições. Para Kant, o conhecimento contém duas energias colaborativas, um componente *a priori* vindo da razão do sujeito, juntamente com um componente *a posteriori* vindo da experiência ativa. A experiência só é significativa a partir da contribuição do sujeito. Sem uma forma de organização ativa proposta pelo sujeito, o mundo não seria nada mais que sensações transitórias passageiras. Por fim, no nível do discernimento, Kant afirma que imagens ou representações passageiras são ligadas pelo sujeito ativo e tomadas como as diversas aparições de uma identidade, de um “objeto”. O realismo de Kant iniciou o empreendimento moderno de psicólogos como Bartlett, Brunner e Piaget. O trabalho de Kant influencia a iniciativa moderna de IA do aprendizado de máquina (Seção IV), bem como o desenvolvimento contínuo de uma epistemologia construtivista (ver Capítulo 16).

### 1.1.3 Desenvolvimento da lógica formal

Uma vez que o pensamento começou a ser considerado como uma forma de cálculo, sua formalização e eventual mecanização foram as etapas seguintes óbvias. Como observamos na Seção 1.1.1, Gottfried Wilhelm von Leibniz, com *Calculus Philosophicus*, introduziu o primeiro sistema de lógica formal, além de propor uma máquina para automatizar suas tarefas (Leibniz, 1887). Além disso, as etapas e os estágios dessa solução mecânica podem ser representadas como movimento pelos estados de uma árvore ou grafo. Leonhard Euler, no século XVIII, com sua análise da “conectividade” das pontes unindo as margens de rios e ilhas da cidade de Königsberg (veja a introdução do Capítulo 3), introduziu o estudo das representações que podem capturar de forma abstrata a estrutura dos relacionamentos no mundo, bem como as etapas distintas dentro de um cálculo sobre esses relacionamentos (Euler, 1735).

A formalização da teoria dos grafos também possibilitou a *busca no espaço de estados*, uma ferramenta conceitual importante da inteligência artificial. Podemos usar grafos para modelar a estrutura mais profunda de um problema. Os nós de um *grafo de espaço de estados* representam estágios possíveis da solução de um problema; os arcos do grafo representam inferências, movimentos em um jogo ou outros passos na solução de um problema. Resolver o problema é um processo de buscar, no grafo de espaço de estados, um caminho para uma solução (introdução à Parte II e Capítulo 3). Por descreverem o espaço completo de soluções do problema, os grafos de espaço de estado são uma ferramenta poderosa para se medir a estrutura e a complexidade de problemas e analisar a eficiência, a correção e a generalidade de estratégias de solução.

Como um dos criadores da ciência da pesquisa operacional e projetista das primeiras máquinas computacionais mecânicas programáveis, Charles Babbage, matemático do século XIX, pode ser também considerado um dos pioneiros da inteligência artificial (Morrison e Morrison, 1961). O *motor diferencial* de Babbage era uma máquina de propósito específico para calcular os valores de certas funções polinomiais e foi o precursor de seu *motor analítico*. Esse motor, concebido, mas não construído com sucesso durante a sua vida, era uma máquina computacional programável de propósito geral que anteviu muitas das suposições arquitetônicas que formam a base do computador moderno.

Descrevendo o motor analítico, Ada Lovelace (1961), amiga, incentivadora e colaboradora de Babbage, disse:

Podemos afirmar muito apropriadamente que o motor analítico tece padrões algébricos do mesmo modo que o tear de Jacquard tece flores e folhas. Aqui, nos parece, reside muito mais originalidade do que se poderia atribuir ao motor diferencial.

A inspiração de Babbage era o desejo de aplicar a tecnologia de seu tempo para libertar o homem da árdua tarefa dos cálculos aritméticos. Nesse sentido, bem como dentro da sua concepção de computadores como dispositivos mecânicos, Babbage pensava puramente em termos do século XIX. O seu motor analítico, entretanto, também incluía muitas noções modernas, como a separação entre memória e processador, o *depósito* e o *moinho*, em termos de Babbage, o conceito de uma máquina digital em vez de uma máquina analógica e a noção de programabilidade baseada na execução de uma série de operações codificadas em cartões de cartolina. A característica mais impressionante na descrição de Ada Lovelace, e em toda a obra de Babbage, é o seu tratamento dos “padrões” de relacionamentos algébricos como entidades que podem ser estudadas, caracterizadas e, enfim, implementadas e manipuladas mecanicamente sem a preocupação com valores particulares que são, por fim, passados pelo “moinho” da máquina de calcular. Esse é um exemplo da implementação “da abstração e da manipulação da forma” que foi descrita pela primeira vez por Aristóteles e Liebniz.

O objetivo de se criar uma linguagem formal para o pensamento também aparece na obra de George Boole, outro matemático do século XIX, cujo trabalho deve ser incluído em qualquer discussão sobre as raízes da inteligência artificial (Boole, 1847, 1854). Embora ele tenha realizado contribuições em várias áreas da matemática, seu trabalho mais conhecido é aquele na formalização das leis da lógica, uma realização que representa o núcleo da ciência da computação moderna. Embora o papel da álgebra booleana no projeto de circuitos lógicos seja bem conhecido, os objetivos pessoais de Boole em desenvolver seu sistema parecem mais próximos dos objetivos dos pesquisadores contemporâneos da IA. No primeiro capítulo de *Uma Investigação das leis do pensamento, sobre as quais estão fundamentadas as teorias matemáticas da lógica e da probabilidade*, Boole (1854) descreveu os seus objetivos como:

investigar as leis fundamentais das operações mentais pelas quais se realiza o raciocínio: expressá-las na linguagem simbólica de um cálculo e, sobre essa base, estabelecer a ciência da lógica e instruir o seu método; ... e, por fim, coletar de vários elementos verdadeiros, revelados no decorrer destas consultas, algumas indicações prováveis relativas à natureza e à constituição da mente humana.

A importância da realização de Boole está no poder extraordinário e na simplicidade do sistema que ele concebeu: três operações, “E” (representada por \* ou  $\wedge$ ), “OU” (representada por + ou  $\vee$ ) e “NÃO” (representada por  $\neg$ ), formam o núcleo do seu cálculo lógico. Essas operações foram a base para todos os desenvolvimentos subsequentes da lógica formal, incluindo a concepção dos computadores modernos. Ao mesmo tempo que mantém o significado desses símbolos praticamente idêntico às operações algébricas correspondentes, Boole observou que “os símbolos da lógica estão adicionalmente sujeitos a uma lei especial, à qual os símbolos de quantidades, como tais, não estão sujeitos”. Essa lei diz que, para qualquer  $X$ , que é um elemento da álgebra,  $X * X = X$  (ou seja, se algo é sabidamente verdadeiro, então a sua repetição não pode aumentar esse conhecimento). Essa noção levou à restrição característica dos valores booleanos a apenas dois números que podem satisfazer essa equação: 1 e 0. As definições padrão da multiplicação booleana (E) e da adição (OU) advêm desse critério.

O sistema de Boole não apenas forneceu a base da aritmética binária, mas também demonstrou que um sistema formal extremamente simples era adequado para captar todo o poder da lógica. Essa suposição e o sistema que Boole desenvolveu para demonstrá-la formam a base de todos os esforços modernos para formalizar a lógica, desde o *Principia Mathematica* (Princípios matemáticos) (Whitehead e Russell, 1950), passando pela obra de Turing e Gödel, até os sistemas modernos de raciocínio automatizado.

Gottlob Frege, em sua obra *Fundamentos da Aritmética* (Frege, 1879, 1884), criou uma linguagem de especificação matemática para descrever a base da aritmética de forma clara e precisa. Com essa linguagem, Frege formalizou muitas das questões que foram tratadas inicialmente pela Lógica de Aristóteles. A linguagem de Frege, agora conhecida como *cálculo de predicados de primeira ordem*, oferece uma ferramenta para descrever as proposições e atribuições de valores verdade que compõem os elementos do raciocínio matemático e descreve a base axiomática do “significado” para essas expressões. O sistema formal do cálculo de predicados, que inclui símbolos de predicados, uma teoria de funções e variáveis quantificadas, foi concebido para ser uma linguagem que descreva a matemática e suas bases filosóficas. Ele tem também um papel importante na criação de uma teoria de representações para a inteligência artificial (Capítulo 2). O cálculo de predicados de primeira ordem fornece as

ferramentas necessárias para o raciocínio automatizado: uma linguagem para expressões, uma teoria para suposições relacionadas com o significado das expressões e um cálculo logicamente correto para inferir novas expressões verdadeiras.

O trabalho de Whitehead e Russell (1950) é particularmente importante para a fundamentação da IA, uma vez que o seu objetivo declarado era derivar a matemática como um todo a partir de operações formais sobre uma coleção de axiomas. Embora muitos sistemas matemáticos tenham sido construídos a partir de axiomas básicos, o interessante aqui é o comprometimento de Russell e Whitehead com a matemática como um sistema puramente formal. Isso significa que os axiomas e teoremas seriam tratados apenas como sequências de caracteres: as provas se processariam apenas pela aplicação de regras bem definidas para manipular essas sequências. Não se poderia confiar na intuição ou no significado de teoremas como base para provas. Cada passo de uma prova é resultado da aplicação estrita de regras formais (sintáticas) a axiomas ou teoremas provados previamente, mesmo quando provas tradicionais possam considerar esse passo como “óbvio”. O “significado” que os teoremas e axiomas do sistema podem ter em relação ao mundo seria independente das suas derivações lógicas. Esse tratamento do raciocínio matemático em termos puramente formais (e consequentemente mecânicos) proporciona uma base essencial para sua automação em computadores físicos. A sintaxe lógica e as regras formais de inferência desenvolvidas por Russell e Whitehead fornecem ainda a base para sistemas de prova automática de teoremas, apresentados no Capítulo 14, bem como para os fundamentos teóricos da inteligência artificial.

Alfred Tarski é outro matemático cujo trabalho é essencial para o embasamento da IA. Tarski criou uma *teoria de referência*, por meio da qual se pode dizer que as *formulas bem formadas* de Frege ou Russell e Whitehead se referem, de um modo preciso, ao mundo físico (Tarski, 1944, 1956; ver Capítulo 2). Essa visão serve de embasamento para a maioria das teorias de semântica formal. No seu artigo “*A concepção semântica da verdade e os fundamentos da semântica*”, Tarski descreve sua teoria de referência e relacionamentos de valores verdade. Vários cientistas da computação moderna, especialmente Scott, Strachey, Burstall (Burstall e Darlington, 1977) e Plotkin, relacionaram essa teoria com linguagens de programação e outras especificações para computação.

Embora nos séculos XVIII, XIX e no início do XX a formulação da ciência e da matemática tenha criado o pré-requisito intelectual para o estudo da inteligência artificial, somente com a introdução do computador digital no século XX é que a IA se tornou uma disciplina científicamente viável. No final dos anos 1940, os computadores eletrônicos digitais demonstraram seu potencial para disponibilizar memória e poder de processamento requeridos pelos programas inteligentes. Era possível, agora, implementar sistemas de raciocínio formal em um computador e testar empiricamente a sua aptidão para exibir inteligência. Um componente essencial da ciência da inteligência artificial é esse compromisso com os computadores digitais como veículo para criar e testar teorias sobre a inteligência.

Os computadores digitais não são apenas um veículo para testar teorias sobre a inteligência. A sua arquitetura sugere também um paradigma específico para essas teorias: inteligência é uma forma de processamento de informação. A noção de busca como uma metodologia para solução de problemas, por exemplo, está mais ligada à natureza sequencial da operação do computador do que a qualquer modelo biológico de inteligência. A maioria dos programas de IA representa o conhecimento em alguma linguagem formal que é, então, manipulada por algoritmos, respeitando a separação entre dados e programa, que é fundamental ao estilo de computação de von Neumann. A lógica formal emergiu como uma ferramenta representacional importante para as pesquisas em IA, assim como a teoria dos grafos desempenha um papel essencial na análise de espaços de problemas e fornece a base para as redes semânticas e outros modelos similares de sentido semântico. Essas técnicas e formalismos são discutidos detalhadamente ao longo de todo o livro e são mencionados aqui para enfatizar a relação simbiótica entre o computador digital e os alicerces da inteligência artificial.

Esquecemos frequentemente que as ferramentas que criamos para os nossos propósitos tendem a moldar a nossa concepção do mundo a partir de sua estrutura e suas limitações. Embora aparentemente restritiva, essa interação é um aspecto essencial da evolução do conhecimento humano: uma ferramenta (e as teorias científicas são, no final das contas, apenas ferramentas) é desenvolvida para solucionar um problema em particular. Conforme essa ferramenta é usada e refinada, ela própria parece sugerir outras aplicações, levando a outras questões e, por fim, ao desenvolvimento de novas ferramentas.

### 1.1.4 Teste de Turing

Um dos primeiros artigos a tratar da questão da inteligência de máquina, especificamente em relação ao computador digital moderno, foi escrito em 1950 pelo matemático britânico Alan Turing. A obra *Maquinismo Computacional e Inteligência* (Turing, 1950) permanece atual tanto em relação à sua ponderação dos argumentos contra a possibilidade de se criar uma máquina computacional inteligente quanto por suas respostas a esses argumentos. Turing, que era conhecido principalmente por suas contribuições à teoria da computabilidade, abordou a questão se seria possível ou não fazer uma máquina pensar. Ao notar que as ambiguidades fundamentais contidas na própria questão (o que é pensar? o que é uma máquina?) obstruam qualquer resposta racional, ele propôs que a questão sobre a inteligência fosse substituída por um teste empírico mais claramente definido.

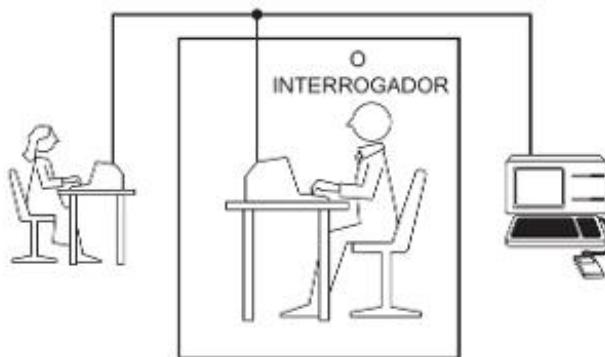
O *teste de Turing* mede o desempenho de uma máquina, aparentemente inteligente, em relação ao desempenho de um ser humano, indiscutivelmente o melhor e único padrão de comportamento inteligente. O teste, que foi chamado de *jogo de imitação* por Turing, coloca a máquina e seu correspondente humano em salas separadas de um segundo ser humano, referido como o *interrogador* (Figura 1.1). O interrogador não é capaz de ver nenhum dos dois participantes ou de falar diretamente com eles. Ele também não sabe qual entidade é a máquina e só pode se comunicar com eles por um dispositivo textual, como um terminal. A tarefa do interrogador é distinguir o computador do ser humano utilizando apenas as respostas de ambos a perguntas formuladas por meio desse dispositivo. Se o interrogador não puder distinguir a máquina do ser humano, então, argumenta Turing, pode-se supor que a máquina seja inteligente.

Isolando-se o interrogador tanto da máquina quanto do outro participante humano, o teste assegura que ele não será influenciado pela aparência da máquina ou por qualquer outra propriedade mecânica de sua voz. Entretanto, o interrogador é livre para fazer qualquer pergunta, não importando quão intrincada ou indireta ela seja, em um esforço para descobrir a identidade do computador. O interrogador pode, por exemplo, solicitar que os dois participantes realizem um cálculo aritmético bastante complicado, presumindo que seja mais provável que o computador, e não o ser humano, realize o cálculo corretamente; para neutralizar essa estratégia, o computador precisa saber quando ele deve deixar de obter a resposta correta a tais problemas, de modo a se parecer com um ser humano. Para descobrir a identidade humana a partir de sua natureza emocional, o interrogador poderia pedir para os dois participantes reagirem a um poema ou a uma obra de arte; essa estratégia requer que o computador tenha conhecimento acerca da constituição emocional dos seres humanos.

As características importantes do teste de Turing são:

1. Ele tenta nos dar uma noção objetiva de inteligência, isto é, o comportamento de um ser sabidamente inteligente em resposta a um conjunto particular de questões. Isso nos fornece um padrão para determinar a inteligência, evitando os debates inevitáveis sobre a sua “verdadeira” natureza.

**Figura 1.1** Teste de Turing.



2. Ele evita que sejamos desviados por essas questões confusas e atualmente irresponsáveis, como, por exemplo, se um computador usa ou não os processos internos adequados ou se a máquina tem ou não consciência de suas ações.
3. Ele elimina qualquer viés em favor dos organismos vivos, forçando o interrogador a focar apenas no conteúdo das respostas às questões.

Por causa dessas vantagens, o teste de Turing fornece uma base para vários esquemas realmente utilizados para avaliar programas de IA modernos. Um programa que potencialmente tenha adquirido inteligência em alguma área de especialidade pode ser avaliado comparando seu desempenho a um especialista humano, diante de certo conjunto de problemas. Essa técnica de avaliação é apenas uma variação do teste de Turing: pede-se que um grupo de pessoas compare, às cegas, o desempenho de um computador e de um ser humano em relação a um conjunto de problemas. Como veremos, essa metodologia se tornou uma ferramenta essencial tanto no desenvolvimento quanto na verificação dos sistemas especialistas modernos.

O teste de Turing, apesar do seu apelo intuitivo, é vulnerável a várias críticas justificáveis. Uma das mais importantes se refere ao seu viés direcionado para as tarefas de solução de problemas puramente simbólicos. Ele não testa as habilidades necessárias para a percepção ou para a destreza manual, muito embora elas sejam componentes importantes da inteligência humana. Por outro lado, existem algumas críticas que sugerem que o teste de Turing restringe desnecessariamente a inteligência de máquina a se encaixar no molde humano. Talvez a inteligência de máquina seja simplesmente diferente da inteligência humana, e tentar avaliá-la em termos humanos seja um erro fundamental. Realmente desejamos uma máquina que realize operações matemáticas de modo tão lento e inexato como o faz um ser humano? Uma máquina inteligente não deveria tirar proveito dos seus próprios recursos, como o fato de dispor de uma memória grande, rápida e confiável, em vez de tentar emular a cognição humana? De fato, vários praticantes da IA moderna (por exemplo, Ford e Hayes, 1995) consideram que responder totalmente ao desafio do teste de Turing é um erro e um grande desvirtuamento da tarefa mais importante que se apresenta: o desenvolvimento de teorias gerais para explicar os mecanismos da inteligência nos seres humanos e em máquinas, com a aplicação dessas teorias no desenvolvimento de ferramentas para solucionar problemas práticos, específicos. Embora concordemos com as preocupações de Ford e Hayes, consideramos ainda o teste de Turing como um componente importante na verificação e na validação do software de IA moderno.

Turing também tratou da real possibilidade de se construir um programa inteligente em um computador digital. Pensando em termos de um modelo específico de computação (uma máquina de computação eletrônica de estados discretos), ele formulou algumas conjecturas bem fundamentadas com relação à capacidade de armazenamento, à complexidade do programa e à filosofia básica de projeto, necessárias para esse tipo de sistema. Por fim, ele discutiu uma série de objeções morais, filosóficas e científicas sobre a possibilidade de se construir um programa assim em termos de uma tecnologia real. O leitor deve consultar o artigo de Turing para ter um quadro resumido ainda atual e relevante do debate sobre as possibilidades das máquinas inteligentes.

Duas das objeções citadas por Turing merecem uma consideração mais detalhada. A *Objeção de Lady Lovelace*, formulada originalmente por Ada Lovelace, argumenta que os computadores podem fazer apenas aquilo que lhes foi anteriormente dito, em consequência não podendo realizar ações originais (e, portanto, inteligentes). Essa objeção se tornou uma parte reconfortante, ainda que um tanto dúbia, do folclore tecnológico contemporâneo. Os sistemas especialistas (Seção 1.2.3 e Capítulo 8), especialmente na área de diagnósticos, têm chegado a conclusões não previstas pelos seus projetistas. Na verdade, vários pesquisadores consideram que a criatividade humana pode ser expressa em um programa de computador.

A outra objeção relacionada, o *Argumento da informalidade do comportamento*, defende a impossibilidade de se criar um conjunto de regras que digam a um indivíduo exatamente o que fazer em todas as circunstâncias possíveis. Certamente, a flexibilidade que possibilita a uma inteligência biológica responder a um conjunto quase infinito de situações de uma forma razoável e, muitas vezes, ótima, é uma característica distintiva do comportamento inteligente. Apesar de ser verdadeira a afirmação de que a estrutura de controle utilizada nos programas de computador mais tradicionais não demonstra grande flexibilidade ou originalidade, não é verdade que todos os programas devam ser escritos dessa forma. Na realidade, grande parte dos esforços em IA nos últimos 25 anos tem se concentrado no desenvolvimento de linguagens de programação e de modelos, tais como sistemas de pro-

dução, sistemas baseados em objetos, representações por redes neurais e outros que são discutidos neste livro, que procuram superar essa deficiência.

Muitos programas de IA modernos consistem, geralmente, em uma coleção de componentes modulares, ou regras de comportamento, que não é executada segundo uma ordenação rígida, mas que é invocada conforme a necessidade em resposta à estrutura de um caso particular de um problema. Os comparadores de padrões permitem a aplicação de regras gerais sobre certo conjunto de casos. Esses sistemas têm uma flexibilidade extrema que possibilita que programas relativamente pequenos exibam um vasto domínio de comportamentos possíveis em resposta a problemas e situações diferentes.

Entretanto, se esses sistemas, no final das contas, podem ou não exibir a flexibilidade que um organismo vivo possui ainda é um tema de muito debate. O ganhador do Prêmio Nobel Herbert Simon argumentou que muito da originalidade e da variabilidade de comportamento demonstradas pelas criaturas vivas é devido à riqueza do seu ambiente e não à complexidade dos seus próprios programas internos. Em *As Ciências do Artificial*, Simon (1981) descreve a progressão tortuosa de uma formiga ao longo de uma extensão de solo irregular e atravessada. Embora a trilha da formiga pareça ser bastante complexa, Simon argumenta que o objetivo da formiga é muito simples: retornar à sua colônia o mais rápido possível. Os giros e voltas na sua trilha são causados pelos obstáculos que ela encontra no seu caminho. Simon conclui:

Uma formiga, vista como um sistema comportamental, é muito simples. A aparente complexidade do seu comportamento ao longo do tempo é, em grande parte, um reflexo da complexidade do ambiente no qual ela se encontra.

Se, no final das contas, for provado que essa ideia se aplica não só às criaturas simples como os insetos, mas também aos organismos com maior inteligência, então ela constitui um argumento poderoso de que, esses sistemas são relativamente simples e, consequentemente, compreensíveis. É interessante notar que, se alguém aplicar essa ideia aos seres humanos, isso se torna um forte argumento para a importância da cultura na formação da inteligência. Em vez de crescer no escuro, como os cogumelos, a inteligência aparentemente depende de uma interação com um ambiente adequadamente rico. Cultura é tão importante na geração de seres humanos quanto os seres humanos são na geração de cultura. Em vez de denegrir nossos intelectos, essa ideia enfatiza a riqueza milagrosa e a coerência das culturas que se formaram a partir das vidas de seres humanos separados. De fato, a ideia de que a inteligência emerge das interações de elementos individuais de uma sociedade é uma das intuições que dão suporte à abordagem para a tecnologia de IA apresentada na próxima seção.

### 1.1.5 Modelos biológicos e sociais da inteligência: teoria de agentes

Até agora, abordamos o problema da construção de máquinas inteligentes do ponto de vista da matemática, com a crença implícita de que o raciocínio lógico é o paradigma da inteligência em si, bem como com o compromisso com a fundamentação “objetiva” do raciocínio lógico. Essa forma de se considerar o conhecimento, a linguagem e o pensamento reflete a tradição racionalista da filosofia ocidental, a qual evolui através de Platão, Galileu, Descartes, Leibniz e muitos dos outros filósofos que discutimos anteriormente neste capítulo. Ela também reflete as suposições que fundamentam o teste de Turing, particularmente a sua ênfase no raciocínio simbólico como teste de inteligência, e a crença de que uma comparação direta com um ser humano é adequada para confirmar a inteligência de máquina.

A confiança na lógica como um meio de representar conhecimento e na inferência lógica como o mecanismo primordial para o raciocínio inteligente é tão dominante na filosofia ocidental que a sua “verdade” frequentemente parece óvia e incontestável. Portanto, não é de surpreender que abordagens baseadas nessas suposições têm dominado a ciência da inteligência artificial desde o seu início até os dias de hoje.

Entretanto, a última metade do século XX presenciou numerosos desafios à filosofia racionalista. Várias formas de relativismo filosófico questionaram a base objetiva da linguagem, da ciência, da sociedade e do próprio pensamento. A filosofia póstuma de Ludwig Wittgenstein (Wittgenstein, 1953) forçou-nos a reconsiderar a base do significado, tanto da linguagem natural quanto da formal. Os trabalhos de Gödel (Nagel e Newman, 1958) e de Turing puseram em dúvida os fundamentos da própria matemática. O pensamento pós-moderno mudou o

nosso entendimento sobre significado e valor nas artes e na sociedade. A inteligência artificial não ficou imune a essas críticas; de fato, as dificuldades que a IA encontrou para alcançar seus objetivos são frequentemente consideradas como evidência da deficiência do ponto de vista racionalista (Winograd e Flores, 1986; Lakoff e Johnson, 1999; Dennett, 2005).

Duas tradições filosóficas, a de Wittgenstein (1953), bem como a de Husserl (1970, 1972) e Heidegger (1962), são centrais a essa reconsideração da tradição filosófica ocidental. Na sua obra póstuma, Wittgenstein questionou muitas das suposições da tradição racionalista, incluindo os fundamentos da linguagem, da ciência e do próprio conhecimento. A linguagem natural foi um foco central da análise de Wittgenstein: ele desafiou a noção de que a linguagem humana tenha derivado o seu significado a partir de qualquer tipo de fundamentação objetiva.

Para Wittgenstein, bem como pela teoria dos atos da fala desenvolvida por Austin (1962) e seus seguidores (Grice, 1975; Searle, 1969), o significado de qualquer expressão vocal depende de onde ela se situa em um contexto humano e cultural. O nosso entendimento da palavra “cadeira”, por exemplo, depende da existência de um corpo físico que corresponda a uma postura sentada e às convenções culturais sobre o uso de cadeiras. Quando, por exemplo, uma pedra grande e plana seria uma cadeira? Por que é estranho se referir ao trono da Inglaterra como uma cadeira? Qual a diferença entre o entendimento humano sobre uma cadeira e o de um cachorro ou gato, que são incapazes de sentar, no sentido humano? Com base nos seus ataques aos fundamentos do significado, Wittgenstein argumentou que deveríamos considerar o uso da linguagem em termos de escolhas feitas e ações realizadas em um contexto cultural variável. Wittgenstein estendeu, inclusive, as suas críticas à ciência e à matemática, argumentando que elas eram construções sociais, assim como o uso da linguagem.

Husserl (1970, 1972), o pai da fenomenologia, estava comprometido com abstrações enraizadas no *mundo da vida (Lebenswelt)* concreto: um modelo racionalista é secundário em relação ao mundo concreto que o suporta. Para Husserl, bem como para seu estudante Heidegger (1962) e seu defensor Merleau-Ponty (1962), a inteligência não é saber o que é verdadeiro, mas saber como lidar com um mundo em constante modificação e evolução. Gadamer (1976) também contribuiu para essa tradição. Assim, pelas tradições existencialista e fenomenologista, a inteligência é vista como sobrevivência no mundo, em vez de um conjunto de proposições lógicas acerca do mundo (combinada com um esquema de inferência).

Muitos autores, por exemplo Dreyfus e Dreyfus (1985) e Winograd e Flores (1986), se inspiraram no trabalho de Wittgenstein e de Husserl/Heidegger para suas críticas à IA. Embora muitos praticantes da IA continuem desenvolvendo a agenda racional/lógica (também conhecida como GOFAI, do inglês *Good Old Fashioned AI* — “a boa e velha IA”), um número crescente de pesquisadores da área incorporaram essas críticas em novos e excitantes modelos de inteligência. Mantendo a ênfase de Wittgenstein sobre as raízes antropológicas e culturais do conhecimento, eles se voltaram, como inspiração, para modelos sociais de comportamento inteligente, algumas vezes chamados de *baseados em agente ou situados*.

Como exemplo de alternativa para uma abordagem baseada em lógica, os pesquisadores da área de aprendizado conexionalista (Seção 1.2.9 e Capítulo 11) retiraram a ênfase na lógica e no funcionamento da mente racional, em um esforço para alcançar a inteligência modelando a arquitetura do cérebro físico. Os modelos neurais da inteligência enfatizam a habilidade do cérebro em se adaptar ao mundo no qual ele está inserido pela modificação dos relacionamentos entre neurônios individuais. Em vez de representar o conhecimento por sentenças lógicas explícitas, eles capturam o conhecimento implicitamente, como uma propriedade de padrões de relacionamentos.

Outro modelo de inteligência baseado na biologia busca sua inspiração nos processos pelos quais espécies inteiras se adaptam ao seu ambiente. Os trabalhos em vida artificial e em algoritmos genéticos (Capítulo 12) aplicam os princípios da evolução biológica aos problemas de encontrar soluções para problemas difíceis. Esses programas não resolvem problemas raciocinando logicamente sobre eles; ao contrário, eles geram populações de soluções candidatas competidoras e as forçam a evoluir para soluções melhores por um processo que imita a evolução biológica: possíveis soluções ruins tendem a se extinguir, enquanto aquelas que se mostram mais promissoras em resolver um problema sobrevivem e se reproduzem, construindo novas soluções a partir de componentes dos seus pais bem-sucedidos.

Os sistemas sociais fornecem outra metáfora para a inteligência, uma vez que eles exibem comportamentos globais que permitem resolver problemas que confundiriam qualquer um de seus membros individuais. Por exemplo, embora nenhum indivíduo possa prever precisamente o número de pães que serão consumidos na

cidade de Nova York em um determinado dia, o sistema de padarias de Nova York como um todo realiza muito bem a tarefa de manter a cidade abastecida com pães, com o mínimo de desperdício. O mercado de ações faz um excelente trabalho em fixar os valores relativos de centenas de empresas, muito embora cada investidor individual tenha somente um conhecimento limitado sobre poucas empresas. Um último exemplo vem da ciência moderna. Indivíduos estabelecidos em universidades, na indústria ou em instituições governamentais enfrentam problemas comuns. Problemas importantes à sociedade como um todo são atacados e solucionados por indivíduos agindo quase que independentemente, tendo conferências e revistas científicas como principais meios de comunicação, embora o progresso, em muitos casos, também seja conduzido por agências de fomento.

Esses exemplos compartilham dois temas: primeiro, a visão da inteligência como enraizada na cultura e na sociedade, e, como consequência, emergente. O segundo, que a inteligência é o reflexo dos comportamentos coletivos de um grande número de indivíduos semiautônomos muito simples que interagem entre si, ou *agentes*. Quer esses agentes sejam células neurais, quer sejam membros individuais de uma espécie, ou ainda uma única pessoa de uma sociedade, as suas interações produzem inteligência.

Quais são os grandes temas que justificam uma visão de inteligência emergente orientada a agentes? Entre eles se incluem:

1. Agentes são autônomos ou semiautônomos, isto é, cada agente tem certas responsabilidades na solução de problemas, com pouco ou nenhum conhecimento do que outros agentes fazem ou de como o fazem. Cada agente realiza a sua parte independentemente da solução do problema: produz um resultado (faz alguma coisa), ou relata seus resultados para outros indivíduos da comunidade (agente comunicante).
2. Os agentes são “situados”. Cada agente é sensível ao seu ambiente particular e (normalmente) não tem conhecimento do domínio completo de todos os agentes. Assim, o conhecimento de um agente é limitado às tarefas atuais: “o arquivo que estou processando” ou “a parede próxima a mim” sem qualquer conhecimento do conjunto total de arquivos ou das restrições físicas da tarefa de solucionar um problema.
3. Agentes são interativos. Isso significa que eles formam uma coleção de indivíduos que cooperam em uma tarefa particular. Nesse sentido, eles podem ser vistos como uma “sociedade” e, como na sociedade humana o conhecimento, as habilidades e as responsabilidades, mesmo quando vistas como coletivos, estão distribuídos entre os indivíduos de uma população.
4. A sociedade de agentes é estruturada. Na maioria das visões de solução de problemas orientada a agentes, cada indivíduo, embora tendo o seu ambiente e seu conjunto de habilidades particulares e únicas, precisa se coordenar com outros agentes para a solução global do problema. Assim, uma solução final não será vista apenas como coletiva, mas também como cooperativa.
5. Finalmente, o fenômeno da inteligência nesse ambiente é “emergente”. Embora agentes individuais sejam vistos como se possuissem conjuntos de habilidades e responsabilidades, o resultado cooperativo global da sociedade de agentes pode ser visto como maior que a soma de suas contribuições individuais. A inteligência é vista como um fenômeno que reside e emerge da sociedade, e não apenas uma propriedade de um agente individual.

Com base nessas observações, definimos um agente como um elemento de uma sociedade que pode perceber aspectos (frequentemente limitados) de seu ambiente e afetá-lo, quer diretamente, quer através da cooperação com outros agentes. A maioria das soluções inteligentes requer uma variedade de agentes. Aí se incluem agentes rotineiros, que simplesmente capturam e comunicam partes da informação; agentes de coordenação, que dão suporte às interações entre outros agentes; agentes de busca, que examinam conjuntos de informações e retornam pedaços importantes deles; agentes aprendizes, que examinam coleções de informações e formam conceitos ou generalizações, e agentes de decisão, que podem tanto disparar tarefas como chegar a conclusões com base em informação e processamento limitados. Retomando uma definição mais antiga de inteligência, agentes podem ser vistos como mecanismos que suportam a tomada de decisão no contexto de recursos computacionais limitados.

Os requisitos principais para projetar e construir uma sociedade de agentes são:

1. estruturas para a representação da informação;
2. estratégias para busca por meio de soluções alternativas;
3. criação de arquiteturas que possam suportar a interação de agentes.

Os capítulos restantes deste livro, especialmente a Seção 7.4, incluem prescrições para a construção de ferramentas de suporte para essa sociedade de agentes, além de muitos exemplos de solução de problemas baseada em agente.

A nossa discussão preliminar sobre a possibilidade de uma teoria da inteligência automática não teve a intenção de exagerar o progresso atingido na atualidade ou minimizar o trabalho que se encontra pela frente. Como enfatizamos ao longo de todo o texto, é importante que tenhamos consciência das nossas limitações e que sejamos honestos acerca de nossos sucessos. Houve, por exemplo, resultados apenas limitados com programas dos quais se pode dizer, em qualquer sentido interessante, que “aprendem”. Nossas realizações na modelagem das complexidades semânticas de uma linguagem natural, como o inglês, também têm sido modestas. Mesmo questões fundamentais, como a organização do conhecimento ou o completo gerenciamento da complexidade e da correção de programas de computadores muito grandes (como grandes bases de conhecimento), requerem pesquisas adicionais consideráveis. Os sistemas baseados em conhecimento, embora tenham atingido um sucesso comercial do ponto de vista da engenharia, ainda têm muitas limitações na qualidade e na generalidade do seu raciocínio. Aqui se incluem a sua incapacidade de realizar *raciocínio de senso comum* ou em exibir conhecimento de uma realidade física rudimentar, por exemplo, como as coisas mudam com o tempo.

Mas devemos manter uma perspectiva razoável. É fácil subestimar as realizações da inteligência artificial quando encaramos honestamente o trabalho que nos resta. Na próxima seção, estabelecemos essa perspectiva a partir de um panorama de várias áreas da pesquisa e do desenvolvimento em inteligência artificial.

## 1.2 Uma visão geral das áreas de aplicação da IA

*O Motor Analítico não tem pretensões de criar algo original. Ele pode fazer qualquer coisa, desde que saibamos como lhe dar a ordem.*

—ADA BYRON, Condessa de Lovelace

*Desculpe-me, Dave; não posso deixar que faça isso.*

—HAL 9000 em 2001: Uma odisseia no espaço, de Arthur C. Clarke

Retornamos agora ao nosso objetivo de definir inteligência artificial por meio de um exame das ambições e das realizações dos pesquisadores da área. As duas preocupações fundamentais dos pesquisadores em IA são a *representação de conhecimento* e a *busca*. A primeira trata do problema de capturar em uma linguagem formal, isto é, em uma linguagem adequada para ser manipulada em computador, toda a extensão de conhecimento necessário para um comportamento inteligente. O Capítulo 2 introduz o cálculo de predicados como uma linguagem para descrever as propriedades e os relacionamentos entre objetos em domínios de problemas que, para a sua solução, necessitam de raciocínio qualitativo, em vez de cálculos aritméticos. Depois, a Seção III discute as ferramentas que a inteligência artificial desenvolveu para representar as ambiguidades e as complexidades de áreas como raciocínio de senso comum e compreensão de linguagem natural.

A busca é uma técnica de solução de problemas que explora sistematicamente o espaço de *estados do problema*, isto é, os estágios sucessivos e alternativos no processo de solução do problema. Exemplos de estados de um problema incluem as diferentes configurações do tabuleiro em um jogo ou os passos intermediários em um processo de raciocínio. Esse espaço de soluções alternativas é, então, explorado para se encontrar uma resposta final. Newell e Simon (1976) propuseram que essa é a base essencial da solução de problemas por seres humanos. De fato, quando um jogador de xadrez examina os efeitos de diferentes movimentos ou um médico considera uma série de diagnósticos alternativos, eles estão realizando uma busca entre diferentes alternativas. As implicações

desse modelo e dessas técnicas para sua implementação são discutidas nos capítulos 3, 4, 6 e 16. A Sala Virtual deste livro oferece implementações desses algoritmos em Lisp, Prolog e Java.

Como a maioria das ciências, a IA é composta de uma série de disciplinas que, ao mesmo tempo em que compartilha uma abordagem essencial para a solução de problemas, está preocupada com aplicações diferentes. Nesta seção, apresentamos algumas das mais importantes áreas de aplicação e suas contribuições para a inteligência artificial como um todo.

### 1.2.1 Jogos

Muitas das pesquisas iniciais sobre busca em espaço de estados foram realizadas utilizando jogos de tabuleiro comuns, como xadrez, damas e o quebra-cabeça dos 15 (*15-puzzle*). Além do seu apelo intelectual inerente, os jogos de tabuleiro têm certas propriedades que os tornaram objetos de estudo ideais para esses trabalhos iniciais. A maioria dos jogos utiliza um conjunto bem definido de regras: isso faz com que seja fácil gerar o espaço de busca e libera o pesquisador de muitas das ambiguidades e complexidades inerentes a problemas menos estruturados. As configurações do tabuleiro usadas nesses jogos são facilmente representáveis em um computador, dispensando o formalismo complexo necessário para capturar as sutilezas semânticas de domínios de problemas mais complexos. Como os jogos podem ser praticados facilmente, o teste de programas de jogos não apresenta nenhum ônus financeiro ou ético. Nos capítulos 3 e 4, é apresentada a busca em espaço de estados, o paradigma que serve de base na maioria das pesquisas sobre jogos.

Os jogos podem gerar espaços de busca extremamente grandes. Eles podem ser grandes e complexos o suficiente para necessitarem de técnicas poderosas para determinar quais alternativas devem ser exploradas no espaço do problema. Essas técnicas são chamadas de *heurísticas* e constituem uma área importante da pesquisa em IA. Uma heurística é uma estratégia de solução útil, mas potencialmente falível, como, por exemplo, checar para assegurar que um equipamento que não está respondendo esteja ligado na tomada antes de admitir que ele esteja quebrado, ou ainda, rocar, a fim de proteger o próprio rei da captura em uma partida de xadrez. Muito do que chamamos de inteligência parece estar relacionado com as heurísticas usadas pelos seres humanos para resolver problemas.

Como a maioria das pessoas tem alguma experiência com esses jogos simples, é possível projetar e testar a eficácia de nossas próprias heurísticas. Não precisamos consultar um especialista como em algumas áreas restritas, por exemplo, a medicina e a matemática (xadrez é uma exceção óbvia a essa regra). Por essas razões, os jogos proporcionam um domínio rico para o estudo da busca heurística. O Capítulo 4 introduz a noção de heurística usando esses jogos simples. Os programas de jogos, apesar da sua simplicidade, apresentam seus próprios desafios, incluindo um adversário cujos movimentos não podem ser antecipados deterministicamente (capítulos 5 e 9), e a necessidade de se considerar fatores tanto psicológicos como táticos na estratégia do jogo.

Os recentes sucessos em jogos baseados em computador incluem campeonatos mundiais de gamão e xadrez. Também é interessante notar que, em 2007, o espaço de estados completo para o jogo de damas foi mapeado, permitindo que ele seja, desde o primeiro movimento, determinístico!

### 1.2.2 Raciocínio automatizado e prova de teoremas

Podemos afirmar que a prova automática de teoremas é o ramo mais antigo da inteligência artificial, sendo possível traçar as suas raízes passando pelo *Teorista Lógico*, de Newell e Simon (Newell e Simon, 1963a), bem como por seu *Resolvedor Geral de Problemas* (Newell e Simon, 1963b), passando pelos esforços de Russell e Whitehead para tratar toda a matemática como uma derivação puramente formal de teoremas, a partir de axiomas básicos, até as suas origens nos escritos de Babbage e Leibniz. De qualquer forma, esse tem sido certamente um dos ramos mais frutíferos da IA. A pesquisa em prova automática de teoremas foi responsável por muitos trabalhos iniciais na formalização de algoritmos de busca e no desenvolvimento de linguagens de representação formais como o cálculo de predicados (Capítulo 2) e a linguagem de programação em lógica Prolog.

Muito do apelo da prova automática de teoremas reside no rigor e na generalidade da lógica. Por ser um sistema formal, a lógica se presta bem a ser automatizada. Uma ampla variedade de problemas pode ser abordada representando a descrição do problema e os conhecimentos básicos relevantes como axiomas lógicos e tratando instâncias do problema como teoremas a serem provados. Essa abordagem é a base dos trabalhos em prova automática de teoremas e em sistemas de raciocínio matemático (Capítulo 14).

Infelizmente, os esforços iniciais para criar provadores de teoremas fracassaram em desenvolver um sistema que pudesse solucionar, de modo consistente, problemas complicados. Isso se deveu à habilidade de qualquer sistema lógico razoavelmente complexo de gerar um número infinito de teoremas prováveis: sem técnicas poderosas (heurísticas) para guiar a sua busca, os provadores automáticos de teoremas provavam um grande número de teoremas irrelevantes antes de encontrar o teorema correto. Em resposta a essa ineficiência, muitos pesquisadores afirmam que métodos sintáticos, puramente formais para guiar a busca, são inherentemente incapazes de lidar com um espaço tão grande e que a única alternativa é confiar nas estratégias *ad hoc* informais que os seres humanos parecem utilizar para resolver problemas. Essa é a estratégia implícita no desenvolvimento de sistemas especialistas (Capítulo 8) e que provou ser adequada.

Apesar disso, o apelo do raciocínio baseado em lógica matemática formal é forte demais para ser ignorado. Muitos problemas importantes, como o projeto e a verificação de circuitos lógicos, a verificação da correção de programas computacionais e o controle de sistemas complexos, parecem corresponder a essa abordagem. De fato, a comunidade de prova de teoremas tem obtido sucesso em conceber heurísticas poderosas para a solução de problemas que dependem apenas de uma estimativa da forma sintática de uma expressão lógica, e, como resultado, capazes de reduzir a complexidade do espaço de busca sem recorrer a técnicas *ad hoc* usadas pela maioria dos peritos humanos.

Outra razão para o interesse continuado em provadores automáticos de teoremas é a compreensão de que tais sistemas não precisam ser capazes de resolver independentemente problemas extremamente complexos sem assistência humana. Muitos provadores de teoremas modernos funcionam como assistentes inteligentes, liberando os seres humanos para realizar as tarefas mais exigentes de decomposição de um problema grande em subproblemas e para conceber heurísticas de busca no espaço de provas possíveis. O provador de teoremas, então, realiza a tarefa mais simples, mas ainda assim exigente, de provar lemas, verificar conjecturas menores e completar os aspectos formais de uma prova delineada por seu associado humano (Boyer e Moore, 1979; Bundy, 1988; Veroff, 1997; Veroff e Spinks, 2006).

### 1.2.3 Sistemas especialistas

Uma das mais importantes conclusões que foi tirada dos trabalhos iniciais em solução de problemas foi a importância do conhecimento específico do domínio. Um médico, por exemplo, não é efetivo em diagnosticar uma doença apenas porque ele possui uma habilidade inata em resolver problemas genéricos; ele é eficaz porque sabe muito sobre medicina. Da mesma forma, um geólogo é eficaz em descobrir depósitos de minérios porque ele é capaz de aplicar uma grande quantidade de conhecimento teórico e empírico sobre geologia ao problema específico. O conhecimento especialista é uma combinação de um entendimento teórico do problema com uma coleção de regras heurísticas para resolver problemas, que a experiência demonstrou ser efetiva no domínio. Os sistemas especialistas são construídos a partir da extração desse conhecimento de um especialista humano, codificando-o de uma forma que um computador possa aplicá-lo a problemas similares.

Uma característica fundamental dos sistemas especialistas é que a sua estratégia para resolver problemas é dependente do conhecimento de um especialista humano no domínio. Embora existam alguns programas em que o projetista é também a fonte do conhecimento do domínio, geralmente é muito mais provável que esses programas sejam um produto da colaboração entre um especialista do domínio, como um médico, um químico, um geólogo ou um engenheiro, e um especialista em inteligência artificial. O especialista no domínio fornece o conhecimento necessário do domínio do problema, tanto por meio de uma discussão geral dos seus métodos de resolução de problema quanto pela demonstração dessas habilidades em um conjunto cuidadosamente escolhido de exemplos de problemas. O especialista em IA, ou *engenheiro do conhecimento*, como frequentemente são co-

nhecidos os projetistas de sistemas especialistas, é responsável por implementar esse conhecimento em um programa que seja tanto efetivo como aparentemente inteligente do ponto de vista de seu comportamento. Uma vez que esse programa esteja escrito, é necessário refiná-lo a partir da apresentação de exemplos de problemas a resolver, sob a supervisão crítica do especialista no domínio, e realizar quaisquer alterações necessárias no conhecimento do programa. Esse processo é repetido até que o programa atinja o nível desejado de desempenho.

Um dos sistemas mais antigos a explorar o conhecimento específico do domínio para a solução de problemas foi o DENDRAL, desenvolvido em Stanford no final dos anos 1960 (Lindsay et al., 1980). Esse sistema foi projetado para inferir a estrutura de moléculas orgânicas a partir de suas fórmulas químicas e de informações de espectrografia de massa das ligações químicas presentes nas moléculas. Como as moléculas orgânicas normalmente são muito grandes, o número de estruturas possíveis para essas moléculas tende a ser enorme. O DENDRAL trata o problema desse grande espaço de busca aplicando o conhecimento heurístico de especialistas em química no problema de elucidação da estrutura. O método utilizado mostrou-se muito efetivo, obtendo rotineiramente a estrutura correta entre milhões de possibilidades após algumas tentativas. A abordagem se mostrou tão bem-sucedida que, hoje em dia, são usados programas descendentes e extensões daquele sistema em laboratórios químicos e farmacêuticos espalhados por todo o mundo.

Enquanto o DENDRAL foi um dos primeiros programas a usar efetivamente o conhecimento específico do domínio para alcançar o desempenho de especialistas na resolução do problema, o MYCIN estabeleceu a metodologia dos sistemas especialistas contemporâneos (Buchanan e Shortliffe, 1984). O MYCIN utiliza conhecimento de especialistas médicos para diagnosticar e prescrever tratamentos para meningite espinhal e infecções bacterianas do sangue. Esse programa, desenvolvido em Stanford em meados dos anos 1970, foi um dos primeiros a tratar os problemas do raciocínio com informações incertas ou incompletas. Ele fornecia explanações lógicas claras do seu próprio raciocínio, utilizava uma estrutura de controle apropriada para o domínio específico do problema e identificava critérios para avaliar o desempenho obtido de forma confiável. Muitas das técnicas de desenvolvimento de sistemas especialistas em uso atualmente foram desenvolvidas originalmente no projeto MYCIN (Capítulo 8).

Outros sistemas especialistas clássicos são o programa PROSPECTOR, para determinar a localização e o tipo prováveis de depósitos de minério com base em informação geológica sobre um sítio (Duda et al., 1979a, 1979b), o programa INTERNIST, para realizar diagnósticos na área da medicina interna, o Dipmeter Advisor, para interpretar os resultados de registros de perfuração de poços petrolíferos (Smith e Baker, 1983), e o XCON, para configurar computadores VAX. O XCON foi desenvolvido em 1981 e, naquele tempo, todo computador VAX vendido pela Digital Equipment Corporation era configurado por esse software. Vários outros sistemas especialistas são atualmente aplicados para resolver problemas em áreas como medicina, educação, negócios, projeto e ciências (Waterman, 1986; Durkin, 1994). Veja também os anais das conferências Innovative Applications of Artificial Intelligence (IAAI).

É interessante notar que a maioria dos sistemas especialistas foi escrita para domínios com nível de pericia relativamente especializado. Esses domínios são geralmente bem estudados e têm estratégias de solução de problemas claramente definidas. Problemas que dependem de uma noção de "senso comum", definida de forma menos rígida, são muito mais difíceis de serem resolvidos por esses meios. Apesar das promessas dos sistemas especialistas, seria um erro superestimar a habilidade dessa tecnologia. Entre as deficiências mais comuns encontradas atualmente, estão:

1. Dificuldade em capturar conhecimento "profundo" do domínio do problema. Ao MYCIN, por exemplo, falta conhecimento real da fisiologia humana. Ele não sabe o que ocorre com o sangue ou qual é a função da medula espinhal. Conta-se que, ao selecionar uma droga para o tratamento de meningite, o MYCIN perguntou se o paciente estava "grávido", mesmo sabendo que ele era do sexo masculino. Mesmo havendo dúvidas sobre o ocorrido, se é verdade ou apenas folclore, isso ilustra a limitação potencial do conhecimento em sistemas especialistas.
2. Falta de robustez e flexibilidade. Se a um ser humano for apresentado um problema que ele não consiga resolver imediatamente, ele geralmente poderá realizar um exame dos princípios fundamentais e chegar a uma estratégia para atacá-lo. Os sistemas especialistas, em geral, não possuem essa habilidade.

3. Incapacidade de fornecer explanações aprofundadas. Como os sistemas especialistas não têm um conhecimento profundo de seus domínios, as suas explanações são geralmente restritas a uma descrição dos passos que eles realizaram para encontrar a solução. Normalmente, por exemplo, eles são incapazes de explicar “por que” adotaram certa abordagem.
4. Dificuldades na verificação. Embora a correção de qualquer sistema computacional extenso seja difícil de ser provada, os sistemas especialistas são particularmente difíceis de serem verificados. Esse é um problema sério, uma vez que a tecnologia dos sistemas especialistas vem sendo utilizada em aplicações críticas, como o controle de tráfego aéreo, a operação de reatores nucleares e os sistemas de armamentos.
5. Pouco aprendizado por experiência. Os sistemas especialistas atuais são artesanais; quando o sistema já se encontra desenvolvido, o seu desempenho não irá melhorar sem a ajuda de seus programadores. Isso levanta severas dúvidas sobre a inteligência desses sistemas.

Apesar dessas limitações, os sistemas especialistas têm provado o seu valor em várias aplicações importantes e constituem um dos principais tópicos deste livro, sendo discutidos nos capítulos 7 e 8. Veja também os anais das conferências Innovative Applications of Artificial Intelligence (IAAI) para novas aplicações.

#### 1.2.4 Compreensão da linguagem natural e modelagem semântica

Um dos objetivos da inteligência artificial, que vem sendo perseguido há muito tempo, é a criação de programas que sejam capazes de entender e gerar a linguagem humana. A habilidade em utilizar e compreender a linguagem natural não apenas parece ser um aspecto fundamental da inteligência humana, mas também a sua automação teria um impacto inacreditável sobre a facilidade de utilização e eficácia dos próprios computadores. Foram feitos grandes esforços no desenvolvimento de programas que compreendem a linguagem natural. Embora esses programas tenham alcançado sucesso em contextos restritos, sistemas que possam usar linguagem natural com a flexibilidade e a generalidade que caracterizam a fala humana ainda estão além das metodologias atuais.

Compreender a linguagem natural envolve muito mais que a simples análise de sentenças, da separação de suas partes individuais e da procura dessas palavras em um dicionário. A compreensão real depende de um extenso conhecimento do domínio do discurso e das expressões idiomáticas utilizadas naquele domínio, bem como da habilidade em aplicar conhecimento contextual genérico para resolver omissões e ambiguidades que são parte usual da fala humana.

Considere, por exemplo, as dificuldades de se estabelecer uma conversação sobre beisebol com um indivíduo que entende o inglês, mas não sabe nada sobre as regras, os jogadores e a história do jogo. Será que essa pessoa poderia compreender o significado da sentença: “sem ninguém no topo da nona e com uma corrida na segunda, o treinador tirou o seu reserva do curral”? Muito embora cada uma das palavras na sentença possa ser compreendida individualmente, a sentença soaria incompreensível mesmo para uma pessoa muito inteligente que não fosse fã de beisebol.

A tarefa de coletar e organizar esse conhecimento de fundo, de forma que ele possa ser aplicado à compreensão da linguagem, constitui o problema fundamental na automação da compreensão da linguagem natural. Em resposta a essa necessidade, os pesquisadores desenvolveram muitas das técnicas para estruturar o significado semântico usado em toda a inteligência artificial (capítulos 7 e 15).

Devido à enorme quantidade de conhecimento necessário para a compreensão da linguagem natural, a maioria dos trabalhos é realizada em áreas de problemas especializadas e bem conhecidas. Um dos primeiros programas a explorar essa metodologia de “micromundo” foi o SHRDLU, de Winograd, um sistema de linguagem natural que podia “conversar” sobre uma configuração simples de blocos de diferentes formas e cores (Winograd, 1973). O SHRDLU podia responder a perguntas como “qual é a cor do bloco que está sobre o cubo azul?”, bem como planejar ações como “coloque a pirâmide vermelha sobre o tijolo verde”. Problemas desse tipo, que envolvem descrição e manipulação de arranjos simples de blocos, apareceram com uma frequência surpreendente em pesquisas iniciais de IA e são conhecidos como problemas de “mundo de blocos”.

Apesar do sucesso do SHRDLU em conversar sobre arranjos de blocos, os seus métodos não podem ser generalizados para além desse domínio. As técnicas representacionais usadas no programa eram simples demais para capturar, de uma maneira útil, a organização semântica de domínios mais ricos e complexos. Grande parte do trabalho atual em compreensão da linguagem natural é dedicada a encontrar formalismos representacionais que sejam gerais o suficiente para serem usados em um amplo espectro de aplicações, mas também que se adaptem bem à estrutura específica de um dado domínio. Várias técnicas diferentes (a maioria das quais são extensões ou modificações de *redes semânticas*) são exploradas para esse propósito e são usadas no desenvolvimento de programas que podem compreender a linguagem natural em domínios de conhecimento restritos, mas interessantes. Por fim, em pesquisas recentes (Marcus, 1980; Manning e Schütze, 1999; Jurafsky e Martin, 2009), modelos estocásticos, descrevendo como conjuntos de palavras “ocorrem” no uso da linguagem, são empregados para caracterizar tanto a sintaxe como a semântica. Entretanto, a compreensão total da linguagem permanece além do estado da arte atual.

### 1.2.5 Modelando o desempenho humano

Embora grande parte da discussão anterior utilize a inteligência humana como um ponto de referência ao considerar a inteligência artificial, os programas discutidos não seguem o modelo de organização da mente humana. De fato, muitos programas de IA são concebidos para resolver algum problema útil sem levar em consideração as suas similaridades com a arquitetura mental humana. Mesmo os sistemas especialistas, que extraem muito do seu conhecimento de especialistas humanos, não procuram realmente simular os processos mentais internos humanos de solução de problemas. Se o desempenho é o único critério pelo qual um sistema é julgado, pode ser que não haja muitos motivos para simular os métodos humanos de solução de problemas; de fato, os programas que utilizam abordagens não humanas para resolver problemas são, frequentemente, mais bem-sucedidos que os seus correspondentes humanos. Ainda assim, o projeto de sistemas que modelam explicitamente algum aspecto do desempenho humano tem sido uma área fértil de pesquisa, tanto em inteligência artificial quanto em psicologia.

A modelagem do desempenho humano, além de proporcionar à IA grande parte de sua metodologia básica, tem se mostrado uma ferramenta poderosa para formular e testar teorias da cognição humana. As metodologias para solução de problemas, desenvolvidas por cientistas da computação, forneceram aos psicólogos uma nova metáfora para explorar a mente humana. Em vez de formular teorias da cognição na linguagem vaga usada nas pesquisas iniciais, ou desistir de descrever completamente os processos internos da mente humana (como sugerido pelos comportamentalistas), muitos psicólogos adotaram a linguagem e a teoria da ciência da computação para formular modelos de inteligência humana. Essas técnicas não apenas fornecem um novo vocabulário para descrever a inteligência humana, mas também as implementações computacionais dessas teorias oferecem aos psicólogos uma oportunidade de testar empiricamente, criticar e refinar as suas ideias (Luger, 1994; ver conferências e periódicos da *Cognitive Science Society*). Uma relação entre a inteligência artificial e inteligência humana é resumida no Capítulo 16.

### 1.2.6 Planejamento e robótica

A pesquisa em planejamento começou como um esforço para projetar robôs que pudessem realizar as suas tarefas com algum grau de flexibilidade e resposta em relação ao mundo externo. Em suma, planejamento pressupõe um robô que seja capaz de realizar certas ações atômicas. Ele procura encontrar uma sequência dessas ações que realize uma tarefa de alto nível, como se mover através de uma sala com vários obstáculos.

Planejamento é um problema difícil por uma série de razões, entre as quais o tamanho do espaço de sequências possíveis de movimentos. Mesmo um robô extremamente simples é capaz de gerar um vasto número de sequências de movimentos potenciais. Imagine, por exemplo, um robô que pode se mover para a frente, para trás, para a direita ou para a esquerda, e considere de quantas maneiras diferentes o robô pode se mover em uma sala. Suponha, também, que haja obstáculos na sala e que o robô deva selecionar um caminho que desvie desses obstáculos de uma forma eficiente. Desenvolver um programa que possa descobrir de modo inteligente o melhor cami-

nho sob essas circunstâncias, sem ser sobrepujado pelo número enorme de possibilidades, requer técnicas sofisticadas para representar conhecimento espacial e controlar a busca através de ambientes possíveis.

Um dos métodos que os seres humanos usam para planejamento é a *decomposição hierárquica do problema*. Ao planejar uma viagem a Londres, você geralmente tratará separadamente dos problemas de organizar o voo, chegar ao aeroporto, fazer as conexões aéreas e o traslado em Londres, apesar de eles serem parte de um plano global maior. Cada um desses problemas pode ser ainda decomposto em subproblemas menores, como encontrar um mapa da cidade, entender o sistema de metrô e encontrar um pub adequado. Essa abordagem não apenas resstringe efetivamente o tamanho do espaço que deve ser buscado, mas também permite armazenar os subplanos frequentemente usados para serem reutilizados no futuro.

Diferentemente dos seres humanos que planejam sem grande esforço, o desenvolvimento de programas de computador que façam o mesmo é um grande desafio. Uma tarefa aparentemente simples, como decompor um problema em subproblemas independentes, na verdade requer heurísticas sofisticadas e conhecimento extensivo sobre o domínio do planejamento. Determinar quais subplanos devem ser armazenados e como eles podem ser generalizados para uso futuro é, também, um problema difícil.

Um robô dificilmente seria considerado inteligente se executasse cegamente uma sequência de ações sem responder a alterações no ambiente ou se não fosse capaz de detectar e corrigir erros no seu plano original. Um robô pode não possuir sensores adequados para localizar todos os obstáculos que se encontram no caminho planejado. Esse robô deve começar movendo-se pelo ambiente com base no que ele “percebeu”, corrigindo o seu caminho, conforme outros obstáculos são detectados. Organizar planos, de modo que seja possível responder a condições variáveis do ambiente, é um dos principais problemas do planejamento (Lewis e Luger, 2000; Thrun et al., 2007).

Por fim, a robótica foi uma das áreas da IA que produziu muitas das ideias que dão suporte à solução de problemas orientada a agentes (Seção 1.1.4). Frustrados tanto com a complexidade de se manter um grande espaço representacional, quanto com a concepção de algoritmos de busca adequados ao planejamento tradicional, pesquisadores como Agre e Chapman (1987), Brooks (1991a) e Thrun et al. (2007) reformularam o problema global em termos da interação de múltiplos agentes semiautônomos. Cada agente é responsável por uma porção individual do problema e, por meio da coordenação entre eles, a solução global emergirá.

A pesquisa em planejamento agora se estende para além dos domínios da robótica, incluindo a coordenação de qualquer conjunto complexo de tarefas e objetivos. Planejadores modernos são aplicados tanto a agentes (Nilsson, 1994) como para controlar aceleradores de feixe de partículas (Klein et al., 1999, 2000).

### 1.2.7 Linguagens e ambientes para IA

Alguns dos produtos secundários mais importantes da pesquisa em inteligência artificial foram avanços em linguagens de programação e em ambientes de desenvolvimento de software. Por uma série de razões, incluindo aí o tamanho de muitos programas de aplicação de IA, a importância de uma metodologia de “prototipação”, a tendência dos algoritmos de busca em gerar espaços enormes e a dificuldade de predizer o comportamento de programas guiados por heurísticas, os programadores em IA foram forçados a desenvolver um conjunto poderoso de metodologias de programação.

Esses ambientes de programação incluem técnicas de estruturação do conhecimento, como a programação orientada a objetos. Linguagens de alto nível, como Lisp e Prolog, que suportam desenvolvimento modular, ajudam a gerenciar o tamanho e a complexidade do sistema. Pacotes de rastreamento permitem que um programador reconstrua a execução de um algoritmo complexo e tornam possível esclarecer as complexidades da busca guiada por heurísticas. Sem essas ferramentas e técnicas, é improvável que muitos dos sistemas de IA significativos tivessem sido construídos.

Muitas dessas técnicas são agora ferramentas-padrão para a engenharia de software e têm pouca relação com a teoria de IA. Outras, como a programação orientada a objetos, são de elevado interesse teórico e prático. Finalmente, muitos dos algoritmos de IA agora são construídos também em linguagens de programação mais tradicionais, como C++ e Java.

As linguagens desenvolvidas para a programação de inteligência artificial estão intimamente ligadas à estrutura teórica da área. Criamos muitas das estruturas representativas neste livro em Prolog, Lisp e Java, colocando-as à

disposição em Luger e Stubblefield (2009) e na Internet. Neste livro, não tomamos partido nas discussões ideológicas sobre seus méritos relativos. Em vez disso, seguimos o ditado “um bom trabalhador conhece todas as ferramentas”.

### 1.2.8 Aprendizado de máquina

O aprendizado permanece sendo uma área desafiadora para a IA. A importância do aprendizado, entretanto, é inquestionável, particularmente porque essa habilidade é um dos componentes mais importantes do comportamento inteligente. Um sistema especialista pode executar cálculos extensivos e custosos para resolver um problema. Entretanto, diferentemente de um ser humano, se em uma outra vez lhe for apresentado o mesmo problema ou outro similar, ele normalmente não se lembrará da solução. Ele realizará a mesma sequência de cálculos novamente. Isso é verdade para a segunda vez, bem como para a terceira, para a quarta e para qualquer outra oportunidade em que ele resolva o problema — o que não é o comportamento esperado de um sistema inteligente para resolver problemas. A solução óbvia é permitir que esses sistemas aprendam por conta própria, seja por sua própria experiência, por analogia, por exemplos, por um professor que lhes “diga” o que fazer, ou por recompensa ou punição, dependendo dos resultados.

Embora o aprendizado seja uma área difícil, há diversos programas que sugerem que isso não é impossível. Um programa pioneiro é o AM (*Automated Mathematician* — matemático automatizado), concebido para descobrir leis matemáticas (Lenat, 1977, 1982). A partir de conceitos e axiomas da teoria dos conjuntos inicialmente fornecidos, o AM foi capaz de induzir conceitos matemáticos importantes como cardinalidade, aritmética inteira e muitos resultados da teoria dos números. O AM formulou novos teoremas a partir da modificação da sua base de conhecimento atual e usou heurísticas para encontrar o “melhor” teorema entre uma série de teoremas possíveis. Mais recentemente, Cotton et al. (2000) concebeu um programa que inventa automaticamente sequências “interessantes” de inteiros.

Entre os trabalhos pioneiros influentes em aprendizado, está a pesquisa de Winston sobre a indução de conceitos estruturais como “arco” a partir de um conjunto de exemplos do mundo de blocos (Winston, 1975a). O algoritmo ID3 obteve sucesso em aprender padrões gerais a partir de exemplos (Quinlan, 1986a). O MetaDENDRAL aprende regras para interpretar dados de espectrografia de massa em química orgânica a partir de exemplos de dados extraídos de compostos de estrutura conhecida. *Tpiresias*, uma interface para sistemas especialistas, converte recomendações descritas em alto nível em regras novas para a sua base de regras (Davis, 1982). *Hacker* realiza o planejamento de manipulações no mundo de blocos a partir de um processo iterativo de conceber um plano, testá-lo e corrigir qualquer falha descoberta no plano candidato (Sussman, 1975). Os trabalhos em aprendizado baseado em explanação mostraram a eficácia do conhecimento prévio sobre o aprendizado (Mitchell et al., 1986; DeJong e Mooney, 1986). Atualmente, existem também muitos modelos biológicos e sociológicos importantes de aprendizado; apresentaremos uma revisão desses modelos nos capítulos sobre aprendizado conexionista e aprendizado emergente.

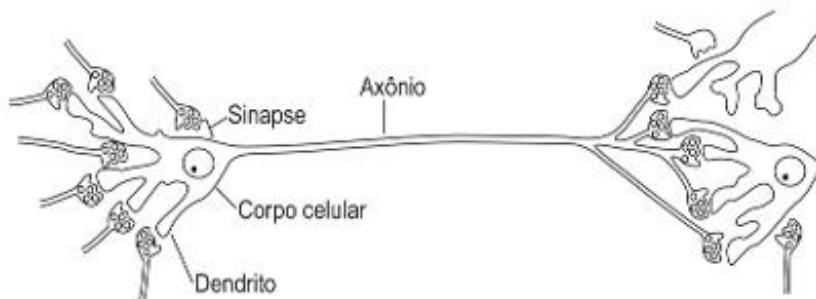
O sucesso dos programas de aprendizado de máquina sugere a existência de um conjunto de princípios gerais do aprendizado que permitirá a construção de programas com capacidade de aprender em domínios realistas. Apresentamos várias abordagens ao aprendizado na Seção IV.

### 1.2.9 Representações alternativas: redes neurais e algoritmos genéticos

A maioria das técnicas apresentadas neste livro sobre IA utiliza o conhecimento representado explicitamente e algoritmos de busca cuidadosamente concebidos para implementar a inteligência. Uma abordagem bem diferente procura construir programas inteligentes utilizando modelos que imitam a estrutura dos neurônios no cérebro humano ou padrões evolutivos encontrados nos algoritmos genéticos e em vida artificial.

Um esquema simples de um neurônio (Figura 1.2) consiste em um corpo celular com várias protuberâncias ramificadas, chamadas de *dendritos*, e um único ramo chamado de *axônio*. Os dendritos recebem sinais de outros neurônios. Quando esses impulsos combinados excedem um determinado limiar, o neurônio dispara e um impulso é

**Figura 1.2** Um diagrama simplificado de um neurônio a partir de Crick e Asanuma (1986).



propagado ao longo do axônio. Os ramos nas terminações do axônio formam *sinapses* com os dendritos de outros neurônios. A sinapse é o ponto de contato entre os neurônios e pode ser *excitatória* ou *inibitória*, dependendo se elas contribuem, respectivamente, para aumentar o sinal global ou, então, para diminuí-lo.

Essa descrição de um neurônio é excessivamente simples, mas capta as características que são relevantes para os modelos neurais de computação. Em particular, cada unidade computacional calcula uma função específica de suas entradas e passa o resultado para outras unidades da rede que estão conectadas a ela: os resultados finais são produzidos pelo processamento paralelo e distribuído dessa rede de conexões neurais e seus limiares de peso.

As arquiteturas neurais são mecanismos que possuem grande apelo para implementar a inteligência por várias razões. Os programas tradicionais de IA podem ser frágeis e excessivamente suscetíveis a ruído. A inteligência humana é muito mais flexível e consegue interpretar muito bem sinais ruidosos, como reconhecer um rosto em uma sala escura ou manter uma conversa durante uma festa barulhenta. As arquiteturas neurais, por capturarem o conhecimento por meio de um grande número de unidades distribuídas em uma rede, têm um potencial maior para reconhecer dados ruidosos e incompletos.

Com os algoritmos genéticos e a vida artificial, desenvolvemos novas soluções para o problema a partir de elementos de soluções anteriores. Os operadores genéticos, como *combinação* e *mutação*, à semelhança de seus equivalentes genéticos do mundo natural, produzem a cada nova geração soluções potenciais cada vez melhores para o problema. A vida artificial produz uma nova geração como uma função da “qualidade” de seus vizinhos em gerações anteriores.

Tanto as arquiteturas neurais como os algoritmos genéticos fornecem um modelo natural para o paralelismo, porque cada neurônio, ou segmento de uma solução, é uma unidade independente. Hillis (1985) observou o fato de que os seres humanos avançam mais rapidamente em uma tarefa à medida que adquirem mais conhecimento, enquanto os computadores tendem a diminuir o ritmo. Essa desaceleração se deve ao custo da busca sequencial em uma base de conhecimento; uma arquitetura maciçamente paralela como o cérebro humano não deveria sofrer desse problema. Por fim, existe algo intrinsecamente atraente ao abordar os problemas de inteligência do ponto de vista neural ou genético. Afinal, o cérebro evoluiu até alcançar a inteligência, utilizando, para isso, uma arquitetura neural. Apresentamos as redes neurais, os algoritmos genéticos e a vida artificial nos capítulos 10 e 11.

### 1.2.10 IA e filosofia

Na Seção 1.1, apresentamos as raízes filosóficas, matemáticas e sociais da inteligência artificial. É importante compreender que a moderna IA não é apenas um produto dessa rica tradição intelectual, mas ela também contribui para isso.

Por exemplo, as questões que Turing propôs sobre programas inteligentes, por exemplo, são refletidas também no nosso entendimento da própria inteligência. O que é inteligência e como ela pode ser descrita? Qual é a natureza do conhecimento? O conhecimento pode ser representado? Como o conhecimento em uma área de aplicação se

relaciona com a habilidade de resolver problemas nesse domínio? Como é que *saber o que* é verdadeiro — a *teoria de Aristóteles* — relaciona-se com *saber como* realizar — a sua *praxis*?

As respostas propostas para essas questões constituem uma parte importante das atividades dos pesquisadores em IA. Em termos científicos, os programas de IA podem ser vistos como experimentos. Uma concepção se torna concreta em um programa e o programa é executado como um experimento. Os projetistas desses programas observam os resultados e, então, modificam o projeto e realizam novamente o experimento. Dessa forma, podemos determinar se as nossas representações e os nossos algoritmos são modelos suficientes de comportamento inteligente. Newell e Simon (1976) propuseram essa abordagem para o entendimento científico na sua palestra do *Turing Award* de 1976, e um modelo mais forte para a inteligência por meio da sua hipótese de sistema simbólico físico: *a condição necessária e suficiente para um sistema físico exibir inteligência é que ele seja um sistema simbólico físico*. Na Seção VI, analisaremos o que essa hipótese significa na prática e como ela tem sido criticada por vários pensadores modernos.

Várias aplicações de IA suscitam, também, questões filosóficas profundas. Em que medida podemos dizer que um computador pode entender expressões em linguagem natural? Para produzir ou compreender uma linguagem, é necessária a interpretação de símbolos. Não é suficiente ser capaz de dizer que uma cadeia de símbolos está bem formada. Um mecanismo para a compreensão deve ser capaz de atribuir significado ou interpretar símbolos dentro de um contexto. O que é significado? O que é interpretação? Em que medida a interpretação requer responsabilidade?

Questões filosóficas similares emergem de muitas áreas de aplicação da IA, desde a construção de sistemas especialistas para cooperar com peritos humanos até o projeto de sistemas de visão computacional, ou ainda, o projeto de algoritmos para aprendizado de máquina. Abordaremos várias dessas questões quando elas surgirem nos diversos capítulos deste livro e retornaremos à questão geral da relevância para a filosofia na Seção VI.

### 1.3 Inteligência artificial — um resumo

Tentamos definir inteligência artificial a partir da discussão de suas áreas mais importantes de pesquisa e aplicação. Essa visão geral revela um campo de estudo jovem e promissor, cujo interesse principal é encontrar um modo efetivo de entender e aplicar técnicas inteligentes para a solução de problemas, para o planejamento e as habilidades de comunicação em uma ampla gama de problemas práticos. Apesar da variedade de problemas tratados pela inteligência artificial, emerge uma série de características importantes que parecem ser comuns a todas as divisões da área; entre elas se incluem:

1. O uso do computador para executar raciocínio, reconhecimento de padrões, aprendizado ou outras formas de inferência.
2. Um foco em problemas que não respondem a soluções algorítmicas. Isso implica a utilização de busca heurística como uma técnica de IA para a solução de problemas.
3. Um interesse na solução de problemas utilizando informação inexata, faltante ou insuficientemente definida, e o uso de formalismos representacionais que possibilitem ao programador compensar esses problemas.
4. Raciocínio que utiliza as características qualitativas significativas de uma situação.
5. Uma tentativa de tratar de questões que envolvem tanto significado semântico como forma sintática.
6. Respostas que não são nem exatas nem ótimas, mas que são “suficientes” em um certo sentido. Isso é resultado de se utilizar essencialmente métodos de solução de problemas baseados em heurísticas em situações em que resultados ótimos, ou exatos, são caros demais ou mesmo impossíveis.
7. O uso de grandes quantidades de conhecimento específico de um domínio para resolver problemas. Essa é a base dos sistemas especialistas.
8. O uso de metaconhecimento para produzir um controle mais sofisticado sobre as estratégias de resolução de problemas. Embora esse seja um problema muito difícil, tratado por relativamente poucos sistemas, ele vem emergindo como uma área essencial de pesquisa.

Esperamos que essa introdução tenha fornecido uma impressão geral da estrutura e do significado da área de inteligência artificial. Esperamos, também, que as breves discussões sobre questões técnicas, como busca e representação, não tenham sido excessivamente enigmáticas e obscuras; elas serão desenvolvidas em uma profundidade mais adequada ao longo deste texto e foram incluídas aqui para demonstrar a sua importância na organização geral da área.

Como mencionamos na discussão sobre solução de problemas orientada a agentes, os objetos adquirem significado a partir da sua relação com outros objetos. Isso é igualmente válido para fatos, teorias e técnicas que constituem uma área de estudo científico. A intenção é fornecer uma ideia geral desses inter-relacionamentos, de modo que, quando os temas técnicos da inteligência artificial forem apresentados separadamente, eles encontrarão o seu lugar em uma compreensão crescente da substância e em direções globais da área. Somos guiados por uma observação feita por Gregory Bateson, o psicólogo e teórico de sistemas (Bateson, 1979):

Se você quebrar o padrão que conecta os itens do aprendizado, você necessariamente destruirá toda a qualidade.

## 1.4 Epílogo e referências

---

A área da IA reflete alguns dos interesses mais antigos da civilização ocidental à luz do modelo computacional moderno. As noções de racionalidade, representação e raciocínio estão agora, talvez mais do que nunca, sob uma observação minuciosa, porque nós pesquisadores da IA necessitamos compreendê-las de modo algorítmico! Ao mesmo tempo, a situação política, econômica e ética da nossa espécie nos força a refletir sobre a nossa responsabilidade pelos efeitos das nossas conquistas.

Várias fontes excelentes disponíveis sobre os tópicos levantados neste capítulo são *Mind Design* (Haugeland, 1997), *Artificial Intelligence: The Very Idea* (Haugeland, 1985), *Brainstorms* (Dennett, 1978), *Mental Models* (Johnson-Laird, 1983), *Elbow Room* (Dennett, 1984), *The Body in the Mind* (Johnson, 1987), *Consciousness Explained* (Dennett, 1991) e *Darwin's Dangerous Idea* (Dennett, 1995), *Prehistory of Android Epistemology* (Glymour, Ford e Hayes, 1995a) e *Sweet Dreams* (Dennett, 2006).

Várias das fontes clássicas também estão disponíveis, incluindo as obras *Física*, *Metafísica* e *Lógica*, de Aristóteles; os artigos de Frege; os escritos de Babbage, Boole, e Russell, e Whitehead. Os artigos de Turing também são muito interessantes, especialmente as suas discussões sobre a natureza da inteligência e a possibilidade de construção de programas inteligentes (Turing, 1950). O famoso artigo de Turing (1937), *On Computable Numbers, with an Application to the Entscheidungsproblem*, elabora a teoria de máquinas de Turing e a definição de computabilidade. A biografia de Turing, *Alan Turing: The Enigma* (Hodges, 1983), também proporciona uma excelente leitura. *Pandemonium* (1959), de Selfridge, é um exemplo pioneiro de aprendizado. Uma importante coleção de artigos pioneiros em IA pode ser encontrada em Webber e Nilsson (1981).

*Computer Power and Human Reason* (Weizenbaum, 1976) e *Understanding Computers and Cognition* (Winograd e Flores, 1986) oferecem comentários soberbos sobre as limitações e questões éticas em IA. *The Sciences of the Artificial* (Simon, 1981) é uma afirmação positiva sobre as possibilidades da inteligência artificial e o seu papel na sociedade.

As aplicações da IA mencionadas na Seção 1.2 visam introduzir o leitor aos variados interesses dos pesquisadores de IA e procuram delinear muitas das questões importantes que estão sendo investigadas. Cada uma dessas subseções se refere às principais áreas deste livro, onde esses tópicos são apresentados. *The Handbook of Artificial Intelligence* (Barr e Feigenbaum, 1989) oferece uma introdução a cada uma dessas áreas. *Encyclopedia of Artificial Intelligence* (Shapiro, 1992) oferece uma visão clara e comprehensível do campo da inteligência artificial.

A compreensão da linguagem natural é um campo de estudo muito dinâmico; alguns pontos de vista importantes se encontram em *Natural Language Understanding* (Allen, 1995), *Language as a Cognitive Process* (Winograd, 1983), *Computer Models of Thought and Language* (Schank e Colby, 1973), *Grammar, Meaning and the Machine Analysis of Language* (Wilks, 1972), *The Language Instinct* (Pinker, 1994), *Philosophy in the Flesh*

(Lakoff e Johnson, 1999) e *Speech and Language Processing* (Jurafsky e Martin, 2009); uma introdução à área se encontra nos nossos capítulos 7 e 15.

O uso de computadores para modelar a capacidade humana, que tratamos rapidamente no Capítulo 17, é discutido com certa profundidade em *Human Problem Solving* (Newell e Simon, 1972), *Computation and Cognition* (Pylyshyn, 1984), *Arguments Concerning Representations for Mental Imagery* (Anderson, 1978) e *Cognitive Science: the Science of Intelligent Systems* (Luger, 1994), *Problem Solving as Model Refinement: Towards a Constructivist Epistemology* (Luger et al., 2002) e *Bayesian Brain* (Doya et al., 2007).

O aprendizado de máquina é discutido na Parte IV; são importantes fontes para esse assunto o conjunto de vários volumes, *Machine Learning* (Michalski et al., 1983, 1986; Kodratoff e Michalski, 1990), o *Journal of Artificial Intelligence* e o *Journal of Machine Learning*. Outras referências poderão ser encontradas nos quatro capítulos da Parte IV.

Por fim, o Capítulo 12 apresenta uma visão de inteligência que enfatiza a sua estrutura modular e a adaptação a um contexto social e natural. O *Society of Mind*, de Minsky (1985), é uma das primeiras e mais provocantes articulações desse ponto de vista. Veja também *Android Epistemology* (Ford et al., 1995b) e *Artificial Life* (Langton, 1995).

## 1.5 Exercícios

---

1. Crie e justifique sua definição para inteligência artificial.
2. Dê vários outros exemplos da distinção aristotélica entre “matéria” e “forma”. Você pode mostrar como os seus exemplos poderiam se encaixar em uma teoria da abstração?
3. Muito do pensamento ocidental tradicional se fundamenta na relação mente-corpo. A mente e o corpo são:
  - a. entidades distintas que interagem de alguma forma; ou
  - b. a mente é uma expressão de “processos físicos”; ou
  - c. o corpo é apenas uma ilusão da mente racional?Discuta suas ideias a respeito do problema mente-corpo e a sua importância para uma teoria da inteligência artificial.
4. Critique os critérios de Turing para que um sistema computacional seja “inteligente”.
5. Descreva seus próprios critérios para que um sistema computacional seja considerado “inteligente”.
6. Embora a computação seja uma disciplina relativamente nova, filósofos e matemáticos têm pensado por milhares de anos sobre as questões envolvidas na solução automática de problemas. Qual é a sua opinião sobre a relevância dessas questões filosóficas para a concepção de um dispositivo para a resolução inteligente de problemas? Justifique a sua resposta.
7. Dadas as diferenças entre as arquiteturas dos computadores modernos e a do cérebro humano, qual é a relevância da pesquisa sobre a estrutura fisiológica e a função de sistemas biológicos para a engenharia de programas de IA? Justifique a sua resposta.
8. Escolha uma área de aplicação na qual você considere que se justifique o trabalho necessário para implementar uma solução por sistema especialista. Explique o problema escolhido detalhadamente. Com base na sua própria intuição, quais aspectos dessa solução seriam mais difíceis de serem automatizados?
9. Cite dois benefícios dos sistemas especialistas, além daqueles já listados no texto. Discuta-os em termos de resultados intelectuais, sociais ou financeiros.
10. Discuta por que você acha que o problema do aprendizado de máquina é tão difícil.
11. Discuta se você acha possível, ou não, que um computador comprehenda e use uma linguagem natural (humana).
12. Liste e discuta dois efeitos potencialmente negativos para a sociedade do desenvolvimento de técnicas de inteligência artificial.



# *Influência artificial como representação e busca*

## **UMA PROPOSTA PARA O PROJETO DE PESQUISA DE VERÃO EM DARTMOUTH SOBRE INTELIGÊNCIA ARTIFICIAL (url IIa)**

*Propomos que um estudo da inteligência artificial com duração de 2 meses e que utilize 10 homens seja executado durante o verão de 1956 no Dartmouth College em Hanover, New Hampshire. O estudo deve prosseguir com base na conjectura de que cada aspecto do aprendizado ou qualquer outro recurso da inteligência pode, em princípio, ser descrito de forma tão precisa que uma máquina pode ser construída para simulá-lo. Será feita uma tentativa de descobrir como fazer máquinas usarem a linguagem, formarem abstrações e conceitos, resolverem tipos de problemas até agora reservados a seres humanos e se aprimorarem. Pensamos que um avanço significativo poderá ser feito em um ou mais desses problemas se um grupo de cientistas selecionados cuidadosamente trabalharem juntos nisso durante um verão.*

J. McCARTHY, *Dartmouth College*

M. L. MINSKY, *Harvard University*

N. ROCHESTER, *I.B.M. Corporation*

C.E. SHANNON, *Bell Telephone Laboratories*

31 de agosto de 1955

## **Introdução à representação e busca**

Do ponto de vista da engenharia, a descrição da inteligência artificial apresentada na Seção 1.3 pode ser resumida como *o estudo da representação e da busca por meio do qual a atividade inteligente pode ser executada em um dispositivo mecânico*. Esse ponto de vista dominou as origens e o crescimento da IA.

O primeiro workshop moderno (e a primeira conferência moderna) para profissionais de IA foi realizado no Dartmouth College no verão de 1956. A proposta para esse workshop é apresentada como a citação introdutória desta Parte II. Esse workshop, em que o próprio nome *inteligência artificial* foi escolhido, reuniu muitos dos então pesquisadores atuais voltados para a integração entre computação e inteligência. Naquela época já havia alguns programas de computador escritos para refletir essas primeiras ideias. Os principais tópicos de discussão nessa conferência, resumidos aqui a partir da proposta original do workshop (url IIa, nas Referências), foram:

**1. Computadores automáticos**

Se uma máquina pode cumprir uma tarefa, então uma calculadora automática pode ser programada para simular a máquina.

**2. Como um computador pode ser programado para usar uma linguagem?**

Pode-se especular que uma grande parte do pensamento humano consiste em manipular palavras de acordo com as regras de raciocínio e as regras de conjectura.

**3. Redes de neurônios**

Como um conjunto de neurônios (hipotéticos) pode ser arrumado de modo a formar conceitos?

**4. Teoria do tamanho de um cálculo**

Se recebermos um problema bem definido (para o qual seja possível testar mecanicamente se uma resposta proposta é ou não uma resposta válida), um modo de solucioná-lo é testar todas as respostas possíveis em ordem. Esse método é ineficaz e, para exclui-lo, deve-se ter algum critério para a eficiência de cálculo.

**5. Autoaperfeiçoamento (Aprendizado de máquina)**

Provavelmente, uma máquina verdadeiramente inteligente executará atividades que podem ser mais bem descritas como autoaperfeiçoamento.

**6. Abstrações**

Diversos tipos de “abstração” podem ser definidos distintamente, e vários outros, menos distintamente. Uma tentativa direta de classificá-los e de descrever métodos de máquina para formar abstrações a partir de dados sensoriais e outros pareceria valiosa.

**7. Aleatoriedade e criatividade**

Uma conjectura bastante atraente, embora ainda claramente incompleta, é que a diferença entre o pensamento criativo e o pensamento competente não imaginativo está na introdução de alguma aleatoriedade.

É interessante observar que os tópicos propostos para essa primeira conferência sobre inteligência artificial captam muitas das questões, como a teoria da complexidade, metodologias para abstração, projeto de linguagem e aprendizado de máquina, que compõem o foco da moderna ciência da computação. De fato, muitas das características de definição da ciência da computação conforme a conhecemos hoje têm suas raízes na IA, que também enfrentou seus próprios conflitos históricos e políticos, com vários desses tópicos propostos para pesquisa, como “redes neurais” e “aleatoriedade e criatividade”, sendo colocados em segundo plano por décadas.

Uma nova ferramenta computacional poderosa, a linguagem Lisp, surgiu nessa época, construída sob a direção de John McCarthy, um dos proponentes originais do Dartmouth Workshop. Lisp tratou de vários dos tópicos do workshop, dando suporte à capacidade de criar relacionamentos que poderiam ser eles mesmos manipulados por outras estruturas da linguagem. Lisp deu à inteligência artificial uma linguagem altamente expressiva, rica em abstração, e também um meio de interpretação dessas expressões.

A disponibilidade da linguagem de programação Lisp modelou grande parte do desenvolvimento inicial da IA, em particular o uso do cálculo de predicados como um meio de representação e também uma busca para explorar a eficácia das diferentes alternativas lógicas, o que agora chamamos de *busca em grafo*. A linguagem Prolog, criada no final da década de 1970, ofereceria à IA uma ferramenta de computação igualmente poderosa.

Uma introdução às técnicas fundamentais de representação e busca que dão suporte ao trabalho em inteligência artificial compõe os cinco capítulos da Parte II. O cálculo de predicados, a busca em grafo, os métodos heurísticos e estocásticos e as arquiteturas (sistemas de controle) para solucionar problemas de forma inteligente compõem o material da Parte II. Essas tecnologias refletem as técnicas dominantes exploradas pela comunidade de IA durante suas duas primeiras décadas.

## Sistemas de representação

A função de qualquer esquema de representação é capturar — ou *abstrair* — as características essenciais do domínio de um problema e fazer com que essa informação se torne disponível para um procedimento de resolução de problemas. A *abstração* é uma ferramenta essencial para gerenciar a complexidade e também um

fator importante para garantir que os programas resultantes sejam computacionalmente eficientes. *Expressividade* (o resultado das características abstraídas) e *eficiência* (a complexidade computacional dos algoritmos usados nas características abstraídas) são as principais dimensões para se avaliar linguagens de representação de conhecimento. Algumas vezes, a expressividade deve ser sacrificada para aumentar a eficiência de um algoritmo. Isso deve ser feito sem limitar a capacidade de capturar o conhecimento essencial para resolução de problemas. Otimizar o compromisso entre eficiência e expressividade é uma tarefa importante para projetistas de sistemas inteligentes.

As linguagens para a representação de conhecimento são também ferramentas para ajudar os seres humanos a resolver problemas. Sendo assim, uma representação deve fornecer um arcabouço *natural* para expressar conhecimento para solução de problemas; ela deve tornar o conhecimento disponível ao computador e ajudar o programador na sua organização.

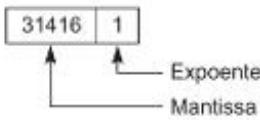
A representação de números em ponto flutuante pelo computador ilustra esses compromissos (ver Figura II.1). Para ser preciso, os números reais necessitam de uma cadeia infinita de dígitos para serem descritos completamente; isso não pode ser realizado em um dispositivo finito, como um computador. Uma solução para esse dilema é representar o número em duas partes: uma com seus dígitos *significativos* e a outra com a localização da vírgula decimal dentro desses dígitos. Embora não seja possível realmente armazenar um número real em um computador, é possível criar uma representação que funcione adequadamente na maioria das aplicações práticas.

A representação em ponto flutuante sacrifica o poder expressivo total para tornar a representação eficiente; nesse caso, para torná-la possível. A representação permite a implementação de algoritmos para aritmética de múltipla precisão, fornecendo precisão infinita pela limitação de erros de arredondamento a qualquer tolerância pré-especificada. Ela garante, também, erros de arredondamento bem comportados. Como toda representação, ela é apenas uma abstração, um padrão simbólico que designa uma entidade desejada, e não a entidade propriamente dita.

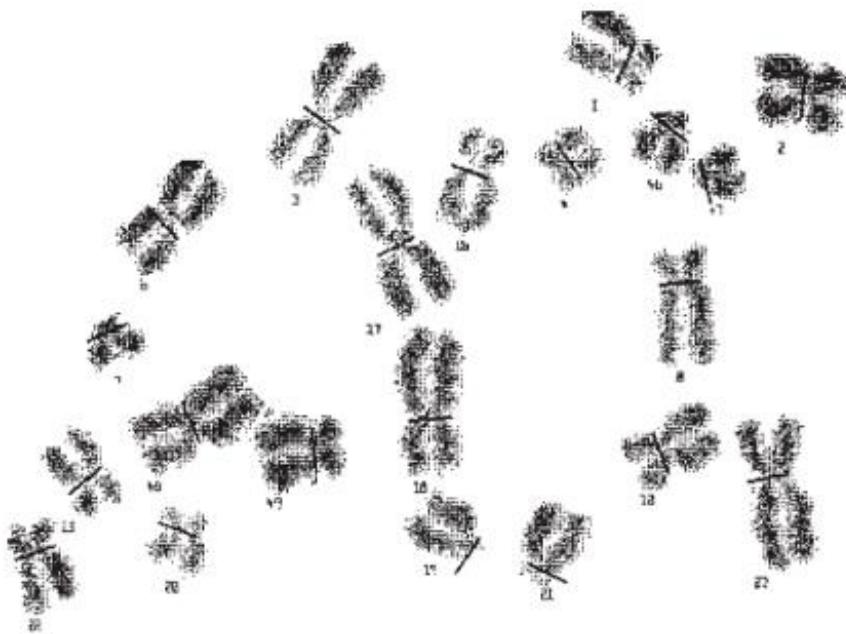
O arranjo é outra representação comum em ciência da computação. Para muitos problemas, ele é mais natural e eficiente que a arquitetura de memória implementada no hardware de computador. Esse ganho em naturalidade e eficiência envolve compromissos em expressividade, como ilustrado pelo exemplo de processamento de imagens a seguir. A Figura II.2 é uma imagem digitalizada de cromossomos humanos em um estágio chamado de *metáfase*. A imagem é processada para determinar o número e a estrutura dos cromossomos, procurando por quebras, partes faltantes e outras anomalias.

A cena visual é composta por um número de pontos de imagem. Cada ponto de imagem, ou *pixel*, tem tanto uma localização como um valor representando a sua intensidade ou o seu *tom de cinza*. É natural, então, capturar a cena completa em um arranjo bidimensional, em que o endereço da linha e da coluna fornece a localização de um pixel (coordenadas X e Y) e o conteúdo do elemento do arranjo é o tom de cinza daquele ponto. Podemos conceber algoritmos para realizar operações como procurar por pontos isolados para remover ruído da imagem, encontrar níveis de limiar para discernir objetos e bordas, somar elementos contíguos para determinar o tamanho e a densidade e transformar os dados dos pontos da imagem de várias outras formas. A implementação desses algoritmos é direta, considerando-se a representação de arranjo e a linguagem FORTRAN, por exemplo. Essa tarefa seria bastante trabalhosa se fossem usadas outras representações, como o cálculo de predicado, os registros ou o código assembly, porque elas não se ajustam naturalmente à matéria sendo representada.

**Figura II.1** Diferentes representações do número real  $\pi$ .

Número real:	$\pi$		
Equivalentes decimal:	3,1415927 ...		
Representação em ponto flutuante:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>31416</td> <td>1</td> </tr> </table> <div style="display: inline-block; vertical-align: middle; margin-left: 10px;">              Exponte            Mantissa         </div>	31416	1
31416	1		
Representação na memória do computador:	11100010		

**Figura II.2** Imagem digitalizada de cromossomos na metáfase.



Quando representamos a imagem como um arranjo de pixels, sacrificamos sua resolução (compare uma foto em um jornal com a impressão original da mesma imagem). Além disso, arranjos de pixels não são capazes de expressar a organização semântica mais profunda da imagem. Um arranjo de pixels não pode expressar, por exemplo, a organização de cromossomos em um núcleo de célula única, sua função genética ou o papel da metáfase na divisão celular. Esse conhecimento é capturado mais facilmente usando-se uma representação como o cálculo de predicado (Capítulo 2) ou as redes semânticas (Capítulo 7). Resumindo, um esquema de representação deverá ser adequado para expressar toda a informação necessária, dar suporte à execução eficaz do código resultante e fornecer um esquema natural para expressar o conhecimento exigido.

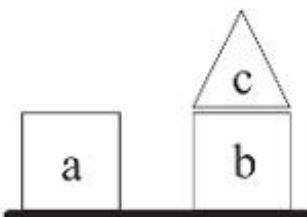
Em geral, os problemas que a IA tenta resolver não são adequados às representações oferecidas pelos formalismos mais tradicionais, como os arranjos. A inteligência artificial está mais preocupada com a solução qualitativa de problemas que com a sua solução quantitativa; com o raciocínio que com o cálculo; com a organização de grandes e variadas quantidades de conhecimento, que com a implementação de um único algoritmo bem definido.

Como um exemplo, considere a Figura II.3, que mostra uma organização de blocos sobre uma mesa. Suponha que queiramos capturar as propriedades e as relações exigidas para controlar o braço de um robô. Temos que determinar quais blocos estão empilhados sobre outros blocos e quais têm seu topo livre, de modo que possam ser agarrados. O *cálculo de predicados* captura diretamente essa informação descritiva. A primeira palavra de cada expressão (*sobre*, *sobre\_a\_mesa* etc.) é um *predicado* que indica uma propriedade ou um relacionamento entre os seus *argumentos* (que aparecem entre parênteses). Os argumentos são símbolos que representam objetos (blocos) do domínio. A coleção de cláusulas lógicas descreve as propriedades e os relacionamentos importantes do *mundo de blocos*:

```

livre(c)
livre(a)
sobre_a_mesa(a)
sobre_a_mesa(b)
sobre(c, b)
cubo(b)
cubo(a)
pirâmide(c)

```

**Figura II.3** Um mundo de blocos.

O cálculo de predicados oferece aos programadores de inteligência artificial uma linguagem bem definida para descrever e raciocinar a respeito dos aspectos qualitativos de um sistema. Suponha, no exemplo do mundo de blocos, que queiramos definir um teste para determinar se um bloco está livre, isto é, se não existe nada sobre ele. Isso é importante no caso de um braço de robô precisar pegá-lo ou colocar um outro bloco sobre ele. Podemos definir uma regra geral:

$$\forall X \rightarrow \exists Y \text{ sobre}(Y, X) \Rightarrow \text{livre}(X)$$

ou, “para todo  $X$ ,  $X$  está livre se não houver um  $Y$ , tal que  $Y$  esteja sobre  $X$ ”. Essa regra pode ser aplicada a uma série de situações distintas, bastando, para isso, atribuir nomes de blocos diferentes como  $a$ ,  $b$ ,  $c$  etc. para  $X$  e  $Y$ . Pelo suporte a regras gerais de inferência, o cálculo de predicados permite uma grande economia de representação e possibilita o projeto de sistemas que são suficientemente flexíveis e gerais para responder intelligentemente em diversas situações.

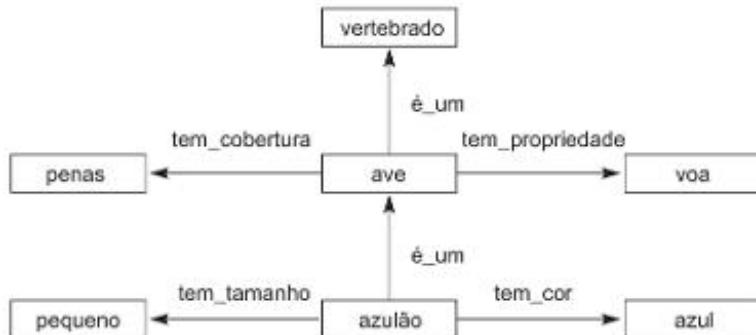
O cálculo de predicados também pode ser usado para representar as propriedades de indivíduos e grupos. Normalmente não é suficiente, por exemplo, descrever um carro simplesmente listando suas partes componentes; podemos querer descrever os modos como essas partes são combinadas e as interações entre elas. Essa visão da estrutura é essencial em diversas situações, incluindo informações taxonômicas, como a classificação de plantas por gênero e espécie, ou uma descrição de objetos complexos, como um motor a diesel ou um corpo humano em termos de suas partes componentes. Por exemplo, uma descrição simples de um pássaro azulão poderia ser “o azulão é uma ave pequena de cor azul, e uma ave é um vertebrado voador emplumado”. Isso pode ser representado como um conjunto de predicados lógicos:

```
tem_tamanho(azulão, pequeno)
tem_cobertura(ave, penas)
tem_cor(azulão, azul)
tem_propriedade(ave, voa)
é_um(azulão, ave)
é_um(ave, vertebrado)
```

Essa descrição de predicados pode ser representada graficamente utilizando *arcos*, ou *elos*, em um grafo, em vez de predicados para indicar os relacionamentos (Figura II.4). Essa descrição, chamada de *rede semântica*, é uma técnica para representar o significado semântico. Como os relacionamentos são representados explicitamente no grafo, um algoritmo para raciocinar sobre o domínio poderia fazer associações relevantes simplesmente seguindo os elos. No exemplo do azulão, um sistema necessita apenas seguir um elo para ver que um azulão voa e dois elos para determinar que um azulão é um vertebrado.

Talvez a aplicação mais importante das redes semânticas seja representar significados semânticos para programas de compreensão de linguagem natural. Quando é necessário compreender uma história de criança, os detalhes de um artigo de jornal ou os conteúdos de uma página Web, redes semânticas podem ser usadas para codificar a informação e os relacionamentos que refletem o conhecimento daquela aplicação. As redes semânticas são discutidas no Capítulo 6, e sua aplicação à compreensão da linguagem, no Capítulo 15.

**Figura II.4** Descrição por rede semântica de um azulão.



## Busca

Dada uma representação, o segundo componente para solucionar problemas de forma inteligente é a *busca*. Os humanos geralmente consideram diversas estratégias alternativas no seu caminho para solucionar um problema. Um jogador de xadrez normalmente analisa movimentos alternativos, selecionando o “melhor” de acordo com critérios como as respostas possíveis do oponente ou o grau ao qual diversos movimentos dão suporte a uma estratégia global do jogo. Um jogador também considera o ganho a curto prazo (como tomar a rainha do oponente), oportunidades de sacrificar uma peça para obter vantagem posicional ou conjecturas com relação à constituição psicológica e ao nível de habilidade do oponente. Esse aspecto do comportamento inteligente está por trás da técnica de solução de problemas da *busca em espaço de estados*.

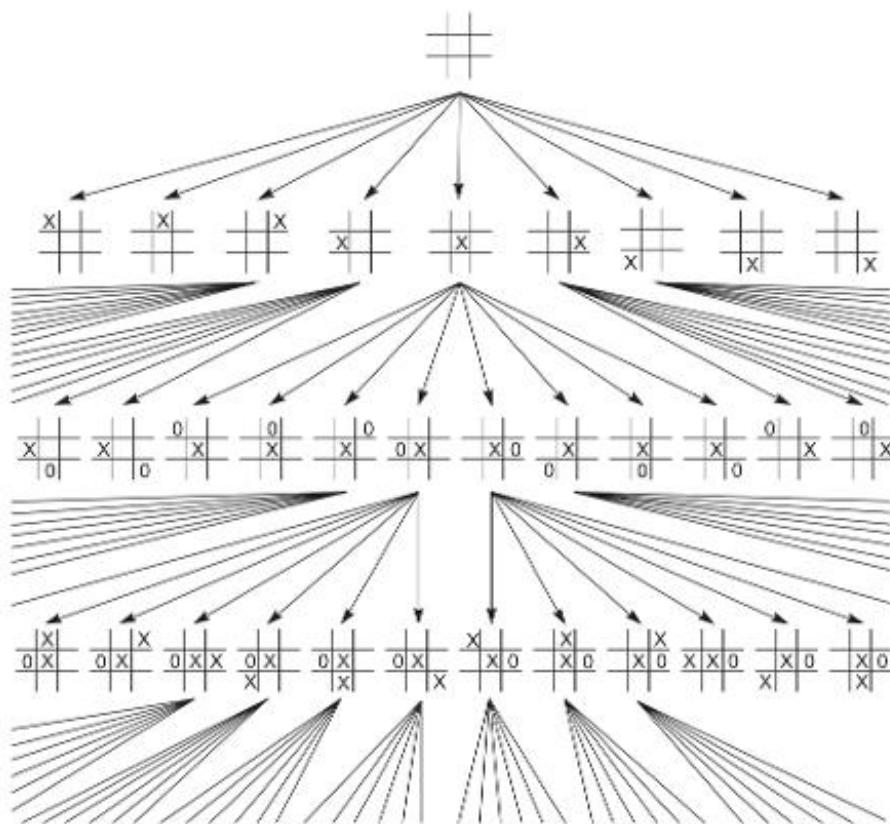
Considere, como um exemplo simples, o jogo da velha. Dada qualquer configuração de tabuleiro, existe apenas um número finito de movimentos que um jogador pode executar. Começando com um tabuleiro vazio, o primeiro jogador pode colocar um X em qualquer uma das nove posições. Cada um desses movimentos produz uma configuração de tabuleiro diferente que permitirá oito respostas possíveis do adversário, e assim por diante. Podemos representar essa coleção de jogadas e respostas possíveis considerando cada configuração de tabuleiro como um *nó* ou *estado* de um grafo. Os *elos* do grafo representam as jogadas válidas que levam de uma determinada configuração para outra. A estrutura resultante é chamada de *grafo de espaço de estados*.

A representação por espaço de estados nos permite, assim, tratar todos os jogos possíveis do jogo da velha como caminhos diferentes por meio desse grafo de espaço de estados. Dada essa representação, uma estratégia de jogo efetiva realizará uma busca através do grafo por caminhos que levem ao máximo de vitórias e ao mínimo de derrotas e jogará de forma que tente sempre forçar o jogo ao longo de um desses caminhos ótimos, como vemos na Figura II.5.

Como um exemplo de como a busca é usada para resolver um problema mais complicado, considere a tarefa de diagnosticar uma pane mecânica em um automóvel. Embora esse problema inicialmente não pareça ser tão adequado à busca em espaço de estados como o jogo da velha ou o jogo de xadrez, ele na verdade se enquadra bem nessa estratégia. Em vez de cada nó do grafo de espaço de estados representar um “estado do tabuleiro”, ele representa um estado de conhecimento parcial sobre problemas mecânicos do automóvel. Podemos pensar no processo de examinar os sintomas da pane e induzir a sua causa como uma realização de busca através de estados com conhecimento crescente. O nó inicial do grafo é vazio, o que indica que nada é conhecido sobre a causa do problema. A primeira coisa que um mecânico faria seria perguntar ao cliente qual o principal sistema (motor, transmissão, direção, freios etc.) que parece estar causando o problema. Isso é representado por uma coleção de arcos que saem do estado inicial para os estados que indicam um foco em um único subsistema do automóvel, como vemos na Figura II.6.

Cada um dos estados do grafo tem arcos (correspondendo a um teste básico de diagnóstico) que levam a estados representando acumulação extra de conhecimento no processo de diagnóstico. Por exemplo, o nó problema no

**Figura II.5** Parte do espaço de estados para o jogo da velha.

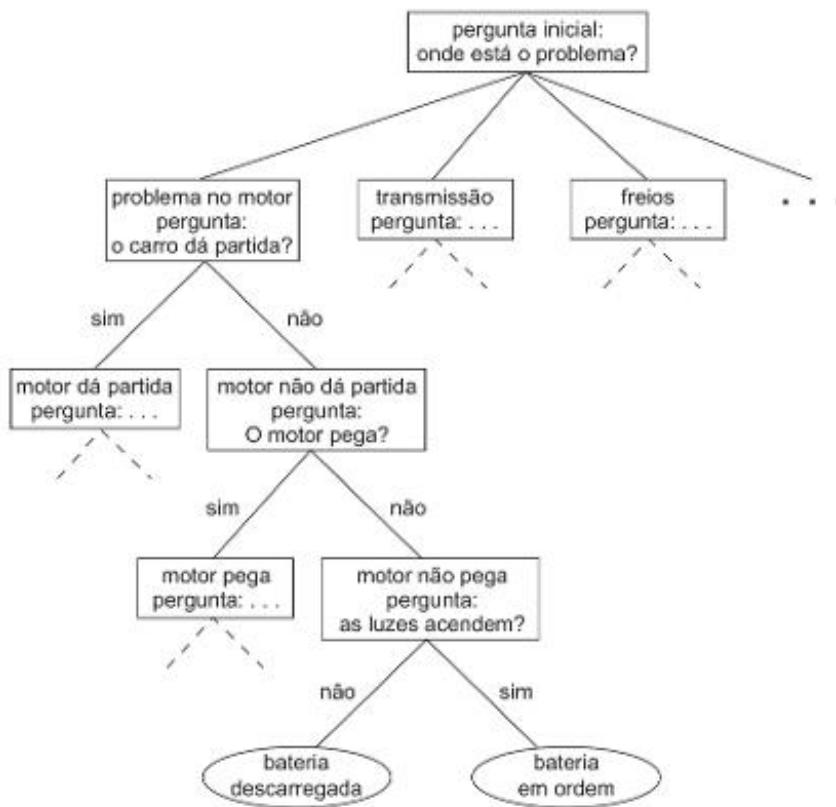


**Figura II.6** Descrição do espaço de estados do primeiro passo no diagnóstico de um problema automotivo.



motor tem arcos rotulados como motor dá partida e motor não dá partida. Do nó não dá partida podemos nos levar para os nós rotulados como pega e não pega. O nó não pega tem arcos para os nós rotulados como bateria desacarregada e bateria em ordem (ver Figura II.7). Um resolvedor de problemas pode diagnosticar um problema em um automóvel ao realizar uma busca por um caminho através desse grafo que seja consistente com os sintomas de um carro defeituoso em particular. Embora esse problema seja muito diferente daquele de encontrar um caminho ótimo para jogar o jogo da velha ou o xadrez, ele é igualmente apropriado para ser solucionado por busca em espaço de estados.

**Figura II.7** Descrição do espaço de estados de diagnóstico do problema automotivo.



Apesar dessa aparente universalidade, a busca em espaço de estados não é, por si só, suficiente para produzir um comportamento inteligente na resolução de problemas; ela é muito mais uma ferramenta importante para o desenvolvimento de programas inteligentes. Se a busca em espaço de estados fosse suficiente, seria muito simples escrever um programa para jogar xadrez, buscando através de todo o espaço completo pelas sequências de jogadas que levassem à vitória, um método conhecido como *busca exaustiva*. Embora a busca exaustiva possa ser aplicada a qualquer espaço de estados, o tamanho descomunal do espaço dos problemas interessantes torna essa abordagem impossível na prática. O jogo de xadrez, por exemplo, tem  $10^{120}$  diferentes estados de tabuleiro. Esse número é maior que o número de moléculas no universo ou o número de nanossegundos que se passaram desde o *big bang*. A busca em um espaço tão grande está muito além da capacidade de qualquer computador, cujas dimensões devem estar confinadas ao universo conhecido e cuja execução deve ser encerrada antes que o universo sucumba à devastação da entropia.

Os seres humanos utilizam busca inteligente: um jogador de xadrez considera uma variedade de jogadas possíveis, um médico examina vários diagnósticos possíveis, um cientista da computação considera diferentes concepções antes de escrever um código. Os seres humanos não utilizam a busca exaustiva: o jogador de xadrez examina apenas as jogadas que a experiência mostrou serem efetivas, o médico não requer testes que não sejam de alguma forma indicados pelo sintoma presente. A resolução de problemas pelo ser humano parece ser baseada em regras de julgamento que guiam a nossa busca áquelas partes do espaço de estados que são, de alguma forma, mais "promissoras".

Essas regras são conhecidas como *heurísticas* e constituem um dos tópicos centrais da pesquisa em IA. Uma heurística (o nome vem da palavra grega que significa "descobrir") é uma estratégia para a busca seletiva de um espaço de problema. Ela guia a nossa busca ao longo de linhas que têm uma alta probabilidade de sucesso, ao mesmo tempo evitando esforços desnecessários e aparentemente estúpidos. Os seres humanos utilizam um grande

número de heurísticas para resolver problemas. Se você perguntar a um mecânico por que o seu carro está superaquecendo, ele pode lhe dizer algo como: “Normalmente isso significa que o termostato está estragado”. Se você perguntar a um médico o que causa náusea e dor de estômago, ele poderá lhe dizer que “provavelmente seja uma virose ou, então, uma intoxicação alimentar”.

A busca em espaço de estados nos dá um meio de formalizar o processo de resolução de problemas, e heurísticas nos permitem adicionar inteligência a esse formalismo. Essas técnicas são discutidas em detalhe nos primeiros capítulos deste livro e permanecem sendo o cerne da maioria dos trabalhos modernos em IA. Em resumo, a busca em espaço de estados é um formalismo, independente de qualquer estratégia particular de busca, e é utilizada como ponto de partida para muitas abordagens na resolução de problemas.

Ao longo deste livro, continuaremos a explorar os aspectos teóricos da representação de conhecimento e da busca e o uso dessa teoria para a construção de programas eficazes. O tratamento da representação do conhecimento começa com o cálculo de predicados (Capítulo 2). O Capítulo 3 introduz a busca no contexto de grafos de jogos e outras aplicações. No Capítulo 4, as heurísticas são introduzidas e aplicadas a algoritmos de busca em grafos, incluindo jogos. No Capítulo 5, apresentamos técnicas estocásticas (probabilísticas) para construir e organizar espaços de busca; estas serão usadas mais adiante em áreas que incluem aprendizado de máquina e processamento de linguagem natural. Por fim, o Capítulo 6 introduz o sistema de produção, quadros-negros e outras arquiteturas de software que integram a representação e a busca, sustentando assim a construção de resolvedores de problemas inteligentes.

# Cálculo de predicados

*Nós alcançamos a posse total de nosso poder de realizar inferências, a última das nossas faculdades; ela não é apenas um dom natural, mas também uma arte longa e difícil.*

— C. S. PIERCE

*A qualidade essencial de uma prova é compelir a crença.*

— FERMAT

## 2.0 Introdução

Neste capítulo, introduzimos o cálculo de predicados como uma linguagem de representação para a inteligência artificial. A importância do cálculo de predicados foi discutida na introdução da Parte II; essas vantagens incluem uma *semântica formal* bem definida e regras de inferência *consistentes* e *completas*. Este capítulo começa com uma breve revisão (opcional) do cálculo proposicional (Seção 2.1). A Seção 2.2 define a sintaxe e a semântica do cálculo de predicados. Na Seção 2.3, discutimos as regras de inferência do cálculo de predicados e seu uso para a solução de problemas. Por fim, o capítulo demonstra o uso do cálculo de predicados para implementar uma base de conhecimento para aconselhamento de investimentos financeiros.

## 2.1 Cálculo proposicional (opcional)

### 2.1.1 Símbolos e sentenças

O cálculo proposicional e, na próxima subseção, o cálculo de predicados são, em primeiro lugar, linguagens. Utilizando suas palavras, frases e sentenças, podemos representar e raciocinar sobre as propriedades e os relacionamentos do mundo. O primeiro passo para descrever uma linguagem é introduzir os elementos que a constituem: o seu conjunto de símbolos.

**Definição****SÍMBOLOS DO CÁLCULO PROPOSICIONAL**

Os *símbolos* do cálculo proposicional são os símbolos proposicionais:

P, Q, R, S, ...

os símbolos de valor verdade:

verdadeiro, falso

e os conectivos:

$\wedge$ ,  $\vee$ ,  $\neg$ ,  $\rightarrow$ ,  $\equiv$

Os símbolos proposicionais denotam *proposições* ou declarações acerca do mundo que podem ser verdadeiras ou falsas, tais como “o carro é vermelho” ou “a água é molhada”. As proposições são denotadas por letras maiúsculas do final do alfabeto. As sentenças do cálculo proposicional são formadas por esses símbolos atômicos de acordo com as seguintes regras:

**Definição****SENTENÇAS DO CÁLCULO PROPOSICIONAL**

Cada símbolo proposicional e símbolo de valor verdade é uma sentença.

Por exemplo: verdadeiro, P, Q e R são sentenças.

A negação de uma sentença é uma sentença.

Por exemplo:  $\neg P$  e  $\neg$  falso são sentenças.

A conjunção, ou e, de duas sentenças é uma sentença.

Por exemplo:  $P \wedge \neg P$  é uma sentença.

A disjunção, ou ou, de duas sentenças é uma sentença.

Por exemplo:  $P \vee \neg P$  é uma sentença.

A implicação de uma sentença a partir de outra é uma sentença.

Por exemplo:  $P \rightarrow Q$  é uma sentença.

A equivalência de duas sentenças é uma sentença.

Por exemplo:  $P \vee Q \equiv R$  é uma sentença.

As sentenças válidas também são chamadas de *fórmulas bem formadas* ou *FBFs*.

Em expressões da forma  $P \wedge Q$ , P e Q são chamados de *membros da conjunção*. Em  $P \vee Q$ , P e Q são chamados de *membros da disjunção*. Em uma implicação,  $P \rightarrow Q$ , P é a *premissa*, ou *antecedente*, e Q é a *conclusão*, ou *consequente*.

Em sentenças do cálculo proposicional, os símbolos ( ) e [ ] são usados para agrupar símbolos em subexpressões e, assim, controlar a sua ordem de avaliação e significado.  $(P \vee Q) = R$ , por exemplo, é bastante diferente de  $P \vee (Q \equiv R)$ , como pode ser demonstrado pela utilização das tabelas verdade (ver Seção 2.1.2).

Uma expressão é uma sentença, ou uma fórmula bem formada, do cálculo proposicional se, e somente se, ela puder ser formada por símbolos válidos a partir de uma sequência dessas regras. Por exemplo,

$$((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$$

é uma sentença bem formada no cálculo proposicional, porque:

- $P, Q$  e  $R$  são proposições e, portanto, são sentenças.
- $P \wedge Q$ , a conjunção de duas sentenças, é uma sentença.
- $(P \wedge Q) \rightarrow R$ , a implicação de uma sentença por uma outra, é uma sentença.
- $\neg P$  e  $\neg Q$ , as negações de sentenças, são sentenças.
- $\neg P \vee \neg Q$ , a disjunção de duas sentenças, é uma sentença.
- $\neg P \vee \neg Q \vee R$ , a disjunção de duas sentenças, é uma sentença.
- $((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$ , a equivalência de duas sentenças, é uma sentença.

Essa é a sentença original, que foi construída a partir de uma série de aplicações de regras válidas e, portanto, “bem formada”.

### 2.1.2 Semântica do cálculo proposicional

A Seção 2.1.1 apresentou a sintaxe do cálculo proposicional definindo um conjunto de regras para produzir sentenças válidas. Nesta seção, definimos formalmente a *semântica* ou o “significado” dessas sentenças. Como os programas de IA devem poder raciocinar com estruturas de representação, é importante demonstrar que a verdade de suas conclusões depende apenas da verdade do seu conhecimento inicial ou premissas, isto é, que não são introduzidos erros lógicos pelos procedimentos de inferência. Um tratamento preciso da semântica é essencial para esse objetivo.

Um símbolo proposicional corresponde a uma declaração sobre o mundo. Por exemplo,  $P$  pode denotar a declaração “está chovendo” ou  $Q$ , a declaração “eu moro em uma casa marrom”. Uma proposição pode ser verdadeira ou falsa, dado algum estado do mundo. A atribuição do valor verdade para sentenças proposicionais é chamada de *interpretação*, uma asserção sobre a sua verdade em um *mundo possível*.

Formalmente, uma interpretação é um mapeamento dos símbolos proposicionais para o conjunto  $\{V, F\}$ . Como mencionado na seção anterior, os símbolos verdadeiro e falso são parte do conjunto de sentenças bem formadas do cálculo proposicional; isto é, eles são distintos do valor verdade atribuído a uma sentença. Para reforçar essa distinção, os símbolos  $V$  e  $F$  são usados para atribuir o valor verdade.

Cada mapeamento possível de valores verdade para proposições corresponde a um mundo de interpretação possível. Se, por exemplo,  $P$  representar a proposição “está chovendo” e  $Q$  representar “estou no trabalho”, então o conjunto de proposições  $\{P, Q\}$  terá quatro mapeamentos funcionais diferentes para os valores verdade  $\{V, F\}$ . Esses mapeamentos correspondem a quatro interpretações diferentes. A semântica do cálculo proposicional, assim como a sua sintaxe, é definida por indução:

#### Definição

#### SEMÂNTICA DO CÁLCULO PROPOSICIONAL

Uma *interpretação* de um conjunto de proposições é a atribuição de um valor verdade,  $V$  ou  $F$ , para cada símbolo proposicional.

Ao símbolo verdadeiro sempre se atribui o valor  $V$ , e ao símbolo falso é atribuído o valor  $F$ .

A interpretação do valor verdade de sentenças é determinada por:

A atribuição do valor verdade da negação,  $\neg P$ , onde  $P$  é um símbolo proposicional qualquer, é  $F$  se a atribuição a  $P$  for  $V$ , e  $V$  se a atribuição a  $P$  for  $F$ .

A atribuição do valor verdade da conjunção,  $\wedge$ , é  $V$  somente quando ambos os termos da conjunção tiverem valor verdade  $V$ ; caso contrário, o seu valor é  $F$ .

A atribuição do valor verdade da disjunção,  $\vee$ , é  $F$  somente quando ambos os termos da disjunção tiverem valor verdade  $F$ ; caso contrário, o seu valor é  $V$ .

A atribuição do valor verdade da *implicação*,  $\rightarrow$ , é F somente quando a premissa, ou o símbolo antes da implicação, for V e o valor verdade do consequente, ou símbolo após a implicação, for F; caso contrário, ele é V.

A atribuição do valor verdade da *equivalência*,  $\equiv$ , é V somente quando ambas as expressões tiverem o mesmo valor verdade para todas as interpretações possíveis; caso contrário, o seu valor é F.

As atribuições de valor verdade de proposições compostas são frequentemente descritas por *tabelas verdade*. Uma tabela verdade lista todas as atribuições de valores verdade para as proposições atômicas de uma expressão e dá o valor verdade da expressão para cada atribuição. Assim, uma tabela verdade enumera todos os mundos de interpretação possíveis que podem ser dados a uma expressão. A tabela verdade para  $P \wedge Q$ , por exemplo, na Figura 2.1, lista os valores verdade para cada atribuição de valor verdade possível dos operandos.  $P \wedge Q$  é verdadeiro apenas quando ambos P e Q são V. Ou ( $\vee$ ), não ( $\neg$ ), implica ( $\rightarrow$ ) e equivalência ( $\equiv$ ) são definidos de forma similar. A construção dessas tabelas verdade é deixada como exercício.

Duas expressões do cálculo proposicional são equivalentes se elas têm o mesmo valor para todas as atribuições de valores verdade. Essa equivalência pode ser demonstrada usando tabelas verdade. Como exemplo, uma prova de equivalência de  $P \rightarrow Q$  e  $\neg P \vee Q$  é dada pela tabela verdade da Figura 2.2.

Pela demonstração de que duas sentenças diferentes do cálculo proposicional têm tabelas verdade idênticas, podemos provar as seguintes equivalências. Sendo P, Q e R expressões proposicionais:

$$\neg(\neg P) \equiv P$$

$$(P \vee Q) \equiv (\neg P \rightarrow Q)$$

$$\text{a lei da contrapositiva: } (P \rightarrow Q) \equiv (\neg Q \rightarrow \neg P)$$

$$\text{a lei de "de Morgan": } \neg(P \vee Q) \equiv (\neg P \wedge \neg Q) \text{ e } \neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$$

$$\text{as leis comutativas: } (P \wedge Q) \equiv (Q \wedge P) \text{ e } (P \vee Q) \equiv (Q \vee P)$$

$$\text{a lei associativa: } ((P \wedge Q) \wedge R) \equiv (P \wedge (Q \wedge R))$$

$$\text{a lei associativa: } ((P \vee Q) \vee R) \equiv (P \vee (Q \vee R))$$

$$\text{a lei distributiva: } P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$

$$\text{a lei distributiva: } P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$$

Identidades como essas podem ser usadas para transformar expressões do cálculo proposicional em formas sintaticamente diferentes, mas logicamente equivalentes. Essas identidades podem ser usadas em vez das tabelas verdade para provar que duas expressões são equivalentes: encontrar uma série de identidades que transforma uma das expressões na outra. O programa de IA, *Logic Theorist* (Newell e Simon, 1956), projetado por Newell, Simon e Shaw, usava transformações entre formas equivalentes de expressões para provar muitos dos teoremas do *Principia Mathematica* (1950), de Whitehead e Russell. A capacidade de transformar uma expressão lógica em uma forma diferente com valores verdade equivalentes é importante também quando se utilizam regras de inferência (*modus ponens*, Seção 2.3, e resolução, Capítulo 14) que requerem que as expressões estejam em uma forma específica.

**Figura 2.1** Tabela verdade para o operador  $\wedge$ .

P	Q	$P \wedge Q$
V	V	V
V	F	F
F	V	F
F	F	F

**Figura 2.2** Tabela verdade demonstrando a equivalência de  $P \rightarrow Q$  e  $\neg P \vee Q$ .

P	Q	$\neg P$	$\neg P \vee Q$	$P \Rightarrow Q$	$(\neg P \vee Q) = (P \Rightarrow Q)$
V	V	F	V	V	V
V	F	F	F	F	V
F	V	V	V	V	V
F	F	V	V	V	V

## 2.2 Cálculo de predicados

No cálculo proposicional, cada símbolo atômico ( $P$ ,  $Q$  etc.) denota uma proposição individual única. Não há como acessar os componentes de uma asserção individual. O cálculo de predicados proporciona essa capacidade. Como exemplo, considere que, em vez de fazermos com que o símbolo proposicional  $P$  denotasse a sentença inteira “choveu na terça-feira”, pudéssemos criar um predicado *tempo* que descrevesse a relação entre uma data e o tempo: *tempo(terça-feira, chuva)*. Por meio de regras de inferência, podemos manipular expressões do cálculo de predicados, acessando seus componentes individuais e inferindo novas sentenças.

O cálculo de predicados permite, também, que as expressões contenham variáveis, que nos permitem criar asserções gerais sobre classes de entidades. Por exemplo, poderíamos afirmar que, para todos os valores de  $X$ , onde  $X$  é um dia da semana, a declaração *tempo(X, chuva)* é verdadeira; isto é, chove todos os dias. Como no cálculo proposicional, primeiro definimos a sintaxe da linguagem e, então, discutimos a sua semântica.

### 2.2.1 Sintaxe dos predicados e das sentenças

Antes de definirmos a sintaxe de expressões corretas no cálculo de predicados, definimos um alfabeto e uma gramática para criar *símbolos* da linguagem. Isso corresponde ao aspecto léxico da definição de uma linguagem de programação. Os símbolos do cálculo de predicados, como os *tokens* de uma linguagem de programação, são elementos sintáticos irredutíveis: eles não podem ser quebrados em suas partes componentes pelas operações da linguagem.

Neste livro, representamos os símbolos do cálculo de predicados como cadeias de letras e dígitos que começam com uma letra. Espaços e caracteres não alfanuméricos não podem aparecer dentro de uma cadeia, embora o sublinhado, \_, possa ser usado para aumentar a legibilidade.

#### Definição

#### SÍMBOLOS DO CÁLCULO DE PREDICADOS

O alfabeto que compõe os símbolos do cálculo de predicados é:

1. O conjunto de letras, tanto minúsculas quanto maiúsculas.
2. O conjunto de dígitos, 0, 1, ..., 9.
3. O caractere de sublinhado, \_.

*Símbolos* no cálculo de predicados começam com uma letra e são seguidos por uma sequência qualquer desses caracteres válidos.

Entre os caracteres válidos do alfabeto de símbolos do cálculo de predicados se incluem

a R 6 9 p \_ z

Exemplos de caracteres não incluídos no alfabeto:

# % @ / &

Entre os símbolos válidos do cálculo de predicados, estão incluídos:

George fogo3 tom\_e\_jerry bill XXXX amigos\_de

Exemplos de cadeias que não são símbolos válidos:

3jack nenhum espaço é permitido ab%cd \*\*\*71 pato!!!

Os símbolos, como vimos na Seção 2.2.2, são usados para indicar objetos, propriedades ou relações do mundo de discurso. Assim como a maioria das linguagens de programação, o uso de “palavras” que sugerem o significado intencional do símbolo nos ajuda a entender o código do programa. Assim, apesar de *a(g, k)* e *ama(george, kate)* serem formalmente equivalentes (isto é, eles têm a mesma estrutura), o segundo pode ser de grande ajuda (para leitores humanos), por indicar qual a relação que a expressão representa. Devemos salientar que esses nomes descritivos têm apenas a intenção de melhorar a legibilidade das expressões. A única maneira de se determinar o “significado” de uma expressão do cálculo de predicados é por meio da sua semântica formal.

Os parênteses “( )”, as vírgulas “,” e os pontos “.” são usados apenas para construir expressões bem formadas, e não denotam objetos ou relações do mundo. Esses símbolos são chamados de *impróprios*.

Os símbolos do cálculo de predicados podem representar *variáveis*, *constants*, *funções* ou *predicados*. Constantes nomeiam objetos específicos ou propriedades do mundo. Símbolos de constantes devem começar com uma letra minúscula. Assim, *george*, *planta*, *alto* e *azul* são exemplos de símbolos de constantes bem formados. As constantes *verdadeiro* e *falso* são reservadas para os *símbolos de valor verdade*.

Os símbolos de variáveis são usados para designar classes gerais de objetos ou propriedades do mundo. Variáveis são representadas por símbolos que começam com uma letra maiúscula. Assim, *George*, *BILL* e *KATE* são variáveis válidas, enquanto *geORGE* e *bill* não o são.

O cálculo de predicados permite, também, funções sobre objetos do mundo de discurso. Símbolos de funções (como as constantes) começam com uma letra minúscula. Funções representam um mapeamento de um ou mais elementos de um conjunto (o *domínio* da função) para um único elemento de um outro conjunto (o *contradomínio* da função). Os elementos do domínio e do contradomínio são objetos do mundo de discurso. Além das funções aritméticas comuns, como a adição e a multiplicação, as funções podem definir mapeamentos entre domínios não numéricos.

Note que a nossa definição de símbolos do cálculo de predicados não inclui números ou operadores aritméticos. O sistema numérico não está incluído nas primitivas do cálculo de predicados; em vez disso, ele é definido axiomaticamente, usando como base o cálculo de predicados “puro” (Manna e Waldinger, 1985). Apesar de os detalhes dessa derivação serem interessantes do ponto de vista teórico, eles são menos importantes para o uso do cálculo de predicados como uma linguagem de representação de IA. Por conveniência, pressupomos essa derivação e incluímos na linguagem a aritmética.

Cada símbolo funcional tem uma *aridade* associada, indicando o número de elementos do domínio que são mapeados para cada elemento do contradomínio. Assim, *pai* pode denotar uma função de aridade 1, que mapeia pessoas para seu (único) pai; mas poderia ser uma função de aridade 2 que mapeia dois números para a sua soma aritmética.

Uma *expressão funcional* é um símbolo funcional seguido de seus argumentos. Os argumentos são elementos do domínio da função; o número de argumentos é igual à aridade da função. Os argumentos se encontram entre parênteses, separados por vírgulas. Por exemplo,

*f(X,Y)*  
*pai(david)*  
*preço(bananas)*

são expressões funcionais bem formadas.

Cada expressão funcional denota o mapeamento dos argumentos para um único objeto do contradomínio, chamado de *valor* da função. Por exemplo, se *pai* é uma função unária, então

*pai(david)*

é uma expressão funcional cujo valor (no mundo de discurso do autor) é *george*. Se *mais* é uma função com aridade 2, no domínio dos inteiros, então

*mais(2,3)*

é uma expressão funcional cujo valor é o inteiro 5. O ato de substituir uma função pelo seu valor é chamado de *avaliação*.

O conceito de um símbolo, ou termo do cálculo de predicados, é formalizado pela seguinte definição:

### Definição

## SÍMBOLOS e TERMOS

Os símbolos do cálculo de predicados incluem:

1. *Símbolos de verdade* verdadeiro e falso (esses são símbolos reservados).
2. *Símbolos de constante* são expressões simbólicas que têm como primeiro caractere uma letra minúscula.
3. *Símbolos de variável* são expressões simbólicas que começam com uma letra maiúscula.
4. *Símbolos de função* são expressões simbólicas que têm como primeiro caractere uma letra minúscula. As funções têm uma aridade associada que indica o número de elementos do domínio que são mapeados para cada elemento do contradomínio.

Uma expressão funcional consiste de um símbolo de função de aridade n, seguido por n termos,  $t_1, t_2, \dots, t_n$ , entre parênteses e separados por vírgulas.

Um termo do cálculo de predicados é uma constante, uma variável ou, então, uma expressão funcional.

Assim, um *termo* do cálculo de predicados pode ser usado para denotar objetos e propriedades de um domínio de problema. São exemplos de termos:

*gato*

*vezes(2,3)*

*X*

*azul*

*mãe(sarah)*

*kate*

Símbolos no cálculo de predicados também podem representar predicados. Símbolos de predicados, como nomes de constantes e de funções, começam com uma letra minúscula. Um predicado denota um relacionamento entre zero ou mais objetos do mundo. O número de objetos assim relacionados corresponde à aridade do predicado. Exemplos de predicados são:

*gosta é igual sobre perto parte\_de*

Uma *sentença atômica*, a unidade mais primitiva da linguagem do cálculo de predicados, é um predicado de aridade n seguido de n termos entre parênteses e separados por vírgulas. São exemplos de sentenças atômicas:

gosta(george,kate)	gosta(X,george)
gosta(george,susie)	gosta(X,X)
gosta(george,sarah,terça)	amigos(bill,richard)
amigos(bill,george)	amigos(pai_de(david),pai_de(andrew))
ajuda(bill,george)	ajuda(richard,bill)

Os símbolos de predicados nessas expressões são gosta, amigos e ajuda. Um símbolo de predicado pode ser usado com diferentes números de argumentos. Nesse exemplo, há dois gosta diferentes, um com dois argumentos e o outro com três. Quando um símbolo de predicado é usado em sentenças com diferentes aridades, considera-se que ele representa duas relações diferentes. Assim, uma relação de predicado é definida por seu nome e por sua aridade. Não há razão para que os dois gosta não possam fazer parte da mesma descrição de mundo; entretanto, isso deve ser evitado porque pode causar confusão.

Nos predicados anteriores, bill, george, kate etc. são símbolos de constantes e representam objetos do domínio do problema. Os argumentos de um predicado são termos e podem incluir, também, variáveis ou expressões funcionais. Por exemplo,

amigos(pai\_de(david),pai\_de(andrew))

é um predicado que descreve uma relação entre dois objetos em um domínio de discurso. Esses argumentos são representados como expressões funcionais cujos mapeamentos (dado que o pai\_de david é george e o pai\_de andrew é allen) formam os parâmetros do predicado. Se as expressões funcionais forem avaliadas, as expressões se tornam

amigos(george,allen)

Essas ideias são formalizadas na definição a seguir.

### Definição

## PREDICADOS e SENTENÇAS ATÔMICAS

Símbolos de predicados são símbolos que começam com uma letra minúscula.

Os predicados têm um inteiro positivo associado referido como *aridade* ou “número de argumentos” do predicado. Predicados com o mesmo nome, mas com diferentes aridades, são considerados distintos.

Uma sentença atômica é uma constante de predicado de aridade n, seguida de n termos, t<sub>1</sub>, t<sub>2</sub>, ..., t<sub>n</sub>, entre parênteses e separados por vírgulas.

Os valores verdade, verdadeiro e falso, são também sentenças atômicas.

Sentenças atômicas são também chamadas de expressões atômicas, átomos ou proposições.

Podemos combinar sentenças atômicas usando operadores lógicos para formar *sentenças* do cálculo de predicados. Esses operadores são os mesmos conectivos lógicos usados no cálculo proposicional:  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\rightarrow$ ,  $\equiv$ .

Quando uma variável aparece como um argumento em uma sentença, ela se refere a objetos não especificados do domínio. O cálculo de predicados de primeira ordem (Seção 2.2.2) inclui dois símbolos, os *quantificadores de variáveis*  $\forall$  e  $\exists$ , que restringem o significado de uma sentença que contém uma variável. Um quantificador é seguido por uma variável e uma sentença, como

$\exists Y \text{ amigos}(Y, \text{peter})$

$\forall X \text{ gosta}(X, \text{sorvete})$

O *quantificador universal*,  $\forall$ , indica que a sentença é verdadeira para todos os valores da variável. No exemplo,  $\forall X \text{ gosta}(X, \text{sorvete})$  é verdadeiro para todos os valores do domínio da definição de X. O *quantificador existencial*,  $\exists$ ,

tencial,  $\exists$ , indica que a sentença é verdadeira para pelo menos um valor do domínio.  $\exists Y \text{ amigos}(Y, \text{peter})$  é verdadeiro se existir pelo menos um objeto indicado por  $Y$  que seja amigo de peter. Os quantificadores são discutidos mais detalhadamente na Seção 2.2.2.

As sentenças no cálculo de predicados são definidas indutivamente.

### Definição

## SENTENÇAS DO CÁLCULO DE PREDICADOS

Cada sentença atômica é uma sentença.

1. Se  $s$  é uma sentença, então a sua negação,  $\neg s$ , também o é.
2. Se  $s_1$  e  $s_2$  são sentenças, então a sua conjunção,  $s_1 \wedge s_2$ , também o é.
3. Se  $s_1$  e  $s_2$  são sentenças, então a sua disjunção,  $s_1 \vee s_2$ , também o é.
4. Se  $s_1$  e  $s_2$  são sentenças, então a sua implicação,  $s_1 \rightarrow s_2$ , também o é.
5. Se  $s_1$  e  $s_2$  são sentenças, então a sua equivalência,  $s_1 \equiv s_2$ , também o é.
6. Se  $X$  é uma variável e  $s$  é uma sentença, então  $\forall X s$  é uma sentença.
7. Se  $X$  é uma variável e  $s$  é uma sentença, então  $\exists X s$  é uma sentença.

A seguir, apresentamos exemplos de sentenças bem formadas. Suponha que `vezes` e `mais` sejam símbolos de funções de aridade 2 e que `igual` e `foo` sejam símbolos de predicados de aridade 2 e 3, respectivamente.

`mais(dois,três)` é uma função e, portanto, não é uma sentença atômica.

`igual(mais(dois, três), cinco)` é uma sentença atômica.

`igual(mais(2, 3), sete)` é uma sentença atômica. Note que essa sentença, dada a interpretação padrão de `mais` e `igual`, é falsa. O fato de ser bem formada e o seu valor verdade são questões diferentes.

$\exists X \text{ foo}(X, \text{dois}, \text{mais}(\text{dois}, \text{três})) \wedge \text{igual}(\text{mais}(\text{dois}, \text{três}), \text{cinco})$  é uma sentença porque ambos os termos da conjunção são sentenças.

$(\text{foo}(\text{dois}, \text{dois}, \text{mais}(\text{dois}, \text{três}))) \rightarrow (\text{igual}(\text{mais}(\text{três}, \text{dois}), \text{cinco}) = \text{verdadeiro})$  é uma sentença porque todos os seus componentes são sentenças, adequadamente conectadas por operadores lógicos.

A definição de sentenças do cálculo de predicados e os exemplos que acabamos de apresentar sugerem um método para verificar se uma expressão é uma sentença. Esse método é escrito como um algoritmo recursivo, `verifica_sentença`. `verifica_sentença` recebe como argumento uma expressão candidata e retorna `SUCESSO` se a expressão for uma sentença.

```

função verifica_sentença(expressão);
    início
        caso
            expressão é uma sentença atômica: retorna SUCESSO;
            expressão é da forma Q X s, onde Q é ∀ ou ∃, X é uma variável,
                se verifica_sentença(s) retorna SUCESSO
                então retorna SUCESSO
                senão retorna FALHA;
            expressão é da forma ¬ s:
                se verifica_sentença(s) retorna SUCESSO
                então retorna SUCESSO
                senão retorna FALHA;
            expressão é da forma s1 op s2, onde op é um operador lógico binário:

```

```

se verifica_sentença(s1) retorna SUCESSO e verifica_sentença(s2) retorna SUCESSO
então retorna SUCESSO
senão retorna FALHA;
senão: retorna FALHA
fim
fim.

```

Concluímos esta seção com um exemplo de cálculo de predicados para descrever um mundo simples. O domínio de discurso é um conjunto de relações familiares de uma genealogia bíblica:

```

mãe(eva,abel)
mãe(eva,caim)
pai(adão,abel)
pai(adão,caim)

 $\forall X \forall Y \text{pai}(X, Y) \vee \text{mãe}(X, Y) \rightarrow \text{genitor}(X, Y)$ 
 $\forall X \forall Y \forall Z \text{genitor}(X, Y) \wedge \text{genitor}(X, Z) \rightarrow \text{irmãos}(Y, Z)$ 

```

Nesse exemplo, usamos os predicados *mãe* e *pai* para definir um conjunto de relações pais-filhos. As implicações fornecem as definições gerais de outras relações, como de *genitor* e de *irmãos*, em termos desses predicados. Intuitivamente, está claro que essas implicações podem ser usadas para inferir fatos como *irmãos(caim,abel)*. Para formalizar o processo, de modo que ele possa ser realizado em um computador, deve-se ter o cuidado de definir algoritmos de inferência e assegurar que eles realmente fornecam as conclusões corretas para um conjunto de asserções do cálculo de predicados. Para tanto, definimos a semântica do cálculo de predicados (Seção 2.2.2) e, então, abordamos a questão das regras de inferência (Seção 2.3).

## 2.2.2 Semântica para o cálculo de predicados

Tendo definido expressões bem formadas do cálculo de predicados, é importante determinar o seu significado em termos de objetos, propriedades e relações no mundo. A semântica do cálculo de predicados fornece uma base formal para determinar o valor verdade de expressões bem formadas. A verdade de expressões depende do mapeamento de constantes, variáveis, predicados e funções para objetos e relações do domínio de discurso. A verdade de relações do domínio determina a verdade das expressões correspondentes.

Informações sobre uma pessoa, por exemplo, George, e suas amigas Kate e Susie, podem ser expressas por

```

amigos(george,susie)
amigos(george,kate)

```

Se realmente for verdade que George é um amigo de Susie e que George é um amigo de Kate, então essas expressões teriam o valor verdade (atribuição) V. Se George é um amigo de Susie, mas não de Kate, então a primeira expressão teria o valor verdade V e a segunda, o valor verdade F.

Para usar o cálculo de predicados como uma representação para a solução de problemas, descrevemos objetos e relações do domínio de interpretação com um conjunto de expressões bem formadas. Os termos e predicados dessas expressões denotam objetos e relações no domínio. Essa base de dados de expressões do cálculo de predicados, cada uma com o valor verdade V, descreve o “estado do mundo”. A descrição de George e seus amigos é um exemplo simples desse tipo de base de dados. Um outro exemplo é o *mundo de blocos* na introdução da Parte II.

Com base nessa percepção, definimos formalmente a semântica do cálculo de predicados. Primeiro, definimos uma *interpretação* de um domínio D. Em seguida, usamos essa interpretação para determinar a *atribuição do valor verdade* de sentenças da linguagem.

### Definição

## INTERPRETAÇÃO

Seja o domínio D um conjunto não vazio.

Uma *interpretação* de D é uma atribuição das entidades de D a cada um dos símbolos de constante, variável, predicado e função de uma expressão do cálculo de predicados, tal que:

1. A cada constante é atribuído um elemento de D.
2. A cada variável é atribuído um subconjunto não vazio de D: essas são as substituições admissíveis para essa variável.
3. Cada função f de aridade m é definida em m argumentos de D e define um mapeamento de  $D^m$  para D.
4. Cada predicado p de aridade n é definido em n argumentos de D e define um mapeamento de  $D^n$  para {V, F}.

Dada uma interpretação, o significado de uma expressão é uma atribuição do valor verdade sobre a interpretação.

### Definição

## VALOR VERDADE DE EXPRESSÕES DO CÁLCULO DE PREDICADOS

Suponha uma expressão E e uma interpretação I para E em um domínio não vazio D. O valor verdade para E é determinado por:

1. O valor de uma constante é o elemento de D ao qual ele é atribuído por I.
2. O valor de uma variável é o conjunto de elementos de D ao qual ela é atribuída por I.
3. O valor de uma expressão funcional é aquele elemento de D obtido pela avaliação da função para os valores dos parâmetros atribuídos pela interpretação.
4. O valor do símbolo de valor verdade "verdadeiro" é V e "falso" é F.
5. O valor de uma sentença atômica é V ou F, conforme determinado pela interpretação I.
6. O valor da negação de uma sentença é V, se o valor da sentença for F, e F, se o valor da sentença for V.
7. O valor da conjunção de duas sentenças é V, se o valor de ambas as sentenças for V, e F, caso contrário.
- 8-10. O valor verdade de expressões que utilizam  $\vee$ ,  $\rightarrow$  e  $\equiv$  é determinado a partir dos seus operandos, como definido na Seção 2.1.2.

Finalmente, para uma variável X e uma sentença S contendo X:

11. O valor de  $\forall X S$  é V se S for V para todas as atribuições a X por I, e F, caso contrário.
12. O valor de  $\exists X S$  é V se existir uma atribuição a X pela interpretação segundo a qual S é V; caso contrário, é F.

A quantificação de variáveis é uma parte importante da semântica do cálculo de predicados. Quando uma variável aparece em uma sentença, como X em  $\text{gosta}(\text{george}, X)$ , a variável funciona como um marcador de lugar. Qualquer constante permitida segundo a interpretação pode substitui-la na expressão. Substituindo X por kate ou susie em  $\text{gosta}(\text{george}, X)$ , formam-se as declarações  $\text{gosta}(\text{george}, \text{kate})$  e  $\text{gosta}(\text{george}, \text{susie})$ , como vimos anteriormente.

A variável X representa todas as constantes que poderiam aparecer como o segundo parâmetro da sentença. Esse nome de variável poderia ser substituído por qualquer outro nome de variável, tal como Y ou PESSOAS, sem

alterar o significado da expressão. Por isso, diz-se que a variável é um símbolo *fictício*. No cálculo de predicados, as variáveis podem ser *quantificadas* de duas formas: *universalmente* ou *existencialmente*. Uma variável é considerada *livre* se ela não estiver contida no escopo de qualquer um dos quantificadores, universal ou existencial. Uma expressão é *fechada* se todas as suas variáveis forem quantificadas. Uma *expressão fundamental* não tem qualquer variável. No cálculo de predicados, todas as variáveis devem ser quantificadas.

Frequentemente, são usados parênteses para indicar o *escopo* da quantificação, isto é, as ocorrências de um nome de variável sobre as quais vale a quantificação. Assim, o símbolo indicador de quantificação universal,  $\forall$ :

$$\forall X (p(X) \vee q(Y) \rightarrow r(X))$$

indica que  $X$  é quantificado universalmente tanto em  $p(X)$  como em  $r(X)$ .

A quantificação universal introduz problemas ao calcular o valor verdade de uma sentença, porque todos os valores possíveis de um símbolo de variável devem ser testados para verificar se a expressão permanece verdadeira. Por exemplo, para testar o valor verdade de  $\forall X$  gosta(george, X), onde  $X$  abrange o conjunto de todos os seres humanos, todos os possíveis valores de  $X$  devem ser testados. Se o domínio de uma interpretação for infinito, o teste exaustivo de todas as substituições para uma variável quantificada universalmente é computacionalmente impossível: provavelmente o algoritmo nunca chegará ao fim. Por causa desse problema, diz-se que o cálculo de predicados é *indecidível*. Como o cálculo proposicional não usa variáveis, as sentenças têm apenas um número finito de atribuições de valores verdade possíveis, e podemos testar exaustivamente todas as atribuições possíveis. Isso é o que é feito com a tabela verdade, Seção 2.1.

Como vimos na Seção 2.2.1, as variáveis podem também ser quantificadas *existencialmente*, indicadas pelo símbolo  $\exists$ . Nesse caso, diz-se que a expressão que contém a variável é verdadeira para pelo menos uma substituição do seu domínio de definição. O escopo de uma variável quantificada existencialmente é indicado, também, colocando-se entre parênteses as ocorrências quantificadas da variável.

Avaliar o valor verdade de uma expressão que contém uma variável quantificada existencialmente pode não ser mais fácil que avaliar o valor verdade de expressões que contêm variáveis quantificadas universalmente. Suponha que desejemos determinar o valor verdade da expressão por meio de tentativas de substituições até que seja encontrada uma que torne a expressão verdadeira. Se o domínio da variável for infinito e a expressão for falsa para todas as substituições, o algoritmo nunca chegará ao fim.

Várias relações entre a negação e os quantificadores universal e existencial são apresentadas abaixo. Essas relações são usadas em sistemas de resolução por refutação descritos no Capítulo 14. A noção de um nome de variável como um símbolo fictício, que representa um conjunto de constantes, é também mencionada. Para predicados  $p$  e  $q$  e variáveis  $X$  e  $Y$ :

$$\begin{aligned} \neg \exists X p(X) &\equiv \forall X \neg p(X) \\ \neg \forall X p(X) &\equiv \exists X \neg p(X) \\ \exists X p(X) &\equiv \exists Y p(Y) \\ \forall X q(X) &\equiv \forall Y q(Y) \\ \forall X (p(X) \wedge q(X)) &\equiv \forall X p(X) \wedge \forall Y q(Y) \\ \exists X (p(X) \vee q(X)) &\equiv \exists X p(X) \vee \exists Y q(Y) \end{aligned}$$

Na linguagem que definimos, variáveis quantificadas universalmente e existencialmente podem se referir somente a objetos (constantes) do domínio de discurso. Predicados e nomes de funções não podem ser substituídos por variáveis quantificadas. Essa linguagem é chamada de *cálculo de predicados de primeira ordem*.

### Definição

## CÁLCULO DE PREDICADOS DE PRIMEIRA ORDEM

O *cálculo de predicados de primeira ordem* permite que variáveis quantificadas se refiram a objetos do domínio de discurso, e não a predicados ou funções.

Por exemplo,

$\forall (\text{Gosta}) \text{Gosta}(\text{george}, \text{kate})$

não é uma expressão bem formada no cálculo de predicados de primeira ordem. Existem cálculos de *ordem mais elevada* em que tais expressões têm significado. Alguns pesquisadores (McCarthy, 1968; Appelt, 1985) utilizaram linguagens de ordem mais elevada para representar conhecimento em programas de compreensão de linguagem natural.

Muitas sentenças gramaticamente corretas (em um determinado idioma) podem ser representadas no cálculo de predicados de primeira ordem usando os símbolos, conectivos e símbolos de variáveis definidos nesta seção. É importante notar que não há um mapeamento único de sentenças para expressões do cálculo de predicados; na realidade, uma sentença em português (ou inglês) pode ter qualquer número de representações diferentes no cálculo de predicados. Um dos maiores desafios para os programadores de IA é encontrar um esquema para utilizar esses predicados que otimize a expressividade e a eficiência da representação resultante. Exemplos de sentenças em português representadas no cálculo de predicados são:

Se não chover na segunda, Tom irá para a serra.

$\neg \text{tempo}(\text{chover}, \text{segunda}) \rightarrow \text{ir}(\text{tom}, \text{serra})$

Emma é um doberman e um bom animal.

$\text{bom\_animal}(\text{emma}) \wedge \text{é\_um}(\text{emma}, \text{doberman})$

Todos os jogadores de basquete são altos.

$\forall X (\text{jogador\_basquete}(X) \rightarrow \text{alto}(X))$

Algumas pessoas gostam de anchova.

$\exists X (\text{pessoa}(X) \wedge \text{gosta}(X, \text{anchova}))$

Se desejos fossem cavalos, mendigos cavalgariam.

$\text{igual}(\text{desejos}, \text{cavalos}) \rightarrow \text{cavalar}(\text{mendigos})$

Ninguém gosta de impostos.

$\neg \exists X \text{gosta}(X, \text{impostos})$

### 2.2.3 Exemplo de significado semântico do “mundo de blocos”

Concluímos esta seção apresentando um exemplo estendido de uma atribuição de valor verdade a um conjunto de expressões do cálculo de predicados. Suponha que desejemos modelar o mundo de blocos da Figura 2.3 para projetar, por exemplo, um algoritmo de controle para um braço de robô. Podemos usar sentenças do cálculo de predicados para representar os relacionamentos qualitativos existentes no mundo: um determinado bloco tem a superfície superior livre? Podemos pegar o bloco a? etc. Suponha que o computador tenha conhecimento da localização de cada bloco e do braço e que ele seja capaz de se manter informado sobre essas localizações (utilizando coordenadas tridimensionais) conforme a garra move os blocos sobre a mesa.

Devemos ser muito precisos sobre o que propomos com esse exemplo de “mundo de blocos”. Primeiro, estamos criando um conjunto de expressões do cálculo de predicados que deve representar um instantâneo do domínio do problema do mundo de blocos. Como veremos na Seção 2.3, esse conjunto de blocos oferece uma *interpretação* e um *modelo* possível para o conjunto de expressões do cálculo de predicados.

Segundo, o cálculo de predicados é *declarativo*, isto é, não há qualquer regulação de tempo nem uma ordem predeterminada para considerar cada expressão. Apesar disso, na Seção 8.4 deste livro, que aborda planejamento, introduziremos uma “semântica procedural”, ou seja, uma metodologia claramente especificada para avaliar essas expressões ao longo do tempo. Um exemplo concreto de uma semântica procedural para expressões do cálculo de predicados é o Prolog, na Seção 14.3. Esse cálculo situacional que criamos introduzirá uma série de questões, incluindo o problema relacionado com o uso de quadros (*frames*) e a questão de as interpretações lógicas serem *não monotônicas*, abordadas mais adiante neste livro. Para esse exemplo, contudo, é suficiente dizer que as nossas expressões do cálculo de predicados serão avaliadas de cima para baixo e da esquerda para a direita.

## 2.3 Usando regras de inferência para produzir expressões do cálculo de predicados

### 2.3.1 Regras de inferência

A semântica do cálculo de predicados fornece uma base para uma teoria formal de *inferência lógica*. A capacidade de inferir novas expressões corretas a partir de um conjunto de asserções verdadeiras é uma característica importante do cálculo de predicados. Essas novas expressões são corretas na medida em que elas são *consistentes* com todas as interpretações prévias do conjunto original de expressões. Primeiro, discutimos essas ideias informalmente e, então, criamos um conjunto de definições para torná-las precisas.

Dizemos que uma interpretação que torna uma sentença verdadeira *satisfaz* essa sentença. Dizemos que uma interpretação que satisfaz todos os membros de um conjunto de expressões satisfaz esse conjunto. Uma expressão  $X$  *resulta logicamente* de um conjunto de expressões  $S$  do cálculo de predicados, se toda a interpretação que satisfaz  $S$  também satisfaz  $X$ . Essa noção nos dá uma base para verificar a correção de regras de inferência: a função da inferência lógica é produzir novas sentenças que resultem logicamente de um dado conjunto de sentenças do cálculo de predicados.

É importante que o significado preciso de *resultar logicamente* seja entendido: para que a expressão  $X$  *resulte logicamente* de  $S$ , ela deve ser verdadeira para toda interpretação que satisfaça o conjunto original de expressões  $S$ . Isso significaria, por exemplo, que qualquer expressão nova do cálculo de predicados que fosse adicionada ao mundo de blocos da Figura 2.3 teria de ser verdadeira naquele mundo, assim como em qualquer outra interpretação que aquele conjunto de expressões pudesse ter.

O próprio termo, “resulta logicamente”, pode ser um pouco confuso. Ele não significa que  $X$  é deduzido de  $S$  nem mesmo que ele é dedutível de  $S$ . Ele apenas significa que  $X$  é verdadeiro para toda interpretação que satisfaz  $S$ . Entretanto, pelo fato de os sistemas de predicados poderem ter um número potencialmente infinito de interpretações possíveis, raramente é prático tentar todas as interpretações. Em vez disso, as *regras de inferência* fornecem um modo computacionalmente viável de determinar quando uma expressão, um componente de uma interpretação, resulta logicamente daquela interpretação. O conceito de “resultar logicamente” fornece uma base formal para provas de correção e consistência de regras de inferência.

Uma regra de inferência é, essencialmente, um meio mecânico de se produzir novas sentenças do cálculo de predicados a partir de outras sentenças, isto é, as regras de inferência produzem novas sentenças com base na forma sintática de asserções lógicas dadas. Quando toda sentença  $X$  produzida por uma regra de inferência que opera sobre um conjunto  $S$  de expressões lógicas resulta logicamente de  $S$ , dizemos que a regra de inferência é *consistente*.

Se a regra de inferência for capaz de produzir toda expressão que resulta logicamente de  $S$ , então dizemos que ela é *completa*. O *modus ponens*, que será introduzido a seguir, e a *resolução*, introduzida no Capítulo 11, são exemplos de regras de inferência que são consistentes e, quando usadas com estratégias de aplicação apropriadas, são completas. Sistemas de inferência lógica geralmente utilizam regras de inferência consistentes, embora alguns capítulos adiante (4, 9 e 15) examinem o raciocínio heurístico e o raciocínio de senso comum, e ambos relaxam esse requisito.

Formalizamos essas ideias por meio das seguintes definições:

#### Definição

#### SATISFAZ, MODELO, VÁLIDO, INCONSISTENTE

Para uma expressão  $X$  do cálculo de predicados e uma interpretação  $I$ :

Se  $X$  tem um valor  $V$  sob  $I$  e sob uma atribuição particular de variáveis, então dizemos que  $I$  *satisfaz*  $X$ .

Se  $I$  satisfaz  $X$  para todas as atribuições de variáveis, então  $I$  é um *modelo* de  $X$ .

$X$  é *satisfazível* se, e somente se, existir uma interpretação e uma atribuição de variável que o satisfaça; caso contrário, ele é *insatisfazível*.

Um conjunto de expressões é *satisfazível* se, e somente se, existir uma interpretação e uma atribuição de variável que satisfaça cada elemento.

Se um conjunto de expressões não é satisfazível, dizemos que é *inconsistente*.

Se  $X$  tem um valor  $V$  para todas as interpretações possíveis, dizemos que  $X$  é *válido*.

No exemplo de mundo de blocos da Figura 2.3, o mundo de blocos foi um modelo para a sua descrição lógica. Todas as sentenças do exemplo são verdadeiras sob essa interpretação. Quando uma base de conhecimento é implementada como um conjunto de asserções verdadeiras sobre um domínio de problema, esse domínio é um modelo para a base de conhecimento.

A expressão  $\exists X (p(X) \wedge \neg p(X))$  é inconsistente porque ela não pode ser satisfeita sob nenhuma interpretação ou atribuição de variável. Por outro lado, a expressão  $\forall X (p(X) \vee \neg p(X))$  é válida.

O método da tabela verdade pode ser usado para testar a validade de qualquer expressão que não contenha variáveis. Como não é sempre possível decidir a validade de expressões contendo variáveis (como mencionado anteriormente, o processo pode não chegar ao fim), o cálculo de predicados completo é “indecidível”. Entretanto, existem *procedimentos de prova* que podem produzir qualquer expressão que resulte logicamente de um conjunto de expressões. Esses procedimentos são chamados de procedimentos de prova *completos*.

### Definição

## PROCEDIMENTOS DE PROVA

Um *procedimento de prova* é uma combinação de uma regra de inferência e de um algoritmo para aplicar essa regra a um conjunto de expressões lógicas para gerar novas sentenças.

No Capítulo 11, apresentamos os procedimentos de prova para a regra de inferência da resolução.

Usando essas definições, podemos definir formalmente o que significa “resulta logicamente”.

### Definição

## RESULTA LOGICAMENTE, CONSISTENTE e COMPLETO

Uma expressão  $X$  do cálculo de predicados *resulta logicamente* de um conjunto  $S$  de expressões do cálculo de predicados se toda interpretação e toda atribuição de variável que satisfaça  $S$  também satisfizer  $X$ .

Uma regra de inferência é *consistente* se toda expressão do cálculo de predicados produzida pela regra aplicada a um conjunto  $S$  de expressões do cálculo de predicados também resultar logicamente de  $S$ .

Uma regra de inferência é *completa* se, dado um conjunto  $S$  de expressões do cálculo de predicados, a regra puder inferir todas as expressões que resultam logicamente de  $S$ .

O *modus ponens* é uma regra de inferência consistente. Se tivermos uma expressão da forma  $P \rightarrow Q$  e uma outra da forma  $P$ , tal que ambas sejam verdadeiras sob uma interpretação  $I$ , então o *modus ponens* nos permite

inferir que Q também é verdadeira para aquela interpretação. De fato, como o *modus ponens* é consistente, Q é verdadeira para *todas* as interpretações para as quais P e  $P \rightarrow Q$  são verdadeiras.

O *modus ponens* e várias outras regras de inferência úteis são definidos a seguir.

### Definição

#### **MODUS PONENS, MODUS TOLLENS, ELIMINAÇÃO DO E, INTRODUÇÃO DO E e INSTANCIACÃO UNIVERSAL**

Se soubermos que as sentenças P e  $P \rightarrow Q$  são verdadeiras, então o *modus ponens* permite inferir Q.

Pela regra de inferência *modus tollens*, se soubermos que  $P \rightarrow Q$  é verdadeira e que Q é falsa, então podemos inferir que P é falsa:  $\neg P$ .

A *eliminação do E* nos permite inferir o valor verdade de cada um dos termos de uma conjunção. Por exemplo,  $P \wedge Q$  nos permite concluir que P e Q são verdadeiras.

A *introdução do E* nos permite inferir o valor verdade de uma conjunção a partir da verdade de seus termos. Por exemplo, se P e Q são verdadeiras, então  $P \wedge Q$  é verdadeira.

A *instanciação universal* determina que, se qualquer variável quantificada universalmente em uma sentença verdadeira, digamos,  $p(X)$ , for substituída por um termo apropriado qualquer do domínio, o resultado é uma sentença verdadeira. Assim, se a é do domínio de X,  $\forall X p(X)$  nos permite inferir  $p(a)$ .

Como um simples exemplo do uso do *modus ponens* no cálculo proposicional, suponha as seguintes observações: “se estiver chovendo o chão estará molhado” e “está chovendo”. Se P denotar “está chovendo” e Q for “o chão está molhado”, então a primeira expressão se torna  $P \rightarrow Q$ . Como realmente está chovendo agora (P é verdadeira), nosso conjunto de axiomas se torna

$$\begin{array}{l} P \rightarrow Q \\ P \end{array}$$

Através da aplicação do *modus ponens*, o fato de que o chão está molhado (Q) pode ser adicionado ao conjunto de expressões verdadeiras.

O *modus ponens* pode ser aplicado também a expressões que contenham variáveis. Considere como exemplo o silogismo comum “todos os homens são mortais e Sócrates é um homem; portanto, Sócrates é mortal”. “Todos os homens são mortais” pode ser representada no cálculo de predicados por

$$\forall X (\text{homem}(X) \rightarrow \text{mortal}(X))$$

“Sócrates é um homem” é representada por

$$\text{homem}(\text{sócrates})$$

Como X na implicação está quantificado universalmente, podemos substituí-lo por qualquer valor do domínio de X que ainda assim teremos uma sentença verdadeira, conforme a regra de inferência da substituição universal. Substituindo X por sócrates na implicação, inferimos a expressão

$$\text{homem}(\text{sócrates}) \rightarrow \text{mortal}(\text{sócrates})$$

Podemos agora aplicar o *modus ponens* e inferir a conclusão  $\text{mortal}(\text{sócrates})$ . Essa sentença é adicionada ao conjunto de expressões que resultam logicamente das asserções originais. Um algoritmo chamado de *unificação* pode ser usado por um resolvedor automático de problemas para determinar que sócrates pode substituir X para que o *modus ponens* possa ser aplicado. A unificação é discutida na Seção 2.3.2.

O Capítulo 14 discute uma regra de inferência mais poderosa chamada de *resolução*, que é a base de vários sistemas de raciocínio automatizado.

### 2.3.2 Unificação

Para aplicar regras de inferência como o *modus ponens*, um sistema de inferência deve ser capaz de determinar quando duas expressões são iguais ou se *correspondem* (ou casam entre si). No cálculo proposicional, isso é trivial: duas expressões se correspondem se, e somente se, elas forem sintaticamente idênticas. No cálculo de predicados, o processo de casamento de duas sentenças é complicado pela existência de variáveis nas expressões. A instanciação universal permite que variáveis quantificadas universalmente sejam substituídas por termos do domínio. Isso requer um processo de decisão para determinar as substituições de variáveis pelas quais duas ou mais expressões possam se tornar idênticas (normalmente com a finalidade de se aplicar regras de inferência).

A *unificação* é um algoritmo para determinar as substituições necessárias para fazer com que duas expressões do cálculo de predicados se correspondam. Vimos isso ser feito na subseção anterior, onde sócrates em homem(sócrates) substituiu X em  $\forall X$  (homem(X)  $\Rightarrow$  mortal(X)). Isso permitiu a aplicação do *modus ponens* e a conclusão mortal(sócrates). Outro exemplo de unificação foi visto anteriormente quando foram discutidas as variáveis fictícias. Como p(X) e p(Y) são equivalentes, Y pode substituir X para fazer com que as sentenças se correspondam.

A unificação e as regras de inferência, como o *modus ponens*, nos permitem inferir um conjunto de asserções lógicas. Para tanto, a base de dados lógica deve ser expressa em uma forma apropriada.

Um aspecto essencial dessa forma é a necessidade de que todas as variáveis sejam quantificadas universalmente. Isso permite liberdade total nas substituições computacionais. Variáveis quantificadas existencialmente podem ser eliminadas de sentenças da base de dados pela sua substituição por constantes que tornem a sentença verdadeira. Por exemplo,  $\exists X$  genitor(X,tom) poderia ser substituída pela expressão genitor(bob,tom) ou genitor(mary,tom), supondo que bob e mary sejam os pais de tom, segundo a interpretação considerada.

O processo de eliminar variáveis quantificadas existencialmente é complicado porque os valores dessas substituições podem depender do valor de outras variáveis da expressão. Por exemplo, na expressão  $\forall X \exists Y$  mae(X,Y), o valor da variável quantificada existencialmente Y depende do valor de X. A *skolemização* substitui cada variável quantificada existencialmente por uma função que retorna a constante apropriada como uma função de uma outra variável ou de todas as outras variáveis da sentença. No exemplo anterior, como o valor de Y depende de X, Y poderia ser substituído por uma *função de skolem*, f, de X. Isso resultaria no predicado  $\forall X$  mae(X,f(X)). A skolemização, um processo que pode também ligar variáveis quantificadas universalmente a constantes, é discutido em mais detalhes na Seção 14.2.

Uma vez que as variáveis quantificadas existencialmente tenham sido removidas de uma base de dados lógica, a unificação pode ser utilizada para corresponder sentenças a fim de aplicar regras de inferência como o *modus ponens*.

A unificação é complicada, pois uma variável pode ser substituída por um termo qualquer, incluindo outras variáveis e expressões funcionais de complexidade arbitrária. Mesmo essas expressões podem conter variáveis. Por exemplo, pai(jack) pode substituir X em homem(X) para inferir que o pai de jack é mortal.

Algumas instâncias da expressão

$\text{foo}(X,a,\text{goo}(Y))$ .

geradas por substituições válidas são:

1.  $\text{foo}(\text{fred},a,\text{goo}(Z))$
2.  $\text{foo}(W,a,\text{goo}(jack))$
3.  $\text{foo}(Z,a,\text{goo}(\text{moo}(Z)))$

Neste exemplo, os casos de substituição, ou *unificações*, que tornariam a expressão inicial idêntica a cada uma das outras três seriam escritos como:

1. {fred/X, Z/Y}
2. {W/X, jack/Y}
3. {Z/X, moo(Z)/Y}

A notação  $X/Y, \dots$  indica que  $X$  substitui a variável  $Y$  na expressão original. As substituições são também denominadas *ligações*. Dizemos que uma variável está *ligada* ao valor que a substituiu.

Ao definir o algoritmo de unificação que determina as substituições necessárias para corresponder duas expressões, várias questões devem ser consideradas. Primeiro, embora uma constante possa substituir sistematicamente uma variável, qualquer constante é considerada uma “ocorrência fundamental” e não pode ser substituída. Duas ocorrências fundamentais diferentes também não podem substituir uma mesma variável. Segundo, uma variável não pode ser unificada com um termo que contém essa variável.  $X$  não pode ser substituída por  $p(X)$ , pois isso criaria uma expressão infinita:  $p(p(p(p(\dots X)\dots)))$ . O teste para essa situação é denominado *verificação de ocorrência*.

Além disso, um processo para resolução de problemas geralmente precisa de inferências múltiplas e, consequentemente, múltiplas unificações sucessivas. Os sistemas para resolução de problemas lógicos devem manter a consistência das substituições de variáveis. É importante que qualquer substituição unificadora seja realizada de modo consistente sobre todas as ocorrências da variável em ambas as expressões que são correspondidas. Isso pode ser constatado quando, anteriormente, Sócrates substituiu não apenas a variável  $X$  em  $\text{homem}(X)$ , mas também a variável  $X$  em  $\text{mortal}(X)$ .

Uma vez que uma variável tenha sido ligada, unificações e inferências futuras devem levar em consideração o valor dessa ligação. Se uma variável for ligada a uma constante, essa variável não pode receber uma nova ligação em uma unificação futura. Se uma variável  $X_1$  for substituída por outra variável  $X_2$  e, mais tarde,  $X_1$  for substituída por uma constante, então  $X_2$  deve refletir também essa ligação. O conjunto de substituições utilizadas em uma sequência de inferências é importante porque ele pode conter a resposta à consulta original (Seção 14.2.5). Por exemplo, se  $p(a, X)$  é unificada com a premissa  $p(Y, Z) \Rightarrow q(Y, Z)$  pela substituição  $\{a/Y, X/Z\}$ , o *modus ponens* nos permite inferir  $q(a, X)$  pela mesma substituição. Se casarmos esse resultado com a premissa de  $q(W, b) \Rightarrow r(W, b)$ , inferimos  $r(a, b)$  pela substituição  $\{a/W, b/X\}$ .

Outro conceito importante é a *composição* de substituições unificadoras. Se  $S$  e  $S'$  são dois conjuntos de substituições, então a composição de  $S$  e  $S'$  (escreve-se  $SS'$ ) é obtida aplicando-se  $S'$  aos elementos de  $S$  e adicionando o resultado a  $S$ . Considere o exemplo da composição das seguintes sequências de substituições:

$$\{X/Y, W/Z\}, \{V/X\}, \{a/V, f(b)/W\}.$$

Compondo o terceiro conjunto,  $\{a/V, f(b)/W\}$ , com o segundo,  $\{V/X\}$ , o resultado é:

$$\{a/X, a/V, f(b)/W\}.$$

Compondo esse resultado com o primeiro conjunto,  $\{X/Y, W/Z\}$ , o resultado é esse conjunto de substituições:

$$\{a/Y, a/X, a/V, f(b)/Z, f(b)/W\}.$$

A composição é o método pelo qual as substituições unificadoras são combinadas e retornadas por meio da função recursiva unificar, apresentada a seguir. Pode-se mostrar que a composição é associativa, mas não comutativa. Os exercícios abordam essas questões mais detalhadamente.

Uma exigência final do algoritmo de unificação é que o unificador seja tão geral quanto possível: deve-se procurar o *unificador mais geral* para as duas expressões. Isso é importante, como será visto no próximo exemplo, porque, se for perdida a generalidade no processo de solução, então o escopo da eventual solução será reduzido ou até mesmo a possibilidade de uma solução poderá ser completamente eliminada.

Por exemplo, na unificação de  $p(X)$  e  $p(Y)$ , qualquer expressão constante, tal como  $\{\text{fred}/X, \text{fred}/Y\}$ , pode ser uma solução. Entretanto, fred não é o unificador mais geral; qualquer variável também funcionaria e produziria uma expressão mais geral:  $\{Z/X, Z/Y\}$ . As soluções obtidas pelo primeiro exemplo de substituição seriam sempre restritas pelo fato de a constante fred limitar as inferências resultantes; isto é, fred seria um unificador, mas ele reduziria a generalidade do resultado.

## Definição

### UNIFICADOR MAIS GERAL (umg)

Se  $s$  é um unificador qualquer das expressões  $E$ , e  $g$  é o unificador mais geral desse conjunto de expressões, então para  $s$  aplicado a  $E$  existe outro unificador  $s'$  tal que  $Es = Eg s'$ , onde  $Es$  e  $Eg s'$  são a composição dos unificadores aplicados à expressão  $E$ .

O unificador mais geral para um conjunto de expressões é único, exceto por variações alfabéticas; isto é, se uma variável se chamar eventualmente  $X$  ou  $Y$ , isso não fará nenhuma diferença em relação à generalidade das unificações resultantes.

A unificação é importante em qualquer sistema de inteligência artificial para resolução de problemas que utiliza o cálculo de predicados para fins de representação. A unificação especifica as condições sob as quais duas (ou mais) expressões do cálculo de predicados podem ser consideradas equivalentes. Isso permite o uso de regras de inferência, tal como a *resolução*, com representações lógicas, um processo que frequentemente requer que se retroceda para encontrar todas as interpretações possíveis.

A seguir, apresentamos o pseudocódigo de uma função, *unificar*, que determina as substituições de unificação (quando estas são possíveis) entre duas expressões do cálculo de predicados. *Unificar* toma como argumentos duas expressões do cálculo de predicados e retorna as substituições de unificação mais gerais ou a constante FALHA, se não for possível a unificação. Ela é definida como uma função recursiva: primeiro, ela tenta recursivamente unificar as componentes iniciais das expressões. Se tiver sucesso, quaisquer substituições que forem retornadas por essa unificação são aplicadas ao resto de ambas as expressões. Essas expressões são passadas, então, para uma segunda chamada recursiva à função *unificar*, que tenta completar a unificação. A recursão é interrompida quando o argumento é um símbolo (um predicado, um nome de função, uma constante ou uma variável) ou, então, todos os elementos da expressão forem correspondidos.

Para simplificar a manipulação de expressões, o algoritmo utiliza uma sintaxe ligeiramente modificada. Como a função *unificar* simplesmente realiza o casamento sintático de padrões, ela pode efetivamente ignorar a distinção do cálculo de predicados entre predicados, funções e argumentos. Representando-se uma expressão como uma *lista* (uma sequência ordenada de elementos), com o predicado ou o nome da função como o primeiro elemento seguido por seus argumentos, simplificamos a manipulação de expressões. Expressões nas quais um argumento é ele mesmo um predicado ou uma expressão funcional são representadas como listas dentro de listas, preservando assim a estrutura da expressão. As listas são delimitadas por parênteses, ( ), e os elementos da lista são separados por espaços. A seguir, apresentamos exemplos de expressões nas sintaxes do cálculo de predicados (CP) e de lista:

#### SINTAXE DO CP

```
p(a,b)
p(f(a),g(X,Y))
igual(eva,mãe(caim))
```

#### SINTAXE DE LISTA

```
(p a b)
(p (f a) (g X Y))
(igual eva (mãe caim))
```

A seguir, apresentamos a função *unificar*:

```
função unificar(E1, E2);
```

```
    início
```

```
        caso
```

```
            E1 e E2 são constantes ou lista vazia:
```

```
%recursão termina
```

```
                se E1 = E2 então retorna {}
```

```
                senão retorna FALHA;
```

```
            E1 é uma variável:
```

```
                se E1 ocorre em E2 então retorna FALHA
```

```

senão retorna {E2/E1};
E2 é uma variável:
  se E2 ocorre em E1 então retorna FALHA
  senão retorna {E1/E2}
E1 ou E2 são vazias então retorna FALHA           %listas com tamanhos diferentes
senão:                                         %E1 e E2 são listas
  início
    HE1 := primeiro elemento de E1;
    HE2 := primeiro elemento de E2;
    SUBS1 := unificar(HE1,HE2);
    se SUBS1 = FALHA então retorna FALHA;
    TE1 := aplicar(SUBS1, resto de E1);
    TE2 := aplicar(SUBS1, resto de E2);
    SUBS2 := unificar(TE1, TE2);
    se SUBS2 = FALHA então retorna FALHA;
    senão retorna composição(SUBS1,SUBS2)
  fim
fim                                              %fim do caso
fim

```

### 2.3.3 Exemplo de unificação

O comportamento do algoritmo apresentado anteriormente pode ser esclarecido pelo rastreamento da chamada `unificar((genitores X (pai X) (mãe bill)), (genitores bill (pai bill) Y))`.

Quando `unificar` é chamada pela primeira vez, como nenhum argumento é um símbolo atômico, a função tenta unificar recursivamente os primeiros elementos de cada expressão, chamando

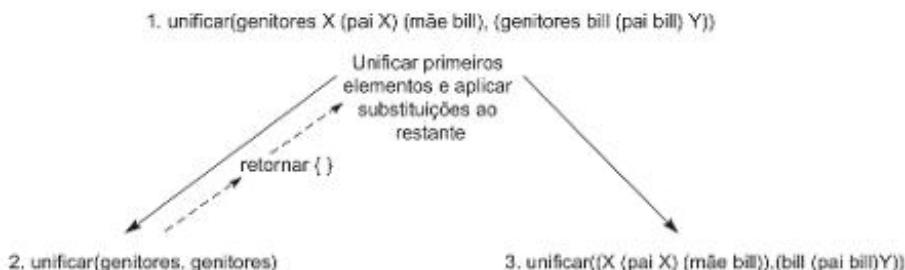
`unificar(genitores, genitores)`.

Essa unificação tem sucesso, retornando a substituição vazia, `{ }`. A aplicação desta ao resto das expressões não cria mudança; o algoritmo, então, chama

`unificar((X (pai X) (mãe bill)), (bill (pai bill) Y))`.

Uma representação de árvore da execução nesse estágio aparece na Figura 2.4.

**Figura 2.4** Primeiras etapas na unificação de `(genitores X (pai X) (mãe bill))` e `(genitores bill (pai bill) Y)`.



Na segunda chamada de unificar, nenhuma expressão é atômica, de modo que o algoritmo separa cada expressão em seu primeiro componente e o restante da expressão. Isso leva à chamada `unificar(X, bill)`.

Essa chamada tem sucesso, pois as duas expressões são atômicas e uma delas é uma variável. A chamada retorna a substituição  $\{bill/X\}$ . Essa substituição é aplicada ao resto de cada expressão e unificar é chamada sobre os resultados, como na Figura 2.5:

`unificar(((pai bill) (mãe bill)), ((pai bill)Y)).`

O resultado dessa chamada é unificar `(pai bill)` com `(pai bill)`. Isso ocasiona as chamadas

`unificar(pai, pai)`

`unificar(bill, bill)`

`unificar((), ())`

Todas essas têm sucesso, retornando o conjunto vazio de substituições conforme visto na Figura 2.6. Unificar é então chamada no restante das expressões:

`unificar(((mãe bill)), (Y)).`

Isso, por sua vez, leva às chamadas

`unificar((mãe bill), Y)`

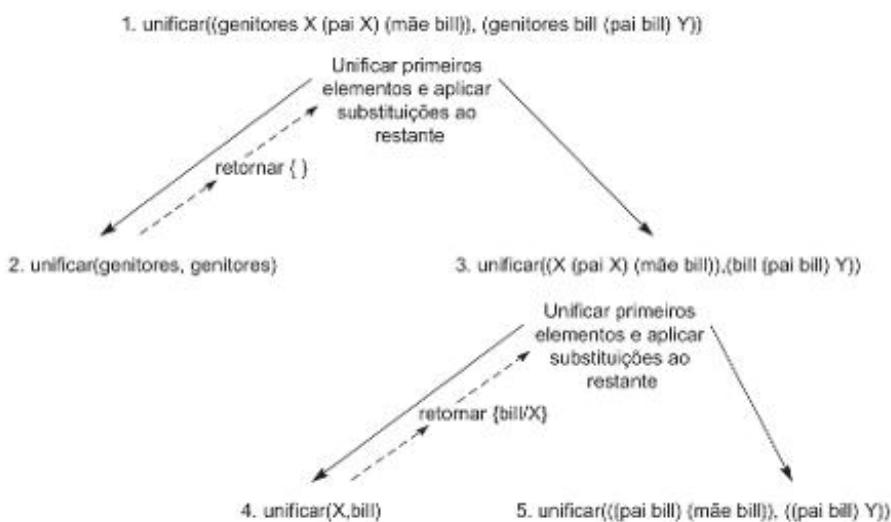
`unificar((), ())`.

Na primeira delas, `(mãe bill)` unifica com `Y`. Observe que a unificação substitui a *estrutura inteira* `(mãe bill)` para a variável `Y`. Assim, a unificação tem sucesso e retorna a substituição  $\{(mãe bill)/Y\}$ . A chamada

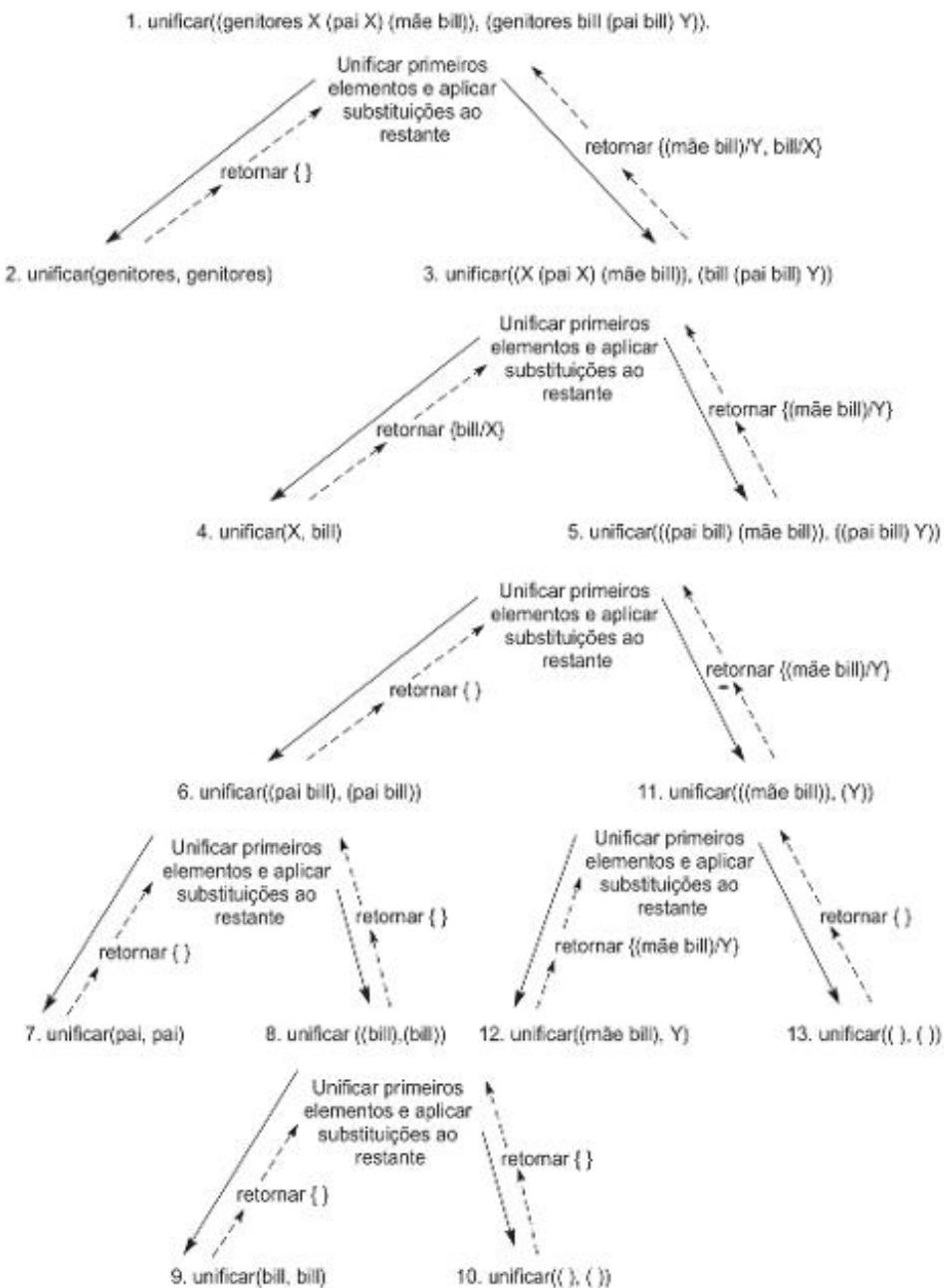
`unificar((), ())`

retorna  $\{\}$ . Todas as substituições são compostas à medida que cada chamada recursiva termina para retornar a resposta  $\{bill/X \ (mãe bill)/Y\}$ . Um rastreamento da execução inteira aparece na Figura 2.6. Cada chamada é numerada para indicar a ordem em que foi feita; as substituições retornadas por cada chamada são anotadas nos arcos da árvore.

**Figura 2.5** Outras etapas na unificação de `(genitores X (pai X) (mãe bill))` e `(genitores bill (pai bill) Y)`.



**Figura 2.6** Rastreamento completo da unificação de (genitores X (pai X) (mãe bill)) e (genitores bill (pai bill) Y).



## 2.4 Aplicação: um consultor financeiro baseado em lógica

Como um exemplo final do uso do cálculo de predicados para representar e raciocinar sobre domínios de problemas, projetaremos um consultor financeiro usando cálculo de predicados. Embora esse seja um exemplo simples, ele ilustra muitas das questões envolvidas em aplicações reais.

A função do consultor é ajudar um usuário a decidir se ele deve investir em uma conta poupança ou no mercado de ações. Alguns investidores podem desejar dividir o seu dinheiro entre as duas opções. O investimento que será recomendado para um determinado investidor depende de sua renda e da sua quantia atual em poupança, de acordo com os seguintes critérios:

1. Indivíduos com uma conta poupança inadequada devem sempre, como prioridade mais alta, aumentar a quantia poupada, independentemente da sua renda.
2. Indivíduos com uma conta poupança adequada e uma renda adequada deveriam considerar um investimento mais arriscado, mas potencialmente mais lucrativo, no mercado de ações.
3. Indivíduos com uma renda pequena que já tenham uma conta poupança adequada podem desejar dividir a sua renda excedente entre poupança e ações para aumentar as reservas em poupança, e tentar, ao mesmo tempo, aumentar a sua renda por meio de ações.

A adequação entre poupança e renda é determinada pelo número de dependentes que um indivíduo mantém. Pela nossa regra, ele deve ter no mínimo \$5.000,00 no banco para cada dependente. Uma renda adequada deve ser estável e fornecer ao menos \$15.000,00 ao ano, mais um adicional de \$4.000,00 por cada dependente.

Para automatizar esse aconselhamento, transcrevemos essas normas em sentenças do cálculo de predicados. A primeira tarefa é determinar as características principais que devem ser consideradas. Aqui, elas são a adequação entre poupança e renda. Elas são representadas pelos predicados `conta_poupança` e `renda`, respectivamente. Ambas as características são predicados unários e seu argumento pode ser adequada ou inadequada. Com isso, seus valores possíveis são:

```
conta_poupança(adequada).
conta_poupança(inadequada).
renda(adequada).
renda(inadequada).
```

As conclusões são representadas pelo predicado unário `investimento`, com os valores possíveis de seu argumento sendo `ações`, `poupança` ou `combinação` (implicando que o investimento deve ser dividido).

Usando esses predicados, as diferentes estratégias de investimento são representadas por implicações. A primeira regra, que indivíduos com poupança inadequada deveriam aumentar a sua poupança como prioridade principal, é representada por

```
conta_poupança(inadequada) → investimento(poupança).
```

De modo semelhante, as outras duas opções restantes de investimentos possíveis são representadas por

```
conta_poupança(adequada) ∧ renda(adequada) → investimento(ações).
```

```
conta_poupança(adequada) ∧ renda(inadequada) → investimento(combinação).
```

A seguir, o consultor deve determinar quando a poupança e a renda são adequadas ou inadequadas. Isso também será feito usando uma implicação. A necessidade de cálculos aritméticos para isso, entretanto, requer o uso de funções. Para determinar a poupança adequada mínima, a função `poupança_min` é definida. Ela toma um argumento, o número de dependentes e retorna 5000 vezes esse argumento.

Usando `poupança_min`, a adequação da poupança é determinada pelas regras

```
∀ X quantia_poupada(X) ∧ ∃ Y (dependentes(Y) ∧ maior(X, poupança_min(Y)))
```

```
→ conta_poupança(adequada).
```

```
∀ X quantia_poupada(X) ∧ ∃ Y (dependentes(Y) ∧ ¬ maior(X, poupança_min(Y)))
```

```
→ conta_poupança(inadequada).
```

onde  $\text{poupança\_min}(X) = 5000 * X$ .

Nessas definições, `quantia_poupada(X)` e `dependentes(Y)` declaram a quantia poupada e o número de dependentes de um investidor; `maior(X,Y)` é o teste aritmético padrão para saber se um número é maior que outro e não será formalmente definido nesse exemplo.

Da mesma forma, a função `renda_min` é definida como

$$\text{renda\_min}(X) \equiv 15000 + (4000 * X).$$

`renda_min` é usada para calcular a renda adequada mínima para um dado número de dependentes. A renda atual de um investidor é representada por um predicado, `ganhos`. Como uma renda adequada deve ser tanto estável como acima do mínimo, `ganhos` toma dois argumentos: o primeiro é a quantia ganha e o segundo deve ser igual a estável ou instável. As demais regras necessárias para o consultor são:

$$\begin{aligned} \forall X \text{ ganhos}(X, \text{estável}) \wedge \exists Y (\text{dependentes}(Y) \wedge \text{maior}(X, \text{renda\_min}(Y))) \\ \rightarrow \text{renda}(\text{adequada}). \end{aligned}$$

$$\begin{aligned} \forall X \text{ ganhos}(X, \text{estável}) \wedge \exists Y (\text{dependentes}(Y) \wedge \neg \text{maior}(X, \text{renda\_min}(Y))) \\ \rightarrow \text{renda}(\text{inadequada}). \end{aligned}$$

$$\forall X \text{ ganhos}(X, \text{instável}) \rightarrow \text{renda}(\text{inadequada}).$$

Para que uma consulta seja realizada, deve-se adicionar a esse conjunto de sentenças do cálculo de predicados uma descrição de um investidor em particular, usando para isso os predicados `quantia_poupada`, `ganhos` e `dependentes`. Assim, um indivíduo com três dependentes, com \$22.000,00 em poupança e uma renda estável de \$25.000,00 seria descrito por

`quantia_poupada(22000).`  
`ganhos(25000, estável).`  
`dependentes(3).`

Isso produz um sistema lógico constituído das seguintes sentenças:

1. `conta_poupança(inadequada) → investimento(poupança).`
2. `conta_poupança(adequada) ∧ renda(adequada) → investimento(ações).`
3. `conta_poupança(adequada) ∧ renda(inadequada) → investimento(combinação).`
4.  $\forall X \text{ quantia_poupada}(X) \wedge \exists Y (\text{dependentes}(Y) \wedge \text{maior}(X, \text{poupança\_min}(Y))) \rightarrow \text{conta\_poupança}(adequada).$
5.  $\forall X \text{ quantia_poupada}(X) \wedge \exists Y (\text{dependentes}(Y) \wedge \neg \text{maior}(X, \text{poupança\_min}(Y))) \rightarrow \text{conta\_poupança}(inadequada).$
6.  $\forall X \text{ ganhos}(X, \text{estável}) \wedge \exists Y (\text{dependentes}(Y) \wedge \text{maior}(X, \text{renda\_min}(Y))) \rightarrow \text{renda}(\text{adequada}).$
7.  $\forall X \text{ ganhos}(X, \text{estável}) \wedge \exists Y (\text{dependentes}(Y) \wedge \neg \text{maior}(X, \text{renda\_min}(Y))) \rightarrow \text{renda}(\text{inadequada}).$
8.  $\forall X \text{ ganhos}(X, \text{instável}) \rightarrow \text{renda}(\text{inadequada}).$
9. `quantia_poupada(22000).`
10. `ganhos(25000, estável).`
11. `dependentes(3).`

onde  $\text{poupança\_min}(X) \equiv 5000 * X$  e  $\text{renda\_min}(X) \equiv 15000 + (4000 * X)$ .

Esse conjunto de sentenças lógicas descreve o domínio do problema. As declarações estão numeradas de modo que possam ser referenciadas no rastreamento a seguir.

Usando a unificação e o *modus ponens*, uma estratégia correta de investimento para esse indivíduo pode ser inferida como uma consequência lógica dessas descrições. Um primeiro passo seria unificar a conjunção de 10 e 11 com as duas primeiras componentes da premissa de 7, isto é,

`ganhos(25000, estável) ∧ dependentes(3)`

pode ser unificada com

`ganhos(X, estável) ∧ dependentes(Y)`

pela substituição {25000/X, 3/Y}. Essa substituição produz a nova implicação

$\text{ganhos}(25000, \text{estável}) \wedge \text{dependentes}(3) \wedge \neg \text{maior}(25000, \text{renda\_min}(3))$   
 $\rightarrow \text{renda}(\text{inadequada}).$

A avaliação da função `renda_min` produz a expressão

$\text{ganhos}(25000, \text{estável}) \wedge \text{dependentes}(3) \wedge \neg \text{maior}(25000, 27000)$   
 $\rightarrow \text{renda}(\text{inadequada}).$

Como todos os três componentes da premissa são verdadeiros individualmente, por 10, 3 e pela definição matemática de maior, a sua conjunção é verdadeira e a premissa inteira é verdadeira. O *modus ponens* pode, então, ser aplicado, produzindo a conclusão `renda(inadequada)`. Essa conclusão será adicionada ao conjunto de sentenças como asserção 12.

**12.** `renda(inadequada)`.

De modo semelhante,

`quantia_poupada(22000) \wedge dependentes(3)`

pode ser unificada com os dois primeiros elementos da asserção 4 pela substituição {22000/X, 3/Y}, produzindo a implicação

$\text{quantia\_poupada}(22000) \wedge \text{dependentes}(3) \wedge \text{maior}(22000, \text{poupança\_min}(3))$   
 $\rightarrow \text{conta\_poupança}(\text{adequada}).$

Aqui, a avaliação da função `poupança_min(3)` produz a expressão

$\text{quantia\_poupada}(22000) \wedge \text{dependentes}(3) \wedge \text{maior}(22000, 15000)$   
 $\rightarrow \text{conta\_poupança}(\text{adequada}).$

Novamente, como todos os componentes da premissa dessa implicação são verdadeiros, a premissa inteira é avaliada como verdadeira e o *modus ponens* pode novamente ser aplicado, produzindo a conclusão `conta_poupança(adequada)`, que é adicionada como asserção 13.

**13.** `conta_poupança(adequada)`.

Como indica um exame das asserções 3, 12 e 13, a premissa da asserção 3 é também verdadeira. Quando aplicamos o *modus ponens* pela terceira vez, a conclusão será `investimento(combinação)`. Esse é o investimento sugerido para esse indivíduo.

Esse exemplo ilustra como o cálculo de predicados pode ser usado para raciocinar acerca de um problema real, chegando a conclusões corretas pela aplicação de regras de inferência à descrição inicial do problema. Não discutimos exatamente como um algoritmo pode determinar quais são as inferências corretas que ele deve fazer para solucionar determinado problema ou a forma como isso pode ser implementado em um computador. Esses tópicos serão apresentados nos capítulos 3, 4 e 6.

## 2.5 Epílogo e referências

Neste capítulo, introduzimos o cálculo de predicados como uma linguagem de representação para a resolução de problemas por IA. Os símbolos, os termos, as expressões e a semântica da linguagem foram descritos e definidos. Com base na semântica do cálculo de predicados, definimos regras de inferência que nos permitem derivar sentenças que resultam logicamente de um determinado conjunto de expressões. Definimos um algoritmo de unificação que determina as substituições de variáveis que tornam duas expressões equivalentes, o que é essencial para a aplicação de regras de inferência. Concluímos o capítulo com o exemplo de um consultor financeiro que representa seu conhecimento financeiro por meio do cálculo de predicados e demonstra a inferência lógica como uma técnica para resolução de problemas.

O cálculo de predicados é discutido em detalhes em vários livros de ciência da computação, incluindo: *The Logical Basis for Computer Programming*, de Zohar Manna e Richard Waldinger (1985); *Logic for Computer Science*, de Jean H. Gallier (1986); *Symbolic Logic and Mechanical Theorem Proving*, de Chin-liang Chang e Richard Char-tung Lee (1973); e *An Introduction to Mathematical Logic and Type Theory*, de Peter B. Andrews (1986). Apresentamos mais técnicas modernas de prova no Capítulo 14, “Raciocínio automatizado”.

Entre os livros que descrevem o uso de cálculo de predicados como uma linguagem de representação para a inteligência artificial, temos: *Logical Foundations of Artificial Intelligence*, de Michael Genesereth e Nils Nilsson (1987); *Artificial Intelligence*, de Nils Nilsson (1998); *The Field of Automated Reasoning*, de Larry Wos (1995); *Computer Modelling of Mathematical Reasoning*, de Alan Bundy (1983, 1988); e *Readings in Knowledge Representation*, de Ronald Brachman e Hector Levesque (1985). Veja também em *Automated Reasoning*, de Bob Veroff (1997), algumas aplicações modernas interessantes de inferência automática. O *Journal for Automated Reasoning (JAR)* e a *Conference on Automated Deduction (CADE)* abordam temas atuais.

## 2.6 Exercícios

1. Usando tabelas verdade, prove as identidades da Seção 2.1.2.
2. Um novo operador,  $\oplus$ , ou *ou-exclusivo*, pode ser definido pela seguinte tabela verdade:

P	Q	$P \rightarrow Q$
V	V	F
V	F	V
F	V	V
F	F	F

Crie uma expressão do cálculo proposicional usando apenas  $\wedge$ ,  $\vee$  e  $\neg$  que seja equivalente a  $P \oplus Q$ .

Prove a sua equivalência usando tabelas verdade.

3. O operador lógico “ $\leftrightarrow$ ” é lido como “se e somente se”.  $P \leftrightarrow Q$  é definido como sendo equivalente a  $(P \rightarrow Q) \wedge (Q \rightarrow P)$ . Baseado nessa definição, mostre que  $P \leftrightarrow Q$  é logicamente equivalente a  $(P \vee Q) \rightarrow (P \wedge Q)$ :
  - a. Usando tabelas verdade.
  - b. Por uma série de substituições, usando as identidades (leis) apresentadas na Seção 2.1.2.
4. Prove que a implicação é transitiva no cálculo proposicional, isto é, que  $((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R)$ .
5. a. Prove que o *modus ponens* é consistente para o cálculo proposicional. Dica: use tabelas verdade para enumerar todas as possíveis interpretações.
  - b. A *abdução* é uma regra de inferência que infere  $P$  de  $P \rightarrow Q$  e  $Q$ . Mostre que a abdução não é consistente (ver Capítulo 7).
  - c. Mostre que o *modus tollens*  $((P \rightarrow Q) \wedge \neg Q) \rightarrow \neg P$  é consistente.
6. Tente unificar os seguintes pares de expressões. Mostre os seus unificadores mais gerais ou explique por que eles não podem ser unificados.
  - a.  $p(X,Y)$  e  $p(a,Z)$
  - b.  $p(X,X)$  e  $p(a,b)$
  - c.  $\text{ancestral}(X,Y)$  e  $\text{ancestral}(\text{bill},\text{pai}(\text{bill}))$
  - d.  $\text{ancestral}(X,\text{pai}(X))$  e  $\text{ancestral}(\text{david},\text{george})$
  - e.  $q(X)$  e  $\neg q(a)$
7. a. Componha os conjuntos de substituição  $\{a/X, Y/Z\}$  e  $\{X/W, b/Y\}$ .
  - b. Prove que a composição dos conjuntos de substituição é associativa.
  - c. Construa um exemplo para mostrar que a composição não é comutativa.

8. Implemente o algoritmo unificar da Seção 2.3.2 na linguagem de computador que você preferir.
9. Ofereça duas interpretações alternativas para a descrição do mundo de blocos da Figura 2.3.
10. Jane Doe tem quatro dependentes, uma renda estável de \$30.000,00 e \$15.000,00 em sua conta poupança. Acrescente os predicados que foram apropriados para descrever sua situação ao consultor de investimentos gerais do exemplo na Seção 2.4 e realize as unificações e inferências necessárias para determinar seu investimento sugerido.
11. Escreva um conjunto de predicados lógicos que realizarão diagnóstico simples de automóvel (por exemplo, se o motor não vira e as luzes não acendem, então a bateria está ruim). Não tente elaborar muito, mas inclua os casos de bateria ruim, falta de combustível, velas com defeito e motor de arranque com problemas.
12. A história a seguir é de N. Wirth (1976), *Algorithms + data structures = programs*.

Eu me casei com uma viúva (vamos chamá-la de V) que tem uma filha já adulta (vamos chamá-la de F). Meu pai (P), que nos visitava com frequência, apaixonou-se por minha enteada e se casou com ela. Logo, meu pai se tornou meu genro e minha enteada se tornou minha mãe. Alguns meses depois, minha esposa teve um filho ( $F_1$ ), que se tornou o cunhado do meu pai, além de meu tio. A esposa do meu pai, ou seja, minha enteada, também teve um filho ( $F_2$ ).

Usando o cálculo de predicados, crie um conjunto de expressões que represente a situação da história citada. Acrescente expressões definindo relacionamentos básicos de família, como a definição de sogro, e use o *modus ponens* nesse sistema para provar a conclusão de que “eu sou meu próprio avô”.

# Estruturas e estratégias para busca em espaço de estados

*Para sobreviver, um organismo necessita ou blindar-se (como uma árvore ou um marisco) e “esperar pelo melhor” ou então desenvolver métodos para escapar de possíveis danos, indo para vizinhanças melhores. Se você escolher essa última opção, será confrontado com o problema primordial que cada agente precisa resolver continuamente: O que eu faço agora?*

— DANIEL C. DENNETT, *Consciousness Explained*

*Duas estradas separavam-se num bosque amarelo,  
Que pena não poder seguir por ambas  
Numa só viagem: muito tempo fiquei  
Mirando uma até onde enxergava  
Quando se perdia entre os arbustos;  
Depois tomei a outra...*

— ROBERT FROST, *The Road Not Taken*

## 3.0 Introdução

O Capítulo 2 introduziu o cálculo de predicados como um exemplo de uma linguagem de representação da inteligência artificial. Expressões de cálculo de predicados bem formadas oferecem um meio de descrever objetos e relações em um domínio de problema, e regras de inferência como *modus ponens* nos permitem inferir novo conhecimento a partir dessas descrições. Essas inferências definem um espaço que é buscado para achar a solução de um problema. O Capítulo 3 introduz a teoria da busca em espaço de estados.

Para projetar e implementar com sucesso os algoritmos de busca, um programador precisa ser capaz de analisar e prever seu comportamento. Algumas perguntas que precisam ser respondidas são:

O resolvidor do problema encontrará, garantidamente, uma solução?

O resolvidor do problema sempre terminará? Ele poderá ficar preso em um laço infinito?

Quando uma solução for encontrada, há garantias de que ela será a ideal?

Qual é a complexidade do processo de busca em termos do uso de tempo? E do uso de memória?

Como o interpretador pode reduzir de modo eficiente a complexidade da busca?

Como se pode projetar um interpretador para que ele utilize uma linguagem de representação da forma mais eficiente possível?

A teoria da *busca em espaço de estados* é a nossa principal ferramenta para responder a essas questões. Representando-se um problema como um *grafo de espaço de estados*, podemos usar a *teoria dos grafos* para analisar a estrutura e a complexidade tanto do problema quanto dos procedimentos de busca que empregamos para resolvê-lo.

Um grafo consiste em um conjunto de *nós* e um conjunto de *arcos*, ou *elos*, conectando pares de nós. No modelo de espaço de estados para resolução de problemas, os nós de um grafo são usados para representar *estados* discretos em um processo de resolução de problemas, tais como, por exemplo, os resultados de inferências lógicas ou as diferentes configurações de um tabuleiro. Os arcos do grafo representam transições entre estados. Essas transições correspondem a inferências lógicas ou movimentos válidos de um jogo. Em sistemas especialistas, por exemplo, os estados descrevem o nosso conhecimento sobre um caso do problema em algum estágio de um processo do raciocínio. O conhecimento do especialista, na forma de regras *se... então*, permite-nos gerar informação nova; o ato de aplicar uma regra é representado como um arco entre estados.

A teoria dos grafos é a nossa melhor ferramenta para raciocinar sobre a estrutura de objetos e relações; na verdade, foi justamente essa necessidade que levou à sua criação no início do século XVIII. O matemático suíço Leonhard Euler inventou a teoria dos grafos para resolver o “problema das pontes de Königsberg”. A cidade de Königsberg ocupava ambas as margens e duas ilhas de um rio. As ilhas e as margens eram conectadas por sete pontes, como indica a Figura 3.1.

O problema das pontes de Königsberg questiona se existe um roteiro pela cidade que atravesse cada ponte exatamente uma vez. Embora os moradores não tenham conseguido encontrar tal roteiro e duvidassem que isso fosse possível, ninguém provou a sua impossibilidade. Concebendo uma forma de teoria de grafos, Euler criou uma representação alternativa para o mapa, apresentada na Figura 3.2. As margens ( $m_1$  e  $m_2$ ) e as ilhas ( $i_1$  e  $i_2$ ) são descritas pelos nós de um grafo; as pontes são representadas por arcos rotulados entre nós ( $p_1, p_2, \dots, p_7$ ). A representação pelo grafo preserva a estrutura essencial do sistema de pontes ao mesmo tempo em que ignora características irrelevantes para o problema, como distâncias, comprimentos e ordem das pontes no roteiro.

Como alternativa, podemos representar o sistema de pontes de Königsberg usando cálculo de predicados. O predíccado *conecta* corresponde a um arco do grafo, declarando que duas massas de terra estão conectadas por uma ponte em particular. Cada ponte requer dois predíccados *conecta*, um para cada direção em que a ponte pode ser cruzada. Uma expressão de predíccados adicional,  $\text{conecta}(X, Y, Z) = \text{conecta}(Y, X, Z)$ , indicando que qualquer ponte pode ser cruzada em ambas as direções, permitiria a remoção de metade dos fatos *conecta* declarados a seguir:

<i>conecta(i1, i2, p1)</i>	<i>conecta(i2, i1, p1)</i>
<i>conecta(m1, i1, p2)</i>	<i>conecta(i1, m1, p2)</i>
<i>conecta(m1, i1, p3)</i>	<i>conecta(i1, m1, p3)</i>
<i>conecta(m1, i2, p4)</i>	<i>conecta(i2, m1, p4)</i>
<i>conecta(m2, i1, p5)</i>	<i>conecta(i1, m2, p5)</i>
<i>conecta(m2, i1, p6)</i>	<i>conecta(i1, m2, p6)</i>
<i>conecta(m2, i2, p7)</i>	<i>conecta(i2, m2, p7)</i>

Figura 3.1 A cidade de Königsberg.

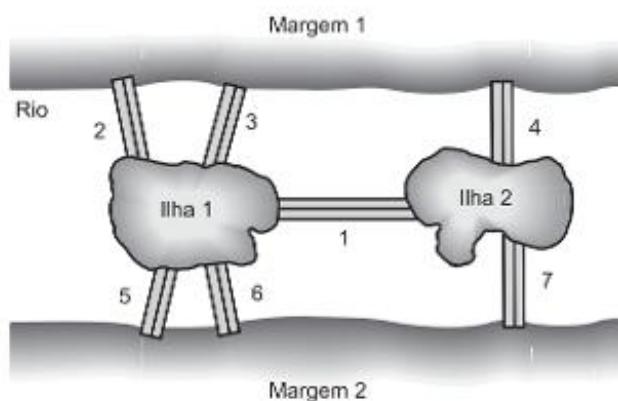
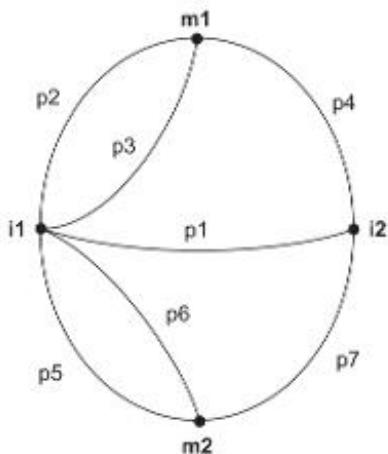


Figura 3.2 Grafo do sistema de pontes de Königsberg.



A representação por cálculo de predicados é equivalente à representação por grafo à medida que a conectividade é preservada. De fato, um algoritmo poderia realizar a tradução entre as duas representações sem perda de informação. Entretanto, a estrutura do problema pode ser visualizada mais diretamente na representação por grafo, enquanto ela se encontra apenas implícita na versão do cálculo de predicados. A prova de Euler ilustra essa distinção.

Para provar que o roteiro desejado é impossível, Euler se concentrou no *grau* dos nós do grafo, observando que um nó poderia ser de *grau par* ou *ímpar*. Um nó de grau *par* tem um número par de arcos ligando-o aos nós vizinhos. Um nó de grau *ímpar* tem um número ímpar de arcos. Com exceção dos seus nós inicial e final, o roteiro desejado deveria deixar cada nó exatamente com a mesma frequência com que ele deveria chegar nele. Os nós de grau ímpar poderiam ser usados apenas como inicio e fim do roteiro, porque esses nós poderiam ser atravessados apenas certo número de vezes antes de se chegar à conclusão de que eles seriam becos sem saída. O viajante não poderia deixar o nó sem usar um arco já previamente atravessado.

Euler observou que, a menos que um grafo contivesse exatamente zero nó ou dois nós de grau ímpar, o roteiro seria impossível. Se houvesse dois nós de grau ímpar, o roteiro poderia começar no primeiro e terminar no segundo; se não houvesse nós de grau ímpar, o roteiro poderia começar e terminar no mesmo nó. O roteiro não é possível para grafos contendo qualquer outro número de nós ímpares, como é o caso da cidade de Königsberg. Esse problema é conhecido agora como “encontrar um *caminho Euleriano* através de um grafo”.

Note que a representação por cálculo de predicados, embora capture os relacionamentos entre pontes e massas de terra da cidade, não sugere o conceito de grau de um nó. Na representação por grafo, há uma única ocorrência para cada nó com arcos entre os nós, enquanto em um conjunto de predicados há múltiplas ocorrências de constantes como argumentos. Por essa razão, a representação por grafo sugere o conceito de grau de um nó e fornece o foco para a prova de Euler. Esse exemplo ilustra o poder da teoria dos grafos para analisar a estrutura de objetos, propriedades e relacionamentos.

Na Seção 3.1, revisamos a teoria básica dos grafos e então apresentamos as máquinas de estados finitos e a descrição de problemas por espaço de estados. Na Seção 3.2, apresentamos a busca pelo grafo como uma metodologia para a solução de problemas. A busca em profundidade e em amplitude são as duas estratégias de busca em espaço de estados. Nós as compararmos e acrescentarmos a distinção entre busca guiada por objetivo e busca guiada por dados. A Seção 3.3 demonstra como a busca em espaço de estados é usada para caracterizar o raciocínio lógico. Ao longo de todo o capítulo, usamos a teoria de grafos para analisar a estrutura e a complexidade de uma série de problemas.

## 3.1 Estruturas para busca em espaço de estados

### 3.1.1 Teoria dos grafos (opcional)

Um *grafo* é um conjunto de *nós* ou *estados* e um conjunto de *arcos* que os conectam. Um *grafo rotulado* tem um ou mais descritores (rótulos) atribuídos a cada nó, que distingue aquele nó de qualquer outro nó do grafo. Em um *grafo de espaço de estados*, esses descritores identificam estados em um processo de solução de problema. Se não há diferenças descriptivas entre dois nós, eles são considerados um mesmo nó. O arco entre dois nós é indicado pelos rótulos dos nós conectados por ele.

Os arcos de um grafo também podem ser rotulados. Os rótulos de arcos são usados para indicar que um arco representa um relacionamento nominado (como em uma rede semântica) ou para atribuir pesos a arcos (como no problema do caixeiro-viajante). Se houver arcos diferentes entre os mesmos dois nós (como na Figura 3.2), eles podem também ser distinguidos por rotulação.

Um *grafo* é *direcionado* se os arcos tiverem uma direcionalidade associada. Os arcos em um grafo direcionado são normalmente desenhados como setas ou têm uma seta associada para indicar a sua direção. Os arcos que podem ser atravessados em qualquer direção podem ter duas setas associadas, porém, mais frequentemente, eles não recebem qualquer indicador de direção. A Figura 3.3 é um grafo direcionado rotulado: o arco (a, b) pode ser atravessado apenas do nó a para o nó b, mas o arco (b, c) pode ser atravessado em qualquer direção.

Um *caminho* ao longo de um grafo conecta uma sequência de nós através de arcos sucessivos. Um caminho é representado por uma lista ordenada que registra os nós na ordem em que eles ocorrem. Na Figura 3.3, [a, b, c, d] representa o caminho através dos nós a, b, c e d, nessa ordem.

Um *grafo radicado* tem um nó especial, chamado de *raiz*, de modo que existe um caminho da raiz para todos os nós do grafo. Ao desenhar um grafo radicado, a raiz é desenhada normalmente no topo da página, acima de todos os demais nós. Os grafos de espaço de estados para jogos são normalmente grafos radicados com o inicio do jogo sendo a raiz. Os movimentos iniciais do grafo do jogo da velha estão representados pelo grafo radicado da Figura II.5. Trata-se de um grafo direcionado em que todos os arcos têm uma única direção. Note que esse grafo não contém ciclos; os jogadores não podem (apesar de algumas vezes o desejarem!) desfazer um movimento.

Uma *árvore* é um grafo no qual dois nós têm, no máximo, um caminho entre eles. As árvores frequentemente têm raízes e, nesse caso, elas são normalmente desenhadas com a raiz no topo, como um grafo radicado. Como cada nó de uma árvore tem apenas um caminho de acesso, a partir de qualquer outro nó é impossível para um caminho circular através de uma sequência de nós originando um *laço* ou *ciclo*.

Para árvores ou grafos radicados, incluem-se como relações entre nós as denominações *genitor*, *filho* e *irmãos*. Essas relações são usadas no mesmo sentido das relações de parentesco com os genitores precedendo os seus filhos ao longo de um arco direcionado. Os filhos de um nó são chamados de *irmãos*. De modo semelhante, um *ancestral* vem antes de um *descendente* em algum caminho de um grafo direcionado. Na Figura 3.4, b é um

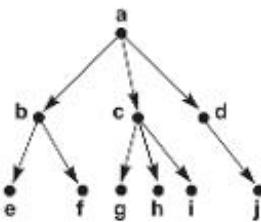
**Figura 3.3** Um grafo direcionado rotulado.



Nós = {a,b,c,d,e}

Arcos = {(a,b),(a,d),(b,c),(c,b),(c,d),(d,a),(d,e),(e,c),(e,d)}

**Figura 3.4** Uma árvore radicada exemplificando relações de parentesco.



genitor dos nós e e f (que são, assim, *filhos* de b e *irmãos* entre si). Os nós a e c são *ancestrais* dos estados g, h e i, e g, h e i são *descendentes* de a e c.

Antes de introduzirmos a representação de espaço de estados de problemas, definimos formalmente esses conceitos.

### Definição

#### GRAFO

Um grafo consiste em:

Um conjunto de nós  $N_1, N_2, N_3, \dots, N_n, \dots$ , que não precisa ser finito.

Um conjunto de arcos que conectam pares de nós.

Arcos são pares ordenados de nós; isto é, o arco  $(N_3, N_4)$  conecta o nó  $N_3$  ao nó  $N_4$ . Isso indicaria uma conexão direcionada do nó  $N_3$  para  $N_4$  mas não de  $N_4$  para  $N_3$ , a menos que  $(N_4, N_3)$  fosse também um arco; nesse caso, o arco ligando  $N_3$  e  $N_4$  é *não direcionado*.

Se um arco *direcionado* conecta  $N_j$  e  $N_k$ , então  $N_j$  é chamado de o *genitor* de  $N_k$ , e  $N_k$ , de o *filho* de  $N_j$ . Se o grafo contém também um arco  $(N_j, N_l)$ , então  $N_k$  e  $N_l$  são *irmãos*.

Um grafo *radicado* tem um único nó  $N_s$  do qual se originam todos os caminhos do grafo. Isto é, a raiz não tem um genitor no grafo.

Um nó de *extremidade* ou *folha* é um nó que não tem filhos.

Uma sequência ordenada de nós  $[N_1, N_2, N_3, \dots, N_n]$ , onde cada par  $N_i, N_{i+1}$  da sequência representa um arco, isto é,  $(N_i, N_{i+1})$ , é chamada de um *caminho* de comprimento  $n - 1$  no grafo.

Em um caminho de um grafo radicado, se diz que um nó é um *ancestral* de todos os nós posicionados depois dele (à sua direita), bem como que ele é um *descendente* de todos os nós que o precedem.

Dizemos que um caminho que contém qualquer nó mais do que uma vez (em que um nó  $N_j$  é repetido na definição anterior de caminho) contém um *ciclo* ou *laço*.

Uma árvore é um grafo no qual existe um único caminho entre qualquer par de nós. (Os caminhos em uma árvore, portanto, não contêm ciclos.)

As arestas em uma árvore radicada são direcionadas para longe da raiz. Cada nó em uma árvore radicada tem um único genitor.

Dizemos que dois nós estão *conectados* se houver um caminho que inclua ambos.

A seguir, introduzimos a máquina de estados finitos, uma representação abstrata para os dispositivos de computação que pode ser vista como um autômato para atravessar os caminhos em um grafo.

### 3.1.2 Máquina de estados finitos (opcional)

Podemos pensar em uma máquina como um sistema que aceita valores de entrada e que possivelmente produz valores de saída e tem algum tipo de mecanismo interno (estados) para registrar informações sobre valores de entrada anteriores. Uma *máquina de estados finitos* (MEF) é um grafo finito, direcionado, conectado, tendo um conjunto de estados, um conjunto de valores de entrada e uma função de transição de estado que descreve o efeito que os elementos do fluxo de entrada têm sobre os estados do grafo. O fluxo de valores de entrada produz um caminho dentro do grafo dos estados dessa máquina finita. Assim, a MEF pode ser vista como um modelo abstrato de computação.

O uso principal de tal máquina é reconhecer componentes de uma linguagem formal. Esses componentes normalmente são sequências de caracteres (“palavras” feitas de caracteres de um “alfabeto”). Na Seção 5.3, estendemos essa definição para uma máquina probabilística de estados finitos. Essas máquinas de estado têm um papel importante na análise de expressões em linguagens, sejam por computação ou humana, como vemos nas seções 5.3, 9.3 e no Capítulo 15.

#### Definição

#### MÁQUINA DE ESTADOS FINITOS (MEF)

Uma *máquina de estados finitos* é uma tripla ordenada  $(S, I, F)$ , onde:

$S$  é um conjunto finito de estados em um grafo conectado  $s_1, s_2, s_3, \dots, s_n$ .

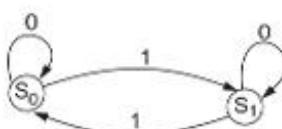
$I$  é um conjunto finito de valores de *entrada* (input)  $i_1, i_2, i_3, \dots, i_m$ .

$F$  é uma função de transição de estado que, para qualquer  $i \in I$ , descreve seu efeito sobre os estados  $S$  da máquina, assim,  $\forall i \in I, F_i : (S \rightarrow S)$ . Se a máquina estiver no estado  $s_j$  e ocorre a entrada  $i$ , o próximo estado da máquina será  $F_i(s_j)$ .

Para ver um exemplo simples de uma máquina de estados finitos, considere  $S = \{s_0, s_1\}$ ,  $I = \{0, 1\}$ ,  $f_0(s_0) = s_0$ ,  $f_0(s_1) = s_1$ ,  $f_1(s_0) = s_1$  e  $f_1(s_1) = s_0$ . Com esse dispositivo, às vezes chamado de *flip-flop*, um valor de entrada igual a zero deixa o estado inalterado, enquanto a entrada 1 muda o estado da máquina. Podemos visualizar essa máquina a partir de dois pontos de vista equivalentes, como um grafo finito com arcos rotulados, direcionados, como na Figura 3.5(a), ou como uma matriz de transição, como na Figura 3.5(b). Na matriz de transição, os valores de entrada são listados ao longo da linha de cima, os estados estão na coluna mais à esquerda, e a saída para uma entrada aplicada a um estado está no ponto de interseção.

Um segundo exemplo de uma máquina de estados finitos é representado pelo grafo direcionado da Figura 3.6(a) e a matriz de transição equivalente da Figura 3.6(b). Pode-se perguntar o que a máquina de estados finitos da Figura 3.6 poderia representar. Com duas suposições, essa máquina poderia ser vista como um reconhecedor de todas as sequências de caracteres do alfabeto  $\{a, b, c, d\}$  que contêm a sequência exata “abc”. As duas suposi-

Figura 3.5 (a) Grafo de estados finitos para um *flip-flop* e (b) sua matriz de transição.

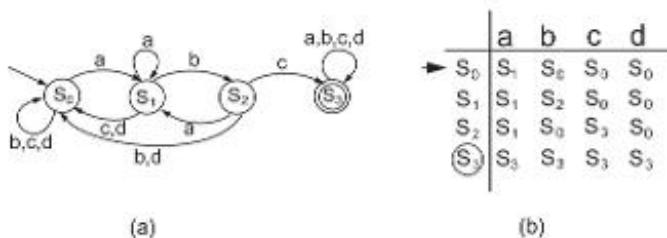


(a)

	0	1
$s_0$	$s_0$	$s_1$
$s_1$	$s_1$	$s_0$

(b)

**Figura 3.6** (a) O grafo de estados finitos e (b) a matriz de transição para o exemplo de reconhecimento de sequência.



ções são, primeiro, que o estado  $s_0$  tem um papel especial como *estado inicial*, e segundo, que  $s_3$  é o *estado de aceitação*. Assim, o fluxo de entrada apresentará seu primeiro elemento ao estado  $s_0$ . Se o fluxo mais tarde terminar com a máquina no estado  $s_3$ , ele terá reconhecido que há a sequência “abc” dentro desse fluxo de entrada.

O que acabamos de descrever é uma *máquina reconhecedora de estados finitos*, às vezes chamada de *máquina de Moore*. Usamos a convenção de colocar uma seta que parte de nenhum estado e termina no estado inicial da máquina de Moore, e de representar o estado (ou estados) de aceitação como especiais, normalmente usando um duplo círculo, como na Figura 3.6. Agora, apresentamos uma definição formal da máquina de Moore:

### Definição

#### RECONHECEDOR DE ESTADOS FINITOS (MÁQUINA DE MOORE)

Um *reconhecedor de estados finitos* é uma máquina de estados finitos  $(S, I, F)$ , onde:

$\exists s_0 \in S$  tal que o fluxo de entrada comece em  $s_0$ , e

$\exists s_n \in S$ , um estado de aceitação. O fluxo de entrada é aceito se terminar nesse estado. De fato, pode haver um conjunto de estados de aceitação.

O reconhecedor de estados finitos é representado como  $(S, s_0, \{s_n\}, I, F)$ .

Apresentamos dois exemplos muitos simples de um conceito poderoso. Como veremos na compreensão de linguagem natural (Capítulo 15), os reconhecedores de estados finitos são uma ferramenta importante para determinar se padrões de caracteres, palavras ou sentenças têm ou não propriedades desejadas. Veremos que um reconhecedor de estados finitos define implicitamente uma linguagem formal com base nos conjuntos de letras (caracteres) e palavras (sequências) que ela aceita.

Também mostramos apenas máquinas de estados finitos *determinísticas*, onde a função de transição para qualquer valor de entrada a um estado resulta em um estado seguinte único. *Máquinas de estados finitos probabilísticos*, em que a função de transição define uma distribuição de estados de saída para cada entrada em um estado, também são uma técnica de modelagem importante. Elas são abordadas na Seção 5.3 e novamente no Capítulo 15. Em seguida, examinaremos uma representação gráfica mais genérica para a análise da solução de problema: o espaço de estados.

### 3.1.3 Representação de problemas por espaço de estados

Na *representação por espaço de estados* de um problema, os nós de um grafo correspondem a *estados* de soluções parciais do problema, e os arcos correspondem a passos de um processo de solução de um problema. Um ou mais *estados iniciais*, que correspondem à informação fornecida em uma instância do problema, formam a raiz do grafo. O grafo define também uma ou mais condições relativas aos *objetivos*, que são soluções para uma instância

do problema. A *busca em espaço de estados* caracteriza a solução de um problema como o processo de procurar um *caminho de solução* partindo do estado inicial até um objetivo.

Um objetivo pode descrever um estado, como um resultado vencedor no jogo da velha (Figura II.5), ou uma configuração alvo no quebra-cabeça dos 8 (Figura 3.7). Como alternativa, um objetivo pode descrever uma propriedade do próprio caminho de solução. No problema do caixeiro-viajante (figuras 3.9 e 3.10), a busca termina quando o caminho “mais curto” por todos os nós do grafo é encontrado. No problema de análise sintética (*parsing*) (Seção 3.3), o caminho de solução é uma análise bem-sucedida da estrutura de uma sentença.

Arcos do espaço de estados correspondem às etapas em um processo de solução, e caminhos pelo espaço representam soluções em diversos estágios de conclusão. Os caminhos são buscados, começando no estado inicial e continuando pelo grafo, até que a descrição do objetivo seja satisfeita ou eles sejam abandonados. A criação real de novos estados ao longo do caminho é feita pela aplicação de operadores, como “movimentos válidos” em um jogo ou regras de inferência em um problema lógico ou sistema especialista, a estados existentes em um caminho. A tarefa de um algoritmo de busca é encontrar um caminho de solução por esse espaço de problemas. Os algoritmos de busca devem acompanhar os caminhos de um nó inicial até um nó objetivo, pois esses caminhos contêm uma série de operações que levam à solução do problema.

Agora, definimos formalmente a representação de problemas por espaço de estados.

### *Definição*

## BUSCA EM ESPAÇO DE ESTADOS

Um *espaço de estados* é representado por uma quadra  $[N, A, I, DO]$ , onde:

$N$  é o conjunto de nós ou estados do grafo. Eles correspondem aos estados de um processo de solução de problema.

$A$  é o conjunto de arcos (ou elos) entre os nós. Eles correspondem aos passos de um processo de solução de problema.

$I$ , um subconjunto não vazio de  $N$ , contém o(s) estado(s) inicial(ais) do problema.

$DO$ , um subconjunto não vazio de  $N$ , contém o(s) estado(s) objetivo(s) do problema. Os estados em  $DO$  são descritos por meio de:

1. Uma propriedade mensurável dos estados encontrados na busca.
2. Uma propriedade mensurável do caminho desenvolvido na busca; por exemplo, a soma dos custos de transição para os arcos do caminho.

Um *caminho de solução* é um caminho através desse grafo de um nó em  $I$  para um nó em  $DO$ .

Uma das características gerais de um grafo, e um dos problemas que surgem no projeto de um algoritmo de busca em grafo, é que os estados às vezes podem ser alcançados por diferentes caminhos. Por exemplo, na Figura 3.3, um caminho pode ser feito do estado  $a$  para o estado  $d$  ou através de  $b$  e  $c$  ou diretamente de  $a$  até  $d$ . Isso torna importante escolher o *melhor* caminho, de acordo com as necessidades de um problema. Além disso, vários caminhos a um estado podem levar a laços ou ciclos em um caminho de solução, que impedem que o algoritmo alcance um objetivo. Uma busca cega para o estado objetivo e no grafo da Figura 3.3 poderia buscar a sequência de estados  $abcdabcdabcd\dots$  para sempre!

Se o espaço a ser buscado é uma árvore, como na Figura 3.4, o problema de ciclos não ocorre. Portanto, é importante distinguir entre problemas cujo espaço de estados é uma árvore e aqueles que podem conter laços. Os algoritmos gerais de busca em grafo deverão detectar e eliminar laços dos caminhos de solução em potencial, enquanto as buscas em árvore podem ganhar eficiência eliminando esse teste e sua sobrecarga.

O jogo da velha e o quebra-cabeça dos 8 exemplificam os espaços de estados de jogos simples. Ambos os exemplos demonstram condições de término do tipo 1 em nossa definição da busca pelo espaço de estados. O Exemplo 3.1.3, o problema do caixeiro-viajante, tem uma descrição de objetivo do tipo 2, o custo total do próprio caminho.

Embora em jogos físicos desse tipo os movimentos sejam feitos movendo peças (“mova a peça 7 para a direita, desde que o vazio esteja à direita da peça” ou “mova a peça 3 para baixo”), é muito mais simples pensar em termos de “mover o espaço vazio”. Isso simplifica a definição das regras de movimento, pois há oito peças, mas somente um único vazio. Para aplicar um movimento, temos que garantir que ele não moverá o vazio para fora do tabuleiro. Portanto, nem todos os quatro movimentos se aplicam o tempo todo; por exemplo, quando o vazio está em um dos cantos, somente dois movimentos são possíveis.

Os movimentos válidos são:

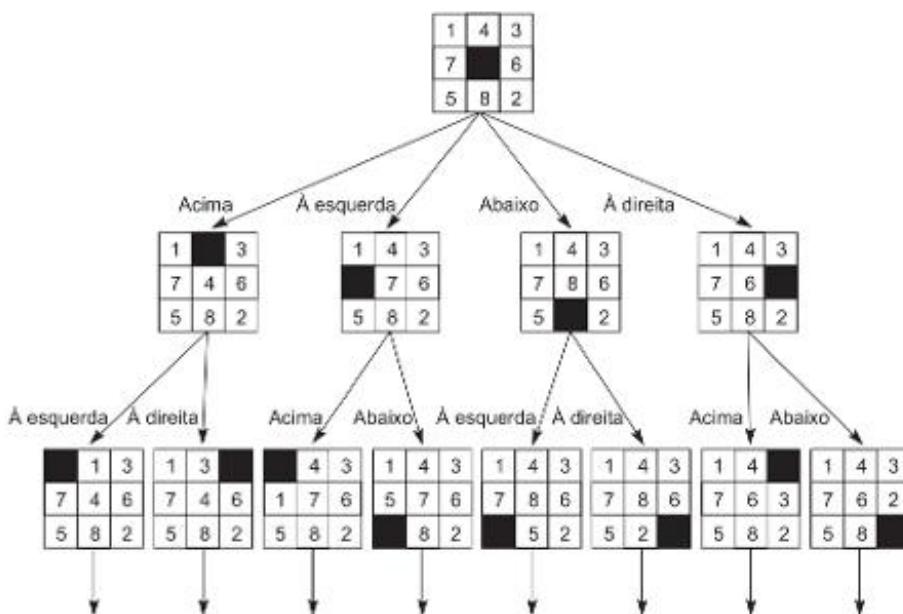
mova o vazio para cima	↑
mova o vazio para a direita	→
mova o vazio para baixo	↓
mova o vazio para a esquerda	←

Se especificarmos um estado inicial e um estado objetivo para o quebra-cabeça dos 8, é possível dar um espaço de estados considerando o processo de solução de problema (Figura 3.8). Os estados poderiam ser representados usando uma matriz  $3 \times 3$  simples. Uma representação de cálculo de predicados poderia usar um predicado de “estado” com nove parâmetros (para os locais dos números na grade). Quatro procedimentos, descrevendo cada um dos movimentos possíveis do vazio, definem os arcos no espaço de estados.

Assim como no jogo da velha, o espaço de estados para o quebra-cabeça dos 8 é um grafo (com a maioria dos estados com vários genitores), mas, diferentemente do jogo da velha, é possível haver ciclos. A DO, ou a descrição do objetivo, do espaço de estados é uma configuração particular do estado ou do tabuleiro. Quando esse estado é encontrado em um caminho, a busca termina. O caminho, do início ao objetivo, é a série de movimentos desejada.

É interessante notar que o espaço de estados completo do quebra-cabeça dos 8 e do quebra-cabeça dos 15 consiste em dois subgrafos desconectados (e, nesse caso, de tamanhos iguais). Isso faz com que metade dos estados possíveis no espaço de busca seja impossível de ser alcançada a partir de qualquer estado inicial. Se trocarmos (soltando) duas peças imediatamente adjacentes, os estados na outra componente do espaço se tornam alcançáveis.

**Figura 3.8** Espaço de estados do quebra-cabeça dos 8 gerado por operações do tipo “mova o vazio”.



### Exemplo 3.1.3 Caixeiro-viajante

Suponha que um caixeiro-viajante tenha que visitar cinco cidades e depois retornar para casa. O objetivo do problema é encontrar o caminho mais curto para o caixeiro-viajante percorrer, visitando cada cidade e, depois, retornando à cidade inicial. A Figura 3.9 apresenta um exemplo desse problema. Os nós do grafo representam cidades, e cada arco é rotulado com um peso indicando o custo para atravessar esse arco. Esse custo poderia ser uma representação dos quilômetros necessários para uma viagem de carro ou os custos de um voo entre as duas cidades. Por conveniência, supomos que o caixeiro-viajante more na cidade A, para onde ele deverá retornar, embora essa suposição simplesmente reduza o problema de  $N$  cidades para um problema de  $(N - 1)$  cidades.

O caminho  $[A,D,C,B,E,A]$ , com o custo associado de 450 quilômetros, é um exemplo de circuito possível. A descrição do objetivo requer um circuito completo com custo mínimo. Note que a descrição do objetivo é uma propriedade do caminho completo, e não de um único estado. Essa é uma descrição de objetivo do tipo 2, segundo a definição de busca em espaço de estados.

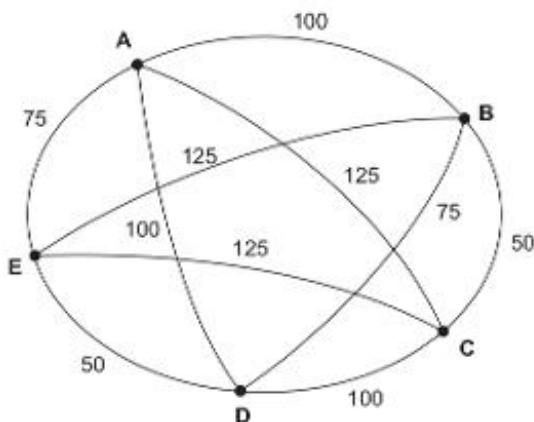
A Figura 3.10 mostra uma maneira de se gerar e comparar caminhos que são possíveis soluções. Começando no nó A, os próximos estados possíveis são acrescentados até que todas as cidades estejam incluídas e o caminho retorne para o ponto inicial. O objetivo é o caminho de custo mais baixo.

Como a Figura 3.10 sugere, a complexidade da busca exaustiva no problema do caixeiro-viajante é  $(N - 1)!$ , onde  $N$  é o número de cidades do grafo. Para 9 cidades, podemos tentar exaustivamente todos os caminhos, mas, para qualquer caso do problema que tenha um tamanho interessante, por exemplo, 50 cidades, a busca exaustiva simples não pode ser realizada em um espaço de tempo prático. De fato, a complexidade de uma busca do tipo  $N!$  cresce tão rapidamente que, em pouco tempo, as combinações da busca se tornam inatratáveis.

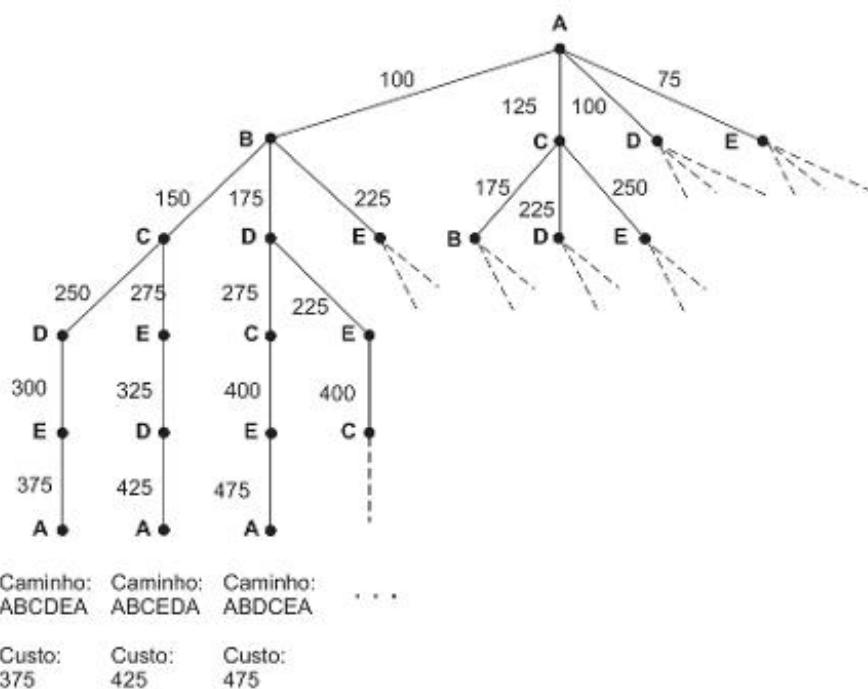
Várias técnicas podem reduzir a complexidade da busca. Uma delas é chamada de *ramificação e poda* (*branch-and-bound*) (Horowitz e Sahni, 1978). Ramificação e poda gera um caminho por vez, mantendo o registro do melhor caminho encontrado até o momento. Esse valor é usado como um *limite* para candidatos futuros. Como os caminhos são construídos com uma cidade por vez, o algoritmo examina cada caminho parcialmente completado. Se o algoritmo determinar que a melhor extensão possível de um caminho, a ramificação, terá um custo maior que o limite, ele eliminará esse caminho parcial e *todas* as suas possíveis extensões. Isso reduz consideravelmente a busca, mas ainda deixa um número exponencial de caminhos ( $1,26^N$ , em vez de  $N!$ ).

Outra estratégia para controlar a busca constrói o caminho de acordo com a regra “vá para a cidade não visitada mais próxima”. O caminho do “vizinho mais próximo” pelo grafo da Figura 3.11 é  $[A,E,D,B,C,A]$ , com um custo de 375 quilômetros. Esse método é muito eficiente, já que existe apenas um caminho que deve ser tentado! Entretanto, a heurística do vizinho mais próximo, algumas vezes chamada de *gulosa*, pode falhar,

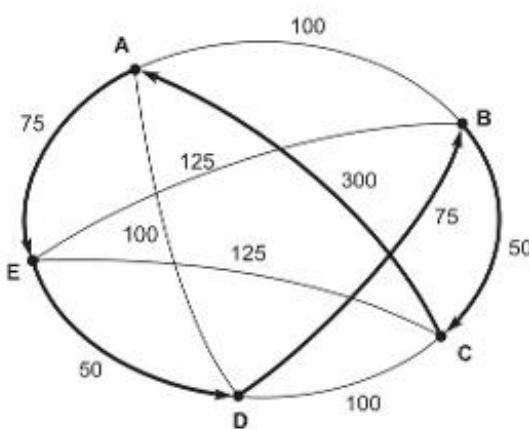
**Figura 3.9** Um exemplo do problema do caixeiro-viajante.



**Figura 3.10** Busca para o problema do caixeiros-viajante. Cada arco é rotulado com o peso total de todos os caminhos desde o nó inicial (A) até o ponto final.



**Figura 3.11** Um exemplo de problema do caixeiros-viajante com o caminho do vizinho mais próximo indicado pelas setas mais grossas. Note que esse caminho (A, E, D, B, C, A), com custo de 550, não é o caminho mais curto. O alto custo relativo do arco (C, A) é responsável pelo insucesso da heurística.



pois existem grafos para os quais ela não encontra o caminho mais curto (veja a Figura 3.11), mas é um compromisso possível quando o tempo requerido torna a busca exaustiva impraticável. A Seção 3.2 examina as estratégias para a busca em espaço de estados.

## 3.2 Estratégias para busca em espaço de estados

### 3.2.1 Busca guiada por dados e busca guiada por objetivo

Um espaço de estados pode ser explorado em duas direções: a partir dos dados fornecidos de uma ocorrência do problema em direção ao objetivo ou, então, a partir do objetivo em direção aos dados.

Na *busca guiada por dados*, algumas vezes chamada de *encadeamento progressivo*, o algoritmo para resolver o problema começa com os fatos fornecidos e um conjunto de movimentos válidos ou regras para a mudança de estado. A busca prossegue pela aplicação de regras aos fatos para produzir novos fatos, que, por sua vez, são usados pelas regras para gerar mais novos fatos. Esse processo continua até que (assim esperamos!) ele gere um caminho que satisfaça a condição-objetivo.

É possível, também, uma abordagem alternativa: comece com o objetivo que desejamos resolver. Veja quais regras ou movimentos válidos poderiam ser usados para gerar esse objetivo e determine que condições devem ser verdadeiras para que eles sejam usados. Essas condições se tornam os novos objetivos, ou *subobjetivos*, para a busca. A busca continua, atuando de forma regressiva através de subobjetivos sucessivos até que (assim esperamos!) ela chegue aos fatos do problema. Esse processo encontra a sequência de movimentos ou regras que conduzem dos dados até o objetivo, embora ele faça isso na ordem inversa. Essa abordagem é chamada de *raciocínio guiado por objetivo*, ou *encadeamento regressivo*, e ele se assemelha ao truque infantil de tentar sair de um labirinto agindo a partir do ponto final em direção ao ponto de partida.

Em resumo: o raciocínio guiado por dados parte dos fatos do problema e aplica as regras ou os movimentos válidos para produzir fatos novos que levem a um objetivo; já o raciocínio guiado por objetivo parte do objetivo, encontra as regras que poderiam produzir o objetivo e segue de forma regressiva pelas regras e pelos subobjetivos sucessivos até os fatos fornecidos do problema.

Fazendo-se uma análise final, chegamos à conclusão que tanto a resolução de problemas guiada por dados quanto a guiada por objetivo realizam a busca no mesmo grafo de espaço de estados; entretanto, a ordem e o número efetivo de estados buscados podem diferir. A estratégia preferida é determinada pelas propriedades do problema propriamente dito. Ai se incluem a complexidade das regras, a “forma” do espaço de estados e a natureza e a disponibilidade dos dados do problema. Todos esses fatores variam para problemas diferentes.

Como um exemplo do efeito que uma estratégia de busca pode ter sobre a complexidade da busca, considere o problema de confirmar ou negar a declaração “eu sou um descendente de Thomas Jefferson”. Uma solução é um caminho de linhagem direta entre “eu” e Thomas Jefferson. Esse espaço pode ser buscado em duas direções, começando com o “eu” e operando sobre linhas ancestrais até Thomas Jefferson, ou começando com Thomas Jefferson e operando por seus descendentes.

Algumas suposições simples nos permitem estimar o tamanho do espaço buscado em cada direção. Thomas Jefferson nasceu há cerca de 250 anos; se supusermos 25 anos por geração, o caminho necessário terá o comprimento aproximado de 10. Como cada pessoa tem exatamente dois genitores (pai e mãe), uma busca regressiva a partir do “eu” examinaria a ordem de  $2^{10}$  ancestrais. Uma busca que procedesse progressivamente a partir de Thomas Jefferson examinaria um número maior de estados, já que as pessoas tendem a ter mais que dois filhos (particularmente nos séculos XVIII e XIX). Se supusermos uma média de apenas três filhos por família, a busca examinaria a ordem de  $3^{10}$  nós da árvore genealógica. Com isso, uma busca regressiva a partir do “eu” examinaria um número bem menor de nós. Note, contudo, que ambas as direções produzem uma complexidade exponencial.

A decisão de escolher entre a abordagem guiada por dados e a guiada por objetivo é baseada na estrutura do problema a ser resolvido. A busca guiada por objetivo é sugerida quando:

1. Um objetivo ou uma hipótese é dado na definição do problema ou pode ser facilmente formulado. Em um provador de teoremas matemáticos, por exemplo, o objetivo é o teorema a ser provado. Muitos sistemas de diagnóstico consideram diagnósticos potenciais em uma forma sistemática, confirmando ou eliminando-os usando o raciocínio guiado por objetivo.
2. Existe um grande número de regras que se aplicam aos fatos do problema, produzindo, assim, um número crescente de conclusões ou objetivos. A seleção prévia de um objetivo pode eliminar a maioria desses ra-

mos, tornando a busca guiada por objetivo mais efetiva em podar o espaço (Figura 3.12). Em um provador de teoremas matemáticos, por exemplo, o número total de regras usadas para produzir um dado teorema é normalmente muito menor que o número de regras que podem ser aplicadas a todo o conjunto de axiomas.

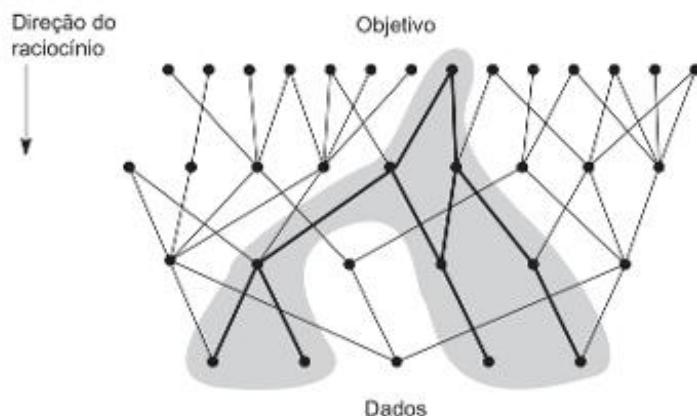
3. Os dados do problema não são fornecidos, mas devem ser adquiridos pelo sistema para a resolução de problemas. Nesse caso, a busca guiada por objetivo pode ajudar a guiar a aquisição de dados. Em um programa de diagnóstico médico, por exemplo, uma grande variedade de testes de diagnóstico pode ser aplicada. Os médicos solicitam apenas o que é necessário para confirmar ou negar uma hipótese particular.

A busca guiada por objetivo usa, dessa forma, conhecimento do objetivo desejado para guiar a busca através das regras relevantes e eliminar os ramos do espaço.

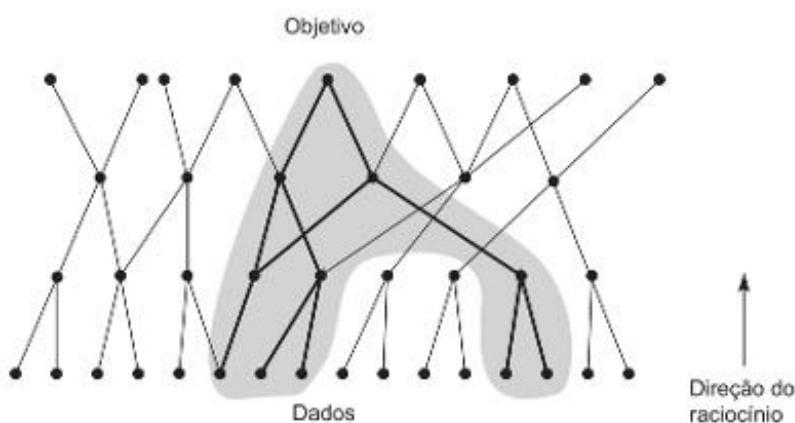
A busca guiada por dados (Figura 3.13) é apropriada a problemas nos quais:

1. Todos os dados (ou a maioria) são fornecidos na formulação inicial do problema. Problemas de interpretação frequentemente se ajustam a essa condição por apresentarem uma coleção de dados e demandarem que o sistema forneça uma interpretação de alto nível. Sistemas que analisam dados particulares (como,

**Figura 3.12** Espaço de estados no qual a busca guiada por objetivo poda efetivamente caminhos de busca irrelevantes.



**Figura 3.13** Espaço de estados no qual a busca guiada por dados poda dados irrelevantes e seus consequentes e determina um dentre vários objetivos possíveis.



por exemplo, o PROSPECTOR ou o programa Dipmeter, que interpretam dados geológicos ou procuram descobrir que minerais são mais prováveis de serem encontrados em um determinado local) se encaixam na abordagem guiada por dados.

2. Existe um grande número de objetivos em potencial, mas há apenas poucas maneiras diferentes de usar os fatos e a informação fornecida de uma instância particular do problema. O programa DENDRAL, um sistema especialista para descobrir a estrutura molecular de compostos orgânicos com base na sua fórmula, em dados de espectrografia de massa e no conhecimento de química, é um exemplo desse tipo. Para qualquer composto orgânico, existe um número enorme de estruturas possíveis. Entretanto, os dados espectrográficos sobre o composto permitem que o programa DENDRAL elimine a maioria deles, restando apenas poucas possibilidades.
3. É difícil formular um objetivo ou hipóteses. No uso do DENDRAL, por exemplo, pouco pode ser conhecido inicialmente sobre a estrutura possível de um composto.

A busca guiada por dados usa o conhecimento e as restrições encontradas nos dados fornecidos de um problema para guiar a busca ao longo das linhas sabidamente verdadeiras.

Para resumir, não há substituto para a análise cuidadosa do problema em particular a ser resolvido, considerando questões como o *fator de ramificação* das aplicações de regra (ver Capítulo 4; na média, quantos novos estados são gerados pelas aplicações de regra nas duas direções?), disponibilidade de dados e facilidade de determinar objetivos potenciais.

### 3.2.2 Implementando a busca de grafo

Na solução de um problema usando a busca guiada por objetivo ou dados, um resolvedor do problema deverá achar um caminho de um estado inicial a um objetivo através do grafo do espaço de estados. A sequência de arcos nesse caminho corresponde às etapas ordenadas da solução. Se um resolvedor do problema recebesse um oráculo ou outro mecanismo infalível para escolher um caminho de solução, a busca não seria necessária. O resolvedor do problema se moveria sem errar pelo espaço até o objetivo desejado, construindo o caminho enquanto seguisse. Como não existem oráculos para problemas interessantes, um resolvedor do problema precisa considerar diferentes caminhos pelo espaço até achar um objetivo. A *busca com retrocesso* (*backtracking*) é uma técnica para tentar sistematicamente todos os caminhos por um espaço de estados.

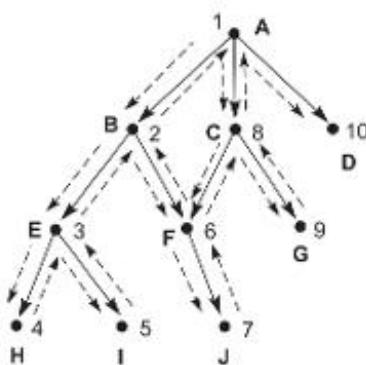
Começamos com a busca com retrocesso porque esse é um dos primeiros algoritmos de busca que os cientistas da computação estudam, e ela tem uma implementação natural em um ambiente recursivo orientado à pilha. Apresentaremos uma versão mais simples do algoritmo de retrocesso com a *busca em profundidade* (Seção 3.2.3).

A busca com retrocesso começa no estado inicial e segue um caminho até alcançar um objetivo ou um “beco sem saída”. Se achar um objetivo, ela termina e retorna o caminho de solução. Se alcançar um beco sem saída, ela “regride” até o nó mais recente no caminho que contenha irmãos não examinados e continua por um desses caminhos, conforme descrito na regra recursiva a seguir:

Se o estado atual  $S$  não atender os requisitos da descrição do objetivo, então gere seu primeiro descendente  $S_{filho1}$  e aplique o procedimento de retrocesso recursivamente a esse nó. Se o retrocesso não achar um nó de objetivo no subgrafo radicado em  $S_{filho1}$ , repita o procedimento para o seu irmão,  $S_{filho2}$ . Continue até que algum descendente de um filho seja um nó de objetivo ou até que todos os filhos tenham sido buscados. Se nenhum dos filhos de  $S$  levar a um objetivo, então o retrocesso “falha” para o genitor de  $S$ , onde é aplicado aos irmãos de  $S$ , e assim por diante.

O algoritmo continua até encontrar um objetivo ou esgotar o espaço de estados. A Figura 3.14 mostra o algoritmo de busca com retrocesso aplicado a um espaço de estados hipotético. A direção das setas tracejadas na árvore indica o progresso da busca acima e abaixo pelo espaço. O número ao lado de cada nó indica a ordem em que ele é visitado. Agora, definimos um algoritmo que realiza um retrocesso, usando três listas para acompanhar os nós no espaço de estados:

**Figura 3.14** Busca com retrocesso de um espaço de estados hipotético.



LE, de lista de estados, lista os estados no caminho atual sendo experimentado. Se um objetivo for achado, LE contém a lista ordenada de estados no caminho da solução.

LNE, de lista de novos estados, contém os nós que aguardam avaliação, ou seja, nós cujos descendentes ainda não foram gerados e buscados.

BSS, de beco sem saída, lista os estados cujos descendentes não contêm um objetivo. Se esses estados forem encontrados novamente, eles serão detectados como elementos de BSS e desconsiderados imediatamente.

Na definição do algoritmo de busca com retrocesso para o caso geral (um grafo em vez de uma árvore), é necessário detectar múltiplas ocorrências de qualquer estado, de modo que ele não seja reentrado e cause laços (infinitos) no caminho. Isso é realizado testando se cada estado recém-gerado pertence a qualquer uma dessas três listas. Se um novo estado pertencer a qualquer uma dessas listas, então ele já terá sido visitado e poderá ser ignorado.

```

função busca_com_retrocesso;
início
  LE := [Inicial]; LNE := [Inicial]; BSS := [ ]; EC := Inicial; % inicializa:
  enquanto LNE ≠ [ ] faça % enquanto há estados para testar
    início
      se EC = objetivo (ou atende descrição do objetivo)
        então retorna LE; % se houver sucesso, retorna lista de estados no caminho.
      se EC não tem filhos (excluindo nós já em BSS, LE e LNE)
        então início
          enquanto LE não está vazio e EC = o primeiro elemento de LE faça
            início
              acrescenta EC em BSS; % registra estado como beco sem saída
              remove primeiro elemento de LE; % regredie
              remove primeiro elemento de LNE;
              EC := primeiro elemento de LNE;
            fim
            acrescenta EC a LE;
          fim
        senão início
          coloca filhos de EC (exceto nós já em BSS, LE ou LNE) em LNE;
          EC := primeiro elemento de LNE;
        fim
      fim
    fim
  fim
fim

```

```

acrescenta EC a LE
fim
fim;
retorna FALHA;
fim.

```

Na busca com retrocesso, o estado atualmente em consideração é chamado de EC, estado atual. EC sempre é igual ao estado acrescentado mais recentemente à LE e representa a “fronteira” do caminho de solução atualmente sendo explorado. As regras de inferência, movimentos em um jogo ou outros operadores apropriados para solução de problemas são ordenados e aplicados ao EC. O resultado é um conjunto ordenado de novos estados, os filhos de EC. O primeiro desses filhos se torna o novo estado atual e o restante é colocado em ordem em LNE para exame futuro. O novo estado corrente é acrescentado à LE e a busca continua. Se EC não tiver filhos, ele é removido de LE (é aí que o algoritmo “retrocede”) e quaisquer filhos restantes do predecessor em LE são examinados.

Um rastro de retrocesso no grafo da Figura 3.14 é dado por:

Inicialize: LE = [A]; LNE = [A]; BSS = []; EC = A;

APÓS ITERAÇÃO	EC	LE	LNE	BSS
0	A	[A]	[A]	[]
1	B	[B A]	[B C D A]	[]
2	E	[E B A]	[E F B C D A]	[]
3	H	[H E B A]	[H I E F B C D A]	[]
4	I	[I E B A]	[I E F B C D A]	[H]
5	F	[F B A]	[F B C D A]	[E I H]
6	J	[J F B A]	[J F B C D A]	[E I H]
7	C	[C A]	[C D A]	[B F J E I H]
8	G	[G C A]	[G C D A]	[B F J E I H]

Como apresentado anteriormente, `busca_com_retrocesso` implementa uma busca guiada por dados, tomando a raiz como um estado inicial e avaliando os seus filhos para buscar o objetivo. O algoritmo pode ser visto como uma busca guiada por objetivo, se fizermos o objetivo ser a raiz do grafo e avaliarmos os descendentes regressivamente para encontrar um estado inicial. Se a descrição do objetivo for do tipo 2 (veja a Seção 3.1.3), o algoritmo deve determinar um estado objetivo através do exame do caminho em LE.

A função `busca_com_retrocesso` é um algoritmo para busca em grafos de espaço de estados. Os algoritmos de busca em grafo no restante do texto, incluindo a busca em profundidade, a busca em amplitude e a busca pela melhor escolha, exploram as ideias usadas no retrocesso, incluindo:

1. O uso de uma lista de estados não processados (LNE) para permitir que o algoritmo retorne (retroceda) a qualquer um desses estados.
2. Uma lista de estados “ruins” (BSS) para evitar que o algoritmo tente novamente caminhos inúteis.
3. Uma lista de nós (LE) sobre o caminho de solução atual, que é retornada se um objetivo for encontrado.
4. Verificação explícita da pertinência de novos estados nessas listas para evitar laços.

A próxima seção introduz algoritmos de busca que, como a `busca_com_retrocesso`, utilizam listas para manter o registro dos estados em um espaço de busca. Esses algoritmos, incluindo as *buscas em profundidade*, *em amplitude* e *melhor escolha* (Capítulo 4), diferem da `busca_com_retrocesso` por fornecerem uma base mais flexível para a implementação de estratégias alternativas de busca.

### 3.2.3 Busca em profundidade e busca em amplitude

Além de especificar uma direção de busca (guiada por dados ou por objetivos), um algoritmo de busca deve determinar a ordem na qual os estados são examinados na árvore ou no grafo. Esta seção considera duas possibilidades para a ordem pela qual os nós do grafo são considerados: *busca em profundidade* e *busca em amplitude*.

Considere o grafo representado na Figura 3.15. Os estados são rotulados (A, B, C, ...) de modo que possam ser referenciados na discussão que se segue. Na busca em profundidade, quando um estado é examinado, todos os seus filhos e os descendentes deles são examinados antes de qualquer um de seus irmãos. A busca em profundidade avança se aprofundando no espaço de estados sempre que possível. Apenas quando não forem encontrados mais descendentes de um estado é que seus irmãos serão considerados. A busca em profundidade examina os estados no grafo da Figura 3.15 na ordem A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R. O algoritmo `busca_com_retrocesso` da Seção 3.2.2 implementa a busca em profundidade.

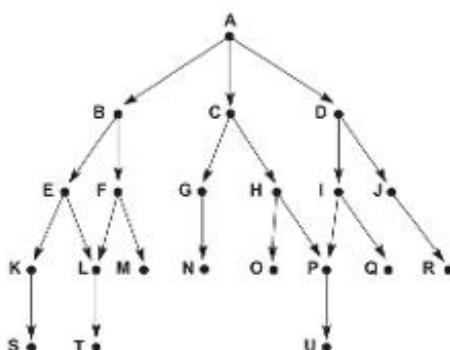
A busca em amplitude, por outro lado, explora o espaço nível por nível. Apenas quando não houver mais estados a serem explorados em um determinado nível é que o algoritmo se moverá para o próximo mais profundo. Uma busca em amplitude do grafo da Figura 3.15 considera os estados na ordem A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U.

Implementamos a busca em amplitude utilizando listas, abertos e fechados, para registrar o progresso através do espaço de estados. A lista abertos, analogamente a LNE no algoritmo `busca_com_retrocesso`, lista os estados que foram gerados, mas cujos filhos não foram examinados. A ordem segundo a qual os estados são removidos de abertos determina a ordem da busca. A lista fechados registra os estados que já foram examinados e corresponde à união das listas BSS e LE do algoritmo `busca_com_retrocesso`.

```

função busca_em_amplitude;
    início
        abertos := [Início];                                % inicialização
        fechados := [ ];
        enquanto abertos ≠ [ ] faça
            início
                remove o estado mais à esquerda em abertos, chame-o de X;
                se X for um objetivo, então retorna SUCESSO          % objetivo encontrado
                senão início
                    gere filhos de X;
                    coloque X em fechados;
                    descarte filhos de X se já estiverem em abertos ou fechados;      % checagem de laços
                    coloque os filhos que restam no final à direita de abertos      % põe na fila
                fim
            fim
            retorna FALHA                                     % não restam estados
        fim.
    
```

**Figura 3.15** Grafo para os exemplos de busca em amplitude e em profundidade.



Os estados filhos são gerados por regras de inferência, movimentos válidos de um jogo, ou por outros operadores de transição de estado. Cada iteração produz todos os filhos do estado X e os adiciona a abertos. Note que abertos é mantido como uma *fila*, ou estrutura de dados do tipo FIFO — *first-in-first-out* (“o primeiro a entrar é o primeiro a sair”). Os estados são adicionados à direita da lista e removidos pela esquerda. Isso orienta a busca em direção aos estados que permanecem em abertos por mais tempo, fazendo com que a busca se dê em amplitude. Os estados filhos que já foram descobertos (já aparecem em abertos ou em fechados) são descartados. Se o algoritmo encerrar porque a condição do laço “enquanto” não for mais satisfeita (`abertos = []`), então ele buscou o grafo inteiro sem encontrar o objetivo desejado: a busca fracassou.

A aplicação de `busca_em_amplitude` sobre o grafo da Figura 3.15 resulta na sequência a seguir. Cada número sucessivo, 2, 3, 4, ... representa uma iteração do laço “enquanto”. U é o estado-objetivo desejado.

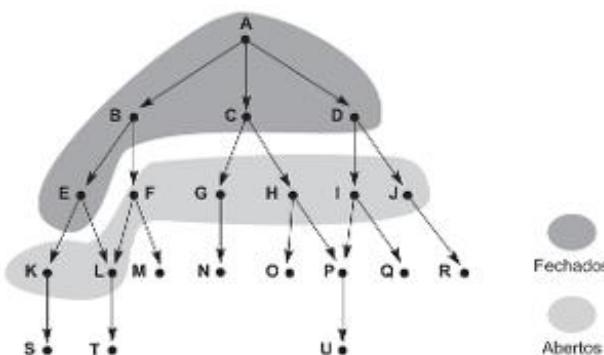
1. abertos = [A]; fechados = []
2. abertos = [B,C,D]; fechados = [A]
3. abertos = [C,D,E,F]; fechados = [B,A]
4. abertos = [D,E,F,G,H]; fechados = [C,B,A]
5. abertos = [E,F,G,H,I,J]; fechados = [D,C,B,A]
6. abertos = [F,G,H,I,J,K,L]; fechados = [E,D,C,B,A]
7. abertos = [G,H,I,J,K,L,M] (pois L já está em abertos); fechados = [F,E,D,C,B,A]
8. abertos = [H,I,J,K,L,M,N]; fechados = [G,F,E,D,C,B,A]
9. e assim por diante, até que U seja encontrado ou abertos = [].

A Figura 3.16 ilustra o grafo da Figura 3.15 após seis iterações de `busca_em_amplitude`. Os estados em abertos e fechados estão ressaltados. Os estados que não estão sombreados ainda não foram descobertos pelo algoritmo. Note que abertos registra os estados na “fronteira” da busca em qualquer estágio, e que fechados registra os estados que já foram visitados.

Como a busca em amplitude considera todos os nós em cada nível do grafo antes de passar para um nível mais profundo do espaço, todos os estados são alcançados primeiro ao longo do caminho mais curto em relação ao estado inicial. Com isso, pode-se garantir que a busca em amplitude encontra o caminho mais curto entre o estado inicial e o objetivo. Além disso, como todos os estados são encontrados primeiro ao longo do caminho mais curto, qualquer estado que seja encontrado por uma segunda vez estará em um caminho de comprimento igual ou maior que o primeiro. Como não pode ocorrer de estados duplicados serem encontrados ao longo de um caminho melhor, o algoritmo simplesmente descarta quaisquer estados duplicados.

Frequentemente, é útil manter outras informações em abertos e fechados além dos nomes dos estados. Note, por exemplo, que `busca_em_amplitude` não mantém uma lista de estados sobre o caminho atual até um objetivo, como `busca_com_retrocesso` fazia com a lista LE; todos os estados visitados são mantidos em fechados. Se o caminho para uma solução for necessário, ele não pode ser retornado por esse algoritmo. A solução pode ser encontrada armazenando-se informação de seus ancestrais juntamente com cada estado. Um estado pode ser arma-

**Figura 3.16** Grafo da Figura 3.15 na iteração 6 da busca em amplitude. Os estados abertos e fechados estão destacados.



zenado juntamente com um registro do estado de seus genitores, isto é, um par (estado, pais). Se isso for feito na busca da Figura 3.15, os conteúdos de abertos e fechados na quarta iteração seriam:

abertos = [(D,A), (E,B), (F,B), (G,C), (H,C)]; fechados = [(C,A), (B,A), (A,nada)]

O caminho (A, B, F) que leva de A a F poderia ser construído facilmente a partir dessa informação. Quando um objetivo é encontrado, o algoritmo pode construir o caminho-solução retrocedendo ao longo dos nós genitores do objetivo até o estado inicial. Note que o estado A tem um genitor do tipo “nada”, indicando que ele é um estado inicial; isso faz com que a reconstrução do caminho seja interrompida. Como a busca em amplitude encontra cada estado ao longo do caminho mais curto e retém a primeira versão de cada estado, esse é o caminho mais curto desde um inicial até um objetivo.

A Figura 3.17 mostra os estados removidos de abertos e examinados em uma busca em amplitude do grafo do quebra-cabeça dos 8. Como anteriormente, os arcos correspondem a movimentos do vazio para cima, para a direita, para baixo e para a esquerda. O número ao lado de cada estado indica a ordem em que ele foi removido de abertos. Não são mostrados os estados que ficaram em abertos quando o algoritmo foi parado.

A seguir, criamos um algoritmo de busca em profundidade, uma simplificação do algoritmo de retrocesso já apresentado na Seção 3.2.3. Nesse algoritmo, os estados descendentes são adicionados e removidos a partir do final *esquerdo* de abertos, que é mantido como uma *pilha*, ou estrutura do tipo LIFO (do inglês *last-in-first-out*, “o último a entrar é o primeiro a sair”). A organização de abertos como uma pilha direciona a busca para os estados gerados mais recentemente, produzindo uma ordem de busca que avança em profundidade:

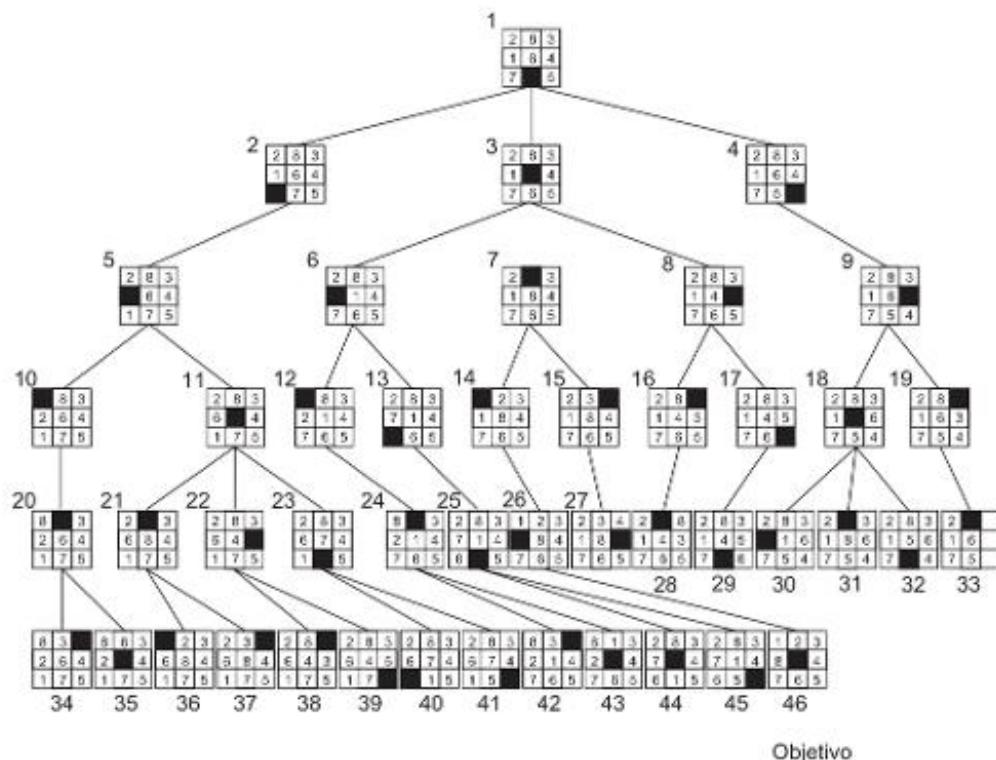
função busca\_em\_profundidade;

```

início
    abertos := [Início];                                % inicializa
    fechados := [ ];
    enquanto abertos ≠ [ ] faça                         % restam estados
        ...
    fim

```

**Figura 3.17** Busca em amplitude do quebra-cabeça dos 8 mostrando a ordem em que os estados foram removidos de abertos.



```

inicio
    remove estado mais à esquerda em abertos, chame-o de X;
    se X é um objetivo, então retorna SUCESSO                                % objetivo encontrado
    senão inicio
        gere filhos de X;
        coloque X em fechados;
        descarte filhos de X se já estiverem em abertos ou fechados;          % checagem de laços
        coloque filhos que restam no final esquerdo de abertos
    fim
    fim;
    retorna FALHA
fim.                                                               % não restam estados

```

A consequência da aplicação de busca\_em\_profundidade no grafo da Figura 3.15 aparece a seguir. Cada iteração sucessiva do laço “enquanto” é indicada por uma única linha (2, 3, 4, ...). Os estados iniciais de abertos e fechados são apresentados na linha 1. Considere que U seja o estado-objetivo.

1. abertos = [A]; fechados = [ ]
2. abertos = [B,C,D]; fechados = [A]
3. abertos = [E,F,C,D]; fechados = [B,A]
4. abertos = [K,L,F,C,D]; fechados = [E,B,A]
5. abertos = [S,L,F,C,D]; fechados = [K,E,B,A]
6. abertos = [L,F,C,D]; fechados = [S,K,E,B,A]
7. abertos = [T,F,C,D]; fechados = [L,S,K,E,B,A]
8. abertos = [F,C,D]; fechados = [T,L,S,K,E,B,A]
9. abertos = [M,C,D] (como L já está em fechados); fechados = [F,T,L,S,K,E,B,A]
10. abertos = [C,D]; fechados = [M,F,T,L,S,K,E,B,A]
11. abertos = [G,H,D]; fechados = [C,M,F,T,L,S,K,E,B,A]

e assim por diante, até que U seja descoberto ou, então, que abertos = [ ].

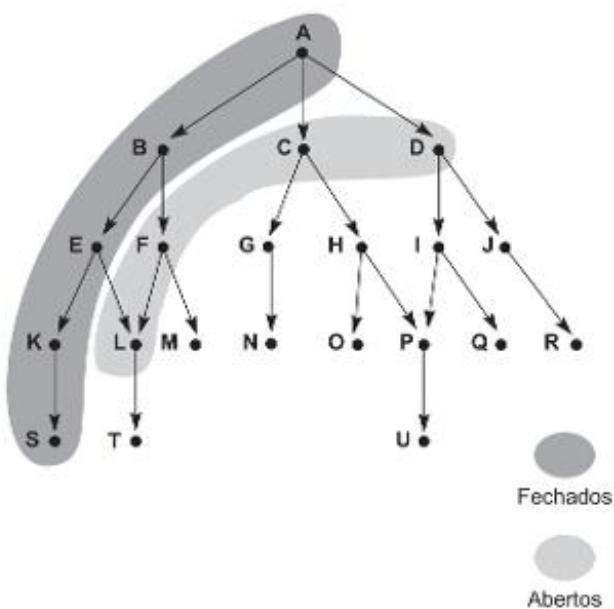
Como no caso de busca\_em\_amplitude, abertos lista todos os estados descobertos, mas ainda não avaliados (a “fronteira” atual da busca), e fechados registra os estados já considerados. A Figura 3.18 mostra o grafo da Figura 3.15 na sexta iteração de busca\_em\_profundidade. Os conteúdos de abertos e fechados estão ressaltados. Como no caso de busca\_em\_amplitude, o algoritmo poderia armazenar um registro do genitor juntamente com cada estado, permitindo que o algoritmo reconstrua o caminho que levou do estado inicial a um objetivo.

Diferentemente da busca em amplitude, uma busca em profundidade não tem garantias de achar o caminho mais curto até um estado na primeira vez que esse estado for encontrado. Mais adiante na busca, um caminho diferente pode ser achado para qualquer estado. Se o tamanho do caminho for importante em um resolvedor do problema, quando o algoritmo encontrar um estado duplicado, ele deverá salvar a versão alcançada ao longo do caminho mais curto. Isso poderia ser feito armazenando cada estado como uma tripla: (estado, genitor, tamanho\_do\_caminho). Quando forem gerados filhos, o valor do tamanho do caminho é simplesmente incrementado em um e salvo com o filho. Se um filho for alcançado ao longo de vários caminhos, essa informação pode ser usada para reter a melhor versão. Isso é tratado com mais detalhes na discussão do *algoritmo A* no Capítulo 4. Observe que reter a melhor versão de um estado em uma busca simples em profundidade não garante que um objetivo será alcançado ao longo do caminho mais curto.

A Figura 3.19 mostra uma busca em profundidade para o quebra-cabeça dos 8. Como já dissemos, o espaço é gerado pelas quatro regras de “mover o vazio” (para cima, para baixo, para a esquerda e para a direita). Os números ao lado dos estados indicam a ordem em que eles são considerados, ou seja, removidos de abertos. Os estados deixados em abertos quando o objetivo é encontrado não aparecem. Uma profundidade limitada a 5 foi imposta nessa busca para evitar que ela se perca na profundidade do espaço.

Assim como na escolha entre a busca guiada por dados e a por objetivos para avaliação de um grafo, a escolha entre a busca em profundidade ou em amplitude depende do problema específico sendo resolvido. Características significativas incluem a importância de achar o caminho mais curto para um objetivo, o fator de ramificação do espaço, o tempo de

**Figura 3.18** Grafo da Figura 3.15 na iteração 6 da busca em profundidade. Os estados abertos e fechados estão destacados.



computação e os recursos de espaço disponíveis, o tamanho médio dos caminhos para um nó de objetivo e se queremos todas as soluções ou apenas a primeira. Ao tomar essas decisões, existem vantagens e desvantagens para cada técnica.

**Amplitude** Por sempre examinar todos os nós no nível  $n$  antes de prosseguir para o nível  $n + 1$ , a busca em amplitude sempre acha o caminho mais curto até um nó de objetivo. Em um problema em que se sabe que existe uma solução simples, essa solução será encontrada. Infelizmente, se houver um alto fator de ramificação, ou seja, se os estados tiverem um grande número médio de filhos, a explosão combinatória poderá impedir que o algoritmo ache uma solução usando a memória disponível. Isso se deve ao fato de que todos os nós não expandidos para cada nível da busca devem ser mantidos em abertos. Para buscas profundas, ou em espaços de estados com um alto fator de ramificação, isso pode se tornar muito complicado.

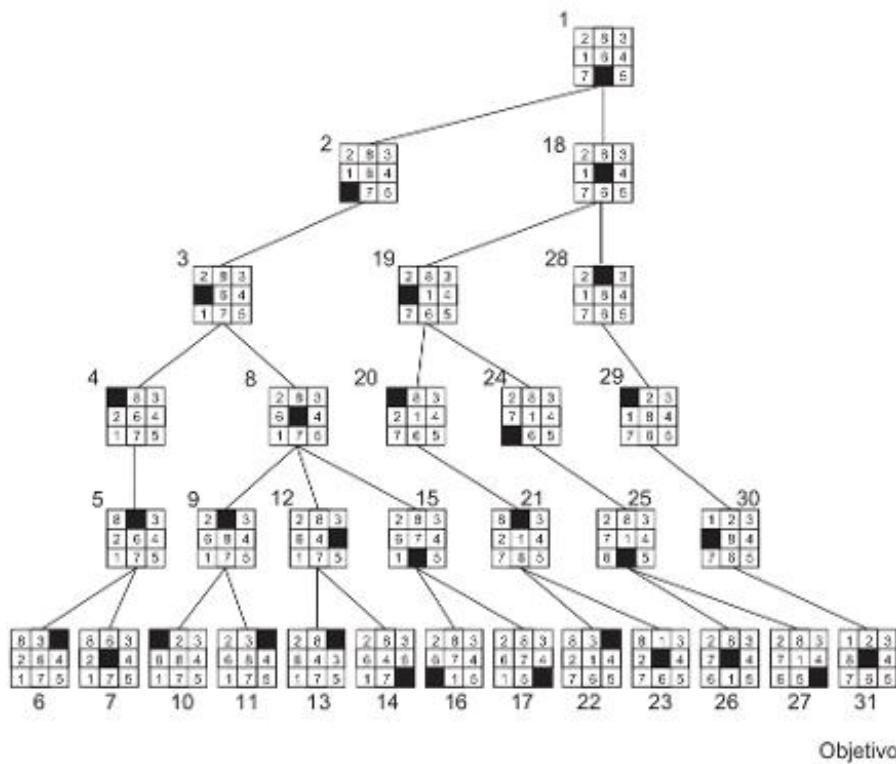
A utilização do espaço da busca em amplitude, medida em termos do número de estados em abertos, é uma função exponencial do tamanho atual do caminho. Se cada estado tiver uma média de  $B$  filhos, o número de estados em determinado nível é  $B$  vezes o número de estados no nível anterior. Isso produz  $B^n$  estados no nível  $n$ . A busca em amplitude colocaria todos esses em abertos ao começar a examinar o nível  $n$ . Isso pode ser quase impossível se os caminhos de solução forem longos, como, por exemplo, no jogo de xadrez.

**Profundidade** A busca em profundidade chega rapidamente a um espaço de busca profundo. Se for sabido que o caminho de solução será longo, a busca em profundidade não desperdiçará tempo buscando um grande número de estados “superficiais” no grafo. Por outro lado, a busca em profundidade pode se “perder” na profundidade de um grafo, ignorando caminhos mais curtos até um objetivo ou até mesmo ficando presa em um caminho infinitamente longo que não leva a um objetivo.

A busca em profundidade é muito mais eficiente para espaços de busca com muitas ramificações, pois não tem que manter todos os nós em determinado nível na lista de abertos. O uso do espaço da busca em profundidade é uma função linear do tamanho do caminho. Em cada nível, abertos retém apenas os filhos de um único estado. Se um grafo tem uma média de  $B$  filhos por estado, isso requer uma utilização total de espaço de  $B \times n$  estados para avançar em profundidade de  $n$  níveis no espaço.

A melhor resposta à questão sobre a disputa entre “busca em profundidade contra busca em amplitude” é examinar cuidadosamente o espaço do problema e consultar especialistas na área. Para o jogo de xadrez, por exemplo, a busca em amplitude simplesmente não é possível. Em jogos mais simples, a busca em amplitude não só pode ser possível, como também a única maneira de evitar a derrota, já que ela retorna o menor caminho.

**Figura 3.19** Busca em profundidade para o quebra-cabeça dos 8 com uma profundidade limitada a 5.



### 3.2.4 Busca em profundidade com aprofundamento iterativo

Um compromisso interessante dessas vantagens e desvantagens é a utilização de um limite de aprofundamento na busca em profundidade. O limite de aprofundamento força uma falha no caminho de busca quando ele ultrapassa certo nível. Isso causa uma varredura do espaço de busca até aquele nível, semelhante a uma busca em amplitude. Quando se sabe que uma solução se encontra dentro de certa profundidade, ou quando restrições de tempo limitam o número de estados que podem ser considerados, como ocorre em um espaço extremamente grande, como no jogo de xadrez, então uma busca em profundidade com limite de aprofundamento pode ser o mais apropriado. A Figura 3.19 mostrou uma busca em profundidade do quebra-cabeça dos 8 na qual um limite de aprofundamento de 5 causou a varredura ao longo do espaço naquele nível.

Essas considerações levaram a um algoritmo de busca que soluciona muitas das desvantagens, tanto da busca em profundidade como da busca em amplitude. A *busca em profundidade com aprofundamento iterativo* (Korf, 1987) realiza uma busca em profundidade do espaço com uma limitação de profundidade de 1. Se ela não conseguir encontrar um objetivo, ela realiza outra busca em profundidade com um limite de aprofundamento de 2. O processo prossegue, aumentando o limite de aprofundamento em 1 a cada iteração. A cada iteração, o algoritmo realiza uma busca completa em profundidade, sob o limite de aprofundamento atual. Não é armazenada nenhuma informação sobre o espaço de estados entre as iterações.

Como o algoritmo busca o espaço nível por nível, é garantido que será encontrado o menor caminho até um objetivo. Como ele realiza apenas uma busca em profundidade a cada iteração, a utilização de espaço em qualquer nível  $n$  é  $B \times n$ , onde  $B$  é o número médio de filhos de um nó.

É interessante notar que, embora possa parecer que a busca em profundidade com aprofundamento iterativo seja muito menos eficiente em termos de tempo, tanto em relação à busca em profundidade como à busca em amplitude, a sua complexidade de tempo é, na realidade, da mesma ordem de magnitude que ambos os algoritmos:  $O(B^n)$ . Uma explanação intuitiva para esse paradoxo aparente é dada por Korf (1987):

Como o número de nós em um determinado nível da árvore cresce exponencialmente com a profundidade, quase todo o tempo é gasto no nível mais profundo, ainda que os níveis mais superficiais sejam gerados um número de vezes aritmeticamente crescente.

Infelizmente, pode-se mostrar que todas as estratégias de busca discutidas neste capítulo — busca em profundidade, busca em amplitude e aprofundamento iterativo — têm complexidade de tempo exponencial no pior caso. Isso é verdade para todos os algoritmos de busca *não informados*. A única abordagem para busca que reduz a sua complexidade é a que emprega heurísticas para guiar a busca. A *busca pela melhor escolha* é um algoritmo de busca que é semelhante aos algoritmos para busca em profundidade e em amplitude apresentados anteriormente. Entretanto, a busca pela melhor escolha ordena os estados na lista abertos, a fronteira atual da busca, de acordo com uma medida heurística de sua qualidade. A cada iteração, não é considerado o estado mais profundo nem o mais superficial, mas sim o “melhor”. A busca pela melhor escolha é o tópico principal do Capítulo 4.

### 3.3 Usando o espaço de estados para representar o raciocínio com o cálculo proposicional e de predicados

#### 3.3.1 Descrição por espaço de estados de um sistema lógico

Quando definimos grafos de espaço de estados na Seção 3.1, observamos que os nós devem ser distinguidos entre si, com cada nó representando algum estado do processo de solução. O cálculo proposicional e o de predicados podem ser usados como uma linguagem de especificação formal para fazer essas distinções, bem como para mapear os nós de um grafo para o espaço de estados. Além disso, podem ser usadas regras de inferência para criar e descrever os arcos entre estados. Dessa forma, problemas do cálculo de predicados, como determinar se uma expressão particular é uma consequência lógica de um determinado conjunto de asserções, podem ser solucionados usando a busca.

A consistência e a completude das regras de inferência do cálculo de predicados garantem a correção das conclusões derivadas através dessa forma de raciocínio baseado em grafo. Essa capacidade de produzir uma prova formal da integridade de uma solução através do mesmo algoritmo que produz a solução é uma característica única entre as soluções de problemas por inteligência artificial e baseadas em prova de teoremas.

Embora muitos estados de problemas — por exemplo, no jogo da velha — possam ser mais naturalmente descritos por outras estruturas de dados — como, por exemplo, os arranjos —, o poder e a generalidade da lógica permitem que muitas soluções de problemas por IA usem descrições e regras de inferências do cálculo de predicados. Outras representações da IA, como as regras (Capítulo 8), redes semânticas, ou quadros (*frames*) (Capítulo 7) também empregam estratégias de busca semelhantes a essas apresentadas na Seção 3.2.

##### **Exemplo 3.3.1 Cálculo proposicional**

O primeiro exemplo de como um conjunto de relacionamentos lógicos pode ser visto como definindo um grafo vem do cálculo proposicional. Se  $p, q, r, \dots$  são proposições, considere as asserções:

$$\begin{aligned} q &\rightarrow p \\ r &\rightarrow p \\ v &\rightarrow q \\ s &\rightarrow r \\ t &\rightarrow r \\ s &\rightarrow u \\ s \\ t \end{aligned}$$

Desse conjunto de asserções e da regra de inferência *modus ponens*, certas proposições ( $p$ ,  $r$  e  $u$ ) podem ser inferidas; outras (como  $v$  e  $q$ ) não podem ser inferidas da mesma forma e, de fato, não resultam logicamente dessas asserções. O relacionamento entre as asserções iniciais e essas inferências é expresso pelo grafo direcionado da Figura 3.20.

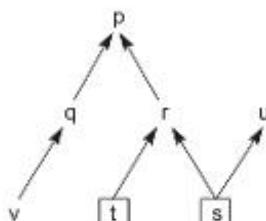
Na Figura 3.20, os arcos correspondem a implicações lógicas ( $\rightarrow$ ). As proposições que são dadas como verdadeiras ( $s$  e  $t$ ) correspondem aos dados fornecidos do problema. Proposições que são consequências lógicas desse conjunto de asserções correspondem aos nós que podem ser alcançados ao longo de um caminho direcionado, partindo de um estado representando uma proposição verdadeira; um caminho assim corresponde a uma sequência de aplicações do *modus ponens*. Por exemplo, o caminho  $[s, r, p]$  corresponde à sequência de inferências:

$s \wedge s \rightarrow r$  produz  $r$ .

$r \wedge r \rightarrow p$  produz  $p$ .

Com essa representação, determinar se uma dada proposição é uma consequência lógica de um conjunto de proposições torna-se um problema para encontrar um caminho que saia de um nó do tipo caixa (o nó inicial) até a proposição (o nó objetivo). Assim, a tarefa pode ser disposta como um problema de busca em grafo. A estratégia de busca usada aqui é guiada por dados, visto que ela parte do que é conhecido (as proposições verdadeiras) em direção ao objetivo. Alternativamente, uma estratégia guiada por objetivo poderia ser aplicada ao mesmo espaço de estados, iniciando com a proposição a ser provada (o objetivo) e buscando, regressivamente ao longo dos arcos, para encontrar suporte para o objetivo entre as proposições verdadeiras. Podemos também realizar a busca nesse espaço de inferências tanto em profundidade quanto em amplitude.

**Figura 3.20** Grafo de espaço de estados de um conjunto de implicações do cálculo proposicional.



### 3.3.2 Grafos E/OU

No exemplo do cálculo proposicional da Seção 3.3.1, todas as asserções eram implicações da forma  $p \rightarrow q$ . Não discutimos a maneira como os operadores lógicos E e OU poderiam ser representados em um grafo como esse. Expressar as relações lógicas definidas por esses operadores requer uma extensão do modelo básico de grafo definido na Seção 3.1 para o chamado *grafo E/OU*. Os grafos E/OU são uma ferramenta importante para descrever os espaços de busca gerados por vários problemas de IA, incluindo aqueles solucionados por provadores de teoremas e sistemas especialistas baseados em lógica.

Em expressões da forma  $q \wedge r \rightarrow p$ , tanto  $q$  quanto  $r$  devem ser verdadeiros para que  $p$  seja verdadeiro. Em expressões da forma  $q \vee r \rightarrow p$ , a verdade de  $q$  ou de  $r$  é suficiente para provar que  $p$  é verdadeiro. Como implicações contendo premissas disjuntivas podem ser escritas como implicações separadas, essa expressão é frequentemente escrita como  $q \rightarrow p$ ,  $r \rightarrow p$ . Para representar essas diferentes relações graficamente, os grafos E/OU fazem distinção entre nós E e nós OU. Se as premissas de uma implicação estiverem conectadas por um operador  $\wedge$ , elas

são chamadas de nós E no grafo e os arcos desse nó são ligados por um elo curvo. A expressão  $q \wedge r \rightarrow p$  é representada pelo grafo E/OU da Figura 3.21.

O elo que conecta os arcos na Figura 3.21 capta a ideia de que tanto q quanto r devem ser verdadeiros para provar p. Se as premissas forem conectadas por um operador OU, elas são vistas como nós OU no grafo. Arcos de nós OU para seus nós genitores não são conectados dessa forma (Figura 3.22). Isso capta a noção de que a verdade de qualquer uma das premissas é independentemente suficiente para determinar a verdade da conclusão.

Um grafo E/OU é, na realidade, uma especialização de um tipo de grafo conhecido como um *hipergrafo*, o qual conecta nós por conjuntos de arcos em vez de arcos únicos. Um hipergrafo é definido como:

### Definição

#### HIPERGRAFO

Um hipergrafo consiste em:

N, um conjunto de nós.

H, um conjunto de hiperarcos definidos por pares ordenados nos quais o primeiro elemento do par é um nó único de N e o segundo elemento é um subconjunto de N.

Um grafo ordinário é um caso especial de hipergrafo no qual todos os conjuntos de nós descendentes têm uma cardinalidade de 1.

Hiperarcos também são conhecidos como *k-conectores*, onde k é a cardinalidade do conjunto de nós descendentes. Se  $k = 1$ , pode-se considerar o descendente como um nó OU. Se  $k > 1$ , os elementos do conjunto de descendentes podem ser vistos como nós E. Nesse caso, um conector é desenhado desde as arestas individuais do nó genitor até os nós descendentes; veja, por exemplo, o elo curvo na Figura 3.21.

**Figura 3.21** Grafo E/OU da expressão  $q \wedge r \rightarrow p$ .



**Figura 3.22** Grafo E/OU da expressão  $q \vee r \rightarrow p$ .



### Exemplo 3.3.2 Busca em grafo E/OU

O segundo exemplo também é do cálculo proposicional, mas gera um grafo que contém descendentes tanto do tipo E quanto do tipo OU. Suponhamos uma situação em que as seguintes proposições sejam verdadeiras:

a  
b  
c  
 $a \wedge b \rightarrow d$   
 $a \wedge c \rightarrow e$   
 $b \wedge d \rightarrow f$   
 $f \rightarrow g$   
 $a \wedge e \rightarrow h$

Esse conjunto de asserções gera o grafo E/OU da Figura 3.23.

As seguintes questões poderiam ser formuladas (e as respostas deduzidas por busca nesse grafo):

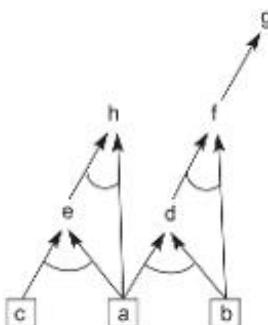
1. h é verdadeira?
2. h é verdadeira se b deixar de ser verdadeira?
3. Qual é o menor caminho (isto é, a menor sequência de inferências) para mostrar que X (uma proposição) é verdadeira?
4. Mostre que a proposição p (note que para p não tem suporte) é falsa. O que isso significa? O que seria necessário para chegar a essa conclusão?

A busca em grafo E/OU requer apenas um pouco mais de armazenamento que a busca em grafos regulares; um exemplo desta última é o algoritmo busca\_com\_retrocesso, discutido anteriormente. Os descendentes OU são verificados da mesma forma que em busca\_com\_retrocesso: uma vez encontrado um caminho que conecte um objetivo a um nó inicial ao longo de nós OU, o problema será resolvido. Se um caminho levar a uma falha, o algoritmo poderá retroceder e tentar outro ramo. Ao realizar a busca através de nós E, entretanto, todos os descendentes E de um nó devem ser resolvidos (ou se deve provar que são verdadeiros) para resolver o nó genitor.

No exemplo da Figura 3.23, uma estratégia guiada por objetivo para determinar a verdade de h primeiro tenta provar tanto a quanto e. A verdade de a é imediata, mas a verdade de e requer a verdade tanto de c quanto de a; estes são dados como verdadeiros. Uma vez que o resolvedor do problema tenha seguido todos esses arcos até as proposições verdadeiras, os valores verdadeiros são recombinaados no nó E para verificar a verdade de h.

Por outro lado, uma estratégia guiada por dados para determinar a verdade de h começa com os fatos conhecidos (c, a e b) e se inicia adicionando novas proposições a esse conjunto de fatos conhecidos, de acordo com as re-

Figura 3.23 Grafo E/OU de um conjunto de expressões em cálculo proposicional.



trições do grafo E/OU.  $e$  ou  $d$  poderiam ser a primeira proposição adicionada ao conjunto de fatos. Essas adições tornam possível a inferência de novos fatos. Esse processo continua até que o objetivo desejado  $\vdash$  seja provado.

Uma forma de se considerar a busca em grafo E/OU é observando que o operador  $\wedge$  (assim como os nós E do grafo) indica uma decomposição do problema, pela qual este é dividido em subproblemas, de forma que todos eles devam ser solucionados para resolver o problema original. Um operador  $\vee$  na representação do problema por cálculo de predicados indica uma seleção, um ponto na solução do problema, onde se deve fazer uma escolha entre caminhos ou estratégias alternativos para solucionar o problema, sendo qualquer um deles, caso bem-sucedido, suficiente para resolvê-lo.

### 3.3.3 Mais exemplos e aplicações

#### *Exemplo 3.3.3 MACSYMA*

Um exemplo natural de um grafo E/OU é um programa para a integração simbólica de funções matemáticas. MACSYMA é um programa bem conhecido que é bastante utilizado por matemáticos. O raciocínio do MACSYMA pode ser representado como um grafo E/OU. Para realizar integrações, uma classe importante de estratégias adotadas envolve quebrar uma expressão em subexpressões que possam ser integradas independentemente, com o resultado sendo combinado algebricamente em uma expressão de solução. Exemplos dessa estratégia incluem a regra para integração por partes e a regra para decompor a integral de uma soma na soma das integrais dos termos individuais. Essas estratégias, que representam a decomposição de um problema em subproblemas independentes, podem ser representadas por nós E no grafo.

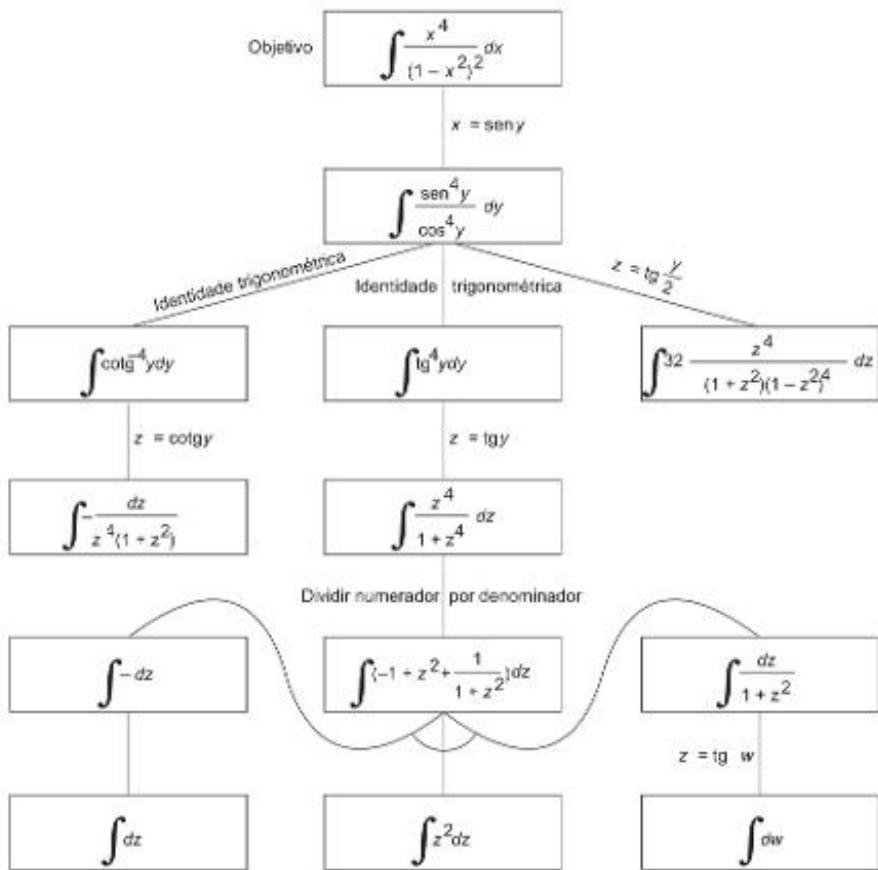
Outra classe de estratégias envolve a simplificação de uma expressão por meio de várias substituições algébricas. Como qualquer expressão pode permitir várias substituições diferentes, cada uma representando uma estratégia de solução independente, essas estratégias são representadas por nós OU no grafo. A Figura 3.24 ilustra o espaço buscado por tal resolvedor de problemas. A busca desse grafo é guiada por objetivo, pois começa com a consulta “determine a integral de uma função em particular” e procura de volta até as expressões algébricas que definem essa integral. Observe que esse é um exemplo em que a busca guiada por objetivo é a estratégia óbvia. Seria praticamente impossível para um resolvedor de problemas determinar as expressões algébricas que formaram a integral desejada sem retroceder a partir da consulta.

#### *Exemplo 3.3.4 Busca E/OU guiada por objetivo*

Esse exemplo é tirado do cálculo de predicados e representa uma busca em grafo guiada por objetivo, onde o objetivo a ser provado como verdadeiro nessa situação é uma expressão de cálculo de predicados contendo variáveis. Os axiomas são as descrições lógicas de uma relação entre um cão, Fred, e seu dono, Sam. Consideraremos que um dia frio não é um dia quente, evitando problemas como a complexidade adicionada por expressões equivalentes para predicados, uma questão discutida com mais detalhes nos capítulos 7 e 14. Os fatos e as regras desse exemplo são dados em sentenças em português seguidas por seus equivalentes no cálculo de predicados:

1. Fred é um collie.  
 $\text{collie}(\text{fred}).$
2. Sam é o dono de Fred.  
 $\text{dono}(\text{fred}, \text{sam}).$
3. O dia é sábado  
 $\text{dia}(\text{sábado}).$
4. Está frio no sábado.  
 $\neg (\text{quente}(\text{sábado})).$

**Figura 3.24** Grafo E/OU de parte do espaço de estados para integração de uma função, de Nilsson (1971).



5. Fred é treinado.  
treinado(fred).
6. Spaniels são bons cachorros, assim como os collies treinados.  
 $\forall X[\text{spaniel}(X) \vee (\text{collie}(X) \wedge \text{treinado}(X)) \rightarrow \text{bom\_cachorro}(X)]$
7. Se um cachorro for bom e tiver um dono, então ele estará com seu dono.  
 $\forall (X,Y,Z) [\text{bom\_cachorro}(X) \wedge \text{dono}(X,Y) \wedge \text{local}(Y,Z) \rightarrow \text{local}(X,Z)]$
8. Se for sábado e estiver quente, então Sam estará no parque.  
(dia(sábado)  $\wedge$  quente(sábado))  $\rightarrow \text{local}(\text{sam},\text{parque})$ .
9. Se for sábado e não estiver quente, então Sam estará no museu.  
(dia(sábado)  $\wedge$   $\neg$  (quente(sábado)))  $\rightarrow \text{local}(\text{sam},\text{museu})$ .

O objetivo é a expressão  $\exists X \text{ local}(\text{fred}, X)$ , significando “onde está Fred?”. Um algoritmo de busca com retrocesso examina meios alternativos de estabelecer esse objetivo: Fred é um bom cachorro e Fred tem um dono e o dono de Fred está em um local, então Fred está nesse local também. As premissas dessa regra são então examinadas: o que significa ser um bom\_cachorro etc.? Esse processo continua, construindo o grafo E/OU da Figura 3.25.

Examinemos essa busca com mais detalhes, principalmente porque é um exemplo de busca guiada por objetivo usando o cálculo de predicados e porque ilustra o papel da unificação na geração do espaço de busca.

O problema a ser resolvido é “onde está Fred?”. Mais formalmente, ele pode ser visto como determinar uma substituição para a variável X, se tal substituição existir, sob a qual  $\text{local}(\text{fred}, X)$  é uma consequência lógica das asserções iniciais.

Quando se deseja determinar o local de Fred, são examinadas cláusulas que têm local como sua conclusão, sendo a primeira a cláusula 7. Essa conclusão,  $\text{local}(X, Z)$ , é então unificada com  $\text{local}(\text{fred}, X)$  pelas substituições  $\{\text{fred}/X, X/Z\}$ . As premissas dessa regra, sob o mesmo conjunto de substituições, formam os descendentes E do objetivo inicial:

bom cachorro(fred)  $\wedge$  dono(fred,Y)  $\wedge$  local(Y,X).

Essa expressão pode ser interpretada como significando que um modo de se encontrar Fred é ver se Fred é um bom cachorro, descobrir quem é seu dono e descobrir onde o dono está. O objetivo inicial foi substituído por três subobjetivos. Esses subobjetivos são nós E e todos devem ser resolvidos.

Para resolver esses subobjetivos, o resolvedor de problemas determina primeiro se Fred é um bom\_cachorro. Isso condiz com a conclusão da cláusula 6 usando a substituição {fred/X}. A premissa da cláusula 6 é o OU de duas expressões:

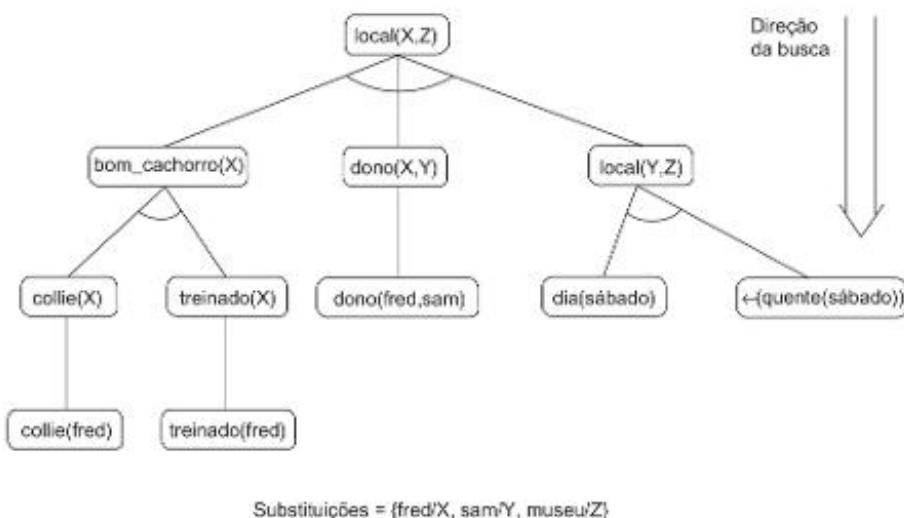
`spaniel(fred) ∨ (collie(fred) ∧ treinado(fred))`

O primeiro desses nós OU é `spaniel(fred)`. A base de dados não contém essa asserção, assim o resolvedor deve supô-la como falsa. O outro nó OU é `(collie(fred) ∧ treinado(fred))`, isto é, Fred é um collie e Fred é treinado. Ambas as asserções devem ser verdadeiras, e realmente elas o são pelas cláusulas 1 e 5.

Isso prova que `bom_cachorro(fred)` é verdadeiro. O resolvidor, então, examina a segunda premissa da cláusula 7: de `dono(X,Y)`. Pela substituição `{fred/X}`, `dono(X,Y)` se torna `dono(fred,Y)`, que se unifica com o fato (cláusula 2) de `dono(fred,sam)`. Isso produz a substituição unificadora `{sam/Y}`, que também fornece o valor de `sam` para o terceiro subobjetivo da cláusula 7, criando o novo objetivo local(`sam,X`).

Ao resolver esse objetivo, supondo-se que o resolvedor experimente as regras em ordem, o objetivo local( $sam,X$ ) primeiro irá unificar-se com a conclusão da cláusula 7. Note que a mesma regra é experimentada com diferentes ligações para  $X$ . Lembre-se de que vimos no Capítulo 2 que  $X$  é uma variável “fictícia” e poderia ter qualquer nome (qualquer cadeia de caracteres iniciando por uma letra maiúscula). Como a extensão do significado de qualquer nome de variável está contida na cláusula na qual ela aparece, o cálculo de predicados não tem variáveis globais. Outro modo de dizer isso é que os valores das variáveis são passados para outras.

**Figura 3.25** Subgrafo da solução mostrando que Fred está no museu.



cláusulas como parâmetros e não têm locais (memórias) fixos. Assim, as múltiplas ocorrências de X, em regras diferentes nesse exemplo, indicam parâmetros formais *diferentes* (Seção 14.3).

Ao tentar resolver as premissas da regra 7 com essas novas ligações, o resolvedor falhará, porque sam não é um bom\_cachorro. Aqui, a busca retrocederá ao objetivo local(sam,X) e tentará o próximo casamento, a conclusão da regra 8. Isso também falhará, causando outro retrocesso e uma unificação com a conclusão da cláusula 9, local(sam, museu).

Como as premissas da cláusula 9 são apoiadas pelo conjunto de asserções (cláusulas 3 e 4), resulta que a conclusão da 9 é verdadeira. Essa unificação final retrocede completamente na árvore para finalmente responder  $\exists X \text{ local}(\text{fred}, X)$  com local(fred,museu).

É importante examinar cuidadosamente a natureza da busca guiada por objetivo de um grafo e compará-la com a busca guiada por dados do Exemplo 3.3.2. No próximo exemplo, continuamos com a discussão dessa questão, incluindo uma comparação mais rigorosa desses dois métodos de busca em grafo, mas todos os detalhes só serão vistos na discussão dos sistemas de produção no Capítulo 6 e na aplicação a sistemas especialistas na Parte IV. Um outro ponto implícito nesse exemplo é que a ordem das cláusulas afeta a ordem da busca. No exemplo anterior, as múltiplas cláusulas local foram examinadas ordenadamente, e a busca com retrocesso eliminou aquelas que não foram provadas como verdadeiras.

### **Exemplo 3.3.5 Consultor financeiro modificado**

No último exemplo do Capítulo 2, usamos o cálculo de predicados para representar um conjunto de regras para fornecer consultoria sobre investimentos. Naquele exemplo, o *modus ponens* foi usado para inferir um investimento adequado para um indivíduo em particular. Não discutimos o modo como um programa determinaria as inferências apropriadas. Certamente, trata-se de um problema de busca; o exemplo atual ilustra uma abordagem para implementar o consultor financeiro baseado em lógica, usando busca em profundidade, guiada por objetivo com retrocesso. A discussão usa os predicados que estão na Seção 2.4 e que não serão reapresentados aqui.

Suponha que o indivíduo tenha dois dependentes, \$20.000 em poupança e uma renda estável de \$30.000. Como foi discutido no Capítulo 2, podemos adicionar expressões do cálculo de predicados que descrevem esses fatos ao conjunto de expressões do cálculo de predicados. Como alternativa, o programa pode iniciar a busca sem essa informação e pedir ao usuário para acrescentá-la quando for necessário. Essa alternativa tem a vantagem de não exigir dados que poderiam não ser necessários para a solução. Essa abordagem, frequentemente adotada em sistemas especialistas, é ilustrada nesse exemplo.

Ao realizar uma consulta, o objetivo é encontrar um investimento; isso é representado pela expressão do cálculo de predicados  $\exists X \text{ investimento}(X)$ , onde X é a variável objetivo que desejamos ligar. Existem três regras (1, 2 e 3) que concluem acerca de investimentos, porque a consulta irá unificar-se com a conclusão dessas regras. Se selecionarmos a regra 1 para ser explorada primeiro, a sua premissa conta\_poupança(inadequada) se torna o subobjetivo, isto é, o nó filho que será expandido a seguir.

Ao gerar os filhos de conta\_poupança(inadequada), a única regra que pode ser aplicada é a regra 5. Isso produz o nó E:

$$\text{quantia\_poupada}(X) \wedge \text{dependentes}(Y) \wedge \neg \text{maior}(X, \text{poupança\_min}(Y)).$$

Se tentarmos satisfazer essas premissas na ordem da esquerda para a direita, *quantia\_poupada(X)* é tomada como o primeiro subobjetivo. Como o sistema não contém regras que concluem esse subobjetivo, ele irá questionar o usuário. Quando *quantia\_poupada(20000)* é adicionada, o primeiro subobjetivo será satisfeito, com a unificação substituindo 20000 por X. Note que como um nó E está sendo buscado, uma falha aqui eliminaria a necessidade de se examinar o restante da expressão.

De modo semelhante, o subobjetivo *dependentes(Y)* leva a uma consulta, e a resposta, *dependentes(2)*, é adicionada à descrição lógica. O subobjetivo condiz com essa expressão pela substituição {2/Y}. Com essas substituições, a busca avaliará a verdade de:

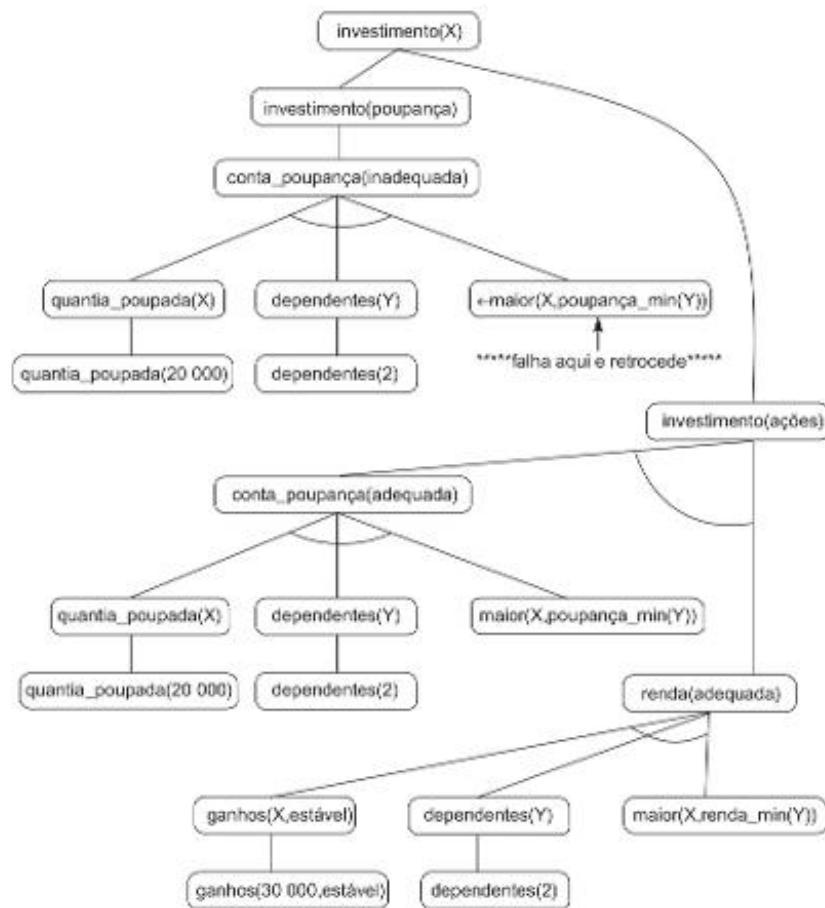
$$\neg \text{maior}(20000, \text{poupança\_min}(2)).$$

O valor resultante será falso, causando a falha do nó E inteiro. A busca, então, retrocede ao nó genitor, `conta_poupança(inadequada)`, e tenta encontrar um modo alternativo para provar que o nó é verdadeiro. Isso corresponde à geração do próximo filho na busca. Como nenhuma outra regra conclui esse subobjetivo, a busca retorna falha para o objetivo do nível mais alto, `investimento(X)`. A próxima regra, cujas conclusões se unificam com esse objetivo, é a regra 2, que produz novos subobjetivos:

`conta_poupança(adequada) ∧ renda(adequada).`

Continuando a busca, prova-se que `conta_poupança(adequada)` é verdadeira como a conclusão da regra 4, e `renda(adequada)` resulta como a conclusão da regra 6. Embora os detalhes do restante da busca sejam deixados para o leitor, o grafo E/OU, que é explorado até o final, aparece na Figura 3.26.

**Figura 3.26** Grafo E/OU buscado pelo consultor financeiro.



### Exemplo 3.3.6 Um analisador sintético da língua portuguesa e gerador de sentenças

O exemplo final não é do cálculo de predicados, mas consiste em um conjunto de regras de reescrita para analisar sentenças em um subconjunto da gramática da língua portuguesa. Regras de reescrita recebem uma expressão e a transformam em outra pela substituição de padrões de um lado da seta ( $\leftrightarrow$ ) por padrões do outro lado da seta. Por exemplo, poderíamos definir um conjunto de regras de reescrita para transformar uma expressão de um idioma, como o português, em outra linguagem (talvez francês ou uma cláusula do cálculo de predicados). As regras de reescrita dadas aqui transformam um subconjunto de sentenças em português em

construções gramaticais de nível mais alto, como frase nominal, frase verbal e sentença. Essas regras são usadas para *analisar* sequências de palavras, isto é, para determinar se elas são sentenças bem formadas (se são gramaticamente corretas ou não) e para modelar a estrutura linguística das sentenças.

Cinco regras para um subconjunto simples da gramática do português são:

1.  $\text{sentença} \leftrightarrow \text{fn fv}$   
(Uma sentença é uma frase nominal seguida de uma frase verbal.)
2.  $\text{fn} \leftrightarrow \text{n}$   
(Uma frase nominal é um nome.)
3.  $\text{fn} \leftrightarrow \text{art n}$   
(Uma frase nominal é um artigo seguido de um nome.)
4.  $\text{fv} \leftrightarrow \text{v}$   
(Uma frase verbal é um verbo.)
5.  $\text{fv} \leftrightarrow \text{v fn}$   
(Uma frase verbal é um verbo seguido de uma frase nominal.)

Além dessas regras de gramática, um analisador sintético necessita de um dicionário de palavras da língua. Essas palavras são chamadas de *terminais* da gramática. Elas são definidas por suas partes do discurso usando regras de reescrita. No dicionário a seguir, “um”, “o”, “homem”, “cão”, “gosta” e “morde” são os terminais da nossa gramática simples:

6.  $\text{art} \leftrightarrow \text{um}$
7.  $\text{art} \leftrightarrow \text{o}$   
 (“um” e “o” são artigos.)
8.  $\text{n} \leftrightarrow \text{homem}$
9.  $\text{n} \leftrightarrow \text{cão}$   
 (“homem” e “cão” são nomes.)
10.  $\text{v} \leftrightarrow \text{gosta}$
11.  $\text{v} \leftrightarrow \text{morde}$   
 (“gosta” e “morde” são verbos.)

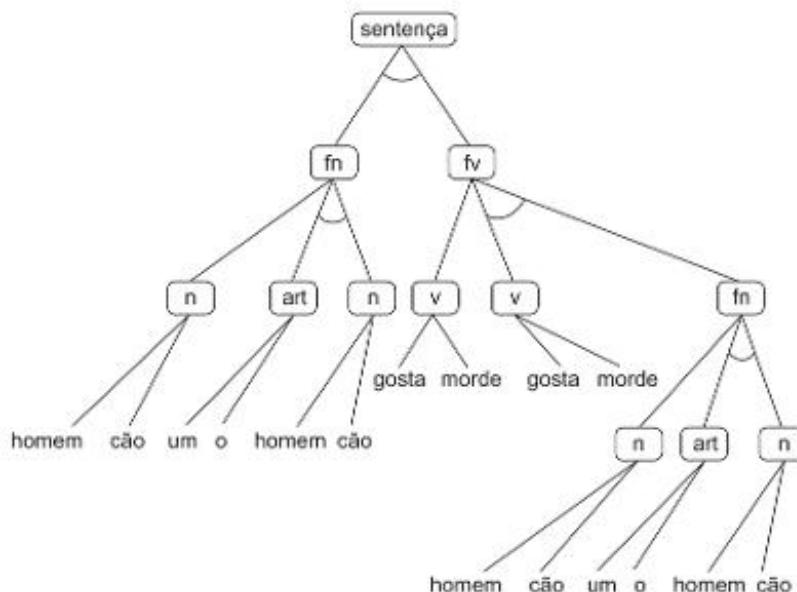
Essas regras de reescrita definem o grafo E/OU da Figura 3.27, onde sentença é a raiz. Os elementos à esquerda de uma regra de reescrita correspondem a nós E do grafo. Múltiplas regras com a mesma conclusão formam os nós OU. Note que os nós folha, ou terminais desse grafo, são as palavras em português da gramática (por isso, eles são chamados de *terminais*).

Uma expressão é *bem formada* em uma gramática se ela consistir inteiramente em símbolos terminais e se existir uma série de substituições usando regras de reescrita que a reduzem ao símbolo sentença. Alternativamente, isso pode ser visto como a construção de uma *árvore de análise* que tem as palavras da expressão como suas folhas e o símbolo sentença como sua raiz.

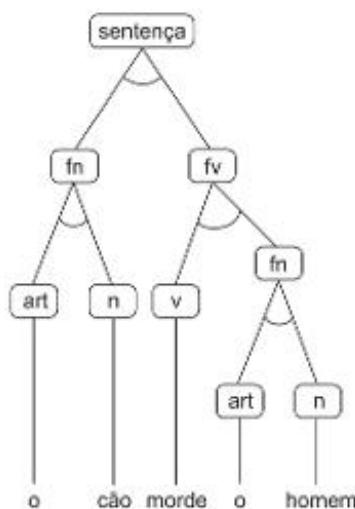
Por exemplo, podemos analisar a sentença *o cão morde o homem* construindo a árvore de análise da Figura 3.28. Essa árvore é uma subárvore do grafo E/OU da Figura 3.27 que é construída por busca nesse grafo. Um algoritmo de *análise sintética guiada por dados* implementaria essa busca correspondendo os lados direitos das regras de reescrita com padrões na sentença, tentando esses casamentos na ordem em que as regras são escritas. Uma vez sendo encontrado um casamento, a parte da expressão que casa com o lado direito da regra é substituída pelo padrão no lado esquerdo. Esse processo continua até que a sentença seja reduzida ao símbolo sentença (indicando uma análise bem-sucedida) ou até que mais nenhuma regra possa ser aplicada (indicando falha). A sequência de análise de *o cão morde o homem* consiste em:

1. A primeira regra que casará é a regra 7, reescrevendo *o* como *art*. Isso produz *art cão morde o homem*.
2. A próxima iteração encontrará um casamento para 7, produzindo *art cão morde art homem*.

**Figura 3.27** Grafo E/OU para a gramática do Exemplo 3.3.6. Alguns dos nós (*fn*, *art* etc.) foram reescritos mais de uma vez para simplificar o grafo.



**Figura 3.28** Árvore de análise para a sentença “O cão morde o homem”. Note que se trata de uma subárvore do grafo da Figura 3.27.



3. A regra 8 disparará, produzindo *art* *cão* *morde* *art* *n*.
4. A regra 3 disparará produzindo *art* *cão* *morde* *fn*.
5. A regra 9 produz *art* *n* *morde* *fn*.
6. A regra 3 pode ser aplicada, gerando *fn* *morde* *fn*.
7. A regra 11 produz *fn* *v* *fn*.
8. A regra 5 produz *fn* *fv*.
9. A regra 1 produz sentença, aceitando a sentença como correta.

O exemplo anterior implementa uma análise sintética em profundidade guiada por dados, pois sempre aplica a regra de nível mais alto à expressão; por exemplo, art n se reduz a fn antes que morde se reduza a v. A análise também poderia ser feita de uma forma guiada por objetivo, tomando sentença como a sequência de partida e determinando uma série de substituições de padrões que combinam o lado esquerdo das regras, levando a uma série de terminais que combinam a sentença alvo.

A análise sintética é importante não apenas para a linguagem natural (Capítulo 15), mas também para construir compiladores e interpretadores para linguagens de computador (Aho e Ullman, 1977). A literatura está repleta de algoritmos de análise sintética para todas as classes de linguagem. Por exemplo, muitos algoritmos de análise sintética guiados por objetivo antecipam o fluxo de entrada para determinar qual regra será aplicada em seguida.

Nesse exemplo, usamos uma técnica muito simples de busca no grafo E/OU em um padrão não informado. Algo interessante nesse exemplo é a implementação da busca. Essa técnica de manter um registro da expressão atual e tentar combinar as regras na ordem é um exemplo de uso de um *sistema de produção* para implementar a busca. Esse é um dos tópicos principais do Capítulo 6.

Regras de reescrita também são usadas para gerar sentenças válidas de acordo com as especificações da gramática. As sentenças podem ser geradas por uma busca guiada por objetivo, começando com sentença como objetivo de alto nível e terminando quando não puderem ser aplicadas mais regras. Isso produz uma sequência de símbolos terminais que é uma sentença válida na gramática. Por exemplo:

Uma sentença é uma fn seguida por uma fv (regra 1).

fn é substituída por n (regra 2), produzindo n fv.

homem é o primeiro n disponível (regra 8), produzindo homem fv.

Agora, fn é satisfeita e fv é tentada. A regra 3 substitui fv por v, homem v.

A regra 10 substitui v por gosta.

homem gosta é encontrado como primeira sentença aceitável.

Se for desejado criar todas as sentenças aceitáveis, essa busca pode ser repetida sistematicamente até que todas as possibilidades sejam experimentadas e o espaço de estados inteiro tenha sido buscado exaustivamente. Isso gera sentenças incluindo um homem gosta, o homem gosta, e assim por diante. Existem cerca de 80 sentenças corretas que são produzidas por uma busca exaustiva. Estas incluem anomalias semânticas como o homem morde o cachorro.

A análise sintética e a geração podem ser usadas juntas de diversas maneiras para lidar com diferentes problemas. Por exemplo, para achar todas as sentenças para completar a sequência “o homem”, o resolvedor de problemas pode receber uma sequência incompleta o homem... . Ele pode trabalhar de baixo para cima, em um padrão guiado por dados, para produzir o objetivo de completar a regra da sentença (regra 1), onde fn é substituída por homem, e depois trabalhar em um padrão guiado por objetivo, para determinar todas as fvs possíveis que completarão a sentença. Isso criaria sentenças como o homem gosta, o homem morde o homem, e assim por diante. Novamente, esse exemplo lida apenas com a correção sintática. A questão da semântica (se a sequência tem um mapeamento em algum “mundo” com “verdade”) é totalmente diferente. O Capítulo 2 examinou a questão de construção de uma semântica baseada em lógica para expressões; para verificar sentenças na linguagem natural, a questão é muito mais difícil, e será discutida no Capítulo 15.

No próximo capítulo, discutiremos o uso de heurística para focalizar a busca na “melhor” parte possível do espaço de estados. O Capítulo 6 discute os sistemas de produção e outras “arquiteturas” de software para controlar a busca no espaço de estados.

## 3.4 Epílogo e referências

O Capítulo 3 introduziu as bases teóricas da busca no espaço de estados, usando a teoria dos grafos para analisar a estrutura e a complexidade das estratégias de solução de problemas. O capítulo comparou o raciocínio

guiado por dados e por objetivo, além da busca em profundidade e em amplitude. Grafos E/OU nos permitem aplicar a busca no espaço de estados à implementação do raciocínio baseado em lógica.

A busca básica em grafo é discutida em diversos livros sobre algoritmos de computação. Entre eles estão *Introduction to Algorithms*, de Thomas Cormen, Charles Leiserson e Ronald Rivest (1990), *Walls and Mirrors*, de Paul Helman e Robert Veroff (1986), *Algorithms*, de Robert Sedgewick (1983), e *Fundamentals of Computer Algorithms*, de Ellis Horowitz e Sartaj Sahni (1978). Autômatos finitos são apresentados em Lindenmayer e Rosenberg (1976). Outros algoritmos para busca E/OU são apresentados no Capítulo 14, “Raciocínio automatizado”.

O uso da busca em grafos para modelar a solução inteligente do problema é apresentado em *Human Problem Solving*, de Alan Newell e Herbert Simon (1972). Entre os textos sobre inteligência artificial que discutem estratégias de busca estão *Artificial Intelligence* (Nils Nilsson, 1998), *Artificial Intelligence* (Patrick Winston, 1992) e *Artificial Intelligence* (Eugene Charniak e Drew McDermott, 1985). *Heuristics* (Judea Pearl, 1984) apresenta algoritmos de busca e estabelece uma base para o material que apresentaremos no Capítulo 4. O desenvolvimento de novas técnicas para a busca em grafos é um tópico constante em conferências anuais sobre IA.

## 3.5 Exercícios

1. Um caminho hamiltoniano é um caminho que usa cada nó do grafo exatamente uma vez. Que condições são necessárias para que haja tal caminho? Existe um caminho assim no mapa de Königsberg?

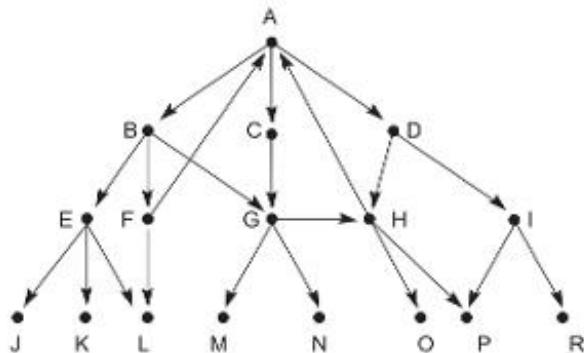
2. Dê a representação em grafo para o problema do fazendeiro, do lobo, da cabra e do repolho:

Um fazendeiro, com seu lobo, sua cabra e seu repolho, chega à margem de um rio que deseja atravessar. Há um barco na margem do rio, mas, naturalmente, somente o fazendeiro pode remar. O barco também só pode transportar duas coisas (incluindo o remador) de cada vez. Se o lobo ficar sozinho com a cabra, ele comerá a cabra; da mesma forma, se a cabra ficar sozinha com o repolho, a cabra comerá o repolho. Crie uma sequência de travessias pelo rio de modo que os quatro personagens cheguem em segurança ao outro lado do rio.

Faça com que os nós representem os estados do mundo; por exemplo, o fazendeiro e a cabra estão na margem oeste e o lobo e o repolho, na margem leste. Discuta as vantagens da busca em amplitude e em profundidade para este problema.

3. Crie um reconhecedor de estados finitos que reconheça todas as sequências de dígitos binários (a) que contêm “111”, (b) que terminam com “111”, (c) que contêm “111” mas não mais do que três “1’s consecutivos.
4. Dê um exemplo do problema do caixeiro-viajante para o qual a estratégia do vizinho mais próximo não consiga achar um caminho ideal. Sugira outra heurística para esse problema.

**Figura 3.29** Um grafo de busca.



5. “Percorra manualmente” o algoritmo de busca com retrocesso no grafo da Figura 3.29. Comece do estado A. Registre os valores sucessivos de LNE, LE, EC etc.
6. Implemente um algoritmo de busca com retrocesso em uma linguagem de programação à sua escolha.
7. Determine se a busca guiada por objetivo ou por dados seria preferível para resolver cada um dos problemas a seguir. Justifique sua resposta.
  - a. Diagnóstico de problemas mecânicos em um automóvel.
  - b. Você encontrou uma pessoa que afirma ser sua prima distante, com um ancestral comum chamado João da Silva. Você gostaria de verificar a afirmação dela.
  - c. Outra pessoa afirma ser seu primo distante. Ele não sabe o nome do ancestral comum, mas sabe que não está distante por mais que oito gerações. Você gostaria de achar esse ancestral ou determinar que ele não existe.
  - d. Um provador de teorema para a geometria plana.
  - e. Um programa para examinar leituras de sonar e interpretá-las, diferenciando, por exemplo, um grande submarino de um pequeno submarino, de uma baleia ou de um cardume.
  - f. Um sistema especialista que, com auxílio de uma pessoa, classifique plantas por espécie, gênero etc.
8. Escolha e justifique a escolha entre busca em amplitude ou em profundidade para os exemplos do Exercício 6.
9. Escreva um algoritmo de busca com retrocesso para grafos E/OU.
10. Solucione o problema guiado por objetivo do *bom cachorro* do Exemplo 3.3.4 de uma maneira guiada por dados.
11. Dê um outro exemplo de um problema de busca em grafo E/OU e desenvolva parte do espaço de busca.
12. Acompanhe uma execução guiada por dados do *consultor financeiro* do Exemplo 3.3.5 para o caso de um indivíduo com quatro dependentes, \$18.000 no banco e uma renda estável de \$25.000 por ano. Comparando esse problema com o exemplo do texto, sugira uma estratégia, em geral, “melhor” para resolver esse problema.
13. Acrescente regras para adjetivos e advérbios à *gramática da língua portuguesa* do Exemplo 3.3.6.
14. Acrescente regras para (múltiplas) frases preposicionais à *gramática da língua portuguesa* do Exemplo 3.3.6.
15. Acrescente regras gramaticais ao Exemplo 3.3.6 que permitam sentenças complexas como:

sentença  $\leftrightarrow$  sentença E sentença.

# Busca heurística

*A tarefa que um sistema simbólico enfrenta, quando a ele é apresentado um problema e o espaço do problema, é como usar os seus limitados recursos de processamento para gerar possíveis soluções, uma após a outra, até encontrar uma que satisfaça o teste que define o problema. Se o sistema simbólico tiver algum controle sobre a ordem em que as potenciais soluções são geradas, então é desejável dispor esta ordem de modo que as soluções reais tenham uma maior probabilidade de surgirem antes que as demais. Um sistema simbólico exibiria inteligência se conseguisse fazer isso. Inteligência para um sistema com recursos limitados de processamento consiste em fazer escolhas sábias sobre o que será feito a seguir...*

—NEWELL E SIMON, 1976, *Turing Award Lecture*

*Eu estou procurando...*

*Procurando... Oh! sim,*

*Procurando todas as possibilidades...*

—LIEBER E STOLLER

## 4.0 Introdução

George Polya define *heurística* como “o estudo dos métodos e das regras de descoberta e invenção” (Polya, 1945). Essa interpretação está relacionada com o termo grego original, o verbo *eurisco*, que significa “eu descubro”. Quando Arquimedes emergiu de seu famoso banho segurando a coroa de ouro, ele gritou “Eureka!”, querendo dizer “Eu descobri!”. Na busca em espaço de estados, *heurísticas* são formalizadas como regras para escolher aqueles ramos em um espaço de estados que têm maior probabilidade de levarem a uma solução aceitável para o problema.

Os resolvidores de problema de IA empregam heurísticas basicamente em duas situações:

1. Um problema pode não ter uma solução exata por causa das ambiguidades inerentes na formulação do problema ou nos dados disponíveis. O diagnóstico médico é um exemplo disso. Determinado conjunto de sintomas pode ter várias causas possíveis; os médicos usam heurísticas para escolher o diagnóstico mais provável e para formular um plano de tratamento. A visão é outro exemplo de um problema inerentemente inexato. Cenas visuais frequentemente são ambíguas, permitindo múltiplas interpretações da conectividade, da extensão e da orientação de objetos. As ilusões de ótica são exemplos dessas ambiguidades. Sistemas de visão tipicamente usam heurísticas para selecionar a mais provável entre várias possíveis interpretações de determinada cena.
2. Um problema pode ter uma solução exata, mas o custo computacional de encontrá-la pode ser proibitivo. Em muitos problemas (como no jogo de xadrez), o crescimento do espaço de estados é combinatorialmen-

te explosivo, com o número de estados possíveis aumentando exponencialmente, ou fatorialmente, com a profundidade da busca. Nesses casos, técnicas de busca por *força bruta*, exaustivas, como a busca em profundidade ou em amplitude, podem não conseguir encontrar uma solução dentro de um intervalo de tempo prático. As heurísticas enfrentam essa complexidade guiando a busca através do espaço ao longo do caminho mais “promissor”. Desconsiderando os estados menos promissores e seus descendentes, um algoritmo heurístico pode (o seu projetista espera) enfrentar essa *explosão combinatória* e encontrar uma solução aceitável.

Infelizmente, como todas as regras de descoberta e de invenção, as heurísticas podem falhar. Uma heurística é apenas uma conjectura informada sobre o próximo passo a ser dado na solução de um problema. Frequentemente, ela é baseada na experiência e na intuição. Como as heurísticas usam informação limitada, como conhecimento sobre a situação atual ou as descrições dos estados que estão nesse momento na lista abertos, elas nem sempre podem prever o comportamento exato do espaço de estados mais adiante na busca. Uma heurística pode levar um algoritmo de busca a uma solução subótima ou, inclusive, levá-lo a não conseguir encontrar uma solução. Essa é uma limitação inerente da busca heurística. Essa limitação não pode ser eliminada por heurísticas “melhores” ou por algoritmos de busca mais eficientes (Garey e Johnson, 1979).

As heurísticas e o projeto de algoritmos para implementar a busca heurística têm sido, há muito tempo, uma preocupação central das pesquisas em inteligência artificial. Os programas de jogos e os provadores de teorema são duas das mais antigas aplicações de inteligência artificial; ambos requerem heurísticas para podar os espaços de soluções possíveis. Não é viável examinar toda a inferência que pode ser feita em um domínio matemático ou todos os movimentos possíveis que podem ser feitos em um tabuleiro. A busca heurística é, frequentemente, a única resposta prática.

A pesquisa em sistemas especialistas tem confirmado a importância de heurísticas como um componente essencial da solução de problemas. Quando um especialista humano resolve um problema, ele examina a informação disponível e toma uma decisão. As “regras práticas” que um especialista humano usa para resolver problemas de forma eficiente são basicamente heurísticas quanto à sua natureza. Essas heurísticas são extraídas e formalizadas por projetistas de sistemas especialistas, conforme veremos no Capítulo 8.

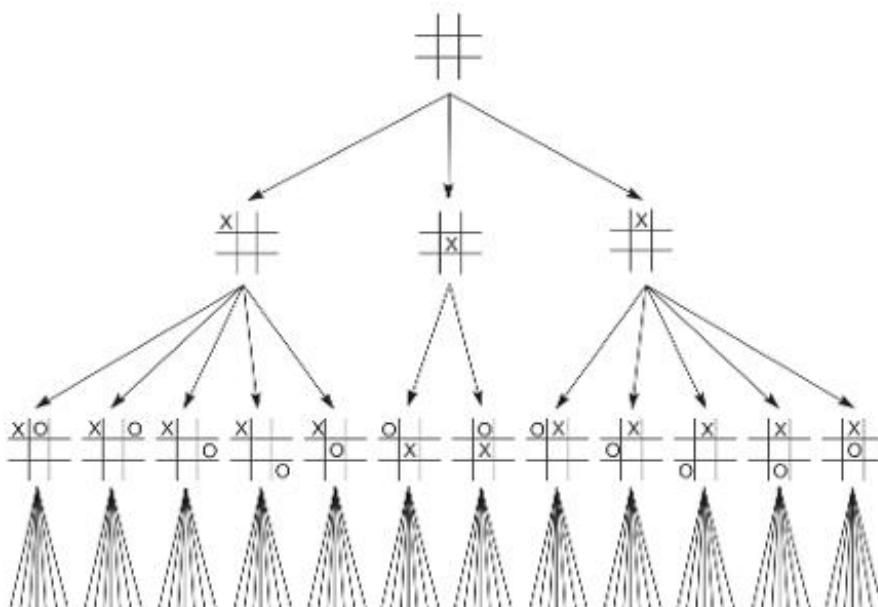
É útil considerar a busca heurística sob duas perspectivas: a medida heurística e um algoritmo que usa heurísticas para buscar o espaço de estados. Na Seção 4.1, implementamos heurísticas com algoritmos de *subida de encosta e programação dinâmica*. Na Seção 4.2, apresentamos um algoritmo para busca pela *melhor escolha*. O projeto e a avaliação da eficácia de heurísticas são apresentados na Seção 4.3, e as heurísticas de jogos, na Seção 4.4.

Considere as heurísticas no jogo da velha, na Figura II.5. As combinações para a busca exaustiva são grandes, mas não intransponíveis. Cada um dos nove primeiros movimentos tem oito respostas possíveis, que, por sua vez, têm sete movimentos possíveis na sequência, e assim por diante. Uma análise simples mostra que o número de estados que necessitam ser considerados em uma busca exaustiva é de  $9 \times 8 \times 7 \times \dots \times 1$ .

O espaço de estados pode ser reduzido por simetria. Muitas configurações do problema são, na realidade, equivalentes em relação a operações simétricas do tabuleiro. Assim, não existem nove movimentos iniciais, mas apenas três: para um canto, para o centro de um lado e para o centro da grade. As reduções por simetria no segundo nível reduzem ainda mais o número de caminhos através do espaço para  $12 \times 7!$ , como pode ser visto na Figura 4.1. Simetrias como essa, em um espaço de jogo, podem ser descritas como invariantes matemáticas, que, quando existirem, podem ser usadas para a redução drástica da busca.

Entretanto, uma heurística simples pode quase eliminar inteiramente a busca: podemos nos mover para a configuração na qual X tem mais oportunidades de vitória. (Os primeiros três estados do jogo da velha são avaliados dessa forma na Figura 4.2.) No caso de estados com números iguais de vitórias potenciais, escolha o primeiro desses estados que for encontrado. O algoritmo realiza, então, a seleção e faz um movimento para o estado com maior número de oportunidades. Nesse caso, X toma o centro da grade. Note que não apenas foram eliminadas as demais alternativas, mas também o foram todos os seus descendentes. Dois terços do espaço são podados com o primeiro movimento (Figura 4.3).

**Figura 4.1** Os primeiros três níveis do espaço de estados do jogo da velha reduzidos por simetria.



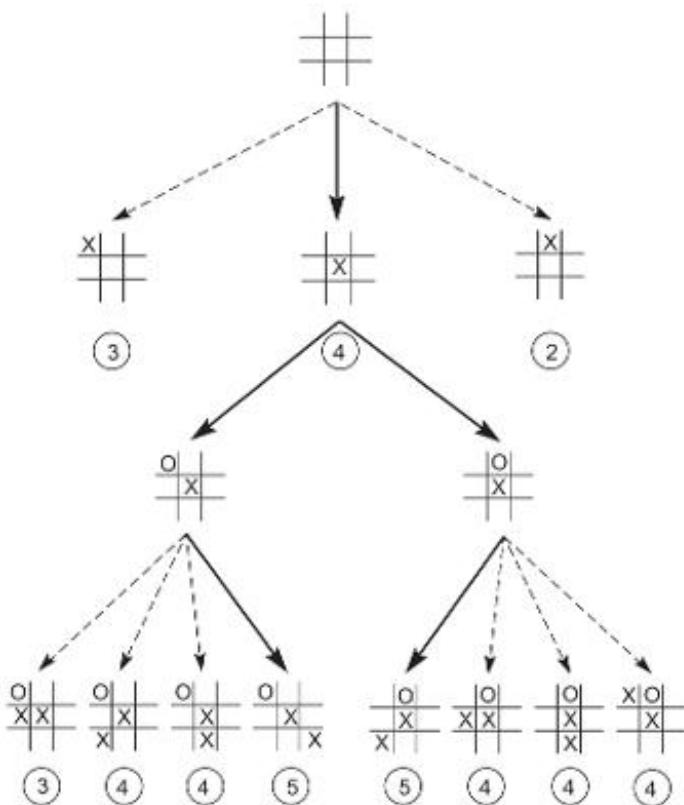
**Figura 4.2** A heurística do “mais vitórias” aplicada aos primeiros filhos do jogo da velha.



Após o primeiro movimento, o oponente pode escolher entre dois movimentos alternativos (como visto na Figura 4.3). Independentemente de qual movimento seja escolhido, podemos aplicar a heurística ao estado resultante do jogo novamente usando a heurística das “mais vitórias” para escolher entre os movimentos possíveis. Conforme a busca continua, cada movimento avalia os filhos de um único nó; uma busca exaustiva não é necessária. A Figura 4.3 mostra a busca reduzida após três passos do jogo. Os estados estão marcados com os seus valores heurísticos. Embora seja difícil calcular o número exato de estados que devem ser examinados com a estratégia de “mais vitórias” para o jogo da velha, pode-se calcular um limite superior grosso, supondo-se um número máximo de cinco movimentos em um jogo com cinco opções por movimento. Na realidade, o número de estados é menor, pois, conforme o tabuleiro é preenchido, o número de opções é reduzido. Esse limite grosso de 25 estados significa uma melhora de quatro ordens de grandeza sobre  $9!$ .

A próxima seção apresenta dois algoritmos de implementação de busca heurística: *subida de encosta* e *programação dinâmica*. A Seção 4.2 usa a fila de prioridades para a *busca pela melhor escolha*. Na Seção 4.3, discu-

Figura 4.3 Espaço de estados reduzido heurísticamente para o jogo da velha.



timos algumas questões teóricas relacionadas à busca heurística, como a *admissibilidade* e a *monotonicidade*. A Seção 4.4 examina o uso de *minimax* e *poda alfa-beta* para aplicar heurísticas a jogos com dois jogadores. A seção final deste capítulo examina a complexidade da busca heurística e enfatiza o seu papel essencial na solução inteligente de problemas.

## 4.1 Subida de encosta e programação dinâmica

### 4.1.1 Subida de encosta

A maneira mais simples de se implementar a busca heurística é por meio de um procedimento chamado de *subida de encosta* (Pearl, 1984). As estratégias de subida de encosta expandem o estado atual da busca e avaliam os seus filhos. O “melhor” filho é selecionado para uma expansão futura; nem os seus irmãos nem os seus genitores são considerados. A subida de encosta é o nome dado para a estratégia que poderia ser usada por um alpinista impetuoso, mas cego: suba pelo caminho mais íngreme possível até você não poder mais avançar. Como o algoritmo não registra o histórico do processo de subida, ele não consegue se recuperar de falhas de sua estratégia. Um exemplo de subida de encosta no jogo da velha foi o “tome o estado com o maior número de vitórias”, que demonstramos na Seção 4.0.

Um problema importante das estratégias de subida de encosta é sua tendência de ficar preso nos *máximos locais*. Se elas alcançarem um estado que tenha uma avaliação melhor que qualquer um de seus filhos, o algoritmo fracassa. Se esse estado não for um objetivo, mas apenas um máximo local, o algoritmo pode não conseguir achar

a melhor solução, ou seja, o desempenho poderia melhorar em uma configuração limitada, mas, devido à forma do espaço inteiro, ele pode nunca alcançar o melhor de todos. Um exemplo de máximos locais em jogos ocorre no quebra-cabeça dos 8. Frequentemente, para mover uma peça específica ao seu destino, outras peças já na posição do objetivo precisam ser afastadas. Isso é necessário para resolver o quebra-cabeças, mas temporariamente piora o estado do tabuleiro. Como “melhor” não precisa ser “o melhor” em um sentido absoluto, os métodos de busca sem retrocesso ou algum outro mecanismo de recuperação são incapazes de distinguir entre máximos locais e globais.

A Figura 4.4 é um exemplo do dilema do máximo local. Suponha, explorando esse espaço de busca, que chegamos ao estado X, desejando maximizar os valores de estado. As avaliações dos filhos, netos e bisnetos de X demonstram que a subida de encosta pode se confundir até mesmo com a antecipação de vários níveis. Existem métodos para contornar esse problema, como perturbar aleatoriamente a função de avaliação, mas em geral não há um modo de garantir o desempenho ideal com as técnicas de subida de encosta. O programa de damas de Samuel (1959) oferece uma variante interessante do algoritmo de subida de encosta.

O programa de Samuel era excepcional para a sua época, 1959, principalmente dadas as limitações dos computadores da década de 1950. O programa de Samuel não apenas aplicava a busca heurística ao jogo de damas, mas também implementava algoritmos para o uso ideal da memória limitada, bem como uma forma simples de aprendizado. De fato, ele foi pioneiro em muitas das técnicas ainda usadas em programas de jogos e aprendizado de máquina.

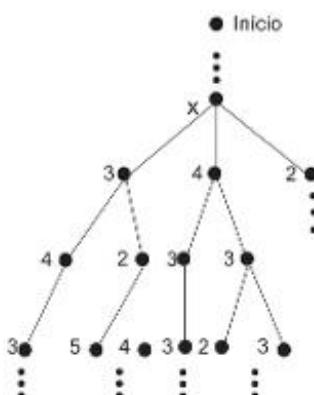
O programa de Samuel avaliava estados do tabuleiro com uma soma ponderada de várias medidas heurísticas diferentes:

$$\sum_i a_i x_i$$

O  $x_i$  nessa soma representava características do tabuleiro como vantagem de peças, local das peças, controle do centro do tabuleiro, oportunidades de sacrificar peças para obter vantagem e até mesmo um cálculo de momentos de inércia das peças de um jogador sobre um eixo do tabuleiro. Os coeficientes  $a_i$  desses  $x_i$  eram pesos ajustados especialmente que tentavam modelar a importância desse fator na avaliação geral do tabuleiro. Assim, se a vantagem em peças fosse mais importante que o controle do centro, o coeficiente de vantagem em peças refletiria isso.

O programa de Samuel antecipava o número desejado de níveis ou *camadas* do espaço de busca (normalmente imposto pelas limitações de espaço e/ou tempo do computador) e avaliava todos os estados naquele nível usando o polinômio de avaliação. Usando uma variação de *minimax* (Seção 4.3), ele propagava esses valores de volta, para cima no grafo. O jogador de damas moveria, então, para o melhor estado; após o movimento do adversário, o processo seria repetido para o novo estado do tabuleiro.

**Figura 4.4** O problema do máximo local para a subida de encosta com antecipação de 3 níveis.



Se o polinômio de avaliação levasse a uma série perdedora de movimentos, o programa ajustaria seus coeficientes em uma tentativa de melhorar o seu desempenho. Às avaliações com grandes coeficientes se atribuía a maior culpa pelas perdas e elas tinham o seu peso diminuído, enquanto os pesos menores eram aumentados para dar mais influência a essas avaliações. Se o programa ganhasse, era feito o oposto. O programa era treinado por meio de jogos contra um adversário humano ou contra outra versão dele mesmo.

O programa de Samuel adotava uma abordagem de aprendizado do tipo subida de encosta, tentando melhorar seu desempenho por meio de melhorias locais no polinômio de avaliação. O jogador de damas de Samuel foi capaz de melhorar seu desempenho conseguindo jogar damas muito bem. Samuel tratou algumas das limitações da subida de encosta verificando a eficiência das medidas heurísticas ponderadas individualmente e substituindo a menos efetiva. O programa apresentava, também, certas limitações interessantes. Por exemplo, por causa do uso de uma estratégia global limitada, ele era vulnerável a funções de avaliação que levassem a mínimos locais. O componente de aprendizado do programa era também vulnerável a inconsistências no jogo do adversário; por exemplo, se o adversário usasse estratégias muito variadas, ou simplesmente jogasse de forma inconsequente, os pesos no polinômio de avaliação começariam a assumir valores “aleatórios”, levando à degradação global do desempenho.

#### 4.1.2 Programação dinâmica

A programação dinâmica (PD) às vezes é chamada de *forward-backward* (*para a frente, para trás*) ou, quando usa probabilidades, algoritmo de *Viterbi*. Criada por Richard Bellman (1956), a programação dinâmica enfrenta a questão da busca com memória restrita em problemas compostos de vários subproblemas interagentes e inter-relacionados. A PD registra e reutiliza subproblemas já pesquisados e resolvidos dentro da solução do problema maior. Um exemplo disso seria o reúso das soluções de subséries dentro da solução das séries de Fibonacci. A técnica de *caching* de subproblema para o reúso às vezes é chamada de *memorização* de soluções de subobjetivo parcial. O resultado é um algoritmo importante, normalmente usado para casamento de sequências, verificação ortográfica e áreas relacionadas no processamento de linguagem natural (ver seções 9.4 e 14.4).

Demonstramos a programação dinâmica usando dois exemplos, ambos de processamento de textos; primeiro, determinando um alinhamento global ideal de duas sequências de caracteres e, segundo, determinando a diferença de edição mínima entre duas sequências de caracteres. Suponha que quiséssemos determinar o melhor alinhamento possível para os caracteres nas sequências BAADDCAABDDA e BBADCBA. Um alinhamento ótimo, entre vários possíveis, seria:

BAAD DCA BDD A  
BBA DC B A

A programação dinâmica requer uma estrutura de dados para registrar os subproblemas relacionados ao estado atualmente sendo processado. Usamos um arranjo. O tamanho das dimensões desse arranjo, devido aos requisitos de inicialização, é um a mais que o tamanho de cada sequência, em nosso exemplo, 12 por 8, como na Figura 4.5. O valor de cada elemento de linha-coluna do arranjo reflete o sucesso do alinhamento global até esse ponto no

**Figura 4.5** O estágio de inicialização e o primeiro passo no preenchimento do arranjo para o alinhamento de caracteres usando a programação dinâmica.

processo de casamento. Existem três custos possíveis para a criação do estado atual: se um caractere for deslocado ao longo da sequência mais curta para o melhor alinhamento possível, o custo é 1 e é registrado pela pontuação de *coluna* do arranjo. Se um novo caractere for inserido, o custo é 1 e é refletido na pontuação de *linha* do arranjo. Se os caracteres a serem alinhados forem diferentes, deslocamento e inserção, o custo é 2; ou então, se forem idênticos, o custo é 0; isso é refletido na *diagonal* do arranjo. A Figura 4.5 mostra a inicialização, o aumento de +1s na primeira linha e coluna reflete o deslocamento continuado ou a inserção de caracteres na sequência “\_” ou vazia.

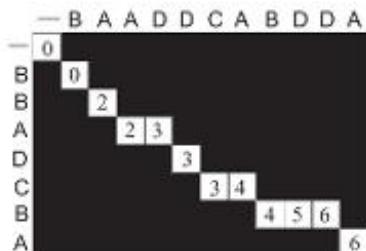
No estágio *forward* do algoritmo de programação dinâmica, preenchemos o arranjo a partir do canto superior esquerdo considerando os sucessos de combinação parciais até o ponto atual da solução. Ou seja, o valor da interseção da linha  $x$  e da coluna  $y$ ,  $(x, y)$ , é uma função (para o problema do alinhamento mínimo, o custo mínimo) de um dos três valores na linha  $x - 1$  coluna  $y$ , linha  $x - 1$  coluna  $y - 1$ , ou linha  $x$  coluna  $y - 1$ . Essas três posições de arranjo mantêm a informação de alinhamento *até* o ponto atual da solução. Se houver um casamento de caracteres na posição  $(x, y)$ , some 0 ao valor na posição  $(x - 1, y - 1)$ , se não houve casamento some 2 (para deslocamento e inserção). Somamos 1 deslocando a sequência de caracteres mais curta (some ao valor anterior da coluna  $y$ ) ou inserindo um caractere (some ao valor anterior da linha  $x$ ). A continuação dessa técnica produz o arranjo preenchido da Figura 4.6. Pode-se observar que a combinação com custo mínimo normalmente ocorre perto da diagonal superior esquerda para a inferior direita do arranjo.

Quando o arranjo estiver preenchido, começamos o estágio *backward* do algoritmo que produz soluções particulares. Ou seja, a partir do melhor alinhamento possível, voltamos pelo arranjo para selecionar uma solução de alinhamento específica. Começamos esse processo no valor linha-coluna do máximo, em nosso exemplo, o 6 na linha 7, coluna 12. De lá, voltamos pelo arranjo, em cada etapa selecionando um dos predecessores imediatos do estado que produziu o estado atual (no estágio *forward*), ou seja, um entre diagonal, linha ou coluna que produziu esse estado. Sempre que há uma diferença de diminuição forçada, como no 6 e no 5 perto do início do rastreamento para trás, selecionamos a diagonal anterior, pois foi daí que veio a combinação; caso contrário, usamos o valor da linha ou coluna anterior. O rastreamento de volta da Figura 4.7, um de vários possíveis, produz o alinhamento ótimo de sequências da página anterior.

**Figura 4.6** O arranjo concluído refletindo a informação de alinhamento máximo para as sequências.

	—	B	A	A	D	D	C	A	B	D	D	A
	0	1	2	3	4	5	6	7	8	9	10	11
B	1	0	1	2	3	4	5	6	7	8	9	10
B	2	1	2	3	4	5	6	7	6	7	8	9
A	3	2	1	2	3	4	5	6	7	8	9	8
D	4	3	2	3	2	3	4	5	6	7	8	9
C	5	4	3	4	3	4	3	4	5	6	7	8
B	6	5	6	5	4	5	4	5	4	5	6	7
A	7	6	5	4	5	6	5	4	5	6	7	6

**Figura 4.7** Um componente *backward* concluído para o exemplo de programação dinâmica dando um alinhamento de sequências (dentre vários possíveis).



No segundo exemplo de uso da PD, consideramos a ideia da *diferença de edição mínima* entre duas sequências. Se estivéssemos criando, por exemplo, um verificador ortográfico inteligente poderíamos querer determinar quais palavras do nosso conjunto de palavras soletradas corretamente mais se aproximam de certa sequência de caracteres não reconhecida. Uma técnica semelhante poderia ser usada para determinar quais palavras conhecidas (sequências de telefones representando palavras) combinam mais de perto com determinada sequência de telefones. O próximo exemplo de estabelecimento da *distância de edição* entre duas sequências de caracteres é adaptado de Jurafsky e Martin (2009).

Suponha que você produza uma sequência de caracteres não reconhecida. A tarefa do corretor ortográfico é produzir uma lista ordenada das palavras mais prováveis, a partir do dicionário, que você deseja digitar. A questão é, então, como uma diferença pode ser medida entre pares de sequências de caracteres, ou seja, a sequência que você digitou e as sequências de caracteres de um dicionário. Para a sua sequência, queremos produzir uma lista ordenada de possíveis palavras corretas no dicionário. Para cada uma dessas palavras, queremos uma medida numérica de como cada palavra é “diferente” da sua sequência.

Uma diferença de edição mínima entre duas sequências pode ser especificada como o número de inserções, exclusões e substituições de caracteres necessário para transformar a primeira sequência, a origem, na segunda sequência, o destino. Isso também é chamado de *distância de Levenshtein* (Jurafsky e Martin, 2009). Agora, implementamos uma busca de programação dinâmica para determinar a diferença de edição mínima entre duas sequências de caracteres. Consideramos que intenção seja a sequência de origem e execução seja o destino. O custo de edição para transformar a primeira sequência na segunda é 1 para uma inserção ou exclusão de caractere e 2 para uma substituição (uma exclusão mais uma inserção). Queremos determinar uma diferença de custo mínima entre essas duas sequências.

Nosso arranjo de subobjetivos novamente será um caractere a mais que cada uma das sequências; nesse caso, 10 por 10. A inicialização é como mostra a Figura 4.8 (uma sequência de inserções é necessária, começando na sequência nula para fazer com que qualquer sequência seja semelhante à outra). A posição (2, 2) do arranjo é 2, pois uma substituição (ou uma exclusão mais uma inserção) é necessária para transformar um i em um e.

A Figura 4.9 dá o resultado completo da aplicação do algoritmo de programação dinâmica para transformar intenção em execução. O valor em cada posição  $(x, y)$  no arranjo é o custo da edição mínima para esse ponto mais o custo (mínimo) de uma inserção, uma exclusão ou uma substituição. Assim, o custo de  $(x, y)$  é o mínimo do custo de  $(x - 1, y)$  mais o custo de inserção, ou o custo de  $(x - 1, y - 1)$  mais o custo de substituição, ou o custo de  $(x, y - 1)$  mais o custo de exclusão. O pseudocódigo para o seu algoritmo é uma função que recebe duas sequências (uma de origem e uma de destino) e seus tamanhos, retornando o custo da diferença de edição mínima:

**Figura 4.8** Inicialização da matriz de diferença de edição mínima entre *intenção* e *execução* (adaptado de Jurafsky e Martin, 2000).

**Figura 4.9** Arranjo completo de diferença de edição mínima entre intenção e execução (adaptado de Jurafsky e Martin, 2000).

	e	x	e	c	u	ç	ã	o	
—	0	1	2	3	4	5	6	7	8
i	1	2	3	4	5	6	7	8	9
n	2	3	4	5	6	7	8	9	10
t	3	4	5	6	7	8	9	10	11
e	4	5	6	5	6	7	8	9	10
n	5	6	7	6	7	8	9	10	11
c	6	7	8	7	8	9	8	9	10
ã	7	8	9	8	9	10	9	8	9
o	8	9	10	9	10	11	10	9	8

função dinâmica(origem, sl, destino, tl) retorna custo (i, j);

```

crie arranjo custo(sl + 1, tl + 1)
custo (0,0) := 0                                     % inicializa
para i := 1 até sl + 1 faça
    para j := 1 até tl + 1 faça
        custo(i, j) := min [ custo (i - 1, j) + custo de inserção destinoi-1           % soma 1
                            custo (i - 1, j - 1) + custo de substituição origemj-1 destinoi-1       % soma 2
                            custo(i, j - 1) + custo de exclusão origemj-1]                      % soma 1

```

Usando os resultados (em negrito) na Figura 4.9, as seguintes edições traduzirão intenção para execução com um custo de edição total de 8.

intenção	
ntenção	exclui i, custo 1
etenção	substitui n por e, custo 2
exenção	substitui t por x, custo 2
exenução	insere u, custo 1
execução	substitui n por c, custo 2

Na situação do corretor ortográfico de propor uma lista ordenada por custo das palavras para substituição de uma sequência não reconhecida, o segmento para trás do algoritmo de programação dinâmica não é necessário. Quando a medição de edição mínima é calculada para o conjunto de sequências relacionadas, é proposta uma ordem de alternativas priorizada, da qual o usuário escolhe uma sequência apropriada.

A justificativa para a programação dinâmica é o custo do tempo/espacão na computação. A programação dinâmica, vista em nossos exemplos, tem custo  $n^2$ , onde n é o comprimento da maior sequência; o custo no pior dos casos é  $n^3$ , se outros subproblemas relacionados precisarem ser considerados (outros valores de linha/coluna) para determinar o estado atual. A busca exaustiva para comparar duas sequências é exponencial, custando entre  $2^n$  e  $3^n$ .

Há uma série de heurísticas óbvias que podem ser usadas para podar a busca na programação dinâmica. Primeiro, soluções úteis normalmente ficam em torno da diagonal da superior esquerda à inferior direita do arranjo; isso leva a ignorar o desenvolvimento de extremos do arranjo. Segundo, pode ser útil podar a busca à medida que ela evolui, por exemplo, para distâncias de edição passando de certo limite, cortar esse caminho de solução ou ainda abandonar o problema inteiro, ou seja, a sequência de origem estará distante demais da sequência de destino dos caracteres para ser útil. Há também uma abordagem estocástica para o problema de comparação de padrões, que veremos na Seção 5.3.

## 4.2 Algoritmo da busca pela melhor escolha

### 4.2.1 Implementando a busca pela melhor escolha

Apesar de suas limitações, os algoritmos como busca com retrocesso, subida de encosta e programação dinâmica podem ser usados efetivamente se suas funções de avaliação forem suficientemente informativas para evitar máximos locais, becos sem saída e anomalias relacionadas em um espaço de busca. Porém, em geral, o uso da busca heurística requer um algoritmo mais flexível: isso é fornecido pela *busca pela melhor escolha*, em que, com uma fila de prioridade, a recuperação dessas situações é possível.

Assim como os algoritmos de busca em profundidade e em amplitude do Capítulo 3, a busca pela melhor escolha usa listas para manter estados: abertos para registrar a fronteira atual da busca e fechados para registrar os estados já visitados. Uma etapa adicional no algoritmo ordena os estados em abertos de acordo com uma estimativa heurística de sua “proximidade” com um objetivo. Assim, cada iteração do laço considera o estado mais “promissor” na lista abertos. O pseudocódigo para a função `busca_melhor_escolha` aparece a seguir.

```

função busca_melhor_escolha;
    início
        abertos := [Inicial];                                % inicializa
        fechados := [ ];
        enquanto abertos ≠ [ ] faça
            % existem estados
            início
                retire o estado mais à esquerda de abertos e chame-o de X;
                se X = objetivo então retorna o caminho de Inicial até X
                senão início
                    gera filhos de X;
                    para cada filho de X faça
                        caso
                            o filho não está em abertos ou em fechados:
                                início
                                    atribua ao filho um valor heurístico;
                                    acrescente o filho a abertos
                                fim;
                            o filho já está em abertos:
                                se o filho foi alcançado por um caminho mais curto
                                    então dê ao estado em abertos o caminho mais curto
                            o filho já está em fechados:
                                se o filho foi alcançado por um caminho mais curto, então
                                    início
                                        retire o estado de fechados;
                                        acrescente o filho em abertos
                                    fim;
                                fim;                                % caso
                            coloque X em fechados;
                            reordene estados em aberto pelo mérito heurístico (melhor mais à esquerda)
                        fim;
                    retorna FALHA                                % abertos está vazio
                fim.

```

A cada iteração, `busca_melhor_escolha` retira o primeiro elemento da lista abertos. Se encontrar as condições do objetivo, o algoritmo retorna o caminho de solução que levou ao objetivo. Note que cada estado retém informa-

ções do ancestral para determinar se ele tinha sido alcançado anteriormente por um caminho mais curto e permitir que o algoritmo retorne o caminho de solução final. (Ver Seção 3.2.3.)

Se o primeiro elemento em abertos não for um objetivo, o algoritmo aplica todas as regras ou operadores de produção que combinam para gerar seus descendentes. Se um estado filho não estiver em abertos ou fechados, busca\_melhor\_escolha aplica uma avaliação heurística a esse estado, e a lista abertos é classificada de acordo com os valores heurísticos desses estados. Isso leva os “melhores” estados para a frente de abertos. Note que, como essas estimativas são heurísticas em natureza, o próximo “melhor” estado a ser examinado pode ser de qualquer nível do espaço de estados. Quando abertos é mantido como uma lista ordenada, essa lista normalmente é conhecida como uma *fila de prioridades*.

Se um estado filho já estiver em abertos ou fechados, o algoritmo verifica para ter certeza de que o estado registra o mais curto dos dois caminhos parciais da solução. Estados duplicados não são retidos. Atualizando o histórico de caminho dos nós em abertos e fechados quando eles são redescobertos, o algoritmo achará um caminho mais curto para um objetivo (dentro dos estados considerados).

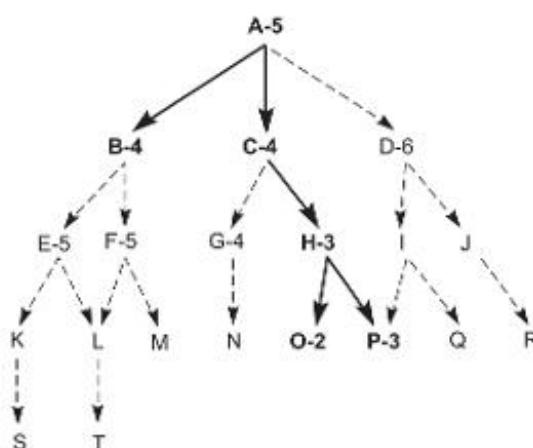
A Figura 4.10 mostra um espaço de estados hipotético com avaliações heurísticas ligadas a alguns de seus estados. Os estados com avaliações conectadas são aqueles realmente gerados em busca\_melhor\_escolha. Os estados expandidos pelo algoritmo de busca heurística são indicados em negrito; note que ele não busca todo o espaço. O objetivo da busca pela melhor escolha é determinar o estado-objetivo examinando o mínimo de estados possível; quanto mais *informada* (Seção 4.2.3) a heurística, menos estados são processados na descoberta do objetivo.

Um rastreamento da execução de busca\_melhor\_escolha nesse grafo aparece a seguir. Suponha que P seja o estado-objetivo no grafo da Figura 4.10. Como P é o objetivo, os estados ao longo do caminho até P tenderão a ter valores heurísticos mais baixos. A heurística é falível: o estado O tem um valor mais baixo que o próprio objetivo e é examinado primeiro. Diferentemente da subida de encosta, que não mantém uma fila de prioridades para a seleção dos “próximos” estados, o algoritmo se recupera desse erro e determina o objetivo correto.

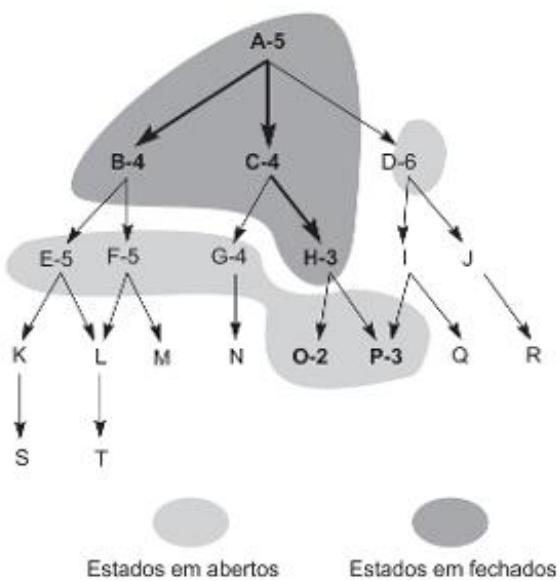
1. abertos = [A5]; fechados = []
2. avaliar A5; abertos = [B4,C4,D6]; fechados = [A5]
3. avaliar B4; abertos = [C4,E5,F5,D6]; fechados = [B4,A5]
4. avaliar C4; abertos = [H3,G4,E5,F5,D6]; fechados = [C4,B4,A5]
5. avaliar H3; abertos = [O2,P3,G4,E5,F5,D6]; fechados = [H3,C4,B4,A5]
6. avaliar O2; abertos = [P3,G4,E5,F5,D6]; fechados = [O2,H3,C4,B4,A5]
7. avaliar P3; a solução foi encontrada!

A Figura 4.11 mostra o espaço como ele se encontra após a quinta iteração do laço “enquanto”. Os estados contidos em abertos e em fechados estão indicados. A lista abertos registra a fronteira atual da busca, e fechados,

**Figura 4.10** Busca heurística de um espaço de estados hipotético.



**Figura 4.11** Busca heurística de um espaço de estados hipotético com os estados abertos e fechados realçados.



os estados que já foram considerados. Note que a fronteira da busca é altamente irregular, refletindo a natureza oportunista da busca pela melhor escolha.

O algoritmo de busca pela melhor escolha sempre seleciona o estado mais promissor em abertos para ser expandido. Entretanto, por usar uma heurística que pode incorrer em erro, ele não abandona todos os outros estados, mas os mantém em abertos. No caso em que a heurística leve a busca por um caminho que se mostre incorreto, o algoritmo recuperará de abertos um “segundo melhor” estado previamente gerado e mudará seu foco para outra parte do espaço. No exemplo da Figura 4.10, após se constatar que os filhos do estado B têm avaliações heurísticas pobres, a busca muda seu foco para o estado C. Os filhos de B são mantidos em abertos para o caso de o algoritmo necessitar retornar a eles mais adiante. Na busca\_melhor\_escolha, assim como nos algoritmos do Capítulo 3, a lista abertos permite o retrocesso de caminhos que falharam em produzir um objetivo.

#### 4.2.2 Implementando funções de avaliação heurísticas

Avaliamos, agora, o desempenho de várias heurísticas diferentes para resolver o quebra-cabeça dos 8. A Figura 4.12 mostra um estado inicial e um estado objetivo para o quebra-cabeça dos 8 com os três primeiros estados gerados na busca.

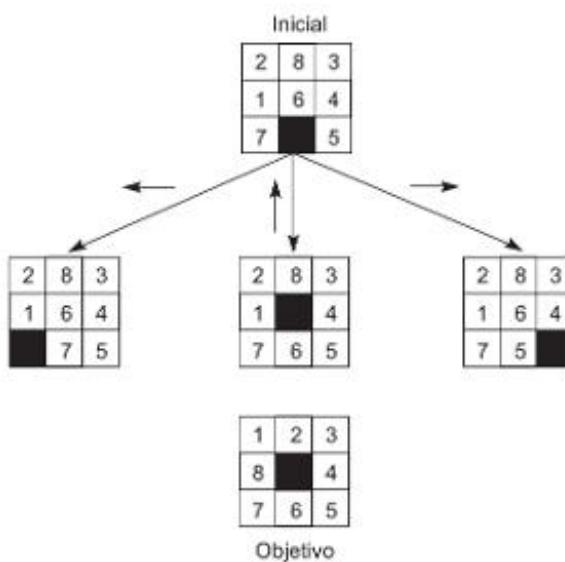
A heurística mais simples conta, em cada estado, as peças que estão fora de lugar em relação ao objetivo. Isso é intuitivamente atraente porque, aparentemente, se o restante for igual, o estado que tiver o menor número de peças fora de lugar estará provavelmente mais próximo do objetivo desejado, sendo o melhor a ser examinado em seguida.

Entretanto, essa heurística não usa toda a informação disponível em uma configuração de tabuleiro, porque ela não leva em conta a distância que as peças devem ser movidas. Uma heurística melhor somaria todas as distâncias em que cada peça está fora do lugar desejado, adicionando uma unidade para cada quadrado que uma peça necessita ser movida para alcançar a sua posição no estado-objetivo.

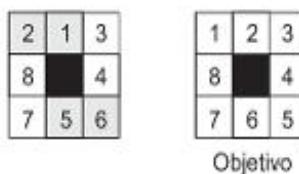
Essas duas heurísticas podem ser criticadas por não perceberem a dificuldade da inversão de peças, isto é, se duas peças forem vizinhas e o objetivo requer que elas estejam em posições opostas, levará (muito) mais que dois movimentos para colocá-las no lugar desejado, já que cada peça precisa “dar a volta” em torno da outra (Figura 4.13).

Uma heurística que leva isso em consideração multiplica um número pequeno (2, por exemplo) por cada inversão direta de peças (em que duas peças adjacentes precisam ser trocadas para ficarem na ordem correspondente

**Figura 4.12** O estado inicial, o primeiro conjunto de movimentos e o estado objetivo para um exemplo do quebra-cabeça dos 8.



**Figura 4.13** Um estado do quebra-cabeça dos 8 com um objetivo e duas inversões: 1 e 2, 5 e 6.



ao objetivo). A Figura 4.14 mostra o resultado da aplicação de cada uma dessas três heurísticas aos três estados filhos da Figura 4.12.

No quadro resumo das funções de avaliação da Figura 4.14, a heurística da “soma das distâncias” realmente parece fornecer uma estimativa mais precisa do trabalho que precisa ser realizado, que simplesmente a contagem do número de peças fora do lugar. Além disso, a heurística da inversão de peças não consegue distinguir entre esses estados, dando a cada um o valor estimado de 0. Embora ela seja uma heurística intuitivamente atraente, ela falha porque nenhum desses estados tem inversões diretas. Uma quarta heurística, que pode contornar as limitações da heurística de inversão de peças, adiciona a soma das distâncias das peças que estão fora do lugar e 2 vezes o número de inversões diretas.

Esse exemplo ilustra a dificuldade de se conceber boas heurísticas. Nossa objetivo é usar a informação limitada disponível em um único descritor de estado para fazer escolhas inteligentes. Cada uma das heurísticas propostas anteriormente ignora uma parcela crítica de informação, ficando sujeita a melhorias. O projeto de boas heurísticas é um problema empírico; discernimento e intuição ajudam, mas a avaliação final de uma heurística deve ser o seu desempenho real sobre instâncias do problema.

Se dois estados tiverem a mesma, ou quase a mesma, avaliação heurística, geralmente é preferível examinar o estado que está mais próximo da raiz do grafo. Esse estado terá uma probabilidade maior de estar no caminho *mais curto* que leva ao objetivo. A distância do estado inicial até os seus descendentes pode ser medida se mantivermos uma contagem da profundidade para cada estado. Essa contagem é 0 para o estado inicial e é incrementada em 1 para cada nível da busca. Essa medida de profundidade pode ser acrescentada à avaliação heurística de cada estado para orientar a busca em favor de estados mais superficiais no grafo.

**Figura 4.14** Três heurísticas aplicadas a estados no quebra-cabeça dos 8.

	<b>5</b>	<b>6</b>	<b>0</b>
	<b>3</b>	<b>4</b>	<b>0</b>
	<b>5</b>	<b>6</b>	<b>0</b>
	Peças fora de lugar	Soma das distâncias fora de lugar	2 × o número de inversões diretas



Objetivo

Isso faz com que a nossa função de avaliação,  $f$ , seja a soma de dois componentes:

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

onde  $g(n)$  mede o comprimento real do caminho de um estado  $n$  qualquer até o estado inicial e  $h(n)$  é uma estimativa heurística da distância entre o estado  $n$  e um objetivo.

No quebra-cabeça dos 8, por exemplo,  $h(n)$  pode ser o número de peças fora de lugar. Quando essa avaliação é aplicada a cada estado filho da Figura 4.12, os seus valores de  $f$  são 6, 4 e 6, respectivamente (veja a Figura 4.15).

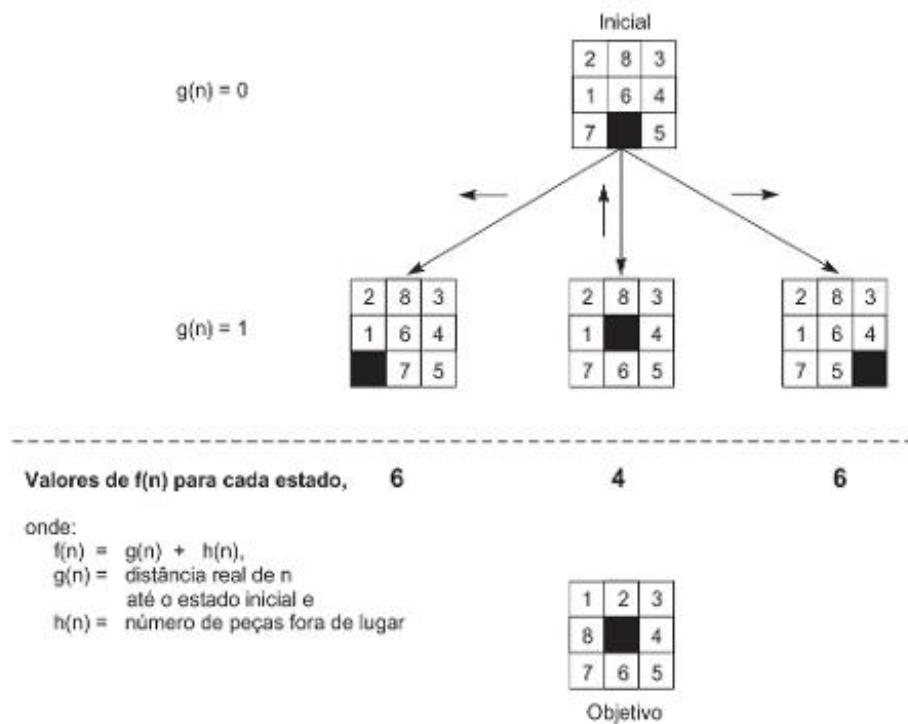
A busca pela melhor escolha completa do grafo do quebra-cabeça dos 8, usando  $f$  como definido anteriormente, aparece na Figura 4.16. Cada estado está rotulado com uma letra e o seu peso heurístico,  $f(n) = g(n) + h(n)$ . O número acima de cada estado indica a ordem na qual ele foi retirado da lista abertos. Alguns estados ( $h$ ,  $g$ ,  $b$ ,  $d$ ,  $n$ ,  $k$  e  $i$ ) não estão numerados dessa forma porque eles ainda estavam em aberto quando o algoritmo terminou.

Os estágios sucessivos de abertos e fechados que geram esse grafo são:

1. abertos = [a4];  
fechados = []
2. abertos = [c4, b6, d6];  
fechados = [a4]
3. abertos = [e5, f5, b6, d6, g6];  
fechados = [a4, c4]
4. abertos = [f5, h6, b6, d6, g6, i7];  
fechados = [a4, c4, e5]
5. abertos = [j5, h6, b6, d6, g6, k7, i7];  
fechados = [a4, c4, e5, f5]
6. abertos = [l5, h6, b6, d6, g6, k7, i7];  
fechados = [a4, c4, e5, f5, j5]
7. abertos = [m5, h6, b6, d6, g6, n7, k7, i7];  
fechados = [a4, c4, e5, f5, j5, l5]
8. sucesso, m = objetivo!

No passo 3, tanto  $e$  quanto  $f$  têm uma heurística de 5. O estado  $e$  é examinado primeiro, produzindo filhos,  $h$  e  $i$ . Embora  $h$ , o filho de  $e$ , tenha o mesmo número de peças fora de lugar que  $f$ , ele está em um nível mais profun-

Figura 4.15 A heurística  $f$  aplicada a estados do quebra-cabeça dos 8.



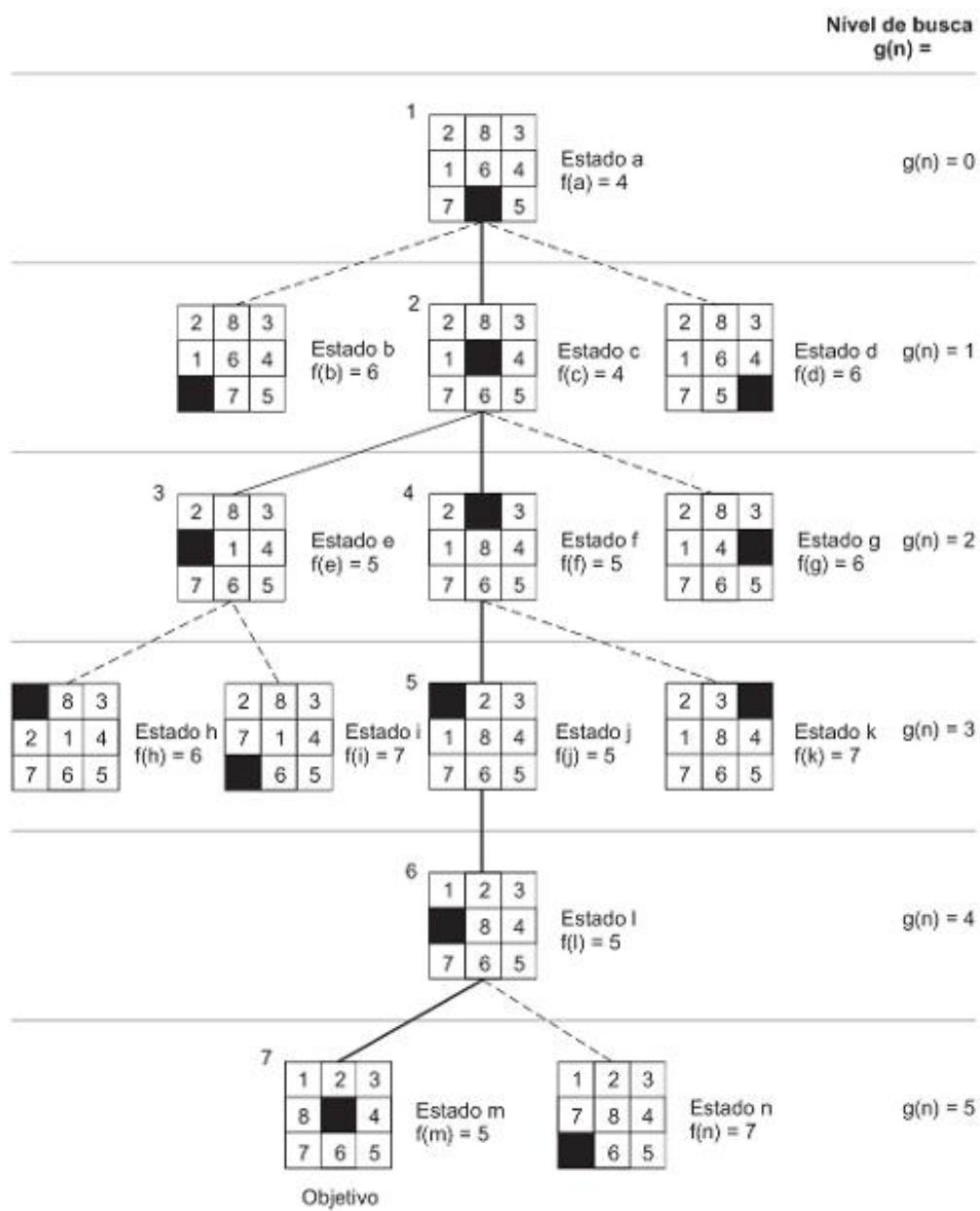
do no espaço. A medida de profundidade,  $g(n)$ , faz com que o algoritmo selecione  $f$  para avaliação no passo 4. O algoritmo retrocede ao estado mais superficial e continua em direção ao objetivo. O grafo do espaço de estados nesse estágio da busca, com abertos e fechados realçados, aparece na Figura 4.17. Observe a natureza oportunista da busca pela melhor escolha.

Na verdade, o componente  $g(n)$  da função de avaliação dá à busca mais característica de busca em amplitude. Ele evita que o algoritmo seja mal conduzido por uma avaliação errada: se uma heurística retoma continuamente “boas” avaliações para estados, ao longo de um caminho que não alcança um objetivo, o valor  $g$  crescerá de modo a dominar  $h$  e a forçar a busca a retornar para um caminho de solução mais curto. Isso garante que o algoritmo não fique permanentemente perdido, descendo por um ramo infinito. A Seção 4.3 examina as condições sob as quais a busca pela melhor escolha, usando essa função de avaliação, pode garantir a produção do caminho mais curto até um objetivo.

Para resumir:

1. Operações sobre estados geram filhos do estado atualmente examinados.
2. Cada estado novo é verificado para ver se ocorreu antes (está em abertos ou fechados), impedindo assim os laços.
3. Cada estado  $n$  recebe um valor  $f$  igual à soma de sua profundidade no espaço de busca  $g(n)$  e uma estimativa heurística de sua distância até um objetivo  $h(n)$ . O valor de  $h$  guia a busca em direção a estados heuristicamente promissores, enquanto o valor de  $g$  pode impedir que a busca persista indefinidamente em um caminho infrutífero.
4. Os estados em abertos são ordenados por seus valores  $f$ . Mantendo todos os estados em abertos até que sejam examinados ou até que um objetivo seja encontrado, o algoritmo se recupera de becos sem saída.
5. Como um ponto de implementação, a eficiência do algoritmo pode ser melhorada a partir da manutenção das listas abertos e fechados, talvez como *heaps* ou *árvores da esquerda* (*leftist trees*).

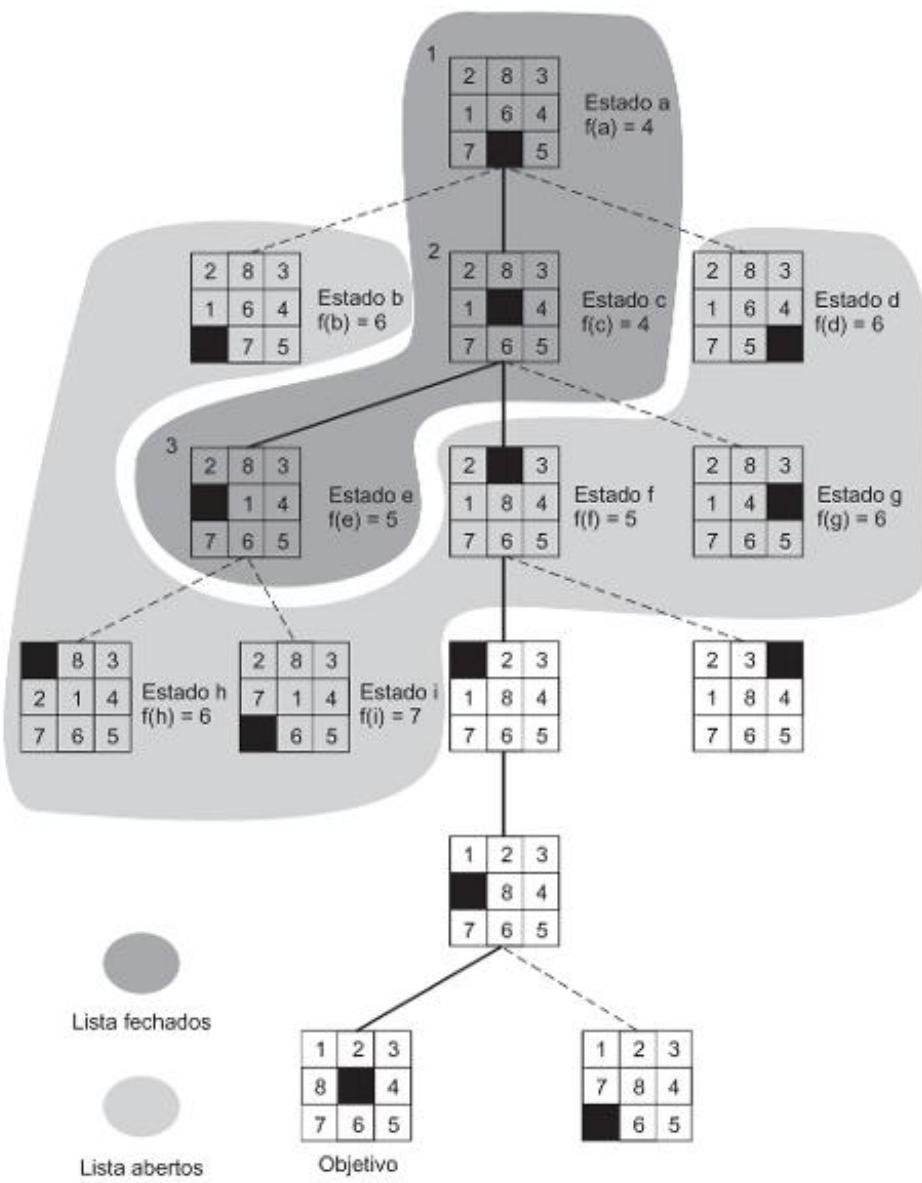
**Figura 4.16** Espaço de estados gerado na busca heurística do grafo do quebra-cabeça dos 8.



A busca pela melhor escolha é um algoritmo genérico para pesquisar, de modo heurístico, qualquer grafo de espaço de estados (assim como os algoritmos em amplitude e em profundidade, apresentados anteriormente). Ela se aplica igualmente a buscas guiadas por dados e por objetivo, e tem suporte em uma série de funções de avaliação heurísticas. Ela continuará (Seção 4.3) a fornecer uma base para examinar o comportamento da busca heurística. Devido à sua generalidade, a busca pela melhor escolha pode ser usada com diversas heurísticas, desde estimativas subjetivas de adequação do estado até medidas sofisticadas baseadas na probabilidade de um estado levar a um objetivo. Medidas estatísticas Bayesianas (capítulos 5 e 9) oferecem um exemplo importante desse método.

Outra abordagem interessante para a implementação de heurísticas é o uso de medidas de confiança por sistemas especialistas para ponderar os resultados de uma regra. Quando especialistas humanos empregam uma heurística, eles normalmente são capazes de dar uma estimativa de sua confiança em suas conclusões. Os siste-

**Figura 4.17** Abertos e fechados, como aparecem após a terceira iteração da busca heurística.



mas especialistas empregam *medidas de confiança* para selecionar as conclusões com a probabilidade de sucesso mais alta. Estados com confiâncias extremamente baixas podem ser totalmente eliminados. Essa técnica de busca heurística é examinada na próxima seção.

### 4.2.3 Busca heurística e sistemas especialistas

Jogos simples, como o quebra-cabeça dos 8, são veículos ideais para explorar o projeto e o comportamento dos algoritmos de busca heurística por diversos motivos:

1. Os espaços de busca são grandes o bastante para exigir a poda heurística.
2. A maior parte dos jogos é complexa o bastante para sugerir uma rica variedade de avaliações heurísticas para comparação e análise.

3. Os jogos geralmente não envolvem aspectos representativos complexos. Um único nó do espaço de estados é apenas uma descrição de tabuleiro e normalmente pode ser capturado de uma forma direta. Isso permite que os pesquisadores se concentrem no comportamento da heurística, e não nos problemas de representação do conhecimento.
4. Como cada nó do espaço de estados tem uma representação comum (por exemplo, uma descrição de tabuleiro), uma heurística simples poderá ser aplicada por todo o espaço de busca. Isso contrasta com sistemas como o consultor financeiro, em que cada nó representa um subobjetivo diferente com sua própria descrição distinta.

Problemas mais realísticos complicam bastante a implementação e a análise da busca heurística, exigindo várias heurísticas para lidar com diferentes situações no espaço do problema. Porém as percepções obtidas por jogos simples se generalizam a problemas como aqueles encontrados em aplicações de sistemas especialistas, planejamento, controle inteligente e aprendizado de máquina. Diferentemente do que ocorre no quebra-cabeça dos 8, uma única heurística pode não se aplicar a todos os estados nesses domínios. Em vez disso, heurísticas de solução de problemas específicas da situação são codificadas na sintaxe e no conteúdo de operadores individuais para a solução do problema. Cada etapa da solução incorpora sua própria heurística, que determina quando ela deve ser aplicada; o comparador de padrões casa a operação apropriada (heuristica) com o estado relevante no espaço.

#### **Exemplo 4.2.1 Consultor financeiro revisto**

O uso de medidas heurísticas para guiar a busca é uma abordagem genérica em IA. Considere novamente o problema do consultor financeiro dos capítulos 2 e 3, em que a base de conhecimento foi tratada como um conjunto de implicações lógicas, cujas conclusões são verdadeiras ou falsas. Na verdade, essas regras são de natureza fortemente heurística. Por exemplo, uma regra determina que um indivíduo com poupança e renda adequadas deve investir em ações:

$\text{conta\_poupança}(\text{adequada}) \wedge \text{renda}(\text{adequada}) \Rightarrow \text{investimento(ações)}$ .

Na realidade, é possível que tal indivíduo prefira a segurança adicional de uma estratégia combinada ou mesmo que todo o dinheiro de investimento seja aplicado em uma poupança. Assim, a regra é uma heurística, e o resolvedor de problemas deveria tentar lidar com essa incerteza. Poderíamos levar em conta fatores adicionais, como a idade do investidor e suas perspectivas de longo prazo em relação à estabilidade e à progressão profissional para tornar as regras mais informadas e capazes de lidar com distinções mais elaboradas. Entretanto, isso não muda a natureza fundamentalmente heurística do consultor financeiro.

Uma forma de os sistemas especialistas abordarem essa questão ocorre por meio da atribuição de um peso numérico (chamado de *medida de confiança* ou *fator de certeza*) à conclusão de cada regra. Esse valor mede a confiança que pode ser depositada nas suas conclusões.

A cada conclusão de uma regra é atribuída uma medida de confiança, um número real entre -1 e 1, com 1 correspondendo à certeza (verdadeiro) e -1 a um valor definitivo de falso. Valores intermediários refletem diversos graus de confiança na conclusão. Por exemplo, a regra anterior pode receber uma confiança de 0,8, refletindo uma possibilidade pequena que ela possa não estar correta. Podem-se tirar outras conclusões com diferentes graus de confiança:

$\text{conta\_poupança}(\text{adequada}) \wedge \text{renda}(\text{adequada}) \Rightarrow \text{investimento(ações)} \text{ com confiança} = 0,8$ .  
 $\text{conta\_poupança}(\text{adequada}) \wedge \text{renda}(\text{adequada}) \Rightarrow \text{investimento(combinação)} \text{ com confiança} = 0,5$ .  
 $\text{conta\_poupança}(\text{adequada}) \wedge \text{renda}(\text{adequada}) \Rightarrow \text{investimento(poupança)} \text{ com confiança} = 0,1$ .

Essas regras refletem a recomendação comum de investimento a um indivíduo com poupança e renda adequadas que seria fortemente aconselhado a investir em ações, mas que poderia fazer uma estratégia combinada de investimentos com uma pequena chance de investir em poupança. Algoritmos de busca heurística podem usar esses fatores de certeza de diversas maneiras. Por exemplo, os resultados de todas as regras aplicáveis poderiam ser gerados com seus valores associados de confiança. Essa busca exaustiva de todas as

possibilidades pode ser apropriada em domínios como a medicina. Alternativamente, o programa poderia retornar apenas o resultado com o maior valor de confiança, caso soluções alternativas não sejam de interesse. Com isso, o programa pode ignorar outras regras, podendo radicalmente o espaço de busca. Uma estratégia de poda mais conservadora poderia ignorar regras que gerassem conclusões com valores de confiança menores que um certo limiar (0,2, por exemplo).

Diversas questões importantes devem ser consideradas ao se usar medidas de confiança para ponderar conclusões de regras. O que realmente significa uma “medida numérica de confiança”? Por exemplo, como os valores de confiança devem ser tratados se a conclusão de uma regra for usada como premissa de outras? Como os valores de confiança devem ser combinados quando mais de uma regra chegar à mesma conclusão? Como devem ser atribuídas medidas de confiança apropriadas às regras? Essas questões são discutidas com mais detalhes no Capítulo 8.

## 4.3 Admissibilidade, monotonicidade e grau de informação

Podemos avaliar o comportamento de heurísticas ao longo de várias dimensões. Por exemplo, podemos não apenas desejar uma solução, mas também podemos exigir que o algoritmo encontre o menor caminho até o objetivo. Isso pode ser importante quando uma aplicação tem um custo excessivo relacionado com passos adicionais de solução, tal como ocorre no planejamento de um caminho para um robô autônomo por um ambiente perigoso. Heurísticas que encontram o caminho mais curto até um objetivo, sempre que ele existir, são chamadas de *admissíveis*. Em outras aplicações, um caminho de solução mínimo pode não ser tão importante quanto a eficiência global em resolver o problema (ver Figura 4.28).

Poderíamos estar interessados em saber se existe uma heurística melhor disponível. Em que sentido uma heurística é “melhor” que outra? Isso é o *grau de informação* de uma heurística.

Quando um estado é descoberto usando busca heurística, há alguma garantia de que o mesmo estado não será descoberto mais adiante na busca com um custo menor (com um caminho mais curto até o estado inicial)? Essa é a propriedade da *monotonicidade*. As respostas a essas e a outras questões relacionadas com a eficácia de heurísticas constituem o conteúdo desta seção.

### 4.3.1 Medidas de admissibilidade

Um algoritmo de busca é *admissível* se ele seguramente encontrar um caminho mínimo até uma solução, sempre que tal solução exista. A busca em amplitude é uma estratégia de busca admissível. Como ela examina todos os estados em um nível  $n$  do grafo antes de considerar qualquer estado no nível  $n + 1$ , quaisquer nós de objetivo são encontrados ao longo do menor caminho possível. Infelizmente, a busca em amplitude quase sempre é muito ineficiente para o uso prático.

Usando a função de avaliação  $f(n) = g(n) + h(n)$ , que foi introduzida na última seção, podemos caracterizar uma classe de estratégias de busca heurística admissíveis. Se  $n$  é um nó no grafo de espaço de estados, então  $g(n)$  mede a profundidade na qual o estado foi encontrado no grafo, e  $h(n)$  é a estimativa heurística da distância de  $n$  até um objetivo. Nesse sentido,  $f(n)$  estima o custo total do caminho partindo do estado inicial, passando por  $n$  e chegando ao estado-objetivo. Para determinarmos as propriedades das heurísticas admissíveis, definimos uma função de avaliação  $f^*$ :

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

onde  $g^*(n)$  é o custo do caminho *mais curto* do nó inicial ao nó  $n$  e  $h^*$  retorna o custo *real* do menor caminho de  $n$  até o objetivo. Com isso,  $f^*(n)$  é o custo real do caminho ótimo partindo de um nó inicial até um nó objetivo, passando pelo nó  $n$ .

Como veremos, quando empregamos busca\_melhor\_escolha com a função de avaliação  $f^*$ , a estratégia de busca resultante é admissível. Embora oráculos como  $f^*$  não existam para a maioria dos problemas reais, desejamos que a função de avaliação  $f$  seja uma estimativa próxima de  $f^*$ . No algoritmo A,  $g(n)$ , o custo do caminho atual até o estado  $n$ , é uma estimativa razoável de  $g^*$ , mas eles podem não ser iguais:  $g(n) \geq g^*(n)$ . Eles serão iguais apenas quando a busca pelo grafo tiver descoberto o caminho ótimo para o estado  $n$ .

De modo semelhante, substituímos  $h^*(n)$  por  $h(n)$ , uma estimativa heurística do custo mínimo até um estado objetivo. Embora usualmente não possamos computar  $h^*$ , frequentemente é possível determinar se a estimativa heurística,  $h(n)$ , tem ou não  $h^*(n)$  como um limite superior, isto é, se ela é sempre menor ou igual ao custo real de um caminho mínimo. Se o algoritmo A usar uma função de avaliação  $f$ , na qual  $h(n) \leq h^*(n)$ , ele será chamado de *algoritmo A\**.

### Definição

#### ALGORITMO A, ADMISSIBILIDADE, ALGORITMO A\*

Considere a função de avaliação  $f(n) = g(n) + h(n)$ , onde

$n$  é um estado qualquer encontrado na busca.

$g(n)$  é o custo de  $n$  a partir do estado inicial.

$h(n)$  é a estimativa heurística do custo de ir de  $n$  até um objetivo.

Se essa função de avaliação for usada com o algoritmo busca\_melhor\_escolha da Seção 4.1, o resultado é o chamado *algoritmo A*.

Um algoritmo de busca é *admissível* se, para qualquer grafo, ele sempre terminar no caminho de solução ótimo, sempre que existir um caminho entre os estados inicial e objetivo.

Se o algoritmo A for usado com uma função de avaliação, na qual  $h(n)$  é menor que o custo do caminho mínimo de  $n$  para um objetivo, ou igual a ele, o algoritmo de busca resultante será chamado de *algoritmo A\** (pronuncia-se “A ESTRELA”).

Agora é possível formular uma propriedade dos algoritmos A\*:

Todos os algoritmos A\* são admissíveis.

A admissibilidade dos algoritmos A\* é um teorema. Um exercício ao final do capítulo fornece as indicações para o desenvolvimento da sua prova (veja também Nilsson, 1980, p. 76-78). O teorema afirma que qualquer algoritmo A\*, isto é, um algoritmo que utiliza uma heurística  $h(n)$ , tal que  $h(n) \leq h^*(n)$  para todo  $n$ , encontrará, certamente, o caminho mínimo de  $n$  até um objetivo, se tal caminho existir.

Note que a busca em amplitude pode ser caracterizada como um algoritmo A\* onde  $f(n) = g(n) + 0$ . A decisão para se considerar um estado é baseada apenas na sua distância em relação ao estado inicial. Mostraremos (Seção 4.3.3) que o conjunto de nós considerados por um algoritmo A\* é um subconjunto dos estados examinados na busca em amplitude.

Várias heurísticas para o quebra-cabeça dos 8 fornecem exemplos de algoritmos A\*. Embora não possamos calcular o valor de  $h^*(n)$  para o quebra-cabeça dos 8, podemos determinar quando uma heurística tem limite superior igual ao custo real do caminho mais curto até um objetivo.

Por exemplo, a heurística de contar o número de peças que não estão na posição objetivo é certamente menor que, ou igual, o número de movimentos necessários para mover-las até a sua posição objetivo. Assim, essa heurística é admissível e garante um caminho de solução ótimo (ou o mais curto) quando tal caminho existe. A soma das distâncias diretas das peças fora de lugar é também menor ou igual ao caminho real mínimo. O uso de pequenos multiplicadores para inversões diretas de peças resulta em uma heurística admissível.

Essa abordagem para a verificação de admissibilidade para as heurísticas do quebra-cabeça dos 8 pode ser aplicada a qualquer problema de busca heurística. Muito embora o custo real do caminho mais curto até um objetivo não possa ser sempre calculável, muitas vezes podemos provar que uma heurística é limitada “por cima” por esse valor. Quando isso puder ser feito, a busca resultante terminará na descoberta do caminho mais curto até o objetivo, quando tal caminho existir.

### 4.3.2 Monotonicidade

Observe que a definição de algoritmos A\* não requer que  $g(n) = g^*(n)$ . Isso significa que heurísticas admissíveis podem inicialmente alcançar estados que não são objetivos ao longo de um caminho subótimo, desde que, no final, o algoritmo encontre um caminho ótimo para todos os estados no caminho até um objetivo. Uma questão natural é saber se existem heurísticas que são “localmente admissíveis”, isto é, que consistentemente encontram o caminho mais curto para cada estado encontrado na busca. Essa propriedade é chamada de *monotonicidade*.

#### Definição

#### MONOTONICIDADE

Uma função heurística  $h$  é monotônica se

1. Para todo estado  $n_i$  e  $n_j$ , onde  $n_j$  é um descendente de  $n_i$ ,  

$$h(n_i) - h(n_j) \leq \text{custo}(n_i, n_j),$$
 onde  $\text{custo}(n_i, n_j)$  é o custo real (em número de movimentos) para ir do estado  $n_i$  para o  $n_j$ .
2. A avaliação heurística do estado objetivo é zero, ou seja,  $h(\text{Objetivo}) = 0$ .

Uma forma de descrever a propriedade da monotonicidade é que, em qualquer lugar, o espaço de busca é localmente consistente com a heurística empregada. A diferença entre as medidas heurísticas para um estado e para qualquer um de seus sucessores é limitada pelo custo real de ir desse estado para o seu sucessor. Isso é o mesmo que dizer que a heurística é admissível em qualquer lugar, alcançando qualquer estado ao longo do caminho mais curto a partir de seus ancestrais.

Se o algoritmo de busca em grafo para a busca pela melhor escolha for usado com uma heurística monotônica, um passo importante pode ser omitido. Como a heurística encontra o caminho mais curto para qualquer estado na primeira vez que o estado é descoberto, quando esse estado é encontrado uma segunda vez, não é necessário verificar se o novo caminho é mais curto. Ele não será! Com isso, qualquer estado que seja redescoberto no espaço pode ser imediatamente descartado sem atualizar a informação do caminho registrada em abertos ou em fechados.

Quando uma heurística monotônica é usada, conforme a busca avança através do espaço, a medida heurística para cada estado  $n$  é substituída pelo custo real para gerar aquela porção do caminho até  $n$ . Como, em qualquer caso, o custo real é igual ou maior que a heurística,  $f$  não diminuirá, isto é,  $f$  é monotonamente não decrescente (dai o nome).

Um argumento simples pode mostrar que qualquer heurística monotônica é admissível. Esse argumento considera qualquer caminho no espaço como uma sequência de estados  $s_1, s_2, \dots, s_n$ , onde  $s_1$  é o estado inicial e  $s_n$  é o objetivo. Para a sequência de movimentos nesse caminho arbitrariamente selecionado:

$s_1$ para $s_2$	$h(s_1) - h(s_2) \leq \text{custo}(s_1, s_2)$	pela propriedade da monotonicidade
$s_2$ para $s_3$	$h(s_2) - h(s_3) \leq \text{custo}(s_2, s_3)$	pela propriedade da monotonicidade
$s_3$ para $s_4$	$h(s_3) - h(s_4) \leq \text{custo}(s_3, s_4)$	pela propriedade da monotonicidade
.	.	pela propriedade da monotonicidade
.	.	pela propriedade da monotonicidade
$s_{n-1}$ para $s_n$	$h(s_{n-1}) - h(s_n) \leq \text{custo}(s_{n-1}, s_n)$	pela propriedade da monotonicidade

Somando cada coluna e usando a propriedade da monotonicidade que  $h(s_0) = 0$ :

$$\text{caminho } s_1 \text{ para } s_0 \quad h(s_1) \leq \text{custo}(s_1, s_0)$$

Isso significa que a heurística monotônica  $h$  é  $A^*$  e admissível. Deixamos como exercício a questão de descobrir se a propriedade da admissibilidade de uma heurística implica monotonicidade.

### 4.3.3 Quando uma heurística é melhor: heurísticas mais informadas

A questão final desta subseção compara a habilidade de duas heurísticas para encontrar o caminho mínimo. Um caso interessante ocorre quando as heurísticas são  $A^*$ .

#### *Definição*

#### GRAU DE INFORMAÇÃO

Para duas heurísticas  $A^*$ ,  $h_1$  e  $h_2$ , se  $h_1(n) \leq h_2(n)$ , para todos os estados  $n$  do espaço de busca, então se diz que a heurística  $h_2$  é *mais informada* que  $h_1$ .

Podemos usar essa definição para comparar as heurísticas propostas para solucionar o quebra-cabeça dos 8. Como mencionado anteriormente, a busca em amplitude é equivalente ao algoritmo  $A^*$  com heurística  $h_1$ , tal que  $h_1(x) = 0$  para todos os estados  $x$ . Isso é menor que  $h^*$  trivialmente. Mostramos também que  $h_2$ , o número de peças fora de lugar em relação ao estado objetivo, é um limite menor para  $h^*$ . Nesse caso,  $h_1 \leq h_2 \leq h^*$ . Daí segue que a heurística do “número de peças fora de lugar” é mais informada que a busca em amplitude. A Figura 4.18 compara os espaços buscados por essas duas heurísticas. Tanto  $h_1$  como  $h_2$  encontram o caminho ótimo, mas  $h_2$  avalia um número muito menor de estados no processo.

De modo semelhante, podemos argumentar que a heurística, que calcula a soma das distâncias diretas de todas as peças que estão fora de lugar, é, por sua vez, mais informada que o cálculo do número de peças que estão fora de lugar em relação ao estado objetivo, e realmente esse é o caso. Podemos visualizar uma sequência de espaços de busca, cada um menor que o anterior, convergindo para a solução direta do caminho ótimo.

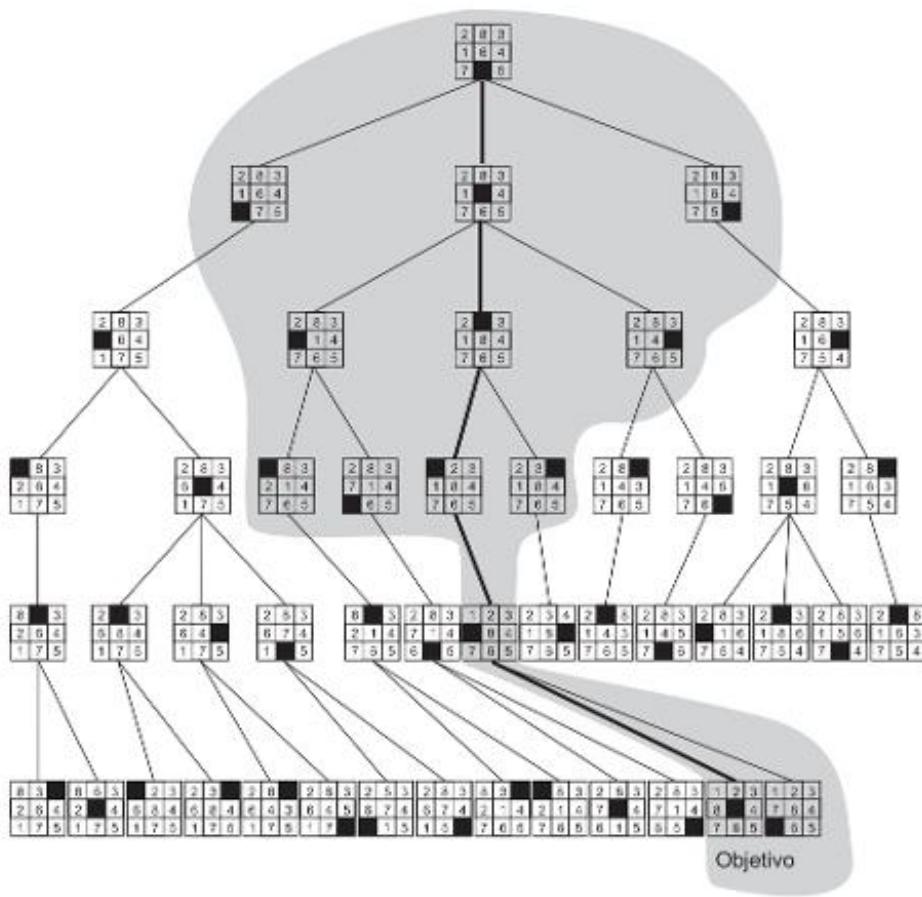
Se uma heurística  $h_2$  é mais informada que  $h_1$ , então o conjunto de estados examinados por  $h_2$  é um subconjunto daqueles expandidos por  $h_1$ . Isso pode ser verificado supondo-se o oposto (que haja pelo menos um estado expandido por  $h_2$ , e não por  $h_1$ ). Mas, como  $h_2$  é mais informada que  $h_1$ , para todo  $n$ ,  $h_2(n) \geq h_1(n)$ , e ambas são limitadas acima por  $h^*$ , nossa suposição é contraditória.

Portanto, em geral, quanto mais informado for um algoritmo  $A^*$ , menos espaço ele precisará expandir até chegar à solução ótima. Porém devemos ter cuidado para que os cálculos necessários para empregar a heurística mais informada não sejam tão ineficientes que superem os ganhos com a redução do número de estados buscados.

Os programas de computador para jogar xadrez oferecem um exemplo interessante desse dilema. Uma escola de pensamento usa heurísticas simples e conta com a velocidade do computador para buscar em profundidade no espaço de busca. Esses programas normalmente utilizam hardware especializado para a avaliação de estado a fim de aumentar a profundidade da busca. Outra escola conta com heurísticas mais sofisticadas para reduzir o número de estados do tabuleiro buscados. Essas heurísticas incluem cálculos de vantagens de peças, controle da geografia do tabuleiro, estratégias de ataque possíveis, peões passados, e assim por diante. O cálculo da própria heurística pode envolver complexidade exponencial (uma questão discutida na Seção 4.5). Como o tempo total para os 40 primeiros movimentos do jogo é limitado, é importante otimizar esse dilema entre busca e avaliação heurística. A mistura ideal de busca cega e heurística continua sendo uma questão empírica em aberto nos jogos de xadrez por computador, como podemos ver na disputa Gary Kasparov × Deep Blue (Hsu, 2002).

**Figura 4.18** Comparação do espaço de estados visitado usando busca heurística com o espaço visitado por busca em amplitude.

A parte do grafo visitada heuristicamente aparece sombreada. O caminho de solução ótimo está em negrito. A heurística usada é  $f(n) = g(n) + h(n)$ , onde  $h(n)$  é o número de peças fora de lugar.



## 4.4 Usando heurísticas em jogos

*Nesse instante, dois conceitos opostos do jogo trazem à tona comentário e discussão. Os jogadores mais destacados distinguiram dois tipos principais do Jogo, o formal e o psicológico...*

—HERMANN HESSE, “Magister Ludi” (O jogo das contas de vidro)

### 4.4.1 Procedimento minimax em grafos exaustivamente buscáveis

Os jogos sempre foram uma área de aplicação importante para algoritmos heurísticos. Jogos para duas pessoas são mais complicados que quebra-cabeças simples, devido à existência de um adversário “hostil” e basicamente imprevisível. Assim, eles oferecem algumas oportunidades interessantes para desenvolver heurísticas, além de maiores dificuldades no desenvolvimento de algoritmos de busca.

Primeiro, consideramos jogos cujo espaço de estados é pequeno o bastante para ser buscado exaustivamente; aqui, o problema é buscar sistematicamente o espaço de movimentos e respostas possíveis pelo adversário. Então, examinamos jogos em que é impossível ou indesejável buscar exaustivamente o grafo de jogos. Como somente

uma parte do espaço de estados pode ser gerada e buscada, o jogador precisa usar heurísticas para orientar a jogada por um caminho até um estado de vitória.

Primeiro, consideramos uma variante do jogo *nim*, cujo espaço de estados pode ser buscado exaustivamente. Para jogar esse jogo, diversas fichas são colocadas em uma mesa entre os dois adversários; a cada movimento, o jogador precisa dividir uma pilha de fichas em duas pilhas não vazias com tamanhos diferentes. Assim, 6 fichas podem ser divididas em pilhas de 5 e 1, ou 4 e 2, mas não de 3 e 3. O primeiro jogador que não conseguir fazer uma jogada perde o jogo. Para um número razoável de fichas, o espaço de estados pode ser buscado exaustivamente. A Figura 4.19 ilustra o espaço para um jogo com 7 fichas.

Ao jogar jogos cujo espaço de estados pode ser delineado exaustivamente, a principal dificuldade está em considerar as ações do oponente. Um modo simples de tratar disso considera que o seu oponente usa o mesmo conhecimento do espaço de estados que você usa e aplica esse conhecimento em um esforço consistente para vencer o jogo. Embora essa suposição tenha suas limitações (que são discutidas na Seção 4.4.2), ela oferece uma base razoável para prever o comportamento de um oponente. *Minimax* busca o espaço do jogo sob essa suposição.

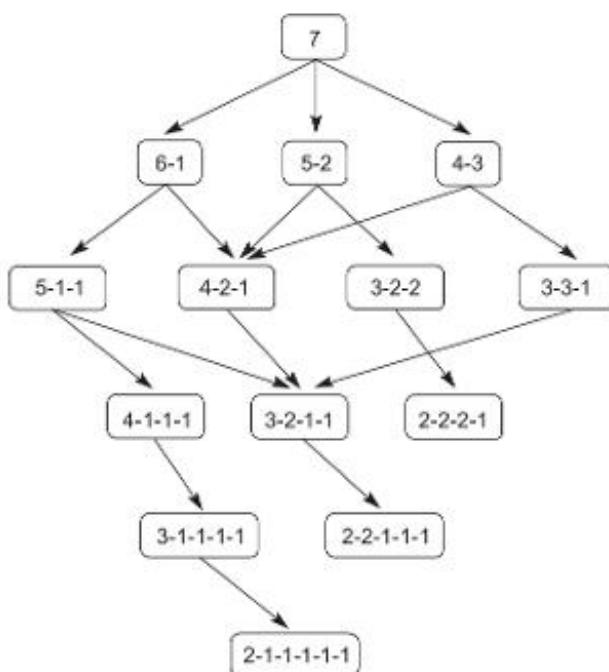
Os oponentes em um jogo são chamados de MIN e MAX. Embora isso seja em parte por motivos históricos, o significado desses nomes é simples: MAX representa o jogador tentando vencer, ou MAXimizar sua vantagem. MIN é o adversário que tenta MINimizar o placar de MAX. Supomos que MIN use as mesmas informações e sempre tente se mover para um estado que é pior para MAX.

Na implementação de minimax, rotulamos cada nível no espaço de busca de acordo com quem joga nesse ponto do jogo, MIN ou MAX. No exemplo da Figura 4.20, MIN pode se mover primeiro. Cada nó folha recebe um valor de 1 ou 0, dependendo se é uma vitória para MAX ou para MIN. Minimax propaga esses valores para cima no grafo, por meio de nós pai sucessivos, de acordo com essa regra:

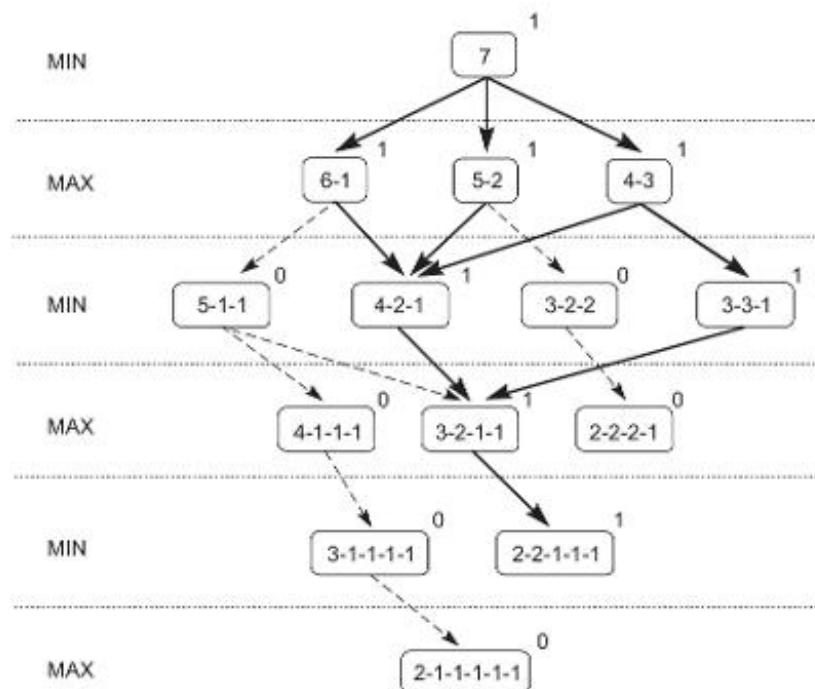
Se o estado pai for um nó MAX, dê-lhe o valor máximo entre seus filhos.

Se o pai for um nó MIN, dê-lhe o valor mínimo de seus filhos.

**Figura 4.19** Espaço de estados para uma variante de nim. Cada estado divide as sete combinações em uma ou mais pilhas.



**Figura 4.20** Minimax exaustivo para o jogo nim. As linhas em negrito indicam vitórias forçadas para MAX. Cada nó é marcado com o seu valor derivado (0 ou 1), segundo minimax.



O valor que é atribuído dessa forma a cada estado indica o valor do melhor estado que esse jogador pode esperar atingir (supondo que o oponente jogue conforme previsto pelo algoritmo minimax). Esses valores derivados são usados para escolher entre jogadas possíveis. O resultado de aplicar minimax ao grafo de espaço de estados para o nim aparece na Figura 4.20.

Os valores dos nós folha são propagados para cima pelo grafo usando minimax. Como todos os primeiros movimentos possíveis de MIN levam a nós com um valor derivado de 1, o segundo jogador, MAX, sempre pode forçar o jogo a uma vitória, independentemente do primeiro movimento de MIN, que só poderia vencer se MAX jogasse de forma inconsequente. Na Figura 4.20, MIN pode escolher qualquer uma das primeiras alternativas de movimento com os caminhos de vitória resultantes para MAX pelas linhas em negrito.

Embora existam jogos em que é possível se buscar exaustivamente o espaço de estados, os casos mais interessantes não permitem a busca exaustiva. Na próxima seção, examinamos a busca em profundidade fixa.

#### 4.4.2 Minimaxizando para profundidade de camada fixa

Ao se aplicar o minimax a jogos mais complicados, raramente é possível expandir o grafo de espaço de estados até os nós folha. Em vez disso, o espaço de estados é buscado até um número de níveis predefinido, conforme os recursos disponíveis de tempo e memória. Essa estratégia é chamada de *antecipação para n níveis*, onde n é o número de níveis a serem explorados. Como as folhas desse subgrafo não são estados finais do jogo, não é possível atribuir-lhes valores que reflitam uma vitória ou uma derrota. Em vez disso, atribui-se a cada nó um valor de acordo com uma função de avaliação heurística. O valor que é propagado de volta ao nó raiz não é uma indicação se uma vitória pode ou não ser alcançada (como no exemplo anterior), mas é simplesmente o valor heurístico do melhor estado que pode ser alcançado em n movimentos a partir da raiz. A estratégia de antecipação aumenta o

poder de uma heurística por permitir que ela seja aplicada em uma área maior do espaço de estados. Minimax consolida essas avaliações separadas em um único valor de um estado ancestral.

Em um jogo de conflito, cada jogador tenta superar o outro; por isso, muitas heurísticas desses jogos medem diretamente a vantagem de um jogador sobre outro. Em um jogo de damas ou de xadrez, a vantagem de peças é importante, assim uma heurística simples poderia calcular a diferença no número de peças pertencentes a MAX e a MIN e tentar maximizar a diferença entre essas medidas de peças. Uma estratégia mais sofisticada poderia atribuir valores diferentes às peças, dependendo de seus valores (por exemplo, rainha em relação a peão, ou dama em relação a uma pedra comum) ou de sua localização no tabuleiro. A maioria dos jogos fornece oportunidades ilimitadas para a especificação de heurísticas.

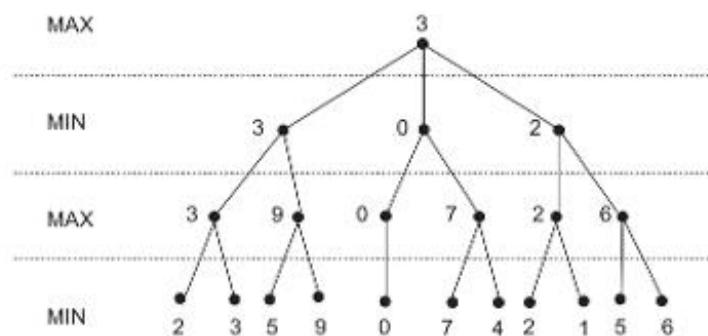
Os grafos de jogos são buscados nível a nível, ou em *camadas*. Como vimos na Figura 4.20, MAX e MIN selecionam alternativamente os movimentos. Cada movimento de um jogador define uma nova camada do grafo. Os programas de jogos geralmente antecipam uma profundidade de camada fixa, determinada pelas limitações de espaço/tempo do computador. Os estados naquela camada são mediados heuristicamente, e os valores são propagados de volta para cima no grafo usando minimax. O algoritmo de busca usa esses *valores derivados* para selecionar o melhor entre os próximos movimentos possíveis.

Após atribuir uma avaliação a cada estado na camada selecionada, o programa propaga um valor para cada estado pai na camada acima. Se o pai estiver em um nível MIN, é propagado para cima o valor mínimo dos filhos. Se o pai é um nó MAX, minimax atribui a ele o valor máximo de seus filhos.

Maximizando para os pais MAX e minimizando para os pais MIN, os valores retornam acima no grafo a partir dos filhos do estado atual. Esses valores, então, são usados pelo estado atual para selecionar um de seus filhos. A Figura 4.21 mostra minimax em um espaço de estados hipotético com uma antecipação de quatro camadas.

Podemos fazer vários comentários finais sobre o procedimento minimax. O primeiro e mais importante é que avaliações em qualquer profundidade de camada fixa (decidida anteriormente) podem ser muito enganosas. Quando uma heurística é aplicada com antecipação limitada, é possível que a profundidade da antecipação não possa detectar que um caminho heuristicamente promissor leve a uma situação ruim mais tarde no jogo. Se o seu adversário no jogo de xadrez oferecer uma torre como uma isca para capturar a sua dama e a avaliação antecipar apenas até a camada em que a torre é oferecida, a avaliação será tendenciosa para esse estado. Infelizmente, a seleção desse estado pode causar a derrota no jogo! Esse é o chamado *efeito de horizonte*. Ele é normalmente enfrentado pelo avanço mais profundo da busca por várias camadas quando estados parecem ser excepcionalmente bons. Entretanto, esse aprofundamento seletivo da busca em áreas importantes não fará desapare-

**Figura 4.21** Minimax aplicado a um espaço de estados hipotético. Estados folha apresentam valores heurísticos; estados internos mostram valores retropropagados.



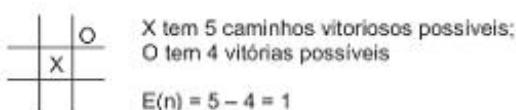
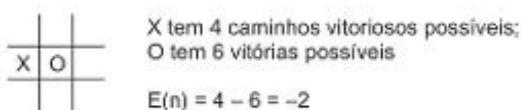
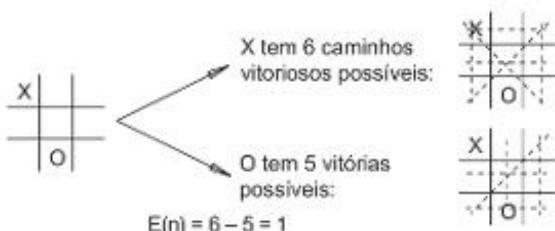
cer o efeito de horizonte. A busca precisa parar em algum lugar e será cega para os estados que estão além desse ponto.

Existe outro efeito que ocorre quando se minimaxifica usando avaliações heurísticas. As avaliações que ocorrem muito profundamente no espaço podem ser tendenciosas devido à sua profundidade excessiva (Pearl, 1984). Do mesmo modo que a média de produtos difere do produto das médias, a estimativa de minimax (que é o que desejamos) é diferente da minimax das estimativas (que é o que estamos fazendo). Nesse sentido, a busca mais profunda com avaliação e minimax significa, mas nem sempre, uma busca melhor. Uma discussão mais aprofundada dessas questões e possíveis soluções podem ser encontradas em Pearl (1984).

Concluindo a discussão sobre minimax, apresentamos uma aplicação ao jogo da velha (Seção 4.0) adaptada de Nilsson (1980). Nesse caso é usada uma heurística um pouco mais complexa que tenta medir o conflito no jogo. A heurística toma o estado a ser medido, conta todas as linhas vencedoras abertas para MAX e, então, subtrai o número total de linhas vencedoras abertas para MIN. A busca tenta maximizar essa diferença. Se um estado for uma vitória forçada para MAX, ele é avaliado como  $+\infty$ ; uma vitória forçada para MIN é avaliada como  $-\infty$ . A Figura 4.22 mostra essa heurística aplicada a vários estados típicos.

As figuras 4.23, 4.24 e 4.25 demonstram a heurística da Figura 4.22 em um minimax de duas camadas. Essas figuras mostram a avaliação heurística, a retropropagação minimax e o movimento de MAX com um tipo de desempate aplicado a movimentos de igual valor, extraído de Nilsson (1971).

**Figura 4.22** Medida heurística de conflito aplicada a estados do jogo da velha.



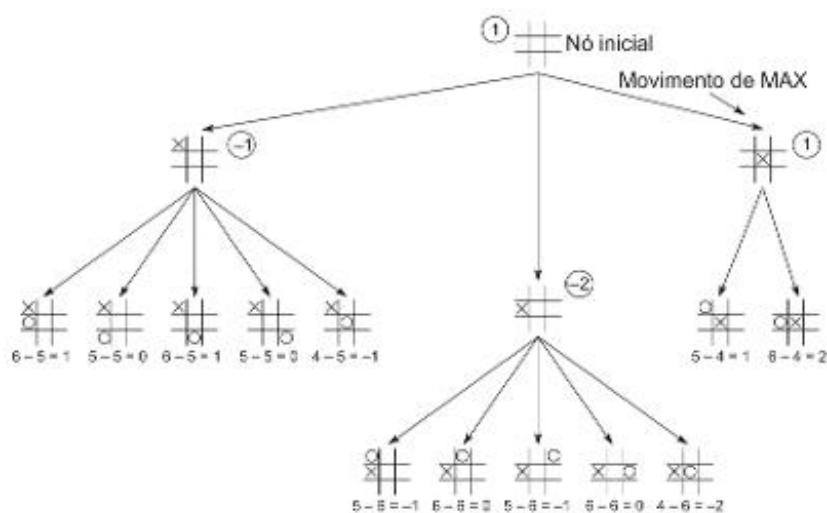
A heurística é  $E(n) = M(n) - O(n)$

onde  $M(n)$  é o total de minhas linhas vitoriosas possíveis

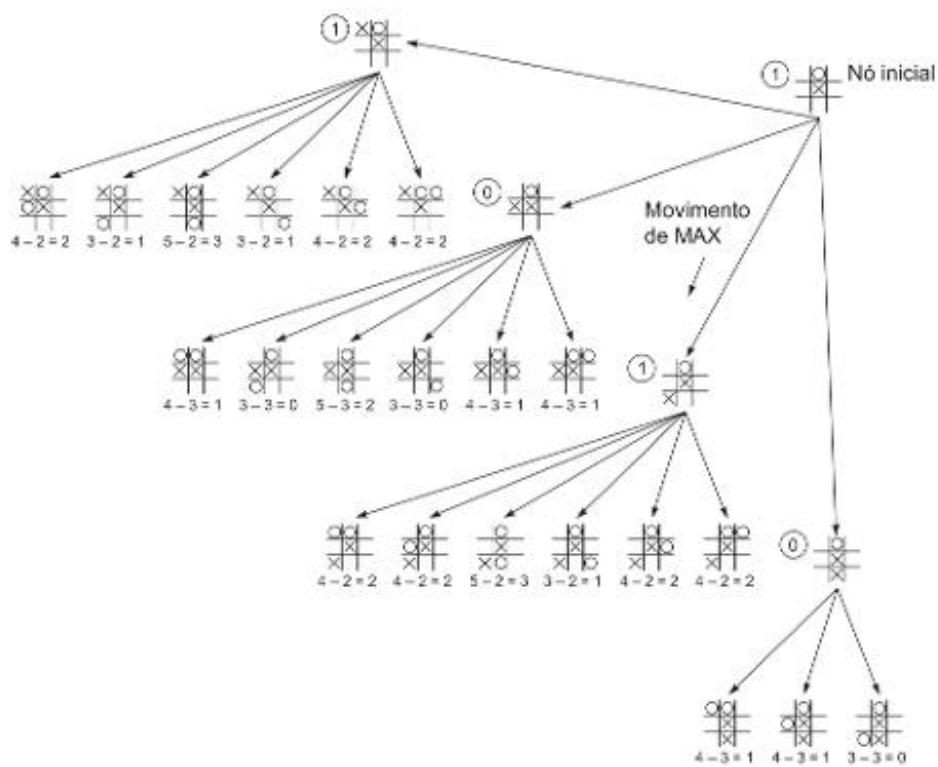
$O(n)$  é o total de linhas vitoriosas possíveis do oponente

$E(n)$  é a avaliação total para o estado  $n$

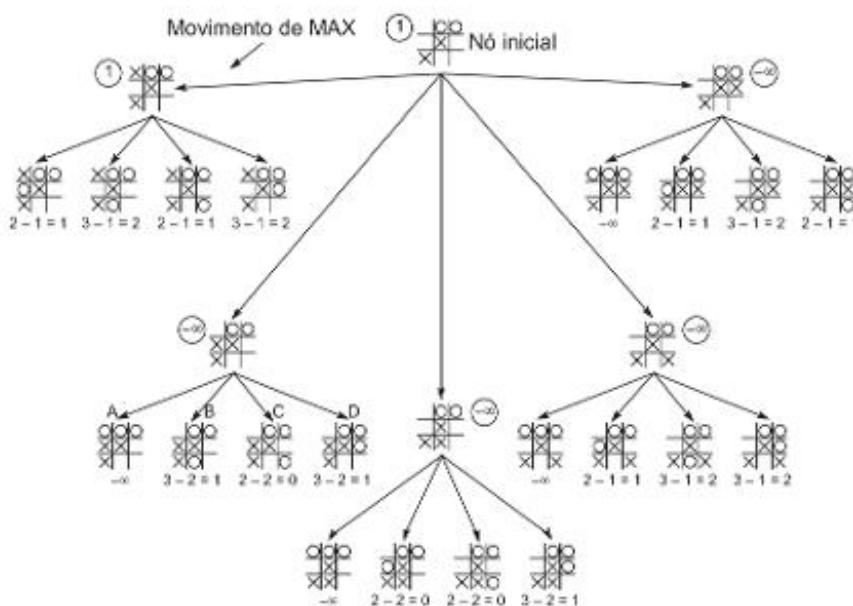
**Figura 4.23** Minimax de duas camadas aplicado ao movimento de abertura do jogo da velha, extraído de Nilsson (1971).



**Figura 4.24** Minimax de duas camadas e um dos dois segundos movimentos possíveis de MAX, extraído de Nilsson (1971).



**Figura 4.25** Minimax de duas camadas aplicado ao movimento do X próximo do final do jogo, extraído de Nilsson (1971).



#### 4.4.3 Procedimento alfa-beta

O minimax original requer uma análise do espaço de busca em dois passos, o primeiro para descer à profundidade da camada e ali aplicar a heurística, e o segundo para retropropagar os valores para cima na árvore. Minimax busca todos os ramos no espaço, incluindo muitos que poderiam ser ignorados ou podados por um algoritmo mais inteligente. Pesquisadores em jogos desenvolveram uma classe de técnicas de busca chamada de poda *alfa-beta*, proposta primeiro no final dos anos 1950 (Newell e Simon, 1976), para melhorar a eficiência de busca em jogos de dois adversários (Pearl, 1984).

A ideia para a busca alfa-beta é simples: em vez de buscar todo o espaço até uma certa profundidade, a busca alfa-beta avança como uma busca em profundidade. Dois valores, chamados de *alfa* e *beta*, são criados durante a busca. O valor alfa, associado aos nós MAX, não pode decrescer, e o valor beta, associado aos nós MIN, não pode aumentar. Suponha que o valor alfa para um nó MAX seja 6. Então, MAX não precisa considerar qualquer valor retropropagado menor ou igual a 6 que seja associado com qualquer nó MIN abaixo dele. Alfa é o pior que MAX pode “marcar”, dado que MIN também faça o “melhor” possível. Da mesma forma, se MIN tem um valor beta de 6, ele não precisa considerar qualquer nó MAX descendente que tenha um valor de 6 ou mais.

Para iniciar a busca alfa-beta, descemos até a profundidade total de camada escolhida como se fosse uma busca em profundidade e aplicamos a nossa avaliação heurística a um estado e a todos os seus irmãos. Suponha que estes sejam nós MIN. O máximo desses valores MIN é, então, retropropagado para o pai (um nó MAX, exatamente como em minimax). Esse valor é oferecido ao avô desses nós MIN como um limiar beta potencial.

Depois, o algoritmo desce até outros netos e termina a exploração de seus pais se qualquer um de seus valores for igual ou maior que esse valor beta. Podemos descrever procedimentos similares para a poda alfa sobre os netos de um nó MAX.

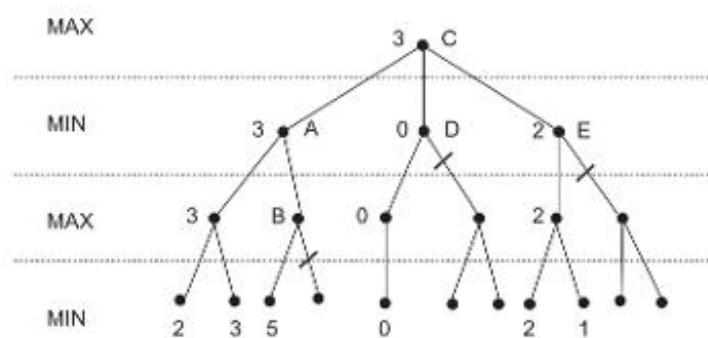
Duas regras para interromper a busca, com base nos valores alfa e beta, são:

1. A busca pode ser interrompida abaixo de qualquer nó MIN que tenha um valor beta menor, ou igual, que o valor alfa de qualquer um de seus ancestrais MAX.
2. A busca pode ser interrompida abaixo de qualquer nó MAX que tenha um valor alfa maior, ou igual, que o valor beta de qualquer um de seus nós MIN ancestrais.

Assim, a poda alfa-beta expressa uma relação entre nós na camada  $n$  e  $n + 2$ , sob as quais podem ser eliminadas subárvore com raízes no nível  $n + 1$ . Como exemplo, a Figura 4.26 toma o espaço da Figura 4.21 e aplica a poda alfa-beta. Note que o valor retropropagado resultante é idêntico ao resultado de minimax, e a economia da busca em relação ao minimax é considerável.

Com uma ordenação fortuita de estados no espaço de busca, alfa-beta pode efetivamente dobrar a profundidade do espaço de busca com um compromisso fixo de espaço/tempo de computação (Nilsson, 1980). Se houver uma ordenação desafortunada, o algoritmo alfa-beta não busca mais espaço que o minimax normal; entretanto, a busca é feita em apenas um passo.

**Figura 4.26** Poda alfa-beta aplicada ao espaço de estados da Figura 4.21. Os estados sem números não são avaliados.



A tem  $\beta = 3$  (A não será maior que 3)

B é  $\beta$  podado, pois  $5 > 3$

C tem  $\alpha = 3$  (C não será menor que 3)

D é  $\alpha$  podado, pois  $0 < 3$

E é  $\alpha$  podado, pois  $2 < 3$

C é 3

## 4.5 Aspectos de complexidade

O aspecto mais difícil dos problemas combinatórios é que a “explosão” quase sempre ocorre sem que os projetistas do programa notem que ela está acontecendo. Como a maioria da atividade humana, computacional ou não, ocorre em um mundo de tempo linear, temos dificuldade em apreciar o crescimento exponencial. Ouvimos a reclamação: “Se eu tivesse um computador maior (ou mais rápido ou maciçamente paralelo), meu problema estaria resolvido”. Essas reclamações, quase sempre feitas como resultado da explosão, geralmente são bobagens. O problema não foi entendido corretamente e/ou etapas apropriadas não foram executadas para resolver as combinatórias da situação.

A extensão total do crescimento combinatório confunde a imaginação. Estima-se que o número de estados produzidos por uma busca completa do espaço de movimentos de xadrez possíveis seja cerca de  $10^{120}$ . Isso não é “apenas outro número grande”; é comparável ao número de moléculas no universo ou ao número de nanosegundos desde o “Big Bang”.

Várias medidas foram desenvolvidas para ajudar a calcular a complexidade. Uma delas é o *fator de ramificação* de um espaço, que é definido como o número médio de ramos (filhos) que são expandidos a partir de qualquer estado no espaço. O número de estados na profundidade  $n$  da busca é igual ao fator de ramificação elevado à  $n$ -ésima potência. Quando o fator de ramificação é calculado para um espaço, é possível estimar o custo da busca

para gerar um caminho de qualquer comprimento em particular. A Figura 4.27 indica a relação entre  $B$  (ramificação),  $L$  (comprimento do caminho) e  $T$  (estados totais na busca) para valores pequenos. A escala da figura é logarítmica em  $T$ , de modo que  $L$  não é uma linha “reta”, como aparece no gráfico.

Vários exemplos usando essa figura mostram como as coisas podem ficar ruins. Se o fator de ramificação for 2, será necessária uma busca de cerca de 100 estados para examinar todos os caminhos que se estendem por seis níveis de profundidade no espaço de busca. É necessária uma busca de cerca de 10.000 estados para considerar caminhos com 12 movimentos de profundidade. Se a ramificação puder ser diminuída para 1,5 (por alguma heurística), então um caminho com o dobro da extensão pode ser examinado para o mesmo número de estados buscados.

A fórmula matemática que produziu as relações da Figura 4.27 é:

$$T = B + B^2 + B^3 + \dots + B^L$$

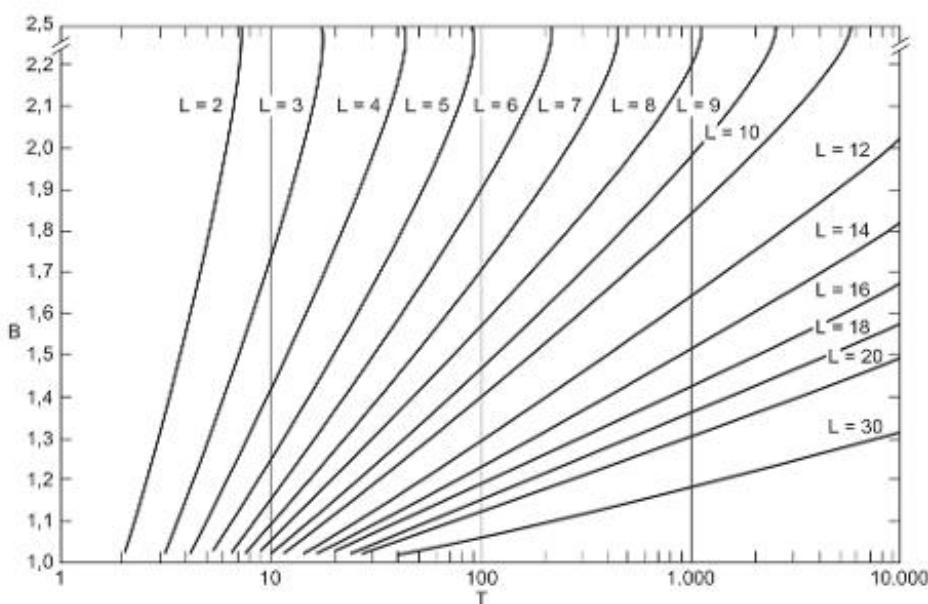
com  $T$  estados totais, comprimento de caminho  $L$  e fator de ramificação  $B$ . Essa equação se reduz a:

$$T = B(B^L - 1)/(B - 1)$$

A medição de um espaço de busca normalmente é um processo empírico feito por testes consideráveis com o problema e suas variantes. Suponha, por exemplo, que queiramos estabelecer o fator de ramificação do quebra-cabeça dos 8. Calculamos o número total de movimentos possíveis: 2 a partir de cada canto para um total de 8 movimentos de canto, 3 do centro de cada lado para um total de 12, e 4 do centro da grade para um total de 24. Isso, dividido por 9, o número de diferentes locais possíveis para o vazio, gera um fator de ramificação médio de 2,67. Como podemos ver na Figura 4.27, isso não é muito bom para uma busca profunda. Se eliminarmos movimentos diretamente de volta a um estado pai (já embutido nos algoritmos de busca desse capítulo), há um movimento a menos a partir de cada estado. Isso gera um fator de ramificação de 1,67, uma melhoria considerável, que poderia (em alguns espaços de estados) possibilitar uma busca exaustiva.

Conforme consideramos no Capítulo 3, o custo da complexidade de um algoritmo também pode ser medido pelos tamanhos das listas abertos e fechados. Um método para manter o tamanho de abertos razoável é salvar em abertos somente alguns dos melhores estados (heuristicamente). Isso pode produzir uma busca mais bem focalizada, porém há o perigo de possivelmente eliminar o caminho de melhor solução ou até mesmo a única. Essa

**Figura 4.27** Número de nós gerados como uma função do fator de ramificação,  $B$ , para vários comprimentos,  $L$ , de caminhos de solução. A equação relacionada é:  $T = B(B^L - 1)/(B - 1)$ , adaptado de Nilsson (1980).



técnica de manter um tamanho limitado em abertos, normalmente uma função do número de etapas esperadas para a busca, é chamada de *busca em feixes*.

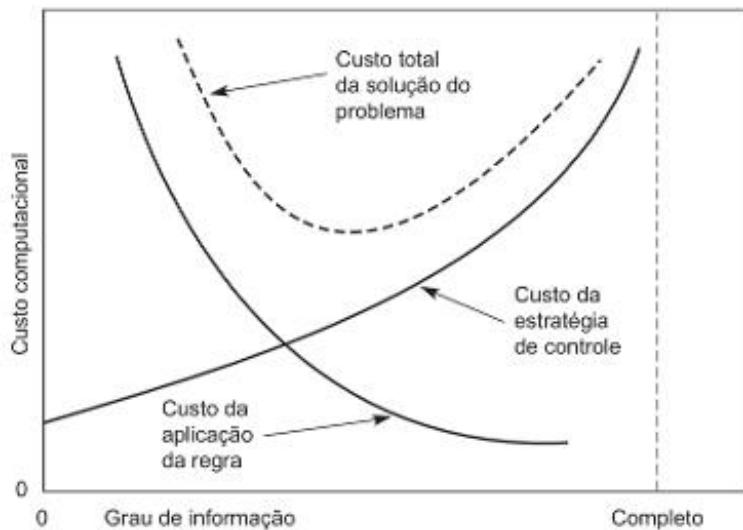
Com o intuito de diminuir a ramificação de uma busca ou de restringir o espaço de busca, apresentamos a noção de heurísticas *mais informadas*. Quanto mais informada for uma heurística, menor será o espaço que precisa ser buscado para se alcançar a solução do caminho mínimo. Como salientamos na Seção 4.4, os custos computacionais da informação adicional necessária para diminuir o espaço de busca podem não ser sempre aceitáveis. Para a solução de problemas usando computadores, não basta encontrar um caminho mínimo. Precisamos também minimizar os custos totais de CPU.

A Figura 4.28, extraída de uma análise feita por Nilsson (1980), é uma tentativa informal para se compreender essas questões. A coordenada de “grau de informação” marca a quantidade de custo de informação que está incluído na heurística de avaliação para melhorar o seu desempenho. A coordenada de CPU marca os custos de CPU correspondentes para implementar a avaliação de estado e outros aspectos da busca. Conforme a informação incluída na heurística aumenta, o custo de CPU da heurística também aumenta. De modo semelhante, conforme a heurística se torna mais informada, o custo de CPU para se avaliar os estados diminui, porque são considerados menos estados. O custo crítico, entretanto, é o custo total formado pelo custo de calcular a heurística MAIS o custo para avaliar os estados, e normalmente é desejável que esse custo seja minimizado.

Por fim, a busca heurística de grafos E/OU é uma área de interesse importante, pois os espaços de estados para sistemas especialistas quase sempre têm essa forma. A busca totalmente geral dessas estruturas é constituída de muitos dos componentes já discutidos neste capítulo e no capítulo precedente. Como todos os filhos E precisam ser pesquisados para que um objetivo seja encontrado, a estimativa heurística do custo da busca de um nó E é a soma das estimativas de se pesquisar os filhos.

Porém há muitas outras questões heurísticas além da avaliação numérica de estados E individuais, no estudo de grafos E/OU, como são usados nos sistemas baseados em conhecimento. Por exemplo, se a satisfação de um conjunto de filhos E for necessária para resolver um estado pai, qual filho deve ser considerado primeiro? O que tem o maior custo de avaliação? O que mais provavelmente falhará? O que o especialista humano considera primeiro? A decisão é importante tanto para eficiência computacional quanto para o custo geral, por exemplo, em testes de diagnóstico médico e outros, do sistema de conhecimento. Essas, além de outras questões heurísticas relacionadas, são analisadas novamente no Capítulo 8.

**Figura 4.28** Gráfico informal dos custos de busca e do cálculo da avaliação heurística em função do grau de informação da heurística, adaptado de Nilsson (1980).



## 4.6 Epílogo e referências

Os espaços de busca para problemas interessantes tendem a crescer exponencialmente; a busca heurística é uma ferramenta fundamental para lidar com a explosão combinatoria. Diversas estratégias de controle para implementar a busca heurística foram apresentadas neste capítulo.

Começamos o capítulo com dois algoritmos tradicionais, ambos herdados da disciplina da Pesquisa Operacional, subida de encosta e programação dinâmica. Recomendamos a leitura do artigo de Arthur Samuel (1959), que discute seu programa para jogar damas e seu uso sofisticado de subida de encosta e busca minimax. Samuel também apresenta primeiros exemplos interessantes de um sofisticado sistema de gerenciamento de memória e um programa que é capaz de aprender. O projeto de algoritmos para programação dinâmica de Bellman (1956) continua sendo importante em áreas como processamento da linguagem natural, onde é necessário comparar sequências de caracteres, palavras ou fonemas. A programação dinâmica muitas vezes é chamada de algoritmos para a *forward/backward* ou algoritmos de Viterbi. Para ver exemplos importantes do uso da programação dinâmica para análise da linguagem, consulte Jurafsky e Martin (2009) e o Capítulo 15.

Em seguida, apresentamos a heurística no contexto da busca tradicional no espaço de estados. Apresentamos os algoritmos A e A\* para implementar a busca pela melhor escolha. A busca heurística foi introduzida usando jogos simples, tal como o quebra-cabeça dos 8, e se estendeu a espaços de problema mais complexos gerados por sistemas especialistas baseados em regras (Capítulo 8). O capítulo aplicou também a busca heurística a jogos de dois adversários, usando antecipação com minimax e poda alfa-beta para tentar prever o comportamento do adversário. Depois de discutir os algoritmos A\*, analisamos seu comportamento, considerando propriedades como admissibilidade, monotonicidade e grau de informação.

A disciplina da teoria da complexidade tem ramificações essenciais para praticamente todos os ramos da ciência da computação, especialmente para a análise do crescimento do espaço de estados e para a poda heurística. A teoria da complexidade examina a complexidade inerente a problemas (em oposição a algoritmos aplicados a esses problemas). A conjectura-chave na teoria da complexidade é a de que existe uma classe de problemas inerentemente intratáveis. Essa classe, referida como NP (Não deterministicamente Polinomial), consiste em problemas que não podem ser resolvidos em um tempo menor que exponencial sem recorrer ao uso de heurísticas. Quase todos os problemas de busca interessantes pertencem a essa classe. Para esse assunto, recomendamos especialmente o livro *Computers and Intractability*, de Michael R. Garey e David S. Johnson (1979), e *Algorithms from P to NP, Vol. I: Design and Efficiency*, de Bernard Moret e Henry Shapiro (1991).

O livro *Heuristics*, de Judea Pearl (1984), fornece um tratamento abrangente sobre o projeto e a análise de heurísticas. R. E. Korf (1987, 1998, 1999, 2004) continua a pesquisar algoritmos de busca, incluindo uma análise do aprofundamento iterativo e o desenvolvimento do algoritmo IDA\*, que integra o aprofundamento iterativo com o A\* para obter limites lineares sobre abertos para a busca heurística. Os programas de jogo de xadrez e outros têm atraído um interesse permanente ao longo da história da IA (Hsu, 2002), sendo os resultados frequentemente apresentados e discutidos em conferências anuais.

Somos gratos a Nils Nilsson (1980) pela abordagem e pelos vários exemplos deste capítulo.

## 4.7 Exercícios

1. Estenda a heurística de “mais vitórias” para o jogo da velha com dois níveis de profundidade no espaço de buscas da Figura 4.3. Qual é o número total de estados examinados usando essa heurística? O algoritmo tradicional de subida de encosta funcionaria nessa situação? Por quê?
2. Use o componente para trás do algoritmo de programação dinâmica para achar outro alinhamento ótimo para os caracteres da Figura 4.6. Quantos alinhamentos ótimos existem?
3. Com a métrica de Levenshtein da Seção 4.1.2, use a programação dinâmica para determinar a distância de edição mínima das sequências de origem “sensação” e “excitação” até a sequência de destino “execução”.

4. Determine uma heurística que um programa de empilhamento de blocos poderia usar para resolver problemas do tipo “empilhe o bloco X sobre o bloco Y”. Essa heurística é admissível? Ela é monotônica?
5. Um quebra-cabeça de peças deslizantes consiste em três peças pretas, três peças brancas e um espaço vazio na configuração mostrada na Figura 4.29.

Esse quebra-cabeça tem dois movimentos válidos com custos associados:

Uma peça pode ser movida para uma posição adjacente vazia. Esse movimento tem um custo de 1. Uma peça pode pular sobre uma ou duas outras peças indo para um espaço vazio. Esse movimento tem um custo igual ao número de peças puladas.

O objetivo é ter todas as peças brancas à esquerda de todas as peças pretas. A posição do vazio não é importante.

- a. Analise o espaço de estados em relação a complexidade e laços.
- b. Proponha uma heurística para resolver este problema e analise-a em relação à admissibilidade, monotonicidade e grau de informação.
6. Compare as três heurísticas para o quebra-cabeça dos 8 da Figura 4.14 com a heurística de adicionar a soma das distâncias fora de lugar a duas vezes o número de inversões diretas. Compare-as em termos de:
  - a. Acurácia em estimar a distância até um objetivo. Isso requer que você primeiro derive a solução do menor caminho e a use como um padrão.
  - b. Grau de informação. Qual heurística poda mais eficientemente o espaço de estados?
  - c. Alguma dessas heurísticas para o quebra-cabeça dos 8 é monotônica?
  - d. Admissibilidade. Quais dessas heurísticas são limitadas acima pelo custo real de um caminho até o objetivo? Prove as suas conclusões para o caso geral ou então dê um contraexemplo.
7. a. Conforme apresentado no texto, a busca pela melhor escolha usa a lista fechados para implementar a detecção de laços. Qual seria o efeito de se eliminar esse teste e se basear no teste da profundidade,  $g(n)$ , para detectar laços? Compare as eficiências das duas abordagens.
  - b. A busca\_melhor\_escolha não testa um estado para ver se ele é um objetivo até que seja removido da lista abertos. Esse teste poderia ser realizado quando são gerados novos estados. Qual o efeito que isso teria sobre a eficiência do algoritmo? E sobre a admissibilidade?
8. Prove que  $A^*$  é admissível. Dica: a prova deve mostrar que:
  - a. A busca  $A^*$  terminará.
  - b. Durante a sua execução, existe sempre um nó em abertos que está em um caminho ótimo que leva até o objetivo.
  - c. Se existir um caminho até um objetivo,  $A^*$  terminará encontrando o caminho ótimo.
9. A admissibilidade implica monotonicidade de uma heurística? Se não implicar, você poderia descrever quando a admissibilidade implica monotonicidade?
10. Prove que o conjunto de estados expandidos pelo algoritmo  $A^*$  é um subconjunto daqueles examinados pela busca em amplitude.
11. Prove que heurísticas mais informadas desenvolvem o mesmo ou um menor espaço de busca. Dica: formalize o argumento apresentado na Seção 4.3.3.
12. Uma cifra de César é um esquema simples de criptografia baseado em permutações cíclicas do alfabeto, sendo a  $i$ -ésima letra substituída pela  $(i + n)$ -ésima letra do alfabeto. Por exemplo, em uma cifra de César com um deslocamento de 4, “Cesar” seria criptografado como “Giwev”.

**Figura 4.29** O quebra-cabeça de blocos deslizantes.



- a.** Determine três heurísticas que poderiam ser usadas para resolver as cifras de César.
- b.** Em uma cifra de substituição simples, cada letra é substituída por outra letra, segundo um mapeamento um para um arbitrário. Quais das heurísticas propostas pela cifra de César podem ser usadas para resolver a cifra de substituição? Explique. (Obrigado a Don Morrison por este problema.)
- 13.** Realize o procedimento minimax na árvore mostrada na Figura 4.30.
- 14.** Realize uma poda alfa-beta da esquerda para a direita na árvore do Exercício 13. Realize uma poda da direita para a esquerda nessa mesma árvore. Discuta por que o resultado é uma poda diferente.
- 15.** Considere o jogo da velha tridimensional. Discuta as questões representacionais; analise a complexidade do espaço de estados. Proponha uma heurística para esse jogo.
- 16.** Realize a poda alfa-beta para a busca do jogo da velha das figuras 4.23, 4.24 e 4.25. Quantos nós folha podem ser eliminados em cada caso?
- 17. a.** Defina um algoritmo para a busca heurística em grafos E/OU. Note que todos os descendentes de um nó E devem ser resolvidos para que o pai seja resolvido. Assim, ao calcular estimativas heurísticas de custos até um objetivo, a estimativa do custo para resolver um nó E deve ser, no mínimo, igual à soma das estimativas para resolver os ramos diferentes.
- b.** Use esse algoritmo para buscar o grafo da Figura 4.31.

Figura 4.30

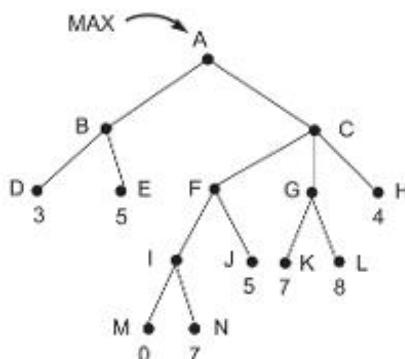
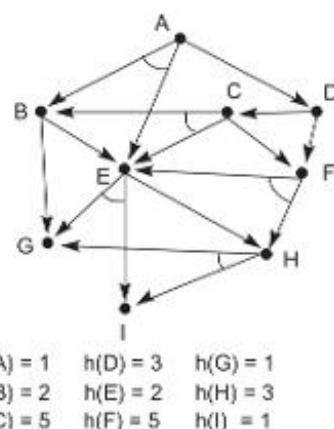


Figura 4.31



# Métodos estocásticos

*Deus não joga dados...*

—ALBERT EINSTEIN (em resposta à credibilidade da Teoria Quântica)

*Deus não apenas joga dados, mas às vezes os lança onde não podem ser vistos...*

—STEPHEN HAWKING

*Impossibilidades prováveis são melhores que possibilidades improváveis...*

—ARISTÓTELES, citado por Bobby Wolfe em “Aces on Bridge”, 2003

## 5.0 Introdução

O Capítulo 4 introduziu a busca heurística como uma abordagem para a solução do problema em domínios em que um problema não tem uma solução exata ou o espaço de estado completo pode ser muito dispendioso para ser calculado. Neste capítulo, propomos uma metodologia estocástica como apropriada para essas situações. O raciocínio probabilístico também é adequado para situações em que a informação de estado é determinada pela amostragem de uma base de informações e por modelos causais que são descobertos a partir dos dados.

Um domínio de aplicação importante para o uso da metodologia estocástica é o raciocínio diagnóstico, em que relações de causa/efeito nem sempre são captadas em um padrão puramente determinístico, como normalmente é possível nas abordagens baseadas em conhecimento para a solução de problemas que vimos nos capítulos 2, 3, 4 e veremos novamente no Capítulo 8. Uma situação de diagnóstico normalmente apresenta evidência, como febre ou dor de cabeça, sem outra justificativa causadora. Na verdade, a evidência muitas vezes pode ser indicação de várias causas diferentes, por exemplo, a febre pode ser causada por uma gripe ou uma infecção. Nessas situações, a informação probabilística quase sempre pode indicar e priorizar explicações possíveis para a evidência.

A metodologia estocástica tem outra aplicação interessante nos jogos de apostas, em que eventos supostamente aleatórios, como o lançamento de dados, a distribuição de cartas embaralhadas ou o giro de uma roda de roleta produzem ganhos possíveis ao jogador. De fato, no século XVIII, a tentativa de oferecer uma base matemática aos jogos foi uma motivação importante para que Pascal (e, depois, Laplace) desenvolvesse um cálculo probabilístico.

Por fim, conforme observado na Seção 1.1.4, um relato “situado” de inteligência sugere que as decisões humanas quase sempre surgem de ambientes complexos, críticos no tempo e materializados em que um cálculo totalmente mecanizado pode simplesmente não ser definível ou, se definido, pode não computar respostas em um es-

paço de tempo utilizável. Nessas situações, ações inteligentes podem ser mais bem-vistas como respostas estocásticas a custos e benefícios antecipados.

A seguir, descrevemos várias áreas de problemas, entre muitas, em que a metodologia estocástica é muito usada na implementação computacional da inteligência; essas áreas serão tópicos importantes em outros capítulos.

- 1. Raciocínio diagnóstico.** Em diagnóstico médico, por exemplo, nem sempre há uma relação óbvia de causa/efeito entre o conjunto de sintomas apresentados pelo paciente e as causas desses sintomas. De fato, os mesmos conjuntos de sintomas muitas vezes sugerem múltiplas causas possíveis. Os modelos probabilísticos também são importantes em situações mecânicas complexas, como o monitoramento de sistemas de voo de aeronave ou helicóptero. Sistemas baseados em regra (Capítulo 8) e probabilísticos (capítulos 9 e 13) têm sido aplicados nesses e em outros domínios de diagnóstico.
- 2. Compreensão da linguagem natural.** Se um computador tiver que entender e usar uma linguagem humana, ele precisará ser capaz de caracterizar como os próprios humanos usam essa linguagem. Palavras, expressões e metáforas são aprendidas, mas também mudam e evoluem ao longo do tempo em que são usadas. A metodologia estocástica admite a compreensão da linguagem; por exemplo, quando um sistema computacional é treinado em um banco de dados de uso da linguagem específica (denominado *corpo linguístico*). Consideramos essas questões da linguagem mais adiante neste capítulo, bem como no Capítulo 15.
- 3. Planejamento e escalonamento.** Quando um agente planeja uma viagem de férias de automóvel, por exemplo, geralmente nenhuma sequência determinística de operações tem garantias de sucesso. O que acontecerá se o carro enguiçar, se uma balsa de travessia for cancelada em um dia específico, se um hotel estiver cheio, mesmo que você tenha feito reserva? Planos específicos, seja para humanos, seja para robôs, normalmente são expressos em linguagem probabilística. O planejamento é considerado na Seção 8.4.
- 4. Aprendizado.** As três áreas mencionadas anteriormente também podem ser vistas como domínios para o aprendizado automatizado. Um componente importante de muitos sistemas estocásticos é que eles têm a capacidade de amostrar situações e aprender com o tempo. Alguns sistemas sofisticados são capazes de amostrar dados e prever resultados, além de aprender novas relações probabilísticas com base em dados e resultados. Consideramos o aprendizado na Parte IV e as técnicas estocásticas de aprendizado no Capítulo 13.

A metodologia estocástica tem suas bases nas propriedades da contagem. A probabilidade de um evento em uma situação é descrita como a razão entre o número de maneiras como o evento pode ocorrer e o número total de resultados possíveis desse evento. Assim, a probabilidade de que um número par resulte do lançamento de um dado comum é o número total de resultados pares (aqui, 2, 4 ou 6) sobre o número total de resultados (1, 2, 3, 4, 5 ou 6), ou  $1/2$ . Novamente, a probabilidade de apanhar uma bola de gude de certa cor dentro de um saco de bolas é a razão entre o número de bolas dessa cor e o número total de bolas nesse saco. Na Seção 5.1, apresentamos as técnicas básicas de contagem, incluindo as regras da soma e do produto. Devido à sua importância na contagem, também apresentamos as *permutações* e as *combinações* de eventos discretos. Esta é uma seção opcional, que pode ser pulada pelos leitores com uma boa base em matemática discreta.

Na Seção 5.2, apresentamos uma linguagem formal para o raciocínio com a metodologia estocástica. Isso inclui as definições de independência e diferentes tipos de variáveis aleatórias. Por exemplo, para proposições probabilísticas, variáveis aleatórias podem ser *lógicas* (verdadeiras ou falsas), *discretas*, tendo os inteiros de 1 a 6 como em um dado, ou *contínuas*, uma função definida sobre os números reais.

Na Seção 5.3, apresentamos o teorema de Bayes, que dá suporte à maioria das abordagens de modelagem e aprendizado estocásticos (Seção 9.3 e Capítulo 13). A regra de Bayes é importante para interpretar a nova evidência no contexto do conhecimento ou da experiência anterior. Na Seção 5.4, apresentamos duas aplicações de métodos estocásticos, incluindo autômatos de estado finito probabilístico e uma metodologia para prever padrões de palavras em inglês com base nos dados amostrados.

Nos capítulos 9 e 13, continuamos a apresentação de abordagens probabilísticas para o raciocínio, incluindo as *redes Bayesianas de crença* (RBCs; ou BBNs, do inglês *Bayesian Belief Networks*), uma introdução aos *Mode-*

*los de Markov* (MOMs) e aos sistemas de representação de primeira ordem que dão suporte à modelagem estocástica. Esses modelos usam o formalismo do grafo acíclico direcionado (ou DAG) e são conhecidos como *modelos gráficos*. No Capítulo 13, apresentamos as abordagens estocásticas para o aprendizado de máquina.

## 5.1 Elementos da contagem (opcional)

A base para a metodologia estocástica é a capacidade de contar os elementos de um domínio de aplicação. A base para coletar e contar elementos, naturalmente, é a teoria dos conjuntos, em que devemos conseguir determinar inequivocamente se um elemento é ou não um membro de um conjunto de elementos. Quando isso é determinado, existem metodologias para a contagem de elementos de conjuntos, do complemento de um conjunto e da união e da interseção de vários conjuntos. Nesta seção, faremos uma revisão dessas técnicas.

### 5.1.1 Regras da adição e da multiplicação

Se tivermos um conjunto  $A$ , o *número de elementos* no conjunto  $A$  é indicado por  $|A|$ , chamado de *cardinalidade* de  $A$ . Naturalmente,  $A$  pode ser vazio (o número de elementos é zero), finito, infinito contável ou infinito incontável. Cada conjunto é definido em termos de um domínio de interesse ou *universo*,  $U$ , de elementos que poderiam estar nesse conjunto. Por exemplo, o conjunto de homens em uma sala de aula pode ser definido no contexto, ou no universo, de todas as pessoas nessa sala. De modo semelhante, o resultado 3 em um lançamento de dados pode ser visto como um resultado de um conjunto de seis possíveis.

O domínio ou o universo de um conjunto  $A$ , portanto, também é um conjunto e é usado para determinar o *complemento* desse conjunto,  $\bar{A}$ . Por exemplo, o complemento do conjunto de todos os homens na sala de aula, que acabamos de mencionar, é o conjunto de todas as mulheres, e o complemento de  $\{3\}$  no lançamento de um dado é  $\{1, 2, 4, 5, 6\}$ . Um conjunto  $A$  é um *subconjunto* de outro conjunto  $B$ ,  $A \subseteq B$ , se cada elemento do conjunto  $A$  for também um elemento do conjunto  $B$ . Assim, de forma trivial, cada conjunto é um subconjunto de si mesmo, qualquer conjunto  $A$  é um subconjunto de seu universo e o conjunto vazio, indicado por  $\{\}$  ou por  $\phi$ , é um subconjunto de todo conjunto.

A *união* de dois conjuntos  $A$  e  $B$ ,  $A \cup B$ , pode ser descrita como o conjunto de todos os elementos em ambos os conjuntos. O número de elementos na união de dois conjuntos é o total de todos os elementos em cada um dos conjuntos menos o número de elementos que estão nos dois conjuntos. A justificativa para isso, logicamente, é o fato de que cada elemento distinto em um conjunto só pode ser contado uma vez. Trivialmente, se os dois conjuntos não tiverem elementos em comum, o número de elementos em sua união é a soma do número de elementos em cada conjunto.

A *interseção* de dois conjuntos  $A$  e  $B$ ,  $A \cap B$ , é o conjunto de todos os elementos comuns aos dois conjuntos. A seguir, damos exemplos de diversos conceitos recém-definidos.

Suponha que o universo,  $U$ , seja o conjunto  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Seja  $A$  o conjunto  $\{1, 3, 5, 7, 9\}$

Seja  $B$  o conjunto  $\{0, 2, 4, 6, 8\}$

Seja  $C$  o conjunto  $\{4, 5, 6\}$

Então,  $|A| = 5$ ,  $|B| = 5$ ,  $|C| = 3$  e  $|U| = 10$ .

Além disso,  $A \subseteq U$ ,  $B \subseteq U$  e  $A \cup B = U$ ,  $|B| = |A|$ ,  $A = \bar{B}$

Ainda,  $|A \cup B| = |A| + |B| = 10$ , pois  $A \cap B = \{\}$ , mas

$|A \cup C| = |A| + |C| - |A \cap C| = 7$ , pois  $A \cap C = \{5\}$

Acabamos de apresentar os componentes principais para a *regra da adição* para combinar dois conjuntos. Para dois conjuntos A e C quaisquer, o número de elementos na união desses conjuntos é:

$$|A \cup C| = |A| + |C| - |A \cap C|$$

Observe que essa regra da adição é verdadeira se os dois conjuntos forem disjuntos ou se possuírem elementos em comum. Uma regra da adição semelhante é novamente verdadeira para três conjuntos, A, B e C, se eles possuirem elementos em comum ou não:

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

Um argumento semelhante ao que foi feito anteriormente pode ser usado para justificar essa equação. Existem equações de inclusão/exclusão semelhantes, e facilmente demonstradas, para a adição de conjuntos de elementos de mais de três conjuntos.

O princípio da multiplicação para contagem afirma que, se tivermos dois conjuntos de elementos A e B de tamanhos a e b, respectivamente, então existem  $a \times b$  maneiras exclusivas de combinar os elementos dos conjuntos. A justificativa para isso, naturalmente, é que, para cada um dos a elementos de A, existem b emparelhamentos para esse elemento. O princípio da multiplicação dá suporte a muitas das técnicas usadas na contagem, incluindo o produto cartesiano dos conjuntos, além de permutações e combinações de conjuntos.

O produto cartesiano de dois conjuntos A e B, indicado por  $A \times B$ , é o conjunto de todos os pares ordenados  $(a, b)$  onde a é um elemento do conjunto A, e b é um elemento do conjunto B; ou mais formalmente:

$$A \times B = \{(a, b) \mid (a \in A) \wedge (b \in B)\}$$

e, pelo princípio da multiplicação da contagem:

$$|A \times B| = |A| \times |B|$$

O produto cartesiano pode, é claro, ser definido para qualquer número de conjuntos. O produto para n conjuntos será o conjunto de n-tuplas, onde o primeiro componente da n-tupla é qualquer elemento do primeiro conjunto, o segundo componente da n-tupla é qualquer elemento do segundo conjunto, e assim por diante. Novamente, o número de n-tuplas distintas resultantes é o produto do número de elementos em cada conjunto.

### 5.1.2 Permutações e combinações

Uma *permutação* de um conjunto de elementos é uma sequência organizada dos elementos desse conjunto. Nessa arrumação de elementos, cada um pode ser usado apenas uma vez. Um exemplo de permutação é uma arrumação ou ordenação de um conjunto de dez livros em uma prateleira que pode conter todos os dez. Outro exemplo é a atribuição de tarefas específicas a quatro de um grupo de seis crianças.

Muitas vezes, queremos saber quantas permutações (distintas) existem de um conjunto de n elementos. Usamos a multiplicação para determinar isso. Se houver n elementos no conjunto A, então o conjunto de permutações desses elementos é uma sequência de tamanho n, onde o primeiro elemento da sequência é qualquer um dos n elementos de A, o segundo elemento da sequência é qualquer um dos  $(n - 1)$  elementos restantes de A, o terceiro elemento da sequência é qualquer um dos  $(n - 2)$  elementos restantes, e assim por diante.

A ordem em que os elementos são colocados na sequência de permutação não é importante, ou seja, qualquer um dos n elementos do conjunto pode ser colocado primeiro em qualquer local na sequência, qualquer um dos n – 1 elementos restantes pode ser colocado em segundo lugar em qualquer um dos n – 1 locais restantes da sequência de permutação, e assim por diante. Por fim, pelo princípio da multiplicação, existem  $n!$  sequências de permutação para esse conjunto de n elementos.

Podemos restringir o número de elementos em uma permutação de um conjunto A para ser qualquer número maior ou igual a zero e menor ou igual ao número de elementos n no conjunto original A. Por exemplo, poderíamos querer saber quantas ordenações distintas existem de dez livros possíveis em uma prateleira que pode conter apenas seis deles de cada vez. Se quiséssemos determinar o número de permutações dos n elementos de A toma-

dos  $r$  de cada vez, onde  $0 \leq r \leq n$ , usamos a multiplicação como antes, exceto que agora só temos  $r$  posições em cada sequência de permutação:

$$n \times (n - 1) \times (n - 2) \times (n - 3) \times \dots \times (n - (r - 1))$$

Como alternativa, podemos representar essa equação como:

$$\frac{n \times (n - 1) \times (n - 2) \times (n - 3) \times \dots \times (n - (r - 1)) \times (n - r) \times (n - r - 1) \times \dots \times 2 \times 1}{(n - r) \times (n - r - 1) \times \dots \times 2 \times 1}$$

ou, de modo equivalente, o número de permutações de  $n$  elementos tomados  $r$  de cada vez, o que é simbolizado como  ${}_nP_r$ , é:

$${}_nP_r = \frac{n!}{(n - r)!}$$

A *combinação* de um conjunto de  $n$  elementos é qualquer *subconjunto* desses elementos que pode ser formado. Assim como as permutações, quase sempre queremos contar o número de combinações de itens que podem ser formadas, dado um conjunto de itens. Assim, há somente uma combinação dos  $n$  elementos de um conjunto de  $n$  itens. A ideia principal aqui é que o número de combinações represente o número de *subconjuntos do conjunto completo de elementos* que podem ser criados. No exemplo da prateleira, combinações representam os diferentes subconjuntos de seis livros, os livros na prateleira, que podem ser formados a partir do conjunto completo de dez livros. Outro exemplo de combinações é a tarefa de formar comitês de quatro membros a partir de um grupo de quinze pessoas. Cada pessoa está no comitê ou não, e não faz diferença se eles são o primeiro ou o último membro escolhido. Outro exemplo é uma mão de cinco cartas em um jogo de pôquer. A ordem em que as cartas são distribuídas não faz diferença para o valor final da mão. (Isso pode fazer uma diferença enorme para o valor da aposta, e as últimas quatro cartas são distribuídas viradas para cima, como em um pôquer stud tradicional, mas o valor final da mão independe da ordem da distribuição.)

O número de combinações de  $n$  elementos tomados  $r$  de cada vez, onde  $0 \leq r \leq n$ , é simbolizado por  ${}_nC_r$ . Um método direto para determinar o número dessas combinações consiste em tomar o número de permutações,  ${}_nP_r$ , como já fizemos, e depois dividir o número de conjuntos duplicados. Como qualquer subconjunto de  $r$  elementos dos  $n$  elementos tem  $r!$  permutações, para chegar ao número de combinações de  $n$  elementos tomados  $r$  de cada vez, dividimos o número de permutações de  $n$  elementos tomados  $r$  de cada vez por  $r!$ . Assim, temos:

$${}_nC_r = \frac{{}_nP_r}{r!} = \frac{n!}{(n - r)! r!}$$

Existem muitas outras variações dos princípios de contagem recém-apresentados e algumas delas aparecem como exercícios no Capítulo 5. Recomendamos qualquer livro-texto de matemática discreta para desenvolver melhor essas técnicas de contagem.

## 5.2 Elementos de teoria da probabilidade

Com base nas regras de contagem apresentadas na Seção 5.1, podemos agora introduzir a teoria da probabilidade. Primeiro, na Seção 5.2.1, consideraremos algumas definições fundamentais, como a noção de se dois ou mais eventos são independentes um do outro. Na Seção 5.2.2, demonstramos como deduzir explicações para determinados conjuntos de dados. Isso nos preparará para considerar vários exemplos de inferência probabilística na Seção 5.3 e o teorema de Bayes na Seção 5.4.

### 5.2.1 Espaço amostral, probabilidades e independência

As definições a seguir, bases para uma teoria da probabilidade, foram formalizadas inicialmente pelo matemático francês Laplace (1816) no início do século XIX. Conforme dissemos na introdução ao Capítulo 5, Laplace estava no processo de criação de um cálculo para jogos de apostas!

## Definição

### EVENTO ELEMENTAR

Um *evento elementar* ou *atômico* é um acontecimento ou uma ocorrência que não pode ser dividido em outros eventos.

#### EVENTO, E

Um *evento* é um conjunto de eventos elementares.

#### ESPAÇO AMOSTRAL, S

O conjunto de todos os resultados possíveis de um evento E é o *espaço amostral* S ou o *universo* para esse evento.

#### PROBABILIDADE, p

A *probabilidade* de um evento E em um espaço amostral S é a razão entre o número de elementos em E e o número total de resultados possíveis do espaço amostral S de E. Assim,  $p(E) = |E| / |S|$ .

Por exemplo, qual é a probabilidade de um 7 ou um 11 serem o resultado do lançamento de dois dados não viciados? Primeiro, determinamos o espaço amostral para essa situação. Usando o princípio da multiplicação da contagem, cada dado tem 6 resultados, de modo que o conjunto total de resultados dos dois dados é 36. O número de combinações dos dois dados que podem gerar um 7 é 1,6; 2,5; 3,4; 4,3; 5,2; e 6,1 – 6 no total. A probabilidade de gerar um resultado 7 é, portanto,  $6/36 = 1/6$ . O número de combinações dos dois dados que podem resultar em 11 é 5,6; 6,5 – ou 2, e a probabilidade de gerar um resultado 11 é  $2/36 = 1/18$ . Usando a propriedade aditiva dos resultados distintos, há uma probabilidade de  $1/6 + 1/18$ , ou  $2/9$ , de que o resultado seja um 7 ou um 11 com dois dados não viciados.

Nesse exemplo de 7/11, os dois eventos são conseguir um 7 e conseguir um 11. Os eventos elementares são os resultados distintos de lançar os dois dados. Assim, o evento de um 7 é composto de seis eventos atômicos (1,6), (2,5), (3,4), (4,3), (5,2) e (6,1). O espaço amostral completo é a união dos trinta e seis eventos atômicos possíveis, o conjunto de todos os pares que resultam do lançamento dos dados. Como veremos logo, como os eventos de conseguir um 7 e conseguir um 11 não possuem eventos atômicos em comum, eles são *independentes*, e a probabilidade de sua soma (união) é simplesmente a soma de suas probabilidades individuais.

Em um segundo exemplo, quantas mãos com quadra podem ser distribuídas em todas as mãos de pôquer possíveis com cinco cartas? Primeiro, o conjunto de eventos atômicos que compõem o espaço completo de todas as mãos de cinco cartas do pôquer é a combinação de 52 cartas tomadas 5 de cada vez. Para obter o número total de mãos de quadra, usamos o princípio da multiplicação. Multiplicamos o número de combinações de 13 cartas tomadas 1 por vez (o número de tipos de cartas diferentes: ás, 2, 3, ..., rei) vezes o número de maneiras de escolher as quatro cartas do mesmo tipo (a combinação de 4 cartas tomadas 4 por vez) vezes o número de outras cartas possíveis que preenchem a mão de 5 cartas (restam 48 cartas). Assim, a probabilidade de uma mão de pôquer com quadra é:

$$(13C_1 \times 4C_4 \times 48C_1) / 52C_5 = 13 \times 1 \times 48 / 2.598.960 \approx 0,00024$$

Vários resultados seguem imediatamente das definições que acabaram de ser feitas. Primeiro, a probabilidade de qualquer evento E no espaço amostral S é:

$$0 \leq p(E) \leq 1, \text{ onde } E \subseteq S$$

Um segundo resultado é que a soma das probabilidades de todos os resultados possíveis em S é 1. Para ver isso, observe que a definição para o espaço amostral S indica que ele é composto da união de todos os eventos individuais E no problema.

Como um terceiro resultado das definições, observe que a *probabilidade do complemento* de um evento é:

$$p(\bar{E}) = (|S| - |E|) / |S| = (|S| / |S|) - (|E| / |S|) = 1 - p(E).$$

O complemento de um evento é uma relação importante. Muitas vezes, é mais fácil determinar a probabilidade de um evento acontecer como uma função de ele não acontecer; por exemplo, determinar a probabilidade de que pelo menos um elemento de uma sequência de bits de tamanho  $n$  gerada aleatoriamente seja um 1. O complemento é que todos os bits na sequência são 0, com a probabilidade  $2^{-n}$ , com  $n$  sendo o tamanho da sequência. Assim, a probabilidade do evento original é  $1 - 2^{-n}$ .

Por fim, pela probabilidade do complemento de um conjunto de eventos, desenvolvemos a probabilidade de quando nenhum evento ocorre, às vezes denominada resultado *contraditório ou falso*:

$$\begin{aligned} p(\{\}) &= 1 - p(\bar{\{\}}) = 1 - p(S) = 1 - 1 = 0, \text{ ou, como alternativa,} \\ &= |\{\}| / |S| = 0 / |S| = 0 \end{aligned}$$

Uma última relação importante, a probabilidade da união de dois conjuntos de eventos, pode ser determinada a partir do princípio da contagem, apresentado na Seção 5.1, a saber, que, para dois conjuntos quaisquer  $A$  e  $B$ :  $|A \cup B| = |A| + |B| - |A \cap B|$ . Por essa relação, podemos determinar a probabilidade da união de dois conjuntos quaisquer tomados do espaço amostral  $S$ :

$$\begin{aligned} p(A \cup B) &= |A \cup B| / |S| = (|A| + |B| - |A \cap B|) / |S| \\ &= |A| / |S| + |B| / |S| - |A \cap B| / |S| = p(A) + p(B) - p(A \cap B) \end{aligned}$$

Naturalmente, esse resultado pode ser estendido para a união de qualquer número de conjuntos, segundo o modelo do princípio de inclusão/exclusão apresentado na Seção 5.1.

Já apresentamos um exemplo para determinar a probabilidade da união de dois conjuntos: a probabilidade de gerar um 7 ou um 11 com dois dados não viciados. Nesse exemplo, a fórmula recém-apresentada foi usada com a probabilidade de os pares de dados que geraram um 7 separados dos pares de dados que geraram um 11. Também podemos usar essa fórmula no caso mais geral em que os conjuntos não são disjuntos. Suponha que quiséssemos determinar, lançando dois dados não viciados, a probabilidade de gerar um 8 ou de gerar pares do mesmo número. Simplesmente calculariamos a probabilidade dessa união em que há um evento elementar — (4,4) — que está na interseção de ambos os eventos de resultados desejados.

Em seguida, consideraremos a probabilidade de dois eventos independentes. Suponha que você seja um jogador em um jogo de cartas para quatro pessoas em que todas as cartas são distribuídas de forma igual. Se você não tem a dama de espadas, pode concluir que há uma probabilidade de  $1/3$  de que cada um dos outros jogadores a tenha. De modo semelhante, você pode concluir que há uma probabilidade de  $1/3$  de que cada jogador tenha o ás de copas e de  $1/3 \times 1/3$ , ou  $1/9$ , de que qualquer jogador tenha as duas cartas. Nessa situação, consideramos que os eventos de ter essas duas cartas são independentes, embora isso seja apenas quase verdadeiro. Formalizamos essa intuição com uma definição.

### *Definição*

#### EVENTOS INDEPENDENTES

Dois eventos  $A$  e  $B$  são *independentes* se, e somente se, a probabilidade de ambos ocorrerem for igual ao produto de sua ocorrência individual. Essa relação de independência é expressa com:

$$p(A \cap B) = p(A) * p(B)$$

Às vezes, usamos a notação equivalente  $p(s,d)$  para  $p(s \cap d)$ . Esclarecemos melhor a noção de independência no contexto de probabilidades condicionais, na Seção 5.2.4.

Como a descrição de independência de eventos conforme apresentada é uma relação *se, e somente se*, podemos determinar se dois eventos são independentes desenvolvendo suas relações probabilísticas. Considere a

situação em que sequências de bits de tamanho quatro são geradas aleatoriamente. Queremos saber se o evento da sequência de bits conter um número par de 1s é independente do evento em que a sequência de bits termina com 0. Usando o princípio da multiplicação, com cada bit tendo 2 valores possíveis, há um total de  $2^4 = 16$  sequências de bits de tamanho 4.

Há 8 sequências de bits de tamanho quatro que terminam com 0: {1110, 1100, 1010, 1000, 0010, 0100, 0110, 0000}. Há também 8 sequências de bits que têm um número par de 1s: {1111, 1100, 1010, 1001, 0110, 0101, 0011, 0000}. O número de sequências de bits que têm um número par de 1s e terminam com 0 é 4: {1100, 1010, 0110, 0000}. Esses dois eventos, portanto, são independentes, pois:

$$\begin{aligned} p(\{\text{número par de 1s}\} \cap \{\text{acaba com 0}\}) &= p(\{\text{número par de 1s}\}) \times p(\{\text{acaba com 0}\}) \\ 4/16 &= 8/16 \times 8/16 = 1/4 \end{aligned}$$

Considere esse mesmo exemplo de sequências de bits de tamanho quatro geradas aleatoriamente. Os dois eventos a seguir são independentes: *as sequências de bits têm um número par de 1s* e *as sequências de bits terminam com 1?* Quando dois ou mais eventos não são independentes, ou seja, a probabilidade de qualquer evento afetar a probabilidade dos outros, é preciso que haja a noção de *probabilidade condicional* para desenvolver suas relações. Vemos isso na Seção 5.2.4.

Antes de encerrar essa seção, notamos que outros sistemas de axioma que dão suporte aos alicerces da teoria da probabilidade são possíveis, por exemplo, como uma extensão ao cálculo proposicional (Seção 2.1). Como um exemplo da abordagem baseada em conjunto que usamos, o matemático russo Kolmogorov (1950) propôs uma variante dos axiomas a seguir, equivalentes às nossas definições. A partir desses três axiomas, Kolmogorov construiu sistematicamente toda a teoria da probabilidade.

1. A probabilidade do evento E no espaço amostral S está entre 0 e 1, ou seja,  $0 \leq p(E) \leq 1$ .
2. Quando a união de todo E = S,  $p(S) = 1$ , e  $p(\bar{S}) = 0$ .
3. A probabilidade da união de dois conjuntos de eventos A e B é:

$$p(A \cup B) = p(A) + p(B) - p(A \cap B)$$

Na próxima seção, apresentamos um exemplo simples de raciocínio probabilístico.

### 5.2.2 Inferência probabilística: um exemplo

Agora, demonstraremos exemplos de raciocínio com as ideias que acabamos de apresentar. Suponha que você esteja dirigindo por uma rodovia interestadual e observe que está gradualmente diminuindo a velocidade devido a um aumento de congestionamento. Você começa a procurar explicações possíveis para a diminuição de velocidade. Poderia ser uma obra na estrada? Houve um acidente? Tudo o que você sabe é que sua velocidade está diminuindo. Mas espere! Você tem acesso às estatísticas de rodovias estaduais e, com sua nova interface gráfica baseada no automóvel e o sistema de dedução, você pode baixar no computador do seu carro as informações estatísticas relevantes. Tudo bem, você tem os dados; mas o que pode fazer com eles?

Para esse exemplo, consideraremos que temos três parâmetros verdadeiro ou falso (definiremos esse parâmetro de tipo como uma *variável aleatória lógica* na Seção 5.2.4). Primeiro, há a questão de se o tráfego — e você — está diminuindo a velocidade ou não. Essa situação será rotulada com V, com definições de v ou f. Segundo, existe a probabilidade de haver ou não ocorrido um acidente, A, com definições de v ou f. Por fim, a probabilidade de a estrada estar em obras no momento, O; novamente, v ou f. Podemos expressar essas relações para o tráfego na rodovia interestadual graças ao nosso sistema de download de dados para o carro, na Tabela 5.1.

**Tabela 5.1** A distribuição de probabilidade conjunta para as variáveis de diminuição de velocidade, V, acidente, A e obras, O, do exemplo da Seção 5.2.2.

V	O	A	p
v	v	v	0,01
v	v	f	0,03
v	f	v	0,16
v	f	f	0,12
f	v	v	0,01
f	v	f	0,05
f	f	v	0,01
f	f	f	0,61

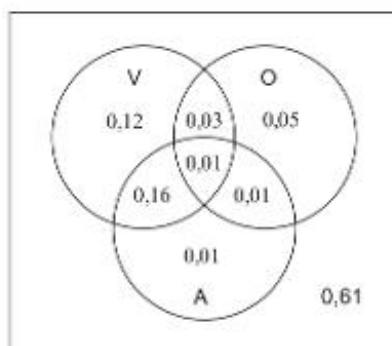
As entradas da Tabela 5.1 são interpretadas, é claro, como nas tabelas verdade da Seção 2.1, exceto que a coluna da direita fornece a probabilidade da situação à esquerda acontecer. Assim, a terceira linha da tabela mostra as probabilidades de diminuição da velocidade e de haver um acidente, mas de nenhuma obra, como 0,16:

$$V \cap \bar{O} \cap A = 0,16$$

Deve-se notar que estivemos desenvolvendo nosso cálculo probabilístico no contexto de *conjuntos* de eventos. A Figura 5.1 demonstra como as probabilidades da Tabela 5.1 podem ser representadas com o diagrama de Venn tradicional. Poderíamos, da mesma forma, ter apresentado essa situação como atribuições de verdade probabilísticas de *proposições*, quando o  $\cap$  seria substituído por um  $\wedge$  e a Tabela 5.1 seria interpretada como valores verdade da conjunção de proposições.

Em seguida, observamos que a soma de todos os resultados possíveis da distribuição conjunta da Tabela 5.1 é 1,0; isso é o que se poderia esperar dos axiomas da probabilidade apresentados na Seção 5.2.1. Também podemos desenvolver a probabilidade de qualquer conjunto de eventos simples ou complexo. Por exemplo, podemos calcular a probabilidade de que haja diminuição de velocidade V. O valor para a diminuição de velocidade é 0,32, a soma das quatro primeiras linhas da Tabela 5.1; ou seja, todas as situações em que  $V = v$ . Isso, às vezes, é chamado de *probabilidade incondicional* ou *marginal* de diminuição de velocidade, V. Esse processo é chamado de *marginalização*, pois todas as probabilidades que não sejam a de diminuição de velocidade são somadas. Ou seja, a distribuição de uma variável pode ser obtida somando-se todas as outras variáveis da distribuição conjunta contendo essa variável.

**Figura 5.1** Uma representação do diagrama de Venn para as distribuições de probabilidade da Tabela 5.1; V é a diminuição da velocidade, A é acidente, O é obras.



De modo semelhante, podemos calcular a probabilidade de obras O sem diminuição de velocidade  $\bar{V}$  — um fenômeno não muito raro no estado do Novo México! Essa situação é capturada por  $p(O \cap \bar{V}) = v$ , como a soma da 5<sup>a</sup> e da 6<sup>a</sup> linhas da Tabela 5.1, ou 0,06. Se considerássemos a negação da situação  $O \cap \bar{V}$ , obteríamos (usando as leis de deMorgan),  $p(\bar{O} \cup V)$ . Calculando a probabilidade da união de dois conjuntos, conforme apresentamos na Seção 5.2.1, obtemos:

$$0,16 + 0,12 + 0,01 + 0,61 + 0,01 + 0,03 + 0,16 + 0,12 - (0,16 + 0,12) = 0,94$$

E, novamente, a probabilidade total de  $O \cap \bar{V}$  e seu complemento (negação) é 1,0.

### 5.2.3 Variáveis aleatórias

Na teoria da probabilidade, probabilidades individuais são calculadas analiticamente, por métodos combinatorios, ou empiricamente, amostrando uma população de eventos. Até este ponto, a maioria de nossas probabilidades tem sido determinada analiticamente. Por exemplo, existem seis lados em um dado, e dois lados em uma moeda. Quando o dado ou a moeda são “não viciados”, dizemos que cada resultado, pelo lançamento do dado ou da moeda, é igualmente provável. Como resultado, é simples determinar o espaço de eventos para essas situações de problema, que mais tarde chamaremos de *paramétrico*.

Porém o raciocínio probabilístico mais interessante resulta da análise baseada em amostragens de situações no mundo real dos eventos. Essas situações muitas vezes não possuem uma especificação bem definida que dê suporte ao cálculo analítico de probabilidades. Também acontece que algumas situações, mesmo quando existe uma base analítica, são tão complexas que os custos em tempo e computação não são suficientes para o cálculo determinístico de resultados probabilísticos. Nessas situações, normalmente adotamos uma metodologia de amostragem empírica.

Mais importante, consideramos que todos os resultados de um experimento não são igualmente prováveis. Porém retemos os axiomas ou as suposições básicas que vimos nas seções anteriores; a saber, que a probabilidade de um evento é um número entre (e incluindo) 0 e 1, e que as probabilidades de todos os resultados somam 1. Também retemos nossa regra para a probabilidade da união de conjuntos de eventos. Definimos a ideia de *variável aleatória* como um método para tornar esse cálculo preciso.

#### *Definição*

#### VARIÁVEL ALEATÓRIA

Uma *variável aleatória* é uma função cujo domínio é um espaço amostral e cujo contradomínio é um conjunto de resultados, quase sempre números reais. Em vez de usar um espaço de eventos específico do problema, uma variável aleatória nos permite falar sobre as probabilidades como valores numéricos que estão relacionados a um espaço de eventos.

#### VARIÁVEIS ALEATÓRIAS LÓGICAS, DISCRETAS e CONTÍNUAS

Uma *variável aleatória lógica* é uma função de um espaço de eventos para {verdadeiro, falso} ou para o subconjunto de números reais {0,0; 1,0}. Uma variável aleatória lógica também é chamada de *tentativa de Bernoulli*.

Uma *variável aleatória discreta*, que inclui variáveis aleatórias lógicas como um subconjunto, é uma função do espaço amostral para (um subconjunto contável de) números reais em [0,0; 1,0].

Uma *variável aleatória contínua* tem como contradomínio o conjunto de números reais.

Em seguida, começamos a apresentação do teorema de Bayes, cuja forma geral é vista na Seção 5.3. A ideia que dá suporte a Bayes é que a probabilidade de uma situação nova (posterior) de uma hipótese dada a evidência pode ser vista como uma função de probabilidades conhecidas para a evidência dada essa hipótese. Podemos dizer que queremos determinar a função  $f$ , tal que  $p(h|e) = f(p(e|h))$ . Normalmente, queremos determinar o valor do lado esquerdo dessa equação dado que quase sempre é muito mais fácil calcular os valores no lado direito.

Agora, provamos o teorema de Bayes para um sintoma e uma doença. Com base nas definições anteriores, a probabilidade *a posteriori* de uma pessoa ter a doença  $d$ , de um conjunto de doenças  $D$ , com sintoma ou evidência  $s$ , de um conjunto de sintomas  $S$ , é:

$$p(d | s) = |d \cap s| / |s|$$

Assim como na Seção 5.1, os “|”s ao redor de um conjunto indicam a cardinalidade ou o número de elementos nesse conjunto. O lado direito dessa equação é o número de pessoas que têm tanto (interseção) a doença  $d$  quanto o sintoma  $s$  dividido pelo número total de pessoas com o sintoma  $s$ . A Figura 5.2 apresenta um diagrama de Venn dessa situação. Expandimos o lado direito dessa equação. Como o espaço amostral para determinar as probabilidades do numerador e do denominador são as mesmas, obtemos:

$$p(d|s) = p(d \cap s) / p(s).$$

Há uma relação equivalente para  $p(s|d)$ ; mas uma vez, veja a Figura 5.2:

$$p(s|d) = p(s \cap d) / p(d).$$

Em seguida, resolvemos a equação  $p(s|d)$  para determinar o valor para  $p(s \cap d)$ :

$$p(s \cap d) = p(s|d) p(d).$$

Substituir esse resultado na equação anterior para  $p(d|s)$  produz a regra de Bayes para uma doença e um sintoma:

$$p(d|s) = \frac{p(s|d)p(d)}{p(s)}$$

Assim, a probabilidade *a posteriori* da doença dado o sintoma é o produto da verossimilhança do sintoma dada a doença e a verossimilhança da doença, normalizada pela probabilidade desse sintoma. Generalizamos essa regra na Seção 5.3.

A seguir, apresentamos a *regra da cadeia*, uma técnica importante usada por muitos domínios de raciocínio estocástico, especialmente no processamento da linguagem natural. Acabamos de desenvolver as equações para dois conjuntos quaisquer,  $A_1$  e  $A_2$ :

$$p(A_1 \cap A_2) = p(A_1 | A_2) p(A_2) = p(A_2 | A_1) p(A_1)$$

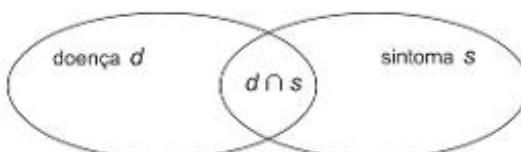
e, agora, a generalização para múltiplos conjuntos  $A_i$ , a chamada regra da cadeia:

$$p(A_1 \cap A_2 \cap \dots \cap A_n) = p(A_1) p(A_2 | A_1) p(A_3 | A_1 \cap A_2) \dots p(A_n | \bigcap_{i=1}^{n-1} A_i)$$

Criamos um argumento indutivo para provar a regra da cadeia; considere o enésimo caso:

$$p(A_1 \cap A_2 \cap \dots \cap A_{n-1} \cap A_n) = p((A_1 \cap A_2 \cap \dots \cap A_{n-1}) \cap A_n).$$

**Figura 5.2** Um diagrama de Venn ilustrando os cálculos de  $p(d|s)$  como uma função de  $p(s|d)$ .



Aplicamos a regra da interseção dos dois conjuntos para obter:

$$p((A_1 \cap A_2 \cap \dots \cap A_{n-1}) \cap A_n) = p(A_1 \cap A_2 \cap \dots \cap A_{n-1}) p(A_n | A_1 \cap A_2 \cap \dots \cap A_{n-1})$$

e depois reduzimos novamente, considerando que:

$$p(A_1 \cap A_2 \cap \dots \cap A_{n-1}) = p((A_1 \cap A_2 \cap \dots \cap A_{n-2}) \cap A_{n-1})$$

até que  $p(A_1 \cap A_2)$  seja alcançado, o caso base, que já demonstramos.

Concluímos esta seção com várias definições baseadas no uso da relação da regra da cadeia. Primeiro, redefinimos eventos independentes (ver Seção 5.2.1) no contexto de probabilidades condicionais, e depois definimos eventos condicionalmente independentes, ou a noção de como os eventos podem ser independentes um do outro, dado um terceiro evento.

### Definição

#### EVENTOS INDEPENDENTES

Dois eventos A e B são *independentes* um do outro se, e somente se,  $p(A \cap B) = p(A)p(B)$ . Quando  $p(B) \neq 0$ , isso é o mesmo que dizer que  $p(A) = p(A|B)$ . Ou seja, saber que B é verdadeiro não afeta a probabilidade de A ser verdadeiro.

#### EVENTOS CONDICIONALMENTE INDEPENDENTES

Dois eventos A e B são considerados *condicionalmente independentes* um do outro, dado o evento C, se, e somente se,  $p((A \cap B) | C) = p(A | C)p(B | C)$ .

Como um resultado da simplificação da regra da cadeia geral oferecida por eventos condicionalmente independentes, sistemas estocásticos maiores podem ser montados com custo computacional menor; ou seja, eventos condicionalmente independentes simplificam distribuições conjuntas. Um exemplo da nossa situação de tráfego lento: suponha que, ao reduzirmos a velocidade, notemos cones laranjas de controle de tráfego ao longo da lateral da estrada. Além de sugerir que a causa mais provável de nossa redução de velocidade agora são obras de reparo sendo feitas na estrada, e não um acidente de trânsito, a presença de cones laranja terá sua própria medida probabilística. De fato, as variáveis representando diminuição de velocidade e a presença de cones na pista são condicionalmente independentes, pois ambas são causadas pelas obras sendo executadas na estrada. Assim, dizemos que a variável obras na estrada *separa* a diminuição de velocidade dos cones laranja.

Devido às eficiências estatísticas obtidas pela independência condicional, uma tarefa importante na construção de grandes sistemas computacionais estocásticos é desmembrar um problema complexo em subproblemas conectados de modo mais fraco. As relações de interação dos subproblemas são então controladas pelas diversas relações de separação condicional. Vemos essa ideia mais bem formalizada com a definição de *d-separação* na Seção 9.3.

Em seguida, na Seção 5.3, apresentamos a forma geral do teorema de Bayes e demonstramos como, em situações complexas, a computação necessária para dar suporte à inferência Bayesiana total pode se tornar intratável. Por fim, na Seção 5.4, apresentamos exemplos de raciocínio baseados nas medidas de probabilidade baseadas em Bayes.

### 5.3 Teorema de Bayes

O Reverendo Thomas Bayes foi um matemático e um ministro. Seu famoso teorema foi publicado em 1763, quatro anos depois de sua morte. Seu artigo, intitulado “Ensaio para a solução de um problema na doutrina das

chances”, foi publicado na *Philosophical Transactions of the Royal Society of London*. O teorema de Bayes relaciona causa e efeito de modo que, compreendendo o efeito, podemos descobrir a probabilidade de suas causas. Como resultado, esse teorema é importante tanto para determinar as causas de doenças, como o câncer, quanto é útil para determinar os efeitos de uma medição em particular sobre essa doença.

### 5.3.1 Introdução

Um dos resultados mais importantes da teoria da probabilidade é a forma geral do teorema de Bayes. Primeiro, revemos um dos resultados da Seção 5.2.4, a equação de Bayes para uma doença e um sintoma. Para ajudar a manter nossas relações de diagnóstico em contexto, renomeamos as variáveis usadas anteriormente para indicar hipóteses individuais,  $h_i$ , a partir de um conjunto de hipóteses,  $H$ , e um conjunto de evidências,  $E$ . Além disso, agora consideraremos o conjunto de hipóteses individuais  $h_i$  como disjuntas, e a união de todo  $h_i$  como sendo igual a  $H$ .

$$p(h_i|E) = (p(E|h_i) \times p(h_i)) / p(E)$$

Essa equação pode ser lida como “a probabilidade de uma hipótese  $h_i$  dado um conjunto de evidências  $E$  é...”. Primeiro, note que o denominador  $p(E)$  no lado direito da equação é, em grande parte, um fator de normalização para qualquer uma das hipóteses  $h_i$  do conjunto  $H$ . Normalmente, acontece que o teorema de Bayes é usado para determinar qual hipótese de um conjunto de hipóteses possíveis é a mais forte, dado um conjunto de evidências em particular  $E$ . Nesse caso, normalmente removemos o denominador  $p(E)$  que é idêntico para todo  $h_i$ , com a economia de um custo computacional possivelmente grande. Sem o denominador, criamos o valor *máximo a posteriori* para uma hipótese:

$$\arg \max(h_i) p(E|h_i) p(h_i)$$

Lemos essa expressão como “o valor máximo sobre todo  $h_i$  de  $p(E|h_i) p(h_i)$ ”. A simplificação que descrevemos é altamente importante para o raciocínio diagnóstico, bem como no processamento da linguagem natural. Naturalmente, o *arg max*, ou a hipótese de máxima verossimilhança, não é mais uma variável aleatória como definida anteriormente.

Em seguida, considere o cálculo do denominador  $p(E)$  na situação em que o espaço amostral inteiro é *particionado* pelo conjunto de hipóteses  $h_i$ . A partição de um conjunto é definida como a divisão desse conjunto em subconjuntos disjuntos não sobrepostos, cuja união compõe o conjunto inteiro. Supondo que o conjunto de hipóteses  $h_i$  particiona o espaço amostral inteiro, obtemos:

$$p(E) = \sum_i p(E|h_i) p(h_i)$$

Essa relação é demonstrada considerando-se o fato de que o conjunto de hipóteses  $h_i$  forma uma partição do conjunto completo de evidência  $E$  com a regra da probabilidade da interseção de dois conjuntos. Então:

$$E = (E \cap h_1) \cup (E \cap h_2) \cup \dots \cup (E \cap h_n)$$

Mas, pela regra generalizada da união de conjuntos desenvolvida na Seção 5.2.1:

$$\begin{aligned} p(E) &= p((E \cap h_1) \cup (E \cap h_2) \cup \dots \cup (E \cap h_n)) \\ &= p(E \cap h_1) + p(E \cap h_2) + \dots + p(E \cap h_n) - p(E \cap h_1 \cap E \cap h_2 \cap \dots \cap h_n) \\ &= p(E \cap h_1) + p(E \cap h_2) + \dots + p(E \cap h_n) \end{aligned}$$

pois o conjunto de hipóteses  $h_i$  particiona  $E$  e sua interseção é vazia.

Esse cálculo de  $p(E)$  produz a forma geral do teorema de Bayes, onde consideramos que o conjunto de hipóteses,  $h_i$ , particiona o conjunto de evidências  $E$ :

$$p(h_i|E) = \frac{p(E|h_i) \cdot p(h_i)}{\sum_{k=1}^n p(E|h_k) \cdot p(h_k)}$$

$p(h_i | E)$  é a probabilidade de que  $h_i$  seja verdadeiro dada a evidência  $E$ .

$p(h_i)$  é a probabilidade de que  $h_i$  seja verdadeiro em geral.

$p(E | h_i)$  é a probabilidade de observar a evidência  $E$  quando  $h_i$  é verdadeiro.

$n$  é o número de hipóteses possíveis.

O teorema de Bayes oferece um modo de calcular a probabilidade de uma hipótese  $h_i$  com uma evidência em particular, dadas apenas as probabilidades com as quais a evidência vem de causas reais (as hipóteses).

Como um exemplo, suponha que queiramos examinar a evidência geológica em um local para ver se ele é apropriado ou não para a mineração de cobre. Devemos conhecer, com antecedência, a probabilidade de achar cada um de um conjunto de minerais e a probabilidade de certa evidência estar presente quando qualquer mineral em particular for achado. Então, podemos usar o teorema de Bayes, com evidência achada no local em particular, para determinar a verossimilhança de cobre. Essa abordagem é usada pelo PROSPECTOR, criado na Universidade de Stanford e SRI International, e empregado na exploração mineral (cobre, molibdênio e outros minerais). O PROSPECTOR encontrou depósitos de minerais comercialmente significativos em diversos locais (Duda et al., 1979a).

Em seguida, apresentamos um exemplo numérico simples demonstrando o teorema de Bayes. Suponha que você saia para comprar um automóvel. A probabilidade de você ir até a agência 1,  $d_1$ , é de 0,2. A probabilidade de ir até a agência 2,  $d_2$ , é de 0,4. Há somente três agências que você está considerando, e a probabilidade de você ir até a terceira,  $d_3$ , também é de 0,4. Em  $d_1$ , a probabilidade de comprar determinado automóvel,  $a_1$ , é de 0,2; na agência  $d_2$ , a probabilidade de comprar  $a_1$  é de 0,4. Por fim, na agência  $d_3$ , a probabilidade de comprar  $a_1$  é de 0,3. Suponha que você compre o automóvel  $a_1$ . Qual é a probabilidade de você o ter comprado na agência  $d_2$ ?

Primeiro, visto que você comprou o automóvel  $a_1$  da agência  $d_2$ , queremos determinar  $p(d_2 | a_1)$ . Apresentamos o teorema de Bayes na forma de variável para determinar  $p(d_2 | a_1)$  e depois com variáveis ligadas à situação no exemplo.

$$\begin{aligned} p(d_2 | a_1) &= (p(a_1 | d_2) p(d_2)) / (p(a_1 | d_1) p(d_1) + p(a_1 | d_2) p(d_2) + p(a_1 | d_3) p(d_3)) \\ &= (0,4)(0,4) / ((0,2)(0,2) + (0,4)(0,4) + (0,4)(0,3)) \\ &= 0,16 / 0,32 \\ &= 0,5 \end{aligned}$$

Há dois compromissos importantes no uso do teorema de Bayes: primeiro, todas as probabilidades nas relações da evidência com as diversas hipóteses devem ser conhecidas, bem como as relações probabilísticas entre as partes da evidência. Segundo, e às vezes mais difícil de determinar, todas as relações entre evidência e hipóteses, ou  $p(E | h_k)$ , devem ser estimadas ou amostradas empiricamente. Lembre-se de que o cálculo de  $p(E)$  para a forma geral do teorema de Bayes também exigiu que o conjunto de hipóteses  $h_i$  dividisse o conjunto de evidências  $E$ . Em geral, e especialmente em áreas como medicina e processamento de linguagem natural, uma suposição dessa partição não pode ser justificada *a priori*.

No entanto, é interessante observar que muitas situações que violam essa suposição (de que as peças individuais de cada evidência particionam o conjunto de evidências) se comportam muito bem! O uso dessa hipótese de partição, até mesmo em situações em que isso não é justificado, é chamado de uso de *Bayes ingênuo*, ou de um *classificador Bayes*. Com o Bayes ingênuo, a suposição é de que, para uma hipótese  $h_i$ :

$$p(E | h_i) = \prod_{j=1}^n p(e_j | h_i)$$

ou seja, consideraremos que as partes da evidência são independentes, dada uma hipótese em particular.

Usando o teorema de Bayes para determinar a probabilidade de uma hipótese  $h_i$  dado um conjunto de evidências  $E$ ,  $p(h_i | E)$ , os números no lado direito da equação normalmente são facilmente obtidos. Isso é verdade especialmente em comparação com a obtenção dos valores para o lado esquerdo da equação ou seja, determinar  $p(h_i | E)$  diretamente. Por exemplo, como a população é menor, é muito mais fácil determinar o número de pacientes com meningite que possuem dores de cabeça que de determinar a porcentagem dos que sofrem de dor de cabeça com meningite. Mais importante ainda, para o caso simples de uma única doença e um único sintoma, não são necessários muitos números. Porém os problemas começam quando consideramos várias doenças  $h_i$  a partir do domínio

de doenças  $H$  e vários sintomas  $e_i$  a partir de um conjunto  $E$  de possíveis sintomas. Quando consideramos cada doença a partir de  $H$  e cada sintoma a partir de  $E$  isoladamente, temos  $m \times n$  medições para coletar e integrar. (Na realidade,  $m \times n$  probabilidades *a posteriori* mais  $m + n$  propriedades *a priori*.)

Infelizmente, nossa análise está para se tornar muito mais complexa. Até este ponto, consideramos cada sintoma  $e_i$  individualmente. Em situações reais, sintomas isolados raramente acontecem. Quando um médico examina um paciente, por exemplo, existem muitas combinações de sintomas que ele precisa considerar. Necessitamos de uma forma de teorema de Bayes que considere qualquer hipótese isolada  $h_i$  no contexto da união de múltiplos sintomas possíveis  $e_i$ .

$$p(h_i | e_1 \cup e_2 \cup \dots \cup e_n) = (p(h_i) p(e_1 \cup e_2 \cup \dots \cup e_n | h_i)) / p(e_1 \cup e_2 \cup \dots \cup e_n)$$

Com uma doença e um único sintoma, só precisamos de  $m \times n$  medições. Agora, para cada par de sintomas  $e_i$  e  $e_j$  e uma hipótese de doença em particular  $h_i$ , precisamos saber  $p(e_i \cup e_j | h_i)$  e  $p(e_i \cup e_j)$ . O número desses pares é  $n \times (n - 1)$ , ou aproximadamente  $n^2$ , quando existem  $n$  sintomas em  $E$ . Agora, se quisermos usar Bayes, haverá cerca de  $(m \times n^2$  probabilidades condicionais) + ( $n^2$  probabilidades de sintomas) + ( $m$  probabilidades de doença) ou cerca de  $m \times n^2 + n^2 + m$  informações para coletar. Em um sistema médico realista com 200 doenças e 2.000 sintomas, esse valor é superior a 800.000.000!

Porém ainda há esperança. Como discutimos quando apresentamos a independência condicional, muitos desses pares de sintomas serão independentes, ou seja,  $p(e_i | e_j) = p(e_i)$ . A independência significa, é claro, a probabilidade de  $e_i$  não ser afetada pela presença de  $e_j$ . Na medicina, por exemplo, a maioria dos sintomas não está relacionada, por exemplo, perda de cabelo e ferida no cotovelo. Porém, mesmo que somente dez por cento de nossos sintomas de exemplo não fossem independentes, ainda haveria cerca de 80.000.000 relações restantes a considerar.

Em muitas situações de diagnóstico, também temos que lidar com informações negativas, por exemplo, quando um paciente não tem um sintoma como pressão sanguínea alta. Exigimos tanto:

$$p(\bar{e}_i) = 1 - p(e_i) \text{ e } p(\bar{h}_i | e_i) = 1 - p(h_i | e_i).$$

Também notamos que  $p(e_i | h_i)$  e  $p(h_i | e_i)$  não são iguais e quase sempre terão valores diferentes. Essas relações, e o impedimento de raciocínio circular, são importantes para o projeto das *redes Bayesianas de crença*, consideradas na Seção 9.3.1.

Um último problema, que novamente torna a manutenção das estatísticas de sistemas Bayesianos complexos praticamente intratável, é a necessidade de reconstruir tabelas de probabilidade quando novas relações entre hipóteses e conjuntos de evidências são descobertas. Em muitas áreas de pesquisa ativas, como na medicina, novas descobertas acontecem continuamente. O raciocínio Bayesiano exige probabilidades completas e atualizadas, incluindo probabilidades conjuntas, para que suas conclusões sejam corretas. Em muitos domínios, essa extensa coleta e verificação de dados não são possíveis ou, se forem, são muito dispendiosas.

Entretanto, em situações em que essas suposições são atendidas, as abordagens Bayesianas oferecem o benefício de um tratamento matematicamente bem fundamentado da incerteza. A maioria dos domínios de sistemas especialistas não atende a esses requisitos e precisa contar com abordagens heurísticas, conforme apresentadas no Capítulo 8. Além disso, devido a questões de complexidade, sabemos que até mesmo computadores muito poderosos não podem usar técnicas Bayesianas para a solução bem-sucedida de problemas em tempo real. Terminaremos este capítulo com dois exemplos que mostram como a abordagem Bayesiana poderia funcionar para organizar relações de hipótese/evidência. Mas primeiro temos que definir máquinas/reconhecedores probabilísticos de estados finitos.

## 5.4 Aplicações da metodologia estocástica

*You say [tow m ey tow] and I say [tow m aa tow]... (Você diz [to ma te] e eu digo [tu ma ti]...)*

—IRA GERSHWIN, “Let’s Call the Whole Thing Off”

Nesta seção, apresentamos dois exemplos que usam medições de probabilidade para raciocinar a respeito da interpretação de informações ambíguas. Primeiro, definimos uma ferramenta de modelagem importante baseada na máquina de estados finitos da Seção 3.1, a *máquina de estados finitos probabilística*.

## Definição

### MÁQUINA DE ESTADOS FINITOS PROBABILÍSTICA

Uma máquina de estados finitos probabilística é uma máquina de estados finitos onde a função do próximo estado é uma distribuição de probabilidade sobre o conjunto completo de estados da máquina.

### RECONHECEDOR DE ESTADOS FINITOS PROBABILÍSTICO

Uma máquina de estados finitos probabilística é um reconhecedor quando um ou mais estados são indicados como os estados *iniciais* e um ou mais são indicados como os estados de aceitação.

Pode-se ver que essas duas definições são extensões simples das máquinas de estados finitos e de Moore apresentadas na Seção 3.1. A adição para o não determinismo é que a função do próximo estado não é mais uma função no sentido estrito da palavra. Ou seja, não há um estado único do contradomínio para cada valor de entrada e cada estado do domínio. Em vez disso, para qualquer estado, a função do próximo estado é uma distribuição de probabilidade sobre todos os próximos estados possíveis.

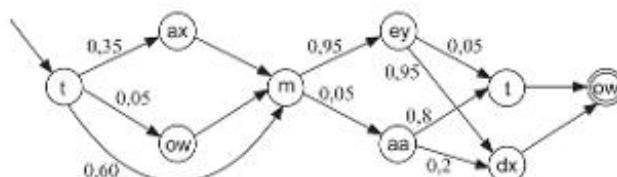
#### 5.4.1 Mas como se pronuncia “tomato”?

A Figura 5.3 apresenta um reconhecedor de estados finitos probabilístico que representa diferentes pronúnrias da palavra inglesa “tomato” (tomate). Uma pronúncia aceitável em particular para a palavra “tomato” é caracterizada por um caminho desde o estado inicial até o estado de aceitação. Os valores decimais que rotulam os arcos do grafo representam a probabilidade de que o orador faça essa transição em particular na máquina de estados. Por exemplo, 60% de todos os oradores nesse conjunto de dados vai diretamente do fonema t para o fonema m sem pronunciar qualquer vogal entre eles.

Além de caracterizar as diversas maneiras como as pessoas nos bancos de dados de pronúncia falam a palavra “tomato”, esse modelo pode ser usado para ajudar a interpretar coleções ambíguas de fonemas. Isso é feito examinando como os fonemas correspondem a possíveis caminhos pelas máquinas de estado de palavras relacionadas. Além disso, dada uma palavra parcialmente formada, a máquina pode tentar determinar o caminho probabilístico que melhor completa essa palavra.

Em um segundo exemplo, também adaptado de Jurafsky e Martin (2009), consideramos o problema do *reconhecimento de fonema*, muitas vezes chamado de *decodificação*. Suponha que um algoritmo de reconhecimento de fonema tenha identificado o fonema ni (como em “knee”), que ocorre logo após a palavra (fonema) reconhecida l, e queiramos associar ni com uma palavra ou a primeira parte de uma palavra. Nesse caso, temos os *corpora* linguísticos, de Brown e de Switchboard, para nos auxiliar.

**Figura 5.3** Um reconhecedor de estados finitos probabilístico para a pronúncia de “tomato”, adaptado de Jurafsky e Martin (2009).



O *corpus de Brown* é uma coleção de um milhão de palavras de sentenças de 500 textos escritos, incluindo jornais, romances e escritos acadêmicos coletados na Brown University na década de 1960 (Kucera e Francis, 1967; Francis, 1979). O *corpus de Switchboard* é uma coleção de 1,4 milhão de palavras de conversas telefônicas. Esses *corpora* contêm juntamente 2.500.000 palavras que nos permitem amostrar bases de informações escritas e faladas.

Há inúmeras maneiras de prosseguir na identificação da palavra mais provável para associar com o fonema ni. Primeiro, podemos determinar qual palavra, começando com esse fonema, é mais provável de ser usada. A Tabela 5.2 apresenta as frequências brutas dessas palavras com a probabilidade de sua ocorrência, ou seja, a frequência da palavra dividida pelo número total de palavras nesses *corpora* combinados. (Essa tabela é adaptada de Jurafsky e Martin (2009); veja o livro para obter uma justificativa de que o “the” pertence a essa coleção.) A partir desses dados, a palavra “the” pareceria ser a primeira escolha para combinar com ni.

Em seguida, aplicamos uma forma do teorema de Bayes que apresentamos na seção anterior. Nossa segunda tentativa de analisar o fonema ni após I usa uma simplificação dessa fórmula que ignora seu denominador:

$$p(\text{palavra} | [ni]) \propto p([ni] | \text{palavra}) \times p(\text{palavra})$$

Os resultados desse cálculo, ordenados do mais recomendado ao menos recomendado, são encontrados na Tabela 5.3 e explicam por que  $p(ni | the)$  é impossível. Os resultados dessa tabela também sugerem que new é a palavra mais provável para decodificar ni. Mas a combinação de duas palavras I new não parece fazer muito sentido, enquanto outras combinações, como I need, fazem sentido. Parte do problema nessa situação é que ainda estamos raciocinando no nível do fonema, ou seja, determinando a probabilidade  $p(ni | new)$ . De fato, existe um modo direto de resolver essa questão, que é procurar combinações explícitas de duas palavras nos *corpora*. Segundo essa linha de raciocínio, I need é um par muito mais provável de palavras consecutivas do que I new, ou quaisquer outras combinações de palavra com I, pelo mesmo motivo.

**Tabela 5.2** As palavras ni com suas frequências e probabilidades dos *corpora* de Brown e Switchboard de 2,5 milhões de palavras, adaptado de Jurafsky e Martin (2009).

Palavra	Frequência	Probabilidade
knee	61	0,000024
the	114834	0,046
neat	338	0,00013
need	1417	0,00056
new	2625	0,001

**Tabela 5.3** As probabilidades de fonema/palavra ni dos *corpora* de Brown e de Switchboard (Jurafsky e Martin, 2009).

palavra	$p([ni]   \text{palavra})$	$p(\text{palavra})$	$p([ni]   \text{palavra}) \times p(\text{palavra})$
new	0,36	0,001	0,00036
neat	0,52	0,00013	0,000068
need	0,11	0,00056	0,000062
knee	1,0	0,000024	0,000024
the	0,0	0,046	0,0

A metodologia para derivar probabilidades a partir de pares ou triplas de combinações de palavras em *corporá* é chamada de *análise de n-gramas*. Com duas palavras, estivemos usando *bigramas*, com três, *trigramas*. As probabilidades derivadas de combinações de palavras usando n-gramas são importantes como veremos novamente no Capítulo 15.

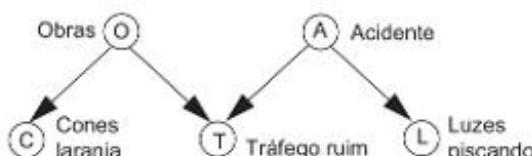
### 5.4.2 Estendendo o exemplo de estrada/tráfego

Apresentamos novamente e estendemos o exemplo da Seção 5.2.2. Suponha que você esteja dirigindo pela rodovia interestadual e observa que está gradualmente diminuindo de velocidade devido a um congestionamento de trânsito intenso. Você começa a buscar possíveis explicações para esse problema. Seria uma obra na estrada? Houve um acidente? Talvez haja outras explicações possíveis. Depois de alguns minutos, você encontra cones laranja na lateral da estrada, que começam a levar para o acostamento. Nesse ponto, você determina que a melhor explicação para a redução da velocidade é a existência de uma obra na estrada. Ao mesmo tempo, a hipótese alternativa de acidente é *descartada*. De modo semelhante, se você encontrasse luzes piscando a uma certa distância, como as de um veículo da polícia ou de uma ambulância, a melhor explicação dadas as evidências seria um acidente de trânsito, e a hipótese de obra na estrada teria sido *descartada*. Quando uma hipótese é descartada, isso não significa que ela não é mais possível. Em vez disso, no contexto de nova evidência, ela é simplesmente menos provável.

A Figura 5.4 apresenta um relato Bayesiano do que acabamos de ver. Obras na estrada está correlacionada a cones laranja e ao tráfego ruim. De modo semelhante, acidente está correlacionado a luzes piscando e tráfego ruim. Examinamos a Figura 5.4 e montamos uma distribuição de probabilidade conjunta para a relação entre obras na estrada e tráfego ruim. Simplificamos essas duas variáveis para verdadeiro (v) e falso (f) e representamos a distribuição de probabilidade na Tabela 5.4. Observe que, se a existência de obras for f, não há probabilidade de que haja tráfego ruim e, se for v, então o tráfego ruim é provável. Observe também que a probabilidade de a rodovia interestadual estar em obras na estrada,  $O = \text{verdadeiro}$ , é de 0,5, e a probabilidade de haver tráfego ruim,  $T = \text{verdadeiro}$ , é de 0,4.

Em seguida, consideraremos a mudança na probabilidade de obras na estrada dado o fato de que temos tráfego ruim, ou  $p(O|T)$  ou  $p(O = v | T = v)$ .

**Figura 5.4** Representação Bayesiana do problema do tráfego com explicações em potencial.



**Tabela 5.4** Distribuição de probabilidade conjunta para as variáveis de tráfego e de obras da Figura 5.4.

	O	T	p	
O é verdadeiro = 0,5	v	v	0,3	T é verdadeiro = 0,4
	v	f	0,2	
	f	v	0,1	
	f	f	0,4	

$$p(O|T) = p(O = v, T = v) / (p(O = v, T = v) + p(O = v, T = \bar{v})) = 0,3 / (0,3 + 0,1) = 0,75$$

Assim, agora, com a probabilidade de haver obras na estrada ser de 0,5, dado que realmente há tráfego ruim, a probabilidade de obras na estrada sobe para 0,75. Essa probabilidade aumentará ainda mais com a presença de cones laranja e descartará a hipótese de acidente.

Além do requisito de que devemos ter conhecimento ou medições para qualquer um dos parâmetros em cada determinado estado, também devemos tratar, como observado na Seção 5.4.1, de questões de complexidade. Considere o cálculo da probabilidade conjunta de todos os parâmetros da Figura 5.4 (usando a regra da cadeia e uma ordem topológica de classificação de variáveis):

$$p(O, A, C, T, L) = p(O) \times p(A|O) \times p(C|O, A) \times p(T|O, A, C) \times p(L|O, A, C, T)$$

Esse resultado é uma decomposição geral das medidas de probabilidade que é sempre verdadeira. O custo de produzir essa tabela de probabilidade conjunta é exponencial no número de parâmetros envolvidos e, nesse caso, requer uma tabela de tamanho  $2^5$ , ou 32. Estamos considerando um exemplo didático, é claro, com apenas cinco parâmetros. Uma situação com tamanho interessante, com trinta ou mais parâmetros, requer uma tabela de distribuição conjunta de aproximadamente um bilhão de elementos. Como veremos na Seção 9.3, as redes Bayesianas de crença e a d-separação nos dão ferramentas para tratar dessas complexidades de representação e computação.

## 5.5 Epílogo e referências

Jogos de azar existem, pelo menos, desde as civilizações gregas e romanas. Porém não foi antes do Renascimento europeu que a análise matemática da teoria da probabilidade começou. Como vimos neste capítulo, o raciocínio probabilístico começa com a determinação de princípios de contagem e combinações. Na verdade, uma das primeiras “máquinas” combinatórias é atribuída a Ramon Llull (Ford et al., 1995), um filósofo espanhol e monge franciscano que criou seu dispositivo, considerado como enumerando automaticamente os atributos de Deus, com a intenção de converter os pagãos. A primeira publicação sobre probabilidades, *De Ratiociniis Ludo Aleae*, é de autoria de Christian Huygens (1657). Huygens descreve os primeiros resultados de Blaise Pascal, incluindo uma metodologia para calcular probabilidades, bem como probabilidades condicionais. Pascal visou tanto a análise “objetiva” do mundo dos jogos quanto a análise mais “subjetiva” dos sistemas de crença, incluindo a existência de Deus.

As definições de probabilidades apresentadas na Seção 5.2.1 são baseadas no formalismo proposto pelo matemático francês Pierre Simon Laplace. O livro de Laplace *Theorie Analytique des Probabilités* (1816) documenta essa técnica. O trabalho de Laplace foi baseado nos primeiros resultados publicados por Gotlob Leibnitz e James Bernoulli.

Thomas Bayes foi um matemático e um ministro. Seu famoso teorema foi publicado em 1764, após sua morte. Seu artigo, intitulado “Essay Towards Solving a Problem in the Doctrine of Chances”, foi publicado na *Philosophical Transactions of the Royal Society of London*. Ironicamente, o teorema de Bayes nunca foi exposto explicitamente em seu artigo, embora esteja lá! Bayes também tem uma discussão extensa da “realidade” das medidas estatísticas.

A pesquisa de Bayes foi motivada parcialmente para responder ao ceticismo filosófico do filósofo escocês David Hume. A rejeição de Hume da causalidade repudiou qualquer base para argumentos por meio de milagres da existência de Deus. De fato, em um artigo de 1763 apresentado à British Royal Society, o ministro Richard Price usou o teorema de Bayes para mostrar que havia uma boa evidência em favor dos milagres descritos no Novo Testamento.

Os matemáticos do início do século XX, incluindo Fisher (1922), Popper (1959) e Carnap (1948), completaram o alicerce da moderna teoria da probabilidade, continuando os debates do “subjetivo/objetivo” sobre a natureza das probabilidades. Kolmogorov (1950, 1965) axiomatizou os alicerces do raciocínio probabilístico (ver Seção 5.2.1).

É possível apresentar o tópico do raciocínio probabilístico a partir de vários pontos de vista. As duas abordagens mais populares são baseadas no cálculo proposicional e na teoria dos conjuntos. Para o cálculo proposicional, Seção 2.1, as proposições recebem uma confiança ou valor verdade probabilístico no intervalo [0,0; 1,0]. Essa

abordagem oferece uma extensão natural à semântica do cálculo proposicional. Escolhemos a segunda abordagem para o raciocínio probabilístico, sentindo que essa orientação é um pouco mais intuitiva, empregando todas as técnicas de contagem e outras técnicas da teoria dos conjuntos, conforme visto na Seção 5.1. Nos capítulos 9 e 13, estendemos nossa apresentação dos sistemas estocásticos aos esquemas de representação de primeira ordem (baseados em variável).

O teorema de Bayes foi um alicerce para diversos sistemas especialistas dos anos 1970 e 1980, incluindo uma extensa análise da dor abdominal aguda no Hospital Universitário de Glasgow (de Dombal et al., 1974) e o PROSPECTOR, um sistema da Universidade de Stanford que dá suporte à exploração de minério (Duda et al., 1979a). A abordagem ingênuo de Bayes tem sido usada em diversos problemas de classificação, incluindo reconhecimento de padrão (Duda e Hart, 1973), processamento de linguagem natural (Mooney, 1996) e outros. Domingos e Pazzani (1997) oferecem justificativa para os sucessos dos classificadores ingênuos de Bayes em situações que não atendem as suposições de independência Bayesianas.

Há muita pesquisa atualmente em raciocínio probabilístico na inteligência artificial, algumas apresentadas nos capítulos 9, 13 e 16. O raciocínio incerto é responsável por um componente das conferências sobre IA, incluindo AAAI, IJCAI, NIPS e UAI. Existem vários textos introdutórios excelentes sobre raciocínio probabilístico (Ross, 1988; DeGroot, 1989), bem como diversas introduções ao uso de métodos estocásticos em aplicações de inteligência artificial (Russell e Norvig, 2003; Jurafsky e Martin, 2009; Manning e Schütze, 1999).

Agradecemos a Dan Pless (Sandia National Laboratories) pela abordagem geral tomada neste capítulo, por vários dos exemplos e também pelas sugestões editoriais.

## 5.6 Exercícios

1. Um dado não viciado com seis lados é lançado cinco vezes, e os números que resultam são registrados em uma sequência. Quantas sequências diferentes existem?
2. Determine o número de permutações distintas das letras na palavra MISSISSIPPI, das letras da palavra ASSOCIATIVE.
3. Suponha que uma urna contenha 15 bolas, das quais oito são vermelhas e sete são pretas. De quantas maneiras cinco bolas podem ser escolhidas de modo que:
  - a. todas sejam vermelhas? todas sejam pretas?
  - b. duas sejam vermelhas e três sejam pretas?
  - c. pelo menos duas sejam pretas?
4. De quantas maneiras um comitê de três membros do corpo docente e dois estudantes pode ser selecionado a partir de um grupo de cinco membros do corpo docente e sete estudantes?
5. Em uma pesquisa com 250 expectadores de televisão, 88 gostam de ver notícias, 98 gostam de ver esportes e 94 gostam de ver comédia. Trinta e três pessoas gostam de ver notícias e esportes, 31 gostam de ver esportes e comédia e 35 gostam de ver notícias e comédia. Dez pessoas gostam de ver os três gêneros. Suponha que uma pessoa desse grupo seja escolhida aleatoriamente:
  - a. Qual é a probabilidade de que ela goste de notícias, mas não de esportes?
  - b. Qual é a probabilidade de que ela goste de notícias ou de esportes, mas não de comédia?
  - c. Qual é a probabilidade de que ela não goste nem de esportes nem de notícias?
6. Qual é a probabilidade de que um inteiro de quatro dígitos sem zeros iniciais:
  - a. tenha 3, 5 ou 7 como um de seus dígitos?
  - b. comece com 3, termine com 5 ou tenha 7 como um de seus dígitos?
7. Dois dados são lançados. Determine a probabilidade de que sua soma seja:
  - a. 4.
  - b. 7 ou um número par.
  - c. 10 ou maior.

8. Uma carta é retirada de um baralho normal de 52 cartas. Qual é a probabilidade de essa carta:
  - a. ser uma carta de figura (valete, rainha, rei ou ás)?
  - b. ser uma rainha ou uma espada?
  - c. ser uma carta de figura ou uma carta de paus?
9. Qual é a probabilidade de se receber as seguintes mãos em um jogo de pôquer com cinco cartas (do baralho normal de 52 cartas)?
  - a. Um “flush” ou todas as cartas do mesmo naipe.
  - b. Um “full house” ou duas cartas do mesmo valor e três cartas de outro valor.
  - c. Um “royal flush” ou dez, valete, rainha, rei e ás, todos do mesmo naipe.
10. A *esperança* é a *média* do valor de uma variável aleatória. Ao lançar um dado, por exemplo, ela pode ser calculada totalizando os valores resultantes a partir de um número grande de lançamentos e depois dividindo pelo número de lançamentos.
  - a. Qual é a esperança do lançamento de um dado não viciado?
  - b. Qual é o valor de uma roda de roleta com 37 resultados igualmente prováveis?
  - c. Qual é o valor de uma carta retirada de um conjunto de cartas (ás tem valor 1, todas as outras cartas de figura têm valor 10)?
11. Suponha que você esteja jogando e que lançamos um dado e depois recebemos o valor em reais igual ao valor do dado. Por exemplo, se vier um 3, recebemos \$3. Se o preço para jogar esse jogo é \$4, isso é lucrativo?
12. Considere a situação em que sequências de bits de tamanho quatro são geradas aleatoriamente. Demonstre se o evento de produção de sequências de bits contendo um número par de 1s é ou não independente do evento de produzir sequências de bits que terminam com 1.
13. Mostre que a declaração  $p(A,B|C) = p(A|C)p(B|C)$  é equivalente a  $p(A|B,C) = p(A|C)$  e  $p(B|A,C) = p(B|C)$ .
14. Na fabricação de um produto, 85% dos produtos que são produzidos não têm defeito. Dos produtos inspecionados, 10% dos bons são vistos como defeituosos e não são entregues, enquanto apenas 5% dos produtos defeituosos são aprovados e entregues. Se um produto é entregue, qual é a probabilidade de ele ser defeituoso?
15. Um teste sanguíneo é 90% eficaz na detecção de uma doença. Ele também diagnostica falsamente que uma pessoa saudável tem a doença 3% das vezes. Se 10% daqueles testados têm a doença, qual é a probabilidade de uma pessoa com teste positivo realmente ter a doença?
16. Suponha que uma companhia de seguros de automóvel classifique um motorista como bom, médio ou ruim. De todos os seus motoristas segurados, 25% são classificados como bons, 50% são médios e 25% são ruins. Suponha que, para o próximo ano, um motorista bom tenha 5% de chance de sofrer um acidente, um motorista médio tenha 15% de chance e um motorista ruim tenha 25% de chance. Se você já sofreu um acidente no ano passado, qual é a probabilidade de você ser um bom motorista?
17. Três prisioneiros, A, B e C, estão em suas celas. Eles são informados de que um deles será executado no dia seguinte e os outros serão perdoados. Somente o governador sabe quem será executado. O prisioneiro A pede um favor ao guarda. “Por favor, pergunte ao governador quem será executado, e depois diga ao prisioneiro B ou ao C que eles serão perdoados.” O guarda faz isso e depois volta e diz ao prisioneiro A que ele disse ao prisioneiro B que ele (B) será perdoado. Quais são as chances de o prisioneiro A ser executado, considerando essa mensagem? Há mais informação agora do que antes do seu pedido ao guarda? (Adaptado de Pearl, 1988.)

# Construção de algoritmos de controle para busca em espaço de estados

*Se excluirmos cuidadosamente as influências dos ambientes de trabalho das influências subjacentes dos componentes e da organização do hardware, revelamos a verdadeira simplicidade do sistema adaptativo. Por isso, como vimos, precisamos postular apenas um sistema de processamento de informação muito simples a fim de levar em conta como o ser humano soluciona problemas, tais como o jogo de xadrez, a lógica e a aritmética criptográfica. O comportamento aparentemente complexo do sistema de processamento de informação, em um dado ambiente, é produzido pela interação das demandas do ambiente com alguns poucos parâmetros básicos do sistema, particularmente característicos de suas memórias.*

— A. NEWELL E H. A. SIMON, “Human Problem Solving” (1972)

*O que denominamos começo é, frequentemente, o fim  
E finalizar é começar.*

*O fim é de onde nós partimos...*

— T. S. ELIOT, “Four Quartets”

## 6.0 Introdução

Até este ponto, a Parte II representou a solução de problemas como uma busca por meio de um conjunto de situações ou estados. O Capítulo 2 apresentou o cálculo de predicados como um meio de descrever os estados de um problema e regras de inferência como método de produzir novos estados. O Capítulo 3 introduziu os grafos para representar e ligar situações de problema. O algoritmo de busca com retrocesso bem como algoritmos para busca em profundidade e em amplitude podem explorar esses grafos. O Capítulo 4 apresentou algoritmos para busca heurística. No Capítulo 5, com os estados probabilísticos do mundo, a inferência estocástica foi usada para produzir novos estados. Resumindo, a Parte II mostrou:

1. Representação de uma solução do problema como um caminho em um grafo do estado inicial para um objetivo.
2. Busca para testar, sistematicamente, caminhos alternativos até um objetivo.
3. Retrocesso, ou algum outro mecanismo, para permitir que um algoritmo se recupere de caminhos que não encontram um objetivo.
4. Listas para manter registros explícitos dos estados que estão sendo considerados.
  - a. A lista abertos permite que o algoritmo explore, se necessário, estados ainda não testados.
  - b. A lista fechados dos estados visitados permite que o algoritmo implemente a detecção de laços e evite a repetição de caminhos infrutíferos.

5. Implementação da lista abertos como uma *pilha* para a busca em profundidade, como uma *fila* para a busca em amplitude e como uma *fila de prioridade* para a busca pela melhor escolha.

O Capítulo 6 introduz técnicas adicionais para implementar algoritmos de busca. Na Seção 6.1, a *busca recursiva* implementa a busca em profundidade, em amplitude e pela melhor escolha de uma maneira natural, mais concisa que aquela apresentada no Capítulo 3. Além disso, a recursão é aumentada com a *unificação* para buscar no espaço de estados gerado pelas asserções do cálculo de predicados. Esse algoritmo de busca *guiado por padrões* é a base da linguagem PROLOG (veja a Seção 14.3) e de várias das estruturas de sistemas especialistas discutidas no Capítulo 8. Na Seção 6.2, introduzimos os *sistemas de produção*, uma arquitetura geral para a solução de problemas guiada por padrões que tem sido usada extensivamente, tanto para modelar a solução humana de problemas, Capítulo 16, quanto para construir outras aplicações de IA, incluindo sistemas especialistas, Capítulo 7. Finalmente, na Seção 6.3, apresentamos outra arquitetura de controle para solução de problemas de IA, o *quadro-negro*.

## 6.1 Busca baseada em recursão (opcional)

### 6.1.1 Busca recursiva

Em matemática, uma definição recursiva usa o termo que está sendo definido como parte de sua própria definição. Na ciência da computação, a recursão é usada para definir e analisar as estruturas de dados e também os procedimentos. Um procedimento recursivo consiste em:

1. Um passo recursivo: o procedimento chama a si próprio para repetir uma sequência de ações.
2. Uma condição de parada, que evita que o procedimento recorra infinitamente (a versão recursiva de um laço infinito).

Esses dois componentes são essenciais e aparecem em todas as definições e em todos os algoritmos recursivos. A recursão é um construtor natural de controle para estruturas de dados que tenham uma estrutura regular e não tenham tamanho definido, tais como listas, árvores e grafos, e é particularmente apropriada para busca em espaço de estados.

Uma tradução direta do algoritmo de busca em profundidade do Capítulo 3 na forma recursiva ilustra a equivalência entre recursão e iteração. Esse algoritmo usa as variáveis globais fechados e abertos para manter listas de estados. A busca em amplitude e pela melhor escolha podem ser criadas praticamente com o mesmo algoritmo, ou seja, mantendo fechados como uma estrutura global de dados e implementando abertos como uma fila ou uma fila de prioridade em vez de uma pilha (criar pilha torna-se criar fila ou criar fila de prioridade):

```

função busca_em_profundidade;                                % abertos e fechados globais
    inicio
        se abertos é vazia
            então retorne FALHA;
        estado_atual := primeiro elemento de abertos;
        se estado_atual é um estado objetivo
            então retorne SUCESSO
        senão
            início
                abertos := resto de abertos;
                fechados := fechados acrescentado estado_atual;
                para cada filho de estado_atual
                    se não estiver em fechados ou em abertos
                        então acrescentar o filho à frente de aberto
                            % constrói pilha
                    fim;
                busca_em_profundidade
            fim.
    fim.

```

A busca em profundidade, como acabamos de apresentar, não utiliza todo o poder da recursão. É possível simplificar ainda mais o procedimento usando a própria recursão (em vez de uma lista abertos explícita) para organizar estados e caminhos através do espaço de estados. Nessa versão do algoritmo, usa-se uma lista fechados global para detectar estados duplicados e para prevenir laços, e a lista abertos está implícita nos registros de ativação do ambiente recursivo. Como a lista abertos não pode ser mais manipulada explicitamente, as buscas em amplitude e pela melhor escolha não são mais extensões naturais do algoritmo a seguir:

```

função busca_em_profundidade (estado_atual);                                % fechados é global
    início
        se estado_atual é um objetivo
            então retorne SUCESSO;
        adicione estado_atual a fechados;
        enquanto estado_atual tiver filhos não examinados
            início
                filho := próximo filho não examinado;
                se filho não for membro de fechados
                    então se busca_em_profundidade(filho) = SUCESSO
                        então retorne SUCESSO
                fim;
                retorne FALHA
            fim.

```

% recursão esgotada

Em vez de gerar todos os filhos de um estado e colocá-los em uma lista abertos, esse algoritmo produz os estados filhos um por vez e, recursivamente, busca os descendentes de cada filho antes de gerar seus irmãos. Note que o algoritmo supõe uma ordem para operadores de geração de estados. Ao buscar recursivamente um estado filho, se algum descendente daquele estado for um objetivo, a chamada recursiva retorna sucesso e o algoritmo ignora os irmãos. Se a busca recursiva do estado filho falhar, ou seja, se não encontrar um objetivo, o próximo irmão é gerado e todos os seus descendentes são buscados. Dessa forma, o algoritmo busca o grafo inteiro em profundidade. O leitor deve se certificar de que o algoritmo realmente busque o grafo inteiro na mesma ordem do algoritmo de busca em profundidade da Seção 3.2.3.

A omissão de uma lista abertos explícita é possível graças à recursão. Os mecanismos pelos quais uma linguagem de programação implementa a recursão incluem um *registro de ativação* separado (Aho e Ullman, 1977) para cada chamada recursiva. Cada registro de ativação captura as variáveis locais e o estado de execução de cada chamada ao procedimento. Quando o procedimento é chamado recursivamente com um novo estado, um novo registro de ativação armazena seus parâmetros (o estado), as variáveis locais e o estado atual da execução. Em um algoritmo de busca recursivo, a série de estados sobre o caminho atual é armazenada conforme a sequência dos registros de ativação das chamadas recursivas. O registro de cada chamada indica, também, a última operação usada para gerar um estado filho; isso permite que o próximo irmão seja gerado quando for necessário.

Quando a busca por todos os descendentes de um estado resulta em falha, ou seja, quando nenhum dos descendentes inclui um objetivo, ocorre um retrocesso, causando a falha da chamada recursiva. Com isso, uma falha é retornada ao procedimento de expansão do estado pai, que então gera o filho seguinte e efetua a recursão. Nessa situação, o mecanismo interno de recursão executa a tarefa da lista abertos da versão iterativa do algoritmo. A implementação recursiva permite que o programador restrinja o seu ponto de vista a um único estado e a seus filhos, em vez de ter que manter explicitamente uma lista de estados abertos. A habilidade da recursão em expressar conceitos globais em uma forma fechada é a principal razão de seu poder.

Como esses dois algoritmos demonstram, a busca em espaço de estados é um processo inherentemente recursivo. Para encontrar um caminho entre um estado atual e um objetivo, é necessário mover-se para um estado filho e executar a recursão. Se esse estado filho não levar a um objetivo, deve-se examinar ordenadamente os seus irmãos. A recursão quebra um problema grande e difícil (a busca em todo o espaço) em partes menores e mais simples (gera os filhos de um único estado) e aplica essa estratégia (recursivamente) a cada uma dessas partes. Esse processo continua até que um estado objetivo seja descoberto ou até que todo o espaço seja esgotado.

Na próxima seção, essa abordagem recursiva para resolver problemas é estendida na forma de um controlador para um resolvedor de problemas baseado em lógica, que usa unificação e inferência para gerar e buscar um espaço de relações lógicas. O algoritmo suporta o E de múltiplos objetivos, bem como o encadeamento regressivo a partir de um objetivo até as premissas.

### 6.1.2 Um exemplo de busca recursiva: raciocínio guiado por padrão

Nesta seção, aplicamos a busca recursiva a um espaço de inferências lógicas; o resultado disso é um procedimento geral de busca para o cálculo de predicados baseado em especificações de problema.

Suponhamos que desejemos escrever um algoritmo que determine se uma expressão do cálculo de predicados é uma consequência lógica de um determinado conjunto de asserções. Isso sugere uma busca guiada por objetivo, com a consulta inicial formando o objetivo, e o *modus ponens* definindo as transições entre estados. Dado um objetivo (como  $p(a)$ ), o algoritmo emprega a unificação para selecionar as implicações cujas conclusões satisfazem o objetivo (por exemplo,  $q(X) \rightarrow p(X)$ ). Como o algoritmo trata implicações como regras potenciais para resolver a consulta, quase sempre elas são chamadas simplesmente de *regras*. Após unificar o objetivo com a conclusão da implicação (ou regra) e aplicar as substituições resultantes na regra inteira, a premissa da regra se torna um novo objetivo ( $q(a)$ ). Isso é chamado de um *subobjetivo*. O algoritmo, então, inicia a recursão para esse subobjetivo. Se um subobjetivo casar com um fato da base de conhecimento, a busca termina. A série de inferências que leva do objetivo inicial aos fatos dados prova a verdade do objetivo inicial.

```

função busca_padrão (objetivo_atual);
    início
        se objetivo_atual é um membro de fechados                                % teste para laços
            então retorne FALHA
        senão acrescente objetivo_atual a fechados;
        enquanto restarem na base de dados fatos ou regras para serem unificados faça
            início
                caso
                    objetivo_atual unifica-se com um fato:
                        retorne SUCESSO;
                    objetivo_atual é uma conjunção ( $p \wedge \dots$ ):
                        início
                            para cada termo da conjunção faça
                                chame busca_padrão para termos da conjunção;
                            se busca_padrão retornar sucesso para todos os termos da conjunção
                                então retorne SUCESSO
                            senão retorne FALHA
                        fim;
                    objetivo_atual unifica-se com a conclusão da regra ( $p$  em  $q \rightarrow p$ ):
                        início
                            aplique substituições para unificação do objetivo à premissa ( $q$ );
                            chame busca_padrão para a premissa;
                            se busca_padrão retornar sucesso
                                então retorne SUCESSO
                            senão retorne FALHA
                        fim;
                    fim;
                fim;
            retorne FALHA
        fim.
    
```

Na função `busca_padrão`, a busca é realizada por uma versão modificada do algoritmo de busca recursivo que usa unificação, Seção 2.3.2, para determinar quando existe casamento entre duas expressões e o *modus ponens* para gerar os filhos dos estados. O foco atual da busca é representado pela variável `objetivo_atual`. Se `objetivo_atual` casar com um fato, o algoritmo retornará `sucesso`. Caso contrário, o algoritmo tentará casar `objetivo_atual` com a conclusão de alguma regra, tentando resolver a premissa de modo recursivo. Se `objetivo_atual` não casar com nenhuma das asserções dadas, o algoritmo retornará `falha`. Esse algoritmo também lida com conjunção de objetivos.

Para simplificar esse algoritmo não considera o problema de manter a consistência entre as substituições de variáveis produzidas pela unificação. Isso é importante quando se resolvem consultas conjuntivas com variáveis compartilhadas (como em  $p(X) \wedge q(X)$ ). Não apenas ambos os termos da conjunção devem retornar sucesso, mas também devem retornar sucesso com as mesmas ligações unificadoras para  $X$ , Seção 2.3.2.

A principal vantagem de usar métodos gerais, como a unificação e o *modus ponens*, para gerar estados é que o algoritmo resultante pode buscar *qualquer* espaço de inferências lógicas no qual as especificidades de um problema são descritas usando asserções do cálculo de predicados. Assim, temos um meio de separar o conhecimento para a solução do problema de seu controle e implementação no computador. A função `busca_padrão` representa o nosso primeiro exemplo da separação entre conhecimento do problema e controle da busca.

Embora a versão inicial de `busca_padrão` tenha definido o comportamento de um algoritmo de busca para expressões do cálculo de predicados, vários aspectos ainda precisam ser tratados. Entre eles, a inclusão da ordem em que o algoritmo tenta casamentos alternativos e o tratamento adequado do conjunto completo de operadores lógicos ( $\wedge$ ,  $\vee$  e  $\neg$ ). A lógica é declarativa e não possui uma estratégia de busca predefinida: ela define um espaço de inferências possíveis, mas não diz quais são as úteis para resolver um problema.

Para raciocinar com o cálculo e com os predicados, necessitamos de um regime de controle que busque sistematicamente o espaço, evitando caminhos sem sentido e laços. Um algoritmo de controle como `busca_padrão` deve tentar casamentos alternativos em uma ordem sequencial. O conhecimento dessa ordem permite que o projetista do sistema controle a busca por meio da ordenação apropriada das regras da base de conhecimento. O modo mais simples de se definir essa ordem é fazer com que o algoritmo experimente as regras e os fatos na ordem em que eles aparecem na base de conhecimento.

Uma segunda questão é a existência de conectivos lógicos nas premissas de regras: por exemplo, implicações do tipo “ $p \leftarrow q \wedge r$ ” ou “ $p \leftarrow q \vee (r \wedge s)$ ”. Como foi dito na discussão sobre grafos E/OU, um operador  $\wedge$  indica que ambas as expressões devem ser verdadeiras para que toda a premissa seja verdadeira. Além disso, os termos da expressão devem ser resolvidos com ligações de variáveis consistentes. Assim, para resolver  $p(X) \wedge q(X)$ , não é suficiente resolver  $p(X)$  com a substituição  $\{a/X\}$  e  $q(X)$  com a substituição  $\{b/X\}$ . Ambos devem ser resolvidos com as mesmas ligações unificadoras para  $X$ . Um operador OU, por outro lado, indica que qualquer uma das expressões deve ser verdadeira. O algoritmo de busca deve levar isso em conta.

A última adição ao algoritmo diz respeito à habilidade de resolver objetivos envolvendo a negação lógica ( $\neg$ ). A função `busca_padrão` trata objetivos negados resolvendo primeiro o operando de  $\neg$ . Se esse subobjetivo resultar em sucesso, então `busca_padrão` retorna `falha`. Se o operando resultar em falha, então `busca_padrão` retorna um conjunto de substituição vazio, indicando sucesso. Note que mesmo se um subobjetivo contiver variáveis, o resultado da resolução de sua negação não conterá qualquer substituição. Isso acontece porque  $\neg$  só pode resultar em sucesso se o seu operando *falha*; assim, ele não pode retornar ligações para o operando.

Por fim, o algoritmo não deve retornar sucesso, mas deve retornar sim as ligações envolvidas na solução. A versão completa da função `busca_padrão`, que retorna o conjunto de unificações que satisfaz cada subobjetivo, é:

`função busca_padrão (objetivo_atual);`

`início`

`se objetivo_atual é um membro de fechados`

`% teste para laços`

`então retorne FALHA`

`senão acrescente objetivo_atual a fechados;`

`enquanto restarem na base dados fatos ou regras para serem unificados faça`

```

    início
    caso
        objetivo_atual unifica-se com um fato;
        retorne substituições unificadoras;
        objetivo_atual é uma negação ( $\neg p$ ):
            início
                chame busca_padrão para p;
                se busca_padrão retornar FALHA
                    então retorne {};
                senão retorne FALHA;
            fim;
        % negação é verdadeira
        objetivo_atual é uma conjunção ( $p \wedge \dots$ ):
            início
                para cada termo da conjunção faça
                    início
                        chame busca_padrão para termos da conjunção;
                        se busca_padrão retornar FALHA
                            então retorne FALHA;
                        senão aplique substituições a outros termos;
                    fim;
                se busca_padrão retornar SUCESSO para todos os termos da conjunção
                    então retorne composição das unificações;
                senão retorne FALHA;
            fim;
        objetivo_atual é uma disjunção ( $p \vee \dots$ ):
            início
                repita para cada termo da disjunção
                    chame busca_padrão para termos da disjunção
                    até não haver mais termos ou obter SUCESSO;
                    se busca_padrão retornar SUCESSO
                        então retorne substituições
                    senão retorne FALHA;
            fim;
        objetivo_atual unifica-se com a conclusão da regra ( $p$  em  $p \leftarrow q$ ):
            início
                aplique substituições para unificação do objetivo à premissa (q);
                chame busca_padrão para a premissa;
                se busca_padrão retornar SUCESSO
                    então retorne composição das substituições de p e q
                senão retorne FALHA;
            fim;
        fim;
    fim;
    retorne FALHA
fim.

```

Esse algoritmo busca\_padrão para buscar um espaço de regras e fatos do cálculo de predicados é a base de Prolog (onde é usada a forma da *cláusula de Horn* dos predicados, Seção 14.3) e em muitos ambientes de sistemas especialistas guiados por objetivo (Capítulo 8). Uma estrutura de controle alternativa para busca guiada por padrão é fornecida pelo *sistema de produção*, discutido na próxima seção.

## 6.2 Sistemas de produção

### 6.2.1 Definição e história

O *sistema de produção* é um modelo de computação que tem provado ser particularmente importante em IA, tanto para a implementação de algoritmos de busca como para a modelagem da solução humana de problemas. Um sistema de produção fornece controle guiado por padrão para um processo de solução de problemas e consiste em um conjunto de *regras de produção*, uma *memória de trabalho* e um ciclo de controle do tipo *reconhece-atua*.

#### Definição

#### SISTEMA DE PRODUÇÃO

Um *sistema de produção* é definido:

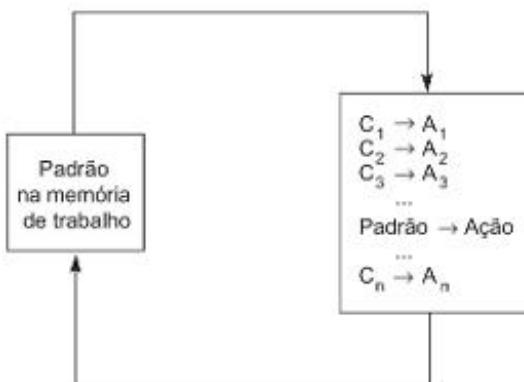
1. *Pelo conjunto de regras de produção.* Essas regras são, com frequência, chamadas simplesmente de *produções*. Uma produção é um par *condição-ação* e define uma porção de conhecimento para a solução de um problema. A *parte da condição* da regra é um padrão que determina quando a regra pode ser aplicada para uma instância do problema. A *parte da ação* define o(s) passo(s) associado(s) da solução do problema.
2. *Pela memória de trabalho*, que contém uma descrição do *estado atual do mundo* em um processo de raciocínio. Essa descrição é um padrão que é comparado com a condição de uma produção para selecionar ações apropriadas na resolução do problema. Quando o elemento da condição de uma regra casa com o conteúdo da memória de trabalho, as ações associadas com essa condição podem ser realizadas. As ações de regras de produção são projetadas especificamente para alterar o conteúdo da memória de trabalho.
3. *Pelo ciclo reconhece-atua.* A estrutura de controle para um sistema de produção é simples: a *memória de trabalho* é inicializada com a descrição inicial do problema. O estado atual da solução do problema é mantido como um conjunto de padrões na memória de trabalho. Esses padrões são comparados com as condições das regras de produção; isso produz um subconjunto de regras de produção, denominado *conjunto de conflito*, cujas condições casam com os padrões na memória de trabalho. Diz-se que as produções no conjunto de conflito estão *habilitadas*. Uma dessas produções do conjunto de conflito é, então, selecionada (*resolução do conflito*) e a produção é *disparada*. Para disparar uma regra, realiza-se a sua ação modificando o conteúdo da memória de trabalho. Após o disparo da regra de produção, o ciclo de controle se repete usando a memória de trabalho modificada. O processo termina quando o conteúdo da memória de trabalho não casa com nenhuma condição de regra.

A *resolução de conflito* escolhe uma regra do conjunto de conflitos para ser disparada. A estratégia para a resolução de conflito pode ser simples, como selecionar a primeira regra cuja condição case com o estado do mundo, ou pode envolver heurísticas complexas de seleção de regras. Esse é um modo importante pelo qual um sistema de produção permite a adição de controle heurístico a um algoritmo de busca.

O modelo *puro* de sistema de produção não possui um mecanismo para se recuperar de becos sem saída que surjam na busca; ele simplesmente continua até que mais nenhuma produção seja habilitada e, então, para. A maioria das implementações práticas de sistemas de produção permite, em tais situações, o retrocesso até um estado anterior da memória de trabalho.

A Figura 6.1 apresenta um desenho esquemático de um sistema de produção.

**Figura 6.1** Um sistema de produção. O controle realiza ciclos até que o padrão contido na memória de trabalho não case mais com nenhuma condição das produções.



Um exemplo muito simples de execução de um sistema de produção aparece na Figura 6.2. Esse é um programa de sistema de produção para ordenar uma sequência de caracteres composta pelas letras a, b e c. Nesse exemplo, uma produção é habilitada se a sua condição casar com uma parte da sequência que se encontra na memória de trabalho. Quando uma regra é disparada, a subsequência que casou com a condição da regra é substituída pela subsequência que se encontra no lado direito da regra. Sistemas de produção formam um modelo geral de computação que pode ser programado para fazer tudo o que pode ser feito com um computador. O seu verdadeiro poder, contudo, consiste em ser uma arquitetura para sistemas baseados em conhecimento.

A ideia para o projeto baseado em *produção* para computação tem a sua origem nos artigos de Post (1943), que propôs um modelo de regras de produção como uma teoria formal de computação. A principal construção dessa teoria era um conjunto de regras para se reescrever sequências de caracteres, de várias formas similares às regras de análise sintética do Exemplo 3.3.6. Esse modelo está também intimamente relacionado com a abordagem seguida pelos algoritmos de Markov (Markov, 1954) e, como eles, é equivalente em poder a uma máquina de Turing.

**Figura 6.2** Rastreamento de um sistema de produção simples.

Conjunto de produção:			
Nº Iteração	Memória de trabalho	Conjunto de conflito	Regra disparada
0	cbaca	1, 2, 3	1
1	cabca	2	2
2	acbca	2, 3	2
3	acbac	1, 3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	Ø	Parar

Uma aplicação interessante das regras de produção para modelar a cognição humana é encontrada no trabalho de Newell e Simon no Carnegie Institute of Technology (agora Carnegie Mellon University) nos anos 1960 e 1970. Os programas que eles desenvolveram, incluindo o *General Problem Solver* (Resolvedor geral de problemas), são em grande parte responsáveis pela importância dos sistemas de produção em IA. Nessa pesquisa, humanos eram monitorados em várias atividades de solução de problemas, como os de lógica de predicados e os de jogos como xadrez. O *protocolo* (padrões de comportamento, incluindo descrições verbais do processo de solução do problema, movimentos dos olhos etc.) desses sujeitos que resolviam problemas era registrado e decomposto em seus componentes elementares. Esses componentes eram tratados como os pedaços básicos do conhecimento para resolver problemas das pessoas humanas e foram dispostos como uma busca através de um grafo (chamado de *grafo de comportamento do problema*). Um sistema de produção era usado para implementar a busca nesse grafo.

As regras de produção representavam o conjunto de habilidades para resolver o problema pela pessoa humana. O foco de atenção presente era representado como o estado atual do mundo. Durante a execução do sistema de produção, a “atenção”, ou “foco atual” do resolvedor de problemas, casaria com uma regra de produção, o que mudaria o estado de “atenção” para casar com outra habilidade codificada como produção, e assim por diante.

É importante notar que, nesse trabalho, Newell e Simon usavam um sistema de produção não apenas como um veículo para implementar busca em grafo, mas também como um modelo real de comportamento humano durante a solução de problemas. As produções correspondiam às habilidades para resolver problemas, armazenadas na *memória de longo prazo* dos seres humanos. Assim como as habilidades armazenadas na memória de longo prazo, essas produções não eram modificadas pela execução do sistema; elas eram invocadas pelo “padrão” de uma ocorrência particular de problema, e novas habilidades poderiam ser acrescentadas sem a necessidade de se “recodificar” o conhecimento previamente existente. A memória de trabalho do sistema de produção corresponde à *memória de curto prazo*, ou ao foco atual de atenção nos seres humanos, e descreve o estágio atual da solução de uma ocorrência de problema. O conteúdo da memória de trabalho geralmente não permanece armazenado após um problema ter sido solucionado.

Essas origens da tecnologia do sistema de produção são descritas no livro *Human Problem Solving*, de Newell e Simon (1972), e em Luger (1978, 1994). Newell, Simon e outros continuaram a usar regras de produção para modelar a diferença entre novatos e peritos (Larkin et al., 1980; Simon e Simon, 1978) em áreas como a solução de problemas de álgebra e de física. Os sistemas de produção formam, também, a base para o estudo do aprendizado tanto de seres humanos como dos computadores (Klahr et al., 1987); ACT\* (Anderson, 1983b) e SOAR (Newell, 1990) usam essa tradição como base.

Os sistemas de produção fornecem um modelo para codificar a pericia humana na forma de regras e para projetar algoritmos de busca guiada por padrão, tarefas que são fundamentais para o projeto de sistemas especialistas baseados em regras. Em sistemas especialistas, não se supõe, necessariamente, que o sistema de produção realmente modele o comportamento humano na solução de problemas; entretanto, os aspectos de sistemas de produção que fazem com que eles sejam úteis como um modelo potencial do comportamento humano na solução de problemas (a estrutura modular das regras, a separação entre conhecimento e controle, a separação entre memória de trabalho e o conhecimento para a solução de problemas) fazem também com que eles se tornem uma ferramenta ideal para projetar e construir sistemas especialistas, como vemos nas seções 8.1 e 8.2.

Uma família importante de linguagens de IA resultou diretamente das pesquisas em linguagens de sistemas de produção realizadas em Carnegie Mellon. São linguagens OPS (do inglês *Official Production System* — sistema de produção oficial). Embora sua origem esteja na modelagem da solução humana de problemas, essas linguagens se mostraram altamente eficazes na programação de sistemas especialistas e em outras aplicações da IA. OPS5 é a linguagem de implementação do XCON, o configurador de VAX, e para outros sistemas especialistas desenvolvidos na Digital Equipment Corporation (McDermott, 1981, 1982; Soloway et al., 1987; Barker e O’Connor, 1989). Interpretadores de OPS estão largamente disponíveis para PCs e estações de trabalho. A CLIPS, implementada na linguagem de programação C, é uma versão orientada a objetos muito utilizada de um sistema de produção construído pela NASA. O JESS, um sistema de produção implementado em Java, foi criado pela Sandia National Laboratories.

Na próxima seção, damos exemplos de como o sistema de produção pode ser usado para resolver uma série de problemas de busca.

## 6.2.2 Exemplos de sistemas de produção

### Exemplo 6.2.1 Quebra-cabeça dos 8 modificado

O espaço de busca gerado pelo quebra-cabeça dos 8, introduzido no Capítulo 3, é suficientemente complexo para ser interessante, mas também pequeno o suficiente para ser tratável, sendo, por isso, frequentemente usado para explorar diferentes estratégias de busca, tais como as buscas em profundidade e em amplitude, bem como as estratégias heurísticas do Capítulo 4. Agora, apresentamos uma solução por sistema de produção.

Lembre-se de que ganhamos em generalidade ao pensar em “movimentar o espaço vazio” em vez de movimentar uma peça numerada. Os movimentos válidos são definidos pelas produções da Figura 6.3. Claro que os quatro movimentos são aplicáveis apenas quando o espaço vazio está no centro; quando ele está em um dos cantos, apenas dois movimentos são possíveis. Se um estado inicial e um estado objetivo para o quebra-cabeça dos 8 forem agora especificados, será possível fazer com que um sistema de produção lide com o espaço de busca do problema.

Uma implementação real desse problema poderia representar cada configuração de tabuleiro por um predicado de “estado” com nove parâmetros (para as nove localizações possíveis das oito peças e o vazio); as regras poderiam ser escritas como implicações cuja premissa realize a verificação da condição requerida. Como alternativa, arranjos ou estruturas de listas poderiam ser usados para os estados do tabuleiro.

Um exemplo, tirado de Nilsson (1980), do espaço buscado para encontrar uma solução para o problema dado na Figura 6.3 é apresentado na Figura 6.4. Como esse caminho-solução pode se estender muito profundamente se não houver restrições, foi acrescentado um limite de profundidade para a busca. (Uma maneira simples de se acrescentar um limite de profundidade é controlar o comprimento do caminho atual e forçar o retrocesso, se esse limite for excedido.) Um limite de profundidade de 5 é usado na solução da Figura 6.4. Note que o número de estados possíveis da memória de trabalho cresce exponencialmente com a profundidade da busca.

### Exemplo 6.2.2 O problema do percurso do cavalo

No jogo de xadrez, um cavalo pode se mover dois quadrados, horizontalmente ou verticalmente, seguidos por um quadrado na direção ortogonal, contanto que ele não saia do tabuleiro. Assim, existem, no máximo, oito movimentos possíveis que o cavalo pode fazer (Figura 6.5).

**Figura 6.3** Quebra-cabeça dos 8 como um sistema de produção.

Estado inicial:		Estado objetivo:
2	8	3
1	6	4
7	5	

#### Conjunto de produção:

##### Condição

##### Ação

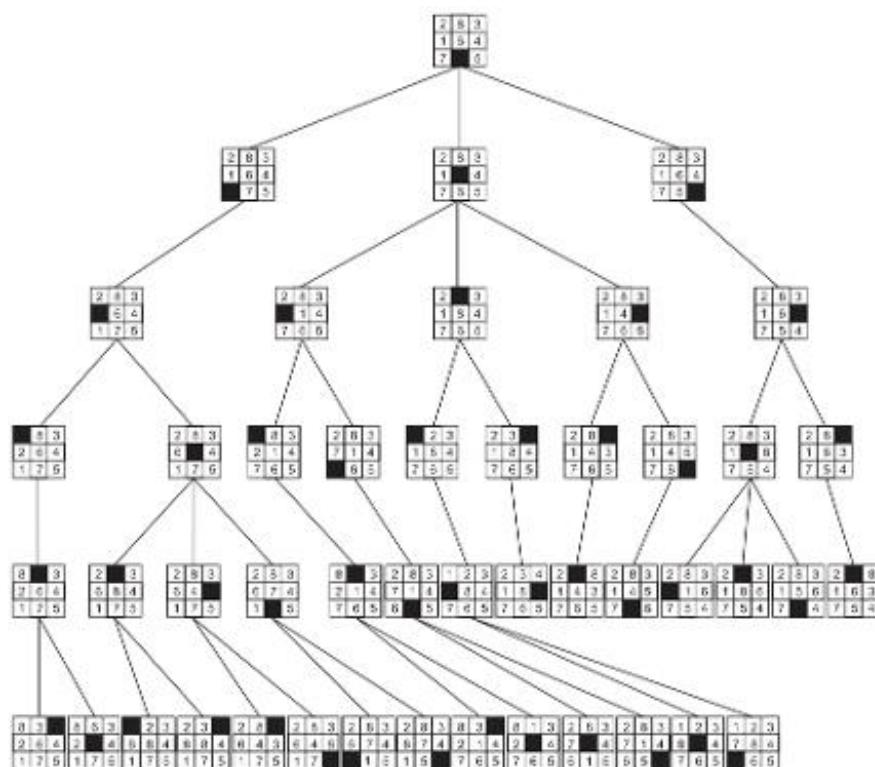
- estado objetivo na memória de trabalho → parar
- vazio não está na borda esquerda → mover o vazio para a esquerda
- vazio não está na borda superior → mover o vazio para cima
- vazio não está na borda direita → mover o vazio para a direita
- vazio não está na borda inferior → mover o vazio para baixo

A memória de trabalho é o estado atual do tabuleiro e o estado objetivo.

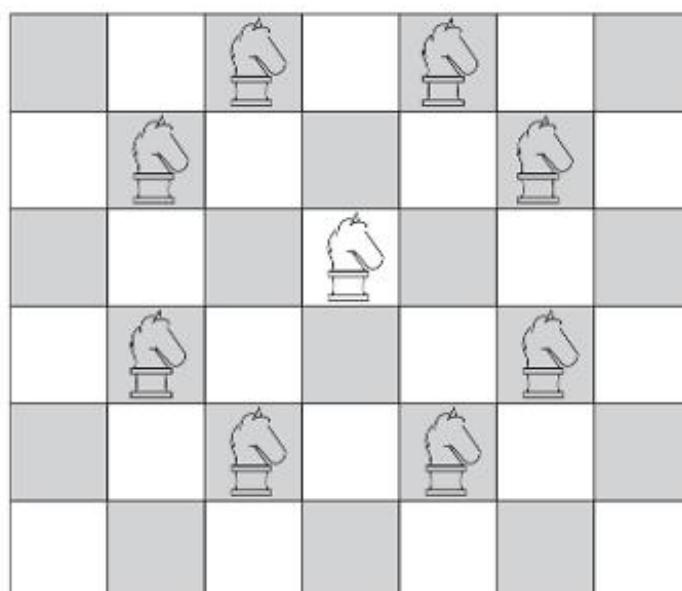
#### Regime de controle:

1. Tentar cada produção na ordem.
2. Não permitir laços.
3. Parar quando o objetivo for encontrado.

**Figura 6.4** Quebra-cabeça dos 8 pesquisado por um sistema de produção com detecção de laços e limite de profundidade de 5, de Nilsson (1971).



**Figura 6.5** Movimentos válidos de um cavalo no jogo de xadrez.



Conforme a definição tradicional, o problema do percurso do cavalo consiste em encontrar uma sequência de movimentos válidos pela qual o cavalo percorra cada quadrado do tabuleiro de xadrez exatamente uma vez. Esse problema tem exercido um papel importante no desenvolvimento e na apresentação de algoritmos de busca. O exemplo que utilizamos neste capítulo é uma versão simplificada do problema do percurso do cavalo, onde indagamos se existe uma sequência de movimentos válidos que levem o cavalo de um determinado quadrado para outro em um tabuleiro de tamanho reduzido ( $3 \times 3$ ).

A Figura 6.6 mostra um tabuleiro de xadrez  $3 \times 3$  com cada quadrado rotulado com inteiros de 1 a 9. Esse esquema de rotulação é usado, em vez da abordagem geral de atribuir a cada quadrado um número de linha e de coluna, com a finalidade de simplificar ainda mais o exemplo. Por causa do tamanho reduzido do problema, simplesmente enumeramos os movimentos alternativos em vez de desenvolver um operador de movimento geral. Com isso, os movimentos válidos sobre esse tabuleiro são descritos em cálculo de predicados usando um predicado `mova`, cujos parâmetros são os quadrados inicial e final de um movimento válido. Por exemplo, `mova(1,8)` leva o cavalo do canto superior esquerdo para o centro da linha inferior. Os predicados da Figura 6.6 enumeram todos os movimentos possíveis para o tabuleiro  $3 \times 3$ .

O problema do percurso do cavalo  $3 \times 3$  pode ser resolvido usando uma abordagem por sistema de produção. Cada movimento seria representado como uma regra cuja condição é a localização do cavalo em um quadrado em particular e cuja ação movimenta o cavalo para um outro quadrado. Dezesseis produções, apresentadas na Tabela 6.1, representam todos os movimentos possíveis do cavalo.

Em seguida, especificamos um procedimento recursivo para implementar um algoritmo de controle para o sistema de produção. Como `caminho(X,X)` irá se unificar somente com predicados como `caminho(3,3)` ou `caminho(5,5)`, ele define a condição de finalização desejada. Se `caminho(X,X)` não tiver sucesso, examinaremos as regras de produção em busca do próximo estado possível e depois faremos a recursão. A definição geral de caminho recursivo é, por isso, dada por duas fórmulas do cálculo de predicados:

$$\begin{aligned} \forall X \text{ caminho}(X,X) \\ \forall X,Y [\text{caminho}(X,Y) \leftarrow \exists Z [\text{mova}(X,Z) \wedge \text{caminho}(Z,Y)]] \end{aligned}$$

A memória de trabalho, parâmetros do predicado do caminho recursivo, contém tanto o estado atual do tabuleiro como o estado objetivo. O regime de controle aplica regras até que o estado atual seja igual ao estado objetivo e, então, para. Um esquema simples de solução de conflito dispararia a primeira regra que não causasse laços na busca. Como a busca pode levar a becos sem saída (dos quais qualquer movimento possível leva a um estado já visitado previamente e, consequentemente, causa um laço), o regime de controle deveria também permitir retrocesso; na Figura 6.7, está representada uma execução desse sistema de produção que determina se existe um caminho do quadrado 1 para o quadrado 2. Essa caracterização da definição do caminho como um sistema de produção é dada na Figura 6.8.

Sistemas de produção são capazes de gerar laços infinitos quando buscam um grafo de espaço de estados. Esses laços são particularmente difíceis de localizar em um sistema de produção, porque as regras podem ser

**Figura 6.6** Um tabuleiro de xadrez  $3 \times 3$  com regras de movimentos para o problema simplificado do percurso do cavalo.

<code>mova(1,8)</code>	<code>mova(6,1)</code>	
<code>mova(1,6)</code>	<code>mova(6,7)</code>	
<code>mova(2,9)</code>	<code>mova(7,2)</code>	
<code>mova(2,7)</code>	<code>mova(7,6)</code>	
<code>mova(3,4)</code>	<code>mova(8,3)</code>	
<code>mova(3,8)</code>	<code>mova(8,1)</code>	
<code>mova(4,9)</code>	<code>mova(9,2)</code>	
<code>mova(4,3)</code>	<code>mova(9,4)</code>	

1	2	3
4	5	6
7	8	9

**Tabela 6.1** Regras de produção para o problema do percurso do cavalo  $3 \times 3$ .

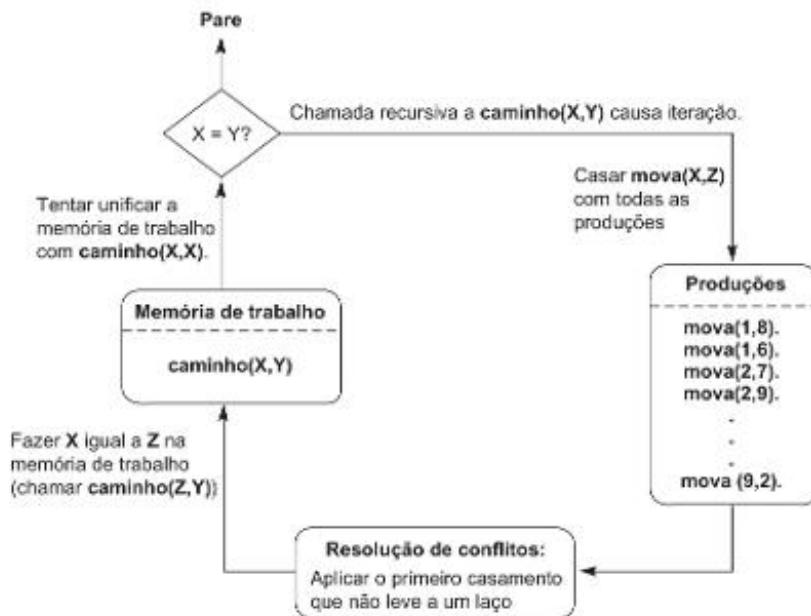
Nº DA REGRA	CONDIÇÃO		AÇÃO
1	cavalo no quadrado 1	→	mova cavalo para quadrado 8
2	cavalo no quadrado 1	→	mova cavalo para quadrado 6
3	cavalo no quadrado 2	→	mova cavalo para quadrado 9
4	cavalo no quadrado 2	→	mova cavalo para quadrado 7
5	cavalo no quadrado 3	→	mova cavalo para quadrado 4
6	cavalo no quadrado 3	→	mova cavalo para quadrado 8
7	cavalo no quadrado 4	→	mova cavalo para quadrado 9
8	cavalo no quadrado 4	→	mova cavalo para quadrado 3
9	cavalo no quadrado 6	→	mova cavalo para quadrado 1
10	cavalo no quadrado 6	→	mova cavalo para quadrado 7
11	cavalo no quadrado 7	→	mova cavalo para quadrado 2
12	cavalo no quadrado 7	→	mova cavalo para quadrado 6
13	cavalo no quadrado 8	→	mova cavalo para quadrado 3
14	cavalo no quadrado 8	→	mova cavalo para quadrado 1
15	cavalo no quadrado 9	→	mova cavalo para quadrado 2
16	cavalo no quadrado 9	→	mova cavalo para quadrado 4

disparadas em qualquer ordem. Isto é, um laço pode aparecer na execução do sistema, mas não pode ser facilmente encontrado a partir de uma inspeção sintática do conjunto de regras. Por exemplo, com as regras “mova” do problema do percurso do cavalo, ordenadas como estão na Tabela 6.1, e com a estratégia de resolução de conflitos correspondente a selecionar pelo primeiro casamento, o padrão *mova(2,X)* casaria com *mova(2,9)*, indicando um

**Figura 6.7** Uma solução por sistema de produção para o problema do percurso do cavalo  $3 \times 3$ .

Nº Iteração	Memória de trabalho		Conjunto de conflito (nº das regras)	Regra disparada
	Quadrado atual	Quadrado objetivo		
0	1	2	1, 2	1
1	8	2	13, 14	13
2	3	2	5, 6	5
3	4	2	7, 8	7
4	9	2	15, 16	15
5	2	2		Parar

Figura 6.8 O algoritmo recursivo de caminho como um sistema de produção.



movimento para o quadrado 9. Na próxima iteração, o padrão  $\text{mova}(9,X)$  casaria com  $\text{mova}(9,2)$ , levando a busca de volta ao quadrado 2, causando um laço.

Para prevenir laços, `busca_padrão` verifica uma lista global (`fechados`) de estados visitados. Portanto, a estratégia real de resolução de conflitos é: selecionar o movimento pelo primeiro casamento *que leve a um estado não visitado*. Em um sistema de produção, o lugar adequado para armazenar esses dados específicos do caso como uma lista de estados previamente visitados não é uma lista fechados global, mas a própria memória de trabalho. Podemos alterar o predicado `caminho` para usar a memória de trabalho na detecção de laços. Suponhamos que a nossa linguagem de cálculo de predicados seja estendida pela adição de um construtor especial, `declare(X)`, que faz com que o seu argumento  $X$  seja adicionado à memória de trabalho. O predicado `declare` não é um predicado comum, mas sim uma ação que é realizada; assim, ele sempre será bem-sucedido.

O predicado `declare` é usado para colocar um “marcador” na memória de trabalho para indicar quando um estado foi visitado. Esse marcador é representado como um predicado unário, `foi(X)`, que toma como argumento um quadrado sobre o tabuleiro. `foi(X)` é adicionado à memória de trabalho quando um novo estado  $X$  é visitado. A resolução de conflitos pode então exigir que `foi(Z)` não esteja na memória de trabalho antes que  $\text{mova}(X,Z)$  possa disparar. Para um valor específico de  $Z$ , essa condição pode ser testada por meio do casamento de um padrão com a memória de trabalho.

O controlador de caminho recursivo modificado para o sistema de produção é:

$$\begin{aligned} &\forall X \text{ caminho}(X,X) \\ &\forall X,Y [\text{caminho}(X,Y) \leftarrow \exists Z [\text{mova}(X,Z) \wedge \neg(\text{foi}(Z)) \wedge \text{declare}(\text{foi}(Z)) \wedge \text{caminho}(Z,Y)]] \end{aligned}$$

Por essa definição,  $\text{mova}(X,Z)$  tem sucesso no primeiro casamento com um predicado de movimento. Isso faz com que um valor seja ligado a  $Z$ . Se  $\text{foi}(Z)$  casar com uma posição na memória de trabalho,  $\neg(\text{foi}(Z))$  causará uma falha (isto é, ele será falso). `busca_padrão`, então, retrocederá e tentará um outro casamento para  $\text{mova}(X,Z)$ . Se o quadrado  $Z$  for um novo estado, a busca continuará, sendo  $\text{foi}(Z)$  adicionado à memória de trabalho a fim de prevenir laços futuros. O disparo da produção acontecerá realmente durante a recursão do algoritmo de caminho. Dessa forma, a presença do predicado `foi` na memória de trabalho implementa a detecção de laços nesse sistema de produção.

### Exemplo 6.2.3 O percurso completo do cavalo

Podemos generalizar a solução do percurso do cavalo para um tabuleiro completo  $8 \times 8$ . Como não faz sentido enumerar os movimentos em um problema tão complexo, substituímos os 16 fatos de movimentos por um conjunto de 8 regras para gerar movimentos válidos do cavalo. Esses movimentos (produções) correspondem aos 8 modos possíveis de se mover um cavalo (Figura 6.5).

Se indexarmos o tabuleiro por números de linhas e colunas, podemos definir uma regra de produção para mover o cavalo dois quadrados abaixo e um quadrado à direita por:

**CONDIÇÃO:** linha atual  $\leq 6 \wedge$  coluna atual  $\leq 7$

**AÇÃO:** linha nova = linha atual + 2  $\wedge$  coluna nova = coluna atual + 1

Se usarmos cálculo de predicados para representar produções, então um quadrado do tabuleiro poderá ser definido pelo predicado `quadrado(L,C)`, representando a L-ésima linha e a C-ésima coluna do tabuleiro. A regra acima poderia ser reescrita em cálculo de predicados como:

```
mova(quadrado(Linha, Coluna), quadrado(Novalinha, Novacoluna)) ←
    menor_ou_igual(Linha, 6) ∧
    igual(Novalinha, mais(Linha, 2)) ∧
    menor_ou_igual(Coluna, 7) ∧
    igual(Novacoluna, mais(Coluna, 1))
```

`mais` é uma função para a adição; `menor_ou_igual` e `igual` têm as interpretações aritméticas óbvias. Podemos projetar ainda sete regras adicionais que calculam os demais movimentos possíveis. Essas regras substituem os fatos `mova` na versão  $3 \times 3$  do problema.

A definição caminho do exemplo  $3 \times 3$  define o laço de controle para este problema. Como vimos, quando descrições do cálculo de predicados são interpretadas de modo procedural, como no caso do algoritmo `busca_padrão`, são feitas modificações sutis à semântica do cálculo de predicados. Uma dessas modificações é a maneira sequencial com que os objetivos são resolvidos. Isso impõe uma ordenação, ou uma *semântica procedural*, à interpretação das expressões do cálculo de predicados. Outra modificação é a introdução de predicados *metalógicos* como `declare`, que indicam ações além da interpretação dos valores verdade de expressões do cálculo de predicados.

### Exemplo 6.2.4 O consultor financeiro como um sistema de produção

Nos capítulos 2 e 3, desenvolvemos um pequeno consultor financeiro usando o cálculo de predicados para representar o conhecimento financeiro e a busca em grafo para realizar as inferências apropriadas em uma consulta. O sistema de produção fornece um veículo natural para a sua implementação. As implicações da descrição lógica formam as produções. A informação específica do caso (o salário e os dependentes de um indivíduo etc.) é armazenada na memória de trabalho. A habilitação de regras se dá quando as suas premissas são satisfeitas. Uma regra é escolhida a partir desse conjunto de conflitos e disparada, sendo a sua conclusão acrescentada à memória de trabalho. Isso continua até que todas as possíveis conclusões de alto nível tenham sido acrescentadas à memória de trabalho. Muitos ambientes de desenvolvimento (*shells*) de sistemas de especialistas, incluindo JESS e CLIPS, são, na verdade, sistemas de produção aos quais foram adicionadas características para que suportassem a interface com o usuário, o tratamento de incertezas no processo de raciocínio, a edição da base de conhecimento e a execução do rastreamento.

## 6.2.3 Controle da busca em sistemas de produção

O modelo de sistema de produção oferece uma gama de oportunidades para adicionar controle heurístico a um algoritmo de busca. Aí se incluem a escolha de estratégias guiada por dados ou por objetivo, a estrutura das próprias regras e a escolha de estratégias para resolução de conflitos.

### Controle por escolha da estratégia de busca guiada por dados ou por objetivo

A busca guiada por dados começa com uma descrição do problema (por exemplo, um conjunto de axiomas lógicos, sintomas de uma doença ou um corpo de dados que necessita interpretação) e infere conhecimento novo a partir dos dados. Isso é realizado pela aplicação de regras de inferência, movimentos válidos de um jogo ou outras operações de geração de estados aplicadas à descrição atual do mundo, e pelo acréscimo dos resultados à descrição do problema. Esse processo continua até que um estado objetivo seja alcançado.

Essa descrição de raciocínio guiado por dados enfatiza a sua adequação ao modelo de computação de sistema de produção. O “estado atual do mundo” (dados que são considerados verdadeiros ou cuja veracidade é deduzida a partir da utilização prévia de regras de produção) é inserido na memória de trabalho. O ciclo reconhece-atua tenta casar então o estado atual com o conjunto (ordenado) de produções. Quando houver casamento (unificação) entre esses dados e as condições de uma das regras de produção, a ação dessa produção acrescenta (por meio de alteração da memória de trabalho) uma nova parcela de informação ao estado atual do mundo.

Todas as produções têm a forma CONDIÇÃO → AÇÃO. Quando a CONDIÇÃO casa com alguns elementos da memória de trabalho, a sua AÇÃO é realizada. Se as regras de produção forem formuladas como implicações lógicas e a AÇÃO acrescentar declarações à memória de trabalho, então o ato de disparar uma regra corresponde a uma aplicação da regra de inferência *modus ponens*. Com isso, é criado um novo estado do grafo.

A Figura 6.9 apresenta uma busca guiada por dados simples de um conjunto de produções expressas como implicações do cálculo proposicional. A estratégia de resolução de conflitos se resume à escolha da regra habilitada que tenha sido disparada há mais tempo (ou que não tenha ainda sido disparada); no caso de empate, escolhe-se a primeira regra. A execução para quando um objetivo for alcançado. A figura apresenta, também, a sequência de disparos de regras e os estágios da memória de trabalho durante a execução com um grafo do espaço buscado.

Até este ponto, tratamos sistemas de produção de uma forma guiada por dados; entretanto, eles podem ser usados, também, para produzir uma busca guiada por objetivo. Como definido no Capítulo 3, a busca guiada por objetivo começa com um objetivo e age retroativamente até os fatos do problema que satisfazem esse objetivo. Para que isso seja implementado em um sistema de produção, o objetivo é colocado na memória de trabalho e comparado com as AÇÕES das regras de produção. Essas AÇÕES são casadas (por exemplo, por unificação) da mesma forma que as CONDIÇÕES das produções foram casadas no raciocínio guiado por dados. Todas as regras de produção cujas conclusões (AÇÕES) casam com o objetivo formam o conjunto de conflito.

**Figura 6.9** Busca guiada por dados em um sistema de produção.

#### Conjunto de produção:

1.  $p \wedge q \rightarrow$  objetivo
2.  $r \wedge s \rightarrow p$
3.  $w \wedge r \rightarrow q$
4.  $t \wedge u \rightarrow q$
5.  $v \rightarrow s$
6. inicio →  $v \wedge r \wedge q$

#### Sequência da execução:

Nº iteração	Memória de trabalho	Conjunto de conflito	Regra disparada
0	inicio	6	6
1	inicio, v, r, q	6, 5	5
2	inicio, v, r, q, s	6, 5, 2	2
3	inicio, v, r, q, s, p	6, 5, 2, 1	1
4	inicio, v, r, q, s, p, objetivo	6, 5, 2, 1	parar

#### Espaço buscado pela execução:



Quando ocorre casamento da AÇÃO de uma regra, as CONDIÇÕES são inseridas na memória de trabalho e se tornam subobjetivos (estados) da busca. Os novos estados são, então, comparados com as AÇÕES de outras regras de produção. Esse processo continua até que fatos sejam encontrados, normalmente na descrição inicial do problema, ou, como ocorre frequentemente em sistemas especialistas, por meio de questionamento direto do usuário acerca de informações específicas. A busca para quando as CONDIÇÕES de todas as produções que foram disparadas dessa forma retroativa são consideradas verdadeiras. Essas CONDIÇÕES e a cadeia de disparos de regras que leva ao objetivo original formam uma prova de sua verdade por meio de inferências sucessivas, tal como o *modus ponens*. Veja, na Figura 6.10, um exemplo de raciocínio guiado por objetivo sobre o mesmo conjunto de regras de produção usado na Figura 6.9. Note que a busca guiada por objetivo dispara uma série diferente de produções e busca um espaço diferente da versão guiada por dados.

Como ilustra essa discussão, o sistema de produção oferece uma caracterização natural tanto da busca guiada por objetivo quanto da guiada por dados. As regras de produção são o conjunto codificado de inferências (o “conhecimento” em um sistema especialista baseado em regras) para mudar de estado dentro do grafo. Quando o estado atual do mundo (o conjunto de declarações verdadeiras que descreve o mundo) casa com as CONDIÇÕES das regras de produção e esse casamento faz com que a AÇÃO da regra crie outro descritor (verdadeiro) do mundo, isso é chamado de busca guiada por dados.

Por outro lado, quando o objetivo é comparado com as AÇÕES das regras do conjunto de regras de produção e as suas CONDIÇÕES são consideradas como subobjetivos a serem demonstrados como “verdadeiros” (pelo casamento das AÇÕES das regras no próximo ciclo do sistema de produção), o resultado é a solução do problema guiada por objetivo.

Como um conjunto de regras pode ser executado tanto de uma maneira guiada por dados como guiada por objetivo, podemos comparar e contrastar a eficiência de cada uma dessas abordagens em controlar a busca. A complexidade da busca para cada estratégia é medida por noções como *fator de ramificação* ou *penetrância* (Seção 4.5). Essas medidas de complexidade de busca podem fornecer uma estimativa de custo para ambas as versões, guiadas por dados e por objetivo, de um sistema para solução de problemas e, portanto, podem ajudar a selecionar a estratégia mais eficaz.

Podemos empregar, também, combinações dessas estratégias. Por exemplo, podemos realizar a busca para a frente até que o número de estados se torne grande e, então, mudar para uma busca guiada por objetivo de modo a

**Figura 6.10** Busca guiada por objetivo em um sistema de produção.

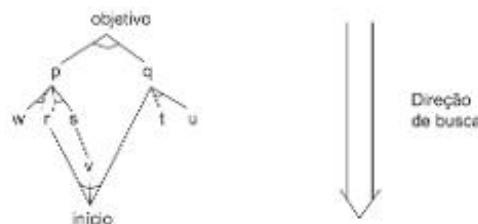
**Conjunto de produção:**

1.  $p \wedge q \rightarrow$  objetivo
2.  $r \wedge s \rightarrow p$
3.  $w \wedge r \rightarrow p$
4.  $t \wedge u \rightarrow q$
5.  $v \rightarrow s$
6. inicio  $\rightarrow v \wedge r \wedge q$

**Sequência da execução:**

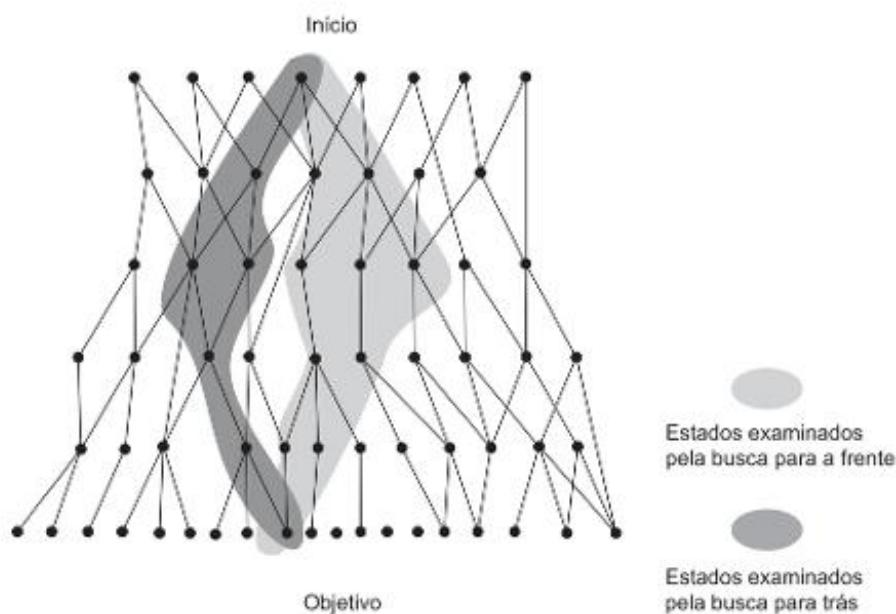
Nº Iteração	Memória de trabalho	Conjunto de conflito	Regra disparada
0	objetivo	1	1
1	objetivo, p, q	1, 2, 3, 4	2
2	objetivo, p, q, r, s	1, 2, 3, 4, 5	3
3	objetivo, p, q, r, s, w	1, 2, 3, 4, 5	4
4	objetivo, p, q, r, s, w, t, u	1, 2, 3, 4, 5	5
5	objetivo, p, q, r, s, w, t, u, v	1, 2, 3, 4, 5, 6	6
6	objetivo, p, q, r, s, w, t, u, v, inicio	1, 2, 3, 4, 5, 6	parar

**Espaço buscado pela execução:**

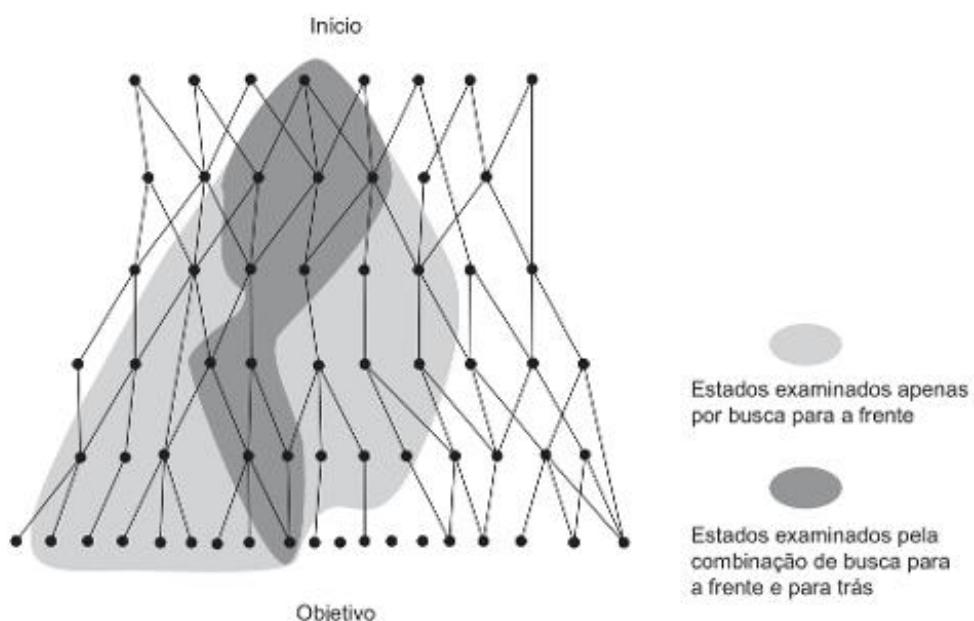


usar possíveis subobjetivos para selecionar entre estados alternativos produzidos pela primeira busca. O perigo nessa situação é que, quando usamos heurísticas ou busca pela melhor escolha (Capítulo 4), as partes dos grafos realmente buscadas podem se desencontrar, demandando mais busca do que uma abordagem mais simples, como na Figura 6.11. Entretanto, quando a ramificação de um espaço for constante e for usada uma busca exaustiva, uma estratégia combinada de busca poderá reduzir drasticamente o tamanho do espaço buscado, como vemos na Figura 6.12.

**Figura 6.11** Busca bidirecional com desencontro em ambas as direções, resultando em busca excessiva.



**Figura 6.12** Busca bidirecional com encontro no meio, eliminando muito do espaço examinado pela busca unidirecional.



### **Controle de busca por meio de estrutura de regras**

A estrutura das regras em um sistema de produção, incluindo a distinção entre a condição e a ação e a ordem na qual as condições são testadas, determina o modo com que o espaço é buscado. Ao introduzirmos o cálculo de predicados como uma linguagem de representação, enfatizamos a natureza *declarativa* de sua semântica. Isto é, as expressões do cálculo de predicados simplesmente definem relacionamentos verdadeiros em um domínio de problema, e não fazem afirmações sobre a ordem na qual os seus componentes são interpretados. Assim, uma regra individual poderia ser  $\forall X (foo(X) \wedge goo(X) \rightarrow moo(X))$ . Pelas regras do cálculo de predicados, uma forma alternativa para essa mesma regra seria  $\forall X (foo(X) \rightarrow moo(X) \vee \neg goo(X))$ . A relação de equivalência entre essas duas cláusulas foi deixada como exercício no Capítulo 2.

Embora essas formulações sejam logicamente equivalentes, elas não levam aos mesmos resultados quando interpretadas como produções, porque o sistema de produção impõe uma ordem para o casamento e para o disparo de regras. Por essa razão, a forma específica das regras determina a facilidade (ou a possibilidade) de casamento de uma regra com uma ocorrência do problema. Esse é um resultado das diferenças no modo com que o sistema de produção *interpreta* as regras. O sistema de produção impõe uma *semântica procedimental* sobre a linguagem declarativa usada para formar as regras.

Como um sistema de produção testa cada uma das suas regras em uma ordem específica, o programador pode controlar a busca a partir da estrutura e da ordem das regras no conjunto de produção. Embora logicamente equivalentes,  $\forall X (foo(X) \wedge goo(X) \rightarrow moo(X))$  e  $\forall X (foo(X) \rightarrow moo(X) \vee \neg goo(X))$  não têm o mesmo comportamento em uma implementação de busca.

Especialistas humanos codificam heurísticas cruciais por meio de regras de competência. A ordem das premissas codifica informações procedimentais geralmente críticas para a solução do problema. É importante que essa forma seja preservada ao se construir um programa que “resolva problemas como o especialista”. Quando um mecânico diz: “se o motor não der partida e as luzes não acenderem, então verifique a bateria”, ele está sugerindo uma sequência específica de ações. Essa informação não é captada pela declaração logicamente equivalente “o motor dá partida, ou as luzes acendem, ou verifique a bateria”. Essa forma das regras é crítica para o controle da busca, fazendo com que o sistema se comporte logicamente, tornando a ordem dos disparos de regras mais compreensível.

### **Controle de busca por meio da resolução de conflitos**

Apesar de os sistemas de produção (como todas as arquiteturas de sistemas baseados em conhecimento) permitirem que heurísticas sejam codificadas no próprio conteúdo do conhecimento das regras, eles oferecem outras oportunidades para o controle heurístico por meio da resolução de conflitos. Embora a estratégia mais simples seja escolher a primeira regra que casar com o conteúdo da memória de trabalho, qualquer estratégia pode ser potencialmente aplicada à resolução de conflitos. Por exemplo, OPS5 (Brownston et al., 1985) suporta as seguintes estratégias para resolução de conflitos:

1. *Regração*. A regração especifica que, uma vez que uma regra tenha sido disparada, ela não poderá disparar novamente até que os elementos da memória de trabalho que casem com essas condições tenham sido modificados. Isso faz com que laços sejam desencorajados.
2. *Atualidade*. A estratégia de atualidade prefere regras cujas condições casem com os padrões que foram acrescentados mais recentemente à memória de trabalho. Isso faz com que a busca seja focada em uma única linha de raciocínio.
3. *Especificidade*. Essa estratégia considera que é apropriado usar uma regra mais específica para resolver um problema do que usar uma regra mais geral. Uma regra é mais específica que outra se ela tiver mais condições, o que implica que ela casará com menos padrões da memória de trabalho.

#### **6.2.4 Vantagens dos sistemas de produção para IA**

Como ilustram os exemplos anteriores, o sistema de produção oferece um arcabouço geral para a implementação da busca. Pela sua simplicidade, capacidade de modificação e flexibilidade em aplicar conhecimento para

resolver problemas, o sistema de produção tem demonstrado ser uma ferramenta importante para a construção de sistemas especialistas e para outras aplicações da IA. Entre as maiores vantagens dos sistemas de produção para a inteligência artificial, estão:

**A separação entre conhecimento e controle.** O sistema de produção é um modelo elegante de separação entre conhecimento e controle em um programa de computador. O controle é fornecido pelo ciclo reconhece-atua do laço do sistema de produção, e o conhecimento para a resolução do problema é codificado nas regras propriamente ditas. Entre as vantagens dessa separação estão a facilidade de modificação da base de conhecimento sem requerer modificação do código para o controle do programa e, por outro lado, a habilidade de alterar o código para o controle do programa sem modificar o conjunto de regras de produção.

**Correspondência natural com a busca de espaço de estados.** Os componentes de um sistema de produção correspondem naturalmente aos construtores da busca em espaço de estados. Os estados sucessivos da memória de trabalho formam os nós de um grafo de espaço de estados. As regras de produção são o conjunto de transições possíveis entre estados, e a resolução de conflitos implementa a seleção de um ramo no espaço de estados. Essas regras simplificam a implementação, a correção e a documentação de algoritmos de busca.

**Modularidade das regras de produção.** Um aspecto importante do modelo de sistema de produção é a ausência de qualquer interação sintática entre regras de produção. As regras podem efetuar o disparo de outras regras apenas pela modificação dos padrões contidos na memória de trabalho; elas não podem “chamar” diretamente outras regras como se fossem sub-rotinas, nem fixar valores de variáveis em outras regras de produção. O escopo das variáveis dessas regras está confinado à regra individual. Essa independência sintática permite o desenvolvimento incremental de sistemas especialistas pelas sucessivas inclusão, exclusão ou modificação do conhecimento (regras) do sistema.

**Controle guiado por padrões.** Os problemas tratados por programas de IA requerem uma flexibilidade particular na execução do programa. Esse objetivo é alcançado pelo fato de que as regras em um sistema de produção podem disparar em qualquer sequência. As descrições de um problema que compõem o estado atual do mundo determinam o conjunto de conflito e, consequentemente, o caminho de busca e a solução particulares.

**Oportunidades para controle heurístico da busca.** (Várias técnicas para controle heurístico foram descritas na seção anterior.)

**Acompanhamento e explanação.** A modularidade das regras e a natureza iterativa de sua execução tornam mais fácil o acompanhamento da execução de um sistema de produção. A cada estágio do ciclo reconhece-atua, a regra selecionada pode ser apresentada. Como a cada regra corresponde uma única “porção” de conhecimento, o conteúdo da regra deve fornecer uma explanação significativa do estado atual do sistema e da ação. Além disso, a sequência de regras usadas em um processo de solução reflete tanto um caminho no grafo como uma “linha de raciocínio” de um especialista humano, como veremos em detalhe no Capítulo 8. Por outro lado, uma única linha de código ou procedimento em uma linguagem tradicional, como Pascal, FORTRAN ou Java, praticamente não tem significado.

**Independência de linguagem.** O modelo de controle por sistema de produção é independente da representação escolhida para as regras e para a memória de trabalho, contanto que essa representação permita casamento de padrões. Descrevemos regras de produção como implicações do cálculo de predicados da forma  $A \Rightarrow B$ , onde a verdade de  $A$  e a regra de inferência do *modus ponens* nos permitem concluir  $B$ . Embora haja muitas vantagens no uso da lógica como base para a representação de conhecimento e como fonte de regras de inferência consistentes, o modelo de sistema de produção pode ser usado também com outras representações, por exemplo, CLIPS e JESS.

Embora o cálculo de predicados ofereça a vantagem da inferência logicamente consistente, muitos problemas requerem raciocínio que não seja consistente em um sentido lógico. Em vez disso, eles envolvem raciocínio probabilístico, uso de evidências incertas e pressuposições. Os capítulos 7, 8 e 9 discutem regras de inferência alternativas que fornecem essas capacidades. A despeito do tipo de regras de inferência que forem empregadas, o sistema de produção fornece um veículo para a busca em espaço de estados.

**Modelo plausível de solução humana de problemas.** A modelagem da solução humana de problemas está entre os primeiros usos de sistemas de produção [ver Newell e Simon (1972)]. Eles continuam a ser usados como modelo de desempenho humano em muitas áreas de pesquisa em ciência cognitiva (Capítulo 16).

A busca guiada por padrões nos fornece a capacidade de explorar o espaço de inferências lógicas do cálculo de predicados. Muitos problemas se baseiam nessa técnica usando cálculo de predicados para modelar aspectos específicos do mundo, como tempo e alterações. Na próxima seção, são apresentados os *sistemas de quadro-negro (blackboard systems)* como uma variação da metodologia do sistema de produção, onde grupos de regras de produção, específicos para determinadas tarefas, são combinados em *fontes de conhecimento* e cooperam no processo de solução do problema pela comunicação por meio de uma memória de trabalho global, ou *quadro-negro*.

### 6.3 Arquitetura de quadro-negro na solução de problemas

O *quadro-negro* é o último mecanismo de controle a ser apresentado neste capítulo. Quando desejamos examinar os estados em um espaço de inferências lógicas de uma maneira bastante determinística, os sistemas de produção fornecem alta flexibilidade, permitindo-nos representar múltiplas soluções parciais simultaneamente na memória de trabalho e escolher o próximo estado por meio da resolução de conflitos. Os quadros-negros estendem os sistemas de produção, permitindo-nos organizar a memória de trabalho em módulos separados, cada qual correspondendo a um subconjunto diferente de regras de produção. Os quadros-negros integram esses conjuntos separados de regras de produção e coordenam as ações desses múltiplos resolvidores de problemas, também denominados *fontes de conhecimento*, dentro de uma única estrutura global: o *quadro-negro*.

Muitos problemas requerem a coordenação de vários tipos diferentes de agentes. Por exemplo, um programa de compreensão da fala deve, primeiro, manipular uma expressão vocal representada como uma forma de onda digitalizada. Conforme o processo de compreensão avança, ele necessita encontrar palavras nessa expressão vocal, agrupando-as em frases e sentenças e, finalmente, produzindo uma representação semântica do significado da expressão vocal.

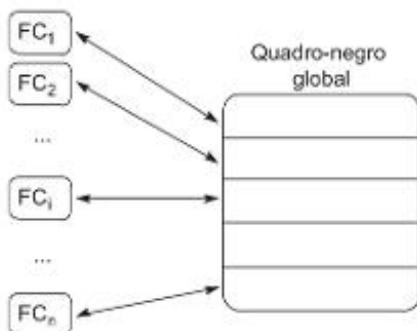
Outro problema relacionado ocorre quando múltiplos processos necessitam cooperar para resolver um único problema. Um exemplo disso é o problema da fusão de sensores (Lesser e Corkill, 1983). Suponha que tenhamos uma rede de sensores, cada qual sendo monitorado por um processo separado. Suponha, também, que os processos possam se comunicar e que a interpretação adequada dos dados de cada sensor dependa dos dados recebidos pelos outros sensores da rede. Esse problema surge em diversas situações, desde a monitoração de uma aeronave por meio de múltiplos pontos de radar até a combinação das leituras de múltiplos sensores em um processo de fabricação.

A *arquitetura de quadro-negro* é um modelo de controle que tem sido aplicado a esses e a outros problemas que requerem coordenação de múltiplos processos ou fontes de conhecimento. Um *quadro-negro* é uma base de dados global central para a comunicação de fontes de conhecimento independentes assíncronas, que focam nos aspectos relacionados de um problema particular. A Figura 6.13 apresenta um esquema do projeto de quadro-negro.

Nessa figura, cada *fonte de conhecimento*  $FC_i$  obtém os seus dados do quadro-negro, processa-os e retorna seus resultados no quadro-negro para serem usados por outras fontes de conhecimento. Cada  $FC_i$  é independente na medida que é um processo separado, operando de acordo com suas próprias especificações e, quando um sistema de multiprocessamento ou multiprocessador for usado, é independente dos outros processamentos envolvidos no problema. Cada  $FC_i$  é um sistema assíncrono, pois começa a sua operação tão logo encontre dados apropriados no quadro-negro. Quando encerra seu processamento, ela divulga seus resultados no quadro-negro e aguarda por novos dados de entrada apropriados.

A abordagem de quadro-negro para organizar um grande programa foi primeiro apresentada no projeto de pesquisa HEARSAY-II (Erman et al., 1980; Reddy, 1976). HEARSAY-II era um programa para a compreensão da fala; ele foi inicialmente concebido como uma interface para um banco de dados de uma biblioteca de artigos

**Figura 6.13** Arquitetura de quadro-negro.



da ciência da computação. O usuário da biblioteca se comunicaria com o computador em inglês falado com consultas do tipo: “existe algo de Feigenbaum e Feldman?”, e o computador responderia à pergunta com a informação do banco de dados da biblioteca. A compreensão da fala requer a integração de vários processos diferentes, sendo que todos eles necessitam conhecimentos e algoritmos bem diferentes, podendo ter complexidade exponencial. O processamento de sinal, o reconhecimento de fonemas, de sílabas e de palavras, bem como as análises sintática e semântica restringem-se mutuamente na interpretação da fala.

A arquitetura de quadro-negro permitiu ao HEARSAY-II coordenar as diferentes fontes de conhecimento necessárias para essa tarefa complexa. A organização do quadro-negro normalmente se dá em duas dimensões. No HEARSAY-II, essas dimensões são o *tempo* no qual o ato da fala foi produzido e o *nível de análise* da expressão vocal. Cada nível de análise é processado por uma classe diferente de fontes de conhecimento. Esses níveis de análise são:

- FC<sub>1</sub> A forma de onda do sinal acústico.
- FC<sub>2</sub> Os fonemas ou os segmentos sonoros possíveis do sinal acústico.
- FC<sub>3</sub> As sílabas que os fonemas poderiam produzir.
- FC<sub>4</sub> As palavras possíveis conforme analisado por uma FC.
- FC<sub>5</sub> As palavras possíveis conforme analisado por uma segunda FC (normalmente considerando palavras de partes diferentes dos dados).
- FC<sub>6</sub> Uma FC para tentar gerar sequências possíveis de palavras.
- FC<sub>7</sub> Uma FC que coloca sequências de palavras em frases possíveis.

Podemos visualizar esses processos como componentes na Figura 6.13. No processamento do discurso falado, a forma de onda do sinal da fala é introduzida no nível mais baixo. As fontes de conhecimento para processar essa entrada são habilitadas e comunicam as suas interpretações para o quadro-negro, a fim de que sejam utilizadas pelo processo apropriado. Por causa das ambiguidades da linguagem falada, várias hipóteses concorrentes podem estar presentes em cada nível do quadro-negro. As fontes de conhecimento dos níveis mais altos tentam dirimir as ambiguidades dessas hipóteses alternativas.

A análise de HEARSAY-II não deve ser vista simplesmente como um nível mais baixo produzindo dados que os níveis mais altos podem analisar. Ela é muito mais complexa que isso. Se uma FC em um nível não é capaz de processar (fazendo sentido) os dados enviados a ela, essa FC pode requisitar que a FC que lhe enviou os dados retroceda para realizar uma nova tentativa ou formule outra hipótese acerca dos dados. Além disso, FCs diferentes podem atuar em partes diferentes da expressão vocal ao mesmo tempo. Todos os processos, como anteriormente mencionado, são assíncronos e guiados por dados; eles agem quando recebem dados de entrada, continuam agindo até que tenham terminado a sua tarefa e, então, comunicam os seus resultados e aguardam sua próxima tarefa.

Uma das FCs, chamada de *escalonador*, trata da comunicação “consumir dados, divulgar resultados” entre as FCs. Esse escalonador coloca classificações nos resultados de cada atividade das FCs e, dessa forma, é capaz de dar,

por meio de uma fila de prioridades, uma direção à solução do problema. Se nenhuma FC estiver ativa, o escalonador determina que a tarefa terminou e encerra seu trabalho.

Quando o programa HEARSAY processava um banco de dados de cerca de 1.000 palavras, ele funcionava relativamente bem, apesar de ser um pouco lento. Quando a base de dados era maior que isso, os dados se tornavam mais complexos do que as fontes de conhecimento eram capazes de tratar. HEARSAY-III (Balzer et al., 1980; Erman et al., 1981) é uma generalização do método adotado por HEARSAY-II. A dimensão de tempo do HEARSAY-II não era mais necessária, mas foram mantidas as múltiplas FCs para os níveis de análise. O quadro-negro do HEARSAY-III foi concebido para interagir com um sistema de banco de dados relacionais de propósito geral. Na verdade, HEARSAY-III é um ambiente para o projeto de sistemas especialistas; ver Seção 8.1.1.

Uma modificação importante em HEARSAY-III foi a divisão da FC escalonadora (como descrito anteriormente para o HEARSAY-II), tornando-a um controlador de quadro-negro separado para o primeiro quadro-negro (do domínio do problema). O segundo quadro-negro permite que o processo de escalonamento seja decomposto, da mesma forma que o problema também foi, em FCs separadas que cuidam dos diferentes aspectos do procedimento da solução (por exemplo, quando e como aplicar o conhecimento do domínio). O segundo quadro-negro pode, assim, comparar e balancear soluções diferentes para cada problema (Nii e Aiello, 1979; Nii, 1986a, 1986b). Um modelo alternativo mantém partes importantes da base de conhecimento no quadro-negro, em vez de distribuí-las entre as fontes de conhecimento (Skinner e Luger, 1991, 1992).

## 6.4 Epílogo e referências

O Capítulo 6 discutiu a implementação das estratégias de busca dos capítulos 3 e 4. Assim, as referências listadas no epílogo daqueles capítulos são também apropriadas para este capítulo. A Seção 6.1 apresentou a recursão como uma ferramenta importante para a programação de busca em grafo, implementando o algoritmo de busca em profundidade e com retrocesso do Capítulo 3 na forma recursiva. A busca guiada por padrões usando unificação e regras de inferência, como apresentadas no Capítulo 2, simplifica a implementação da busca através de um espaço de inferências lógicas.

Na Seção 6.2, o sistema de produção foi apresentado como uma arquitetura natural para modelar a solução de um problema e para implementar algoritmos de busca. A seção foi concluída com exemplos de implementações, por sistema de produção, de buscas guiadas por dados e por objetivo. Na verdade, o sistema de produção sempre foi um importante paradigma para a programação de IA, começando com o trabalho de Newell e Simon e seus colegas da Carnegie Mellon University (Newell e Simon, 1976; Klahr et al., 1987; Neches et al., 1987; Newell et al., 1989). O sistema de produção tem sido, também, uma arquitetura importante de apoio às pesquisas em ciências cognitivas (Newell e Simon, 1972; Luger, 1994; ver também o Capítulo 16).

Como referências sobre implementação de sistemas de produção, especialmente as linguagens OPS, citamos *Programming Expert Systems in OPS5*, de Lee Brownston et al. (1985), e *Pattern Directed Inference Systems*, de Donald Waterman e Frederick Hayes-Roth (1978). Sistemas de produção modernos em C e em Java podem ser encontrados nos sites para CLIPS e JESS.

Trabalhos iniciais em modelos de quadro-negro são descritos no projeto de pesquisa HEARSAY-II (Reddy, 1976; Erman et al., 1980). Progressos mais recentes em arquiteturas de quadro-negro são descritos no trabalho sobre o HEARSAY-III (Lesser e Corkill, 1983; Nii, 1986a; Nii, 1986b) e em *Blackboard Systems*, editado por Robert Engelmore e Tony Morgan (1988).

A pesquisa em sistemas de produção, planejamento e arquiteturas de quadro-negro continua sendo um tema ativo na inteligência artificial. Recomendamos que os leitores interessados consultem os anais recentes da *American Association for Artificial Intelligence Conference* e da *International Joint Conference on Artificial Intelligence*. A Morgan Kaufmann e a AAAI Press publicam anais de conferências e coleções de leituras sobre tópicos de IA.

## 6.5 Exercícios

1.
  - a. Escreva um algoritmo de verificação de pertinência para determinar recursivamente se determinado elemento é um membro de uma lista.
  - b. Escreva um algoritmo para contar o número de elementos em uma lista.
  - c. Escreva um algoritmo para contar o número de átomos em uma lista.  
(A distinção entre átomos e elementos é que um elemento pode ser ele mesmo uma lista.)
2. Escreva um algoritmo recursivo (usando as listas abertos e fechados) para implementar a busca em amplitude. A recursão permite a omissão da lista abertos quando se implementa a busca em amplitude? Explique.
3. Faça o acompanhamento da execução do algoritmo recursivo de busca em profundidade (a versão que não usa a lista abertos) sobre o espaço de estados da Figura 3.14.
4. Em uma antiga cerimônia hindu do chá, há três participantes: um ancião, um servente e uma criança. As quatro tarefas que eles executam são: alimentar o fogo, servir bolos, servir o chá e ler poesia; essa ordem reflete a importância decrescente das tarefas. No inicio da cerimônia, a criança executa as quatro tarefas. Elas são então passadas, uma por vez, para o servente e o ancião até que, no final da cerimônia, o ancião passa a realizar as quatro tarefas. Ninguém pode receber uma tarefa menos importante do que aquela que já realizou. Gere uma sequência de movimentos para transferir todas as tarefas da criança para o ancião. Escreva um algoritmo recursivo para executar a sequência de movimentos.
5. Usando as definições de *mova* e *caminho* para o percurso do cavalo da Seção 6.2.2, rastreie a execução de *busca\_padrão* para os objetivos:
  - a. *caminho(1,9)*.
  - b. *caminho(1,5)*.
  - c. *caminho(7,6)*.Quando os predicados *mova* são tentados em ordem, frequentemente existem laços na busca. Discuta a detecção de laços e o retrocesso nessa situação.
6. Escreva a definição em pseudocódigo para uma versão em amplitude da função *busca\_padrão* (Seção 6.1.2). Discuta a eficiência de tempo e espaço desse algoritmo.
7. Usando a regra do Exemplo 6.2.3 como um modelo, escreva as oito regras de movimento necessárias para a versão completa  $8 \times 8$  do percurso do cavalo.
8. Usando os estados inicial e objetivo da Figura 6.3, execute à mão a solução por sistema de produção para o quebra-cabeça dos 8:
  - a. de maneira guiada por objetivo.
  - b. de maneira guiada por dados.
9. Considere o problema do consultor financeiro discutido nos capítulos 2, 3 e 4. Usando o cálculo de predicados como uma linguagem de representação:
  - a. escreva o problema explicitamente como um sistema de produção.
  - b. gere o espaço de estados e os estágios da memória de trabalho para a solução guiada por dados para o exemplo do Capítulo 3.
10. Repita o Problema 9(b) para produzir uma solução guiada por objetivo.
11. A Seção 6.2.3 apresentou as estratégias gerais para resolução de conflitos por refração, atualidade e especificidade. Proponha e justifique duas novas estratégias desse tipo.
12. Sugira duas aplicações apropriadas para solução usando a arquitetura de quadro-negro. Caracterize brevemente a organização do quadro-negro e as fontes de conhecimento para cada implementação.

# Capturando inteligência: o desafio da IA

*A nossa era de ansiedade é, em grande parte, o resultado de se tentar realizar as tarefas de hoje em dia com as ferramentas de antigamente...*

— MARSHALL McLUHAN

*Será que não é mais proveitoso considerar-se o cérebro como um controlador da atividade incorporada? Essa pequena mudança de ponto de vista tem grandes implicações em como construímos uma ciência da mente. Isso demanda, de fato, uma reforma radical na nossa forma de considerar o comportamento inteligente. Isso exige que abandonemos as ideias (comuns desde Descartes) da mente como um domínio distinto do domínio do corpo; que abandonemos a ideia de linhas divisórias claras entre percepção, cognição e ação; que abandonemos a ideia de um centro executivo onde o cérebro realiza raciocínio de alto nível; e, principalmente, que abandonemos métodos de investigação que dissociem artificialmente o pensamento da execução de uma ação incorporada...*

— ANDY CLARK, *Being There* (1997)

## Representação e inteligência

A questão da *representação*, ou de como captar, da melhor forma possível, os aspectos críticos da atividade inteligente para uso em um computador, ou mesmo para a comunicação com os seres humanos, tem sido um tema constante ao longo dos sessenta anos da história da IA. Iniciamos a Parte III com uma revisão das três abordagens para a representação, predominantes na comunidade de pesquisa em IA durante esse período de tempo. O primeiro tema, articulado nos anos 1950 e 1960 por Newell e Simon em seus trabalhos com o *Logic Theorist* (Newell e Simon, 1956, 1963a), é conhecido como *solução de problemas por métodos fracos*. O segundo tema, comum ao longo dos anos 1970 e 1980, e apoiado pelos primeiros projetistas de sistemas especialistas, é a *solução de problemas por métodos fortes* (veja a citação de Feigenbaum no início do Capítulo 8). Nos anos mais recentes, especialmente nos domínios da robótica e da Internet (Brooks, 1987, 1989; Clark, 1997), a ênfase está em abordagens de inteligência *distribuídas e incorporadas ou baseadas em agente*. Introduzimos, a seguir, cada uma dessas metodologias para representar a inteligência. Os três capítulos que constituem a Parte III oferecem descrições mais detalhadas de cada uma dessas abordagens.

No final dos anos 1950 e começo dos anos 1960, Alan Newell e Herbert Simon escreveram vários programas de computador para testar a hipótese de que o comportamento inteligente resultava de busca heurística. O *Logic Theorist*, desenvolvido com J. C. Shaw (Newell e Simon, 1963a), provava teoremas em lógica elementar usando a

notação e os axiomas do *Principia Mathematica*, de Whitehead e Russell (1950). Os autores descrevem sua pesquisa como se ela objetivasse a compreensão.

dos complexos processos (heurísticas) que são eficazes na solução de problemas. Assim, não estamos interessados em métodos que garantam soluções, mas que requeiram muita computação. Em vez disso, desejamos compreender, por exemplo, como um matemático é capaz de provar um teorema mesmo que, de inicio, ele não saiba como, ou se conseguirá fazê-lo.

Em um programa mais recente, o *General Problem Solver* (GPS), Newell e Simon (1963b, 1972) continuaram esse esforço para encontrar princípios gerais da solução inteligente de problemas. O GPS solucionava problemas formulados como busca em espaço de estados; os passos válidos para solucionar um problema eram um conjunto de operações para modificar representações de estado. O GPS buscava uma sequência de operações que transformasse o estado inicial no estado objetivo, realizando a busca da mesma maneira que os algoritmos discutidos nos capítulos anteriores deste livro.

O GPS usava a *análise de meios e fins*, uma heurística geral para realizar uma seleção entre operações alternativas de transformação de estado, para guiar a busca através do espaço do problema. A análise de meios e fins examina as diferenças *sintáticas* entre o estado atual e o estado objetivo e seleciona um operador que reduza essas diferenças. Suponha, por exemplo, que o GPS tente provar a equivalência de duas expressões lógicas. Se o estado atual contiver um operador  $\wedge$  e o objetivo não contiver um  $\wedge$ , então a análise de meios e fins selecionará uma transformação como a lei de De Morgan, para remover os  $\wedge$  das expressões (veja a Seção 14.1 para obter uma descrição mais detalhada e exemplos dessa pesquisa).

Usando uma heurística que examina apenas a forma sintática dos estados, esperava-se que o GPS fosse uma arquitetura *geral* para a solução inteligente de problemas, não importando qual fosse o domínio. Programas como o GPS, que ficam restritos a estratégias baseadas em sintaxe e visam uma grande variedade de aplicações, são conhecidos como *resolvedores de problemas por método fraco*.

Infelizmente, parece não haver uma heurística única que possa ser aplicada com sucesso a todos os domínios de problemas. Em geral, os métodos que usamos para solucionar problemas empregam extensivamente o conhecimento de uma situação. Os médicos são capazes de diagnosticar doenças porque possuem bastante conhecimento de medicina, além das suas habilidades gerais para a solução de problemas. Os arquitetos projetam casas porque eles têm conhecimento de arquitetura. De fato, as heurísticas usadas no diagnóstico médico seriam inúteis para se projetar um prédio comercial.

Os métodos fracos contrastam diretamente com os *métodos fortes*, os quais usam o conhecimento explícito de um domínio de problema particular. Considere uma regra para a análise de diagnóstico de problemas automotivos:

se  
o motor não pega e  
as luzes não acendem  
então  
o problema está na bateria ou nos cabos (0,8)

Essa heurística, expressa como uma regra *se... então...*, foca a busca no subsistema bateria/cabo do automóvel, eliminando outros componentes e podando o espaço de busca. Note que essa heurística em particular, diferentemente da análise de meios e fins, usa o conhecimento empírico sobre como funciona um automóvel, tal como o conhecimento das relações entre a bateria, as luzes e o motor de partida. Esse conhecimento é inútil em qualquer outro domínio de problema, exceto no conserto de automóveis.

Os resolvedores de problemas por método forte não apenas usam conhecimentos específicos do domínio, mas geralmente necessitam de grandes quantidades de conhecimento para serem eficazes. A heurística da bateria ruim, por exemplo, não seria de muita utilidade no diagnóstico de um problema no carburador e seria totalmente inútil em qualquer outro domínio fora do diagnóstico automotivo. Assim, dois dos principais desafios do projeto de programas baseados em conhecimento são a aquisição e a organização de grandes quantidades de conhecimentos específicos baseados em domínio. Regras do domínio podem conter, também, uma medida, 0,8 na regra anterior, para refletir a confiança do diagnosticador nessa parte de conhecimento do domínio.

Ao usar a abordagem de método forte, os projetistas de programas fazem certas suposições acerca da natureza dos sistemas inteligentes. Essas suposições são formalizadas por Brian Smith (1985) como a *hipótese da representação do conhecimento*. Essa hipótese afirma que:

qualquer processo inteligente mecanicamente incorporado será composto de ingredientes estruturais que (a) nós, como observadores externos, naturalmente percebemos que representam uma explicação proposicional do conhecimento que o processo global exibe e (b), independentemente de tal atribuição semântica externa, desempenham um papel formal na geração do comportamento que manifesta esse conhecimento.

Os aspectos importantes dessa hipótese incluem a suposição de que o conhecimento será representado *de modo proposicional*, isto é, em uma forma que represente explicitamente o conhecimento em questão e que possa ser vista por um observador externo como uma descrição “natural” daquele conhecimento. A segunda suposição fundamental é a de que o comportamento do sistema poderá ser diretamente causado pelas proposições contidas na base de conhecimento e que esse comportamento poderá ser consistente com o significado percebido por nós daquelas proposições.

O tema final para a representação de IA é descrito, frequentemente, como a solução de problemas *baseada em agentes, incorporada ou emergente*. Vários pesquisadores que trabalham em domínios aplicados, tais como robótica, projeto de jogos e Internet, incluindo Brooks (1987, 1989), Agre e Chapman (1987), Jennings (1995), Wooldridge (2000), Wooldridge et al. (2006, 2007), Fatima et al. (2004, 2005, 2006) e Vieira et al. (2007), têm desafiando a necessidade de se ter uma base de conhecimento centralizada ou um esquema de inferência de propósito geral. Os resolvedores de problemas são projetados como agentes *distribuídos: situados, autônomos e flexíveis*.

Por esse ponto de vista, a solução de problemas é vista como distribuída, com agentes que realizam tarefas em diferentes subcontextos de seus domínios, por exemplo, a atividade de um navegador da Internet ou de um agente de segurança. A tarefa de solucionar o problema é dividida em seus vários componentes com pouca ou nenhuma coordenação geral de tarefas. O agente situado é capaz de receber informações sensoriais de seu ambiente particular, bem como de reagir naquele contexto sem esperar por instruções de um controlador geral. Em jogos interativos, por exemplo, o agente trataria de uma questão local específica, defendendo-se de um ataque em particular ou disparando um alarme específico, sem uma visão geral da situação completa do problema.

Agentes são autônomos na medida em que frequentemente são solicitados a agir sem a intervenção direta de seres humanos ou de um processo de controle geral. Agentes autônomos têm controle sobre as suas próprias ações e seu estado interno. Alguns sistemas de agentes são capazes até de aprender a partir de sua própria experiência. Por fim, agentes são flexíveis porque reagem a situações em seus ambientes locais. Eles também são proativos, pois podem antecipar situações. Finalmente, eles devem ser capazes de reagir colaborativamente em relação a outros agentes incorporados no domínio do problema, comunicando-se acerca de tarefas, objetivos e processos apropriados.

Vários pesquisadores do domínio da robótica construíram sistemas baseados em agentes. Rodney Brooks (1987, 1989), no MIT Robotics Laboratory, projetou o que ele denominou *arquitetura de subsunção (subsumption architecture)*, uma sequência de camadas de máquinas de estados finitos, cada uma agindo no seu contexto particular, mas permitindo, também, a funcionalidade em níveis mais altos. Manuela Veloso et al. (2000), trabalhando no Robotics Lab da Carnegie Mellon, projetou um time de agentes robóticos para jogar futebol, que colaboraram no ambiente competitivo dos campeonatos de futebol.

Por fim, essa abordagem situada e incorporada para a representação é descrita por vários filósofos da ciência, incluindo Dan Dennett (1991, 1995) e Andy Clark (1997), como caracterizações apropriadas da inteligência humana. Mais adiante, apresentamos outros esquemas representacionais, incluindo as abordagens conexionista (Capítulo 11) e genética (Capítulo 12). No Capítulo 16, são discutidas novamente questões representacionais gerais.

No Capítulo 7, examinamos em detalhes muitas das principais abordagens que a comunidade de IA adotou para a representação. Começamos com o uso inicial de representações, incluindo *redes semânticas*, roteiros (*scripts*), quadros (*frames*) e *objetos*. Apresentamos esses esquemas de um ponto de vista evolucionário, mostrando como as ferramentas modernas se originaram desses modelos iniciais da pesquisa em IA. Apresentamos, a seguir, os *grafos conceituais* de John Sowa, uma representação para ser usada em compreensão de linguagem natural. Finalmente, apresentamos os métodos de representação situada e baseada em agentes, incluindo a *arqui-*

tetura de subsunção de Rodney Brooks, que questiona a necessidade de uma base central de conhecimento explícito usada com um controlador de propósito geral.

No Capítulo 8, discutimos os sistemas de conhecimento intensivo e examinamos os problemas envolvidos na aquisição, na formalização e na verificação de uma base de conhecimento. Apresentamos diferentes esquemas de inferência para sistemas de regras, incluindo os raciocínios guiados por objetivo e por dados. Além dos sistemas baseados em regras, apresentamos os sistemas baseados em modelos e em casos. A primeira abordagem tenta representar explicitamente a fundamentação teórica e a funcionalidade de um domínio, por exemplo, um circuito eletrônico, enquanto a segunda constrói uma base de dados cumulativa dos sucessos e dos fracassos passados obtidos no domínio do problema, para auxiliar a solução de problemas no futuro. Concluímos o Capítulo 8 com uma revisão sobre *planejamento*, em que o conhecimento é organizado para controlar a solução do problema em domínios complexos como em robótica.

No Capítulo 9, apresentamos várias técnicas que tratam do problema da representação para raciocinar em situações de vagueza e/ou incerteza. Nessas situações, tentamos obter a melhor explanação possível para a informação que é quase sempre ambígua. Muitas vezes, esse tipo de raciocínio é chamado de *abolutivo*. Apresentamos inicialmente lógicas não monotônicas e de manutenção da verdade, nas quais a lógica de predicados tradicional é estendida para captar situações com incerteza. Entretanto, muitas soluções de problemas interessantes e importantes não se enquadram bem no conceito de lógica dedutiva. Para que se possa raciocinar nessas situações, introduzimos várias outras ferramentas poderosas, incluindo técnicas Bayesianas, a abordagem de Dempster-Shafer e a álgebra do Fator de Certeza de Stanford (*Stanford Certainty Factor*). Temos, também, uma seção sobre raciocínio difuso. Concluímos a Parte III com a metodologia estocástica para a incerteza, apresentando as redes Bayesianas de crença, os modelos de Markov, os modelos ocultos de Markov e métodos relacionados.

# Representação do conhecimento

*Este grande livro, o universo,... está escrito na linguagem da matemática, e os seus personagens são triângulos, círculos e outras figuras geométricas sem as quais é humanamente impossível entender uma palavra dele; sem elas, estariam vagando em um labirinto escuro...*

— GALILEU GALILEI, *Discorsi e Dimonstrazioni Matematiche Intronio a Due Nuove Scienze* (1638)

*Como nenhum organismo pode lidar com a infinita diversidade, uma das funções básicas de todos os organismos é repartir o ambiente em classes pelas quais estímulos não idênticos possam ser tratados como equivalentes...*

— ELEANOR ROSCH, *Principles of Categorization* (1978)

*Nós temos sempre dois universos de discurso — chame-os “físico” e “fenomenal”, ou como preferir —, um que lida com questões de estruturas quantitativas e formais, o outro com aquelas qualidades que constituem um “mundo”. Todos temos os nossos próprios mundos mentais distintos, nossas próprias jornadas e paisagens internas e estas, para a maioria de nós, não exigem uma “correlação” neurológica clara.*

— OLIVER SACKS, *The Man Who Mistook His Wife for a Hat* (1987)

## 7.0 Questões da representação do conhecimento

A representação da informação para ser usada na solução inteligente de problemas oferece desafios importantes e difíceis que se encontram no âmago da IA. Na Seção 7.1, apresentamos uma breve retrospectiva histórica das primeiras pesquisas em representação; entre os tópicos, incluímos as *redes semânticas*, as *dependências conceituais*, os *roteiros (scripts)* e *quadros (frames)*. A Seção 7.2 oferece uma representação mais moderna usada em programas de linguagem natural, os *grafos conceituais* de John Sowa. Na Seção 7.3, criticamos a necessidade de se criar esquemas representacionais centralizados e explícitos. A alternativa de Brooks é a *arquitetura de subsunção* para robótica. A Seção 7.4 apresenta os *agentes*, uma alternativa ao controle centralizado. Nos capítulos seguintes, estendemos nossa discussão sobre representações para incluir as representações estocástica (Seção 9.3 e Capítulo 13), conexionista (Capítulo 11) e genética/emergente (Capítulo 12).

Iniciamos a nossa discussão sobre representação de uma perspectiva histórica, em que uma base de conhecimento é descrita como um mapeamento entre os objetos e as relações de um domínio de problema e entre os objetos computacionais e as relações de um programa (Bobrow, 1975). Os resultados de inferências na base de conhecimento devem corresponder aos resultados de ações ou às observações no mundo. Os objetos, as relações e as inferências computacionais disponíveis para os programadores são mediados pela linguagem de representação de conhecimento.

Existem princípios gerais de organização do conhecimento que se aplicam a uma série de domínios e que podem ser diretamente suportados por uma linguagem de representação. Por exemplo, são encontradas hierar-

quias de classe tanto em sistemas de classificação científicos como nos de senso comum. Como poderíamos conceber um mecanismo geral para representá-las? Como podemos representar definições? E exceções? Quando um sistema inteligente deve fazer suposições preconcebidas sobre uma informação faltante e como ele poderia ajustar o seu raciocínio caso essas suposições se mostrassem erradas? Como podemos representar o tempo da melhor forma possível? E a causalidade? E a incerteza? O progresso na construção de sistemas inteligentes depende da descoberta dos princípios da organização do conhecimento e do suporte a eles por meio de ferramentas representacionais de nível mais alto.

É útil fazer a distinção entre um *esquema representacional* e o *meio* de sua implementação. Isso é semelhante à distinção entre estruturas de dados e linguagens de programação. As linguagens de programação são o *meio* de implementação; a estrutura de dados é o *esquema*. Geralmente, as linguagens para a representação de conhecimento são mais restritas que o cálculo de predicados ou as linguagens de programação. Essas restrições tomam a forma de estruturas explícitas para representar categorias de conhecimento. O meio pelo qual elas são implementadas pode ser o Prolog, o Lisp ou as linguagens mais comuns como C++ ou Java.

A nossa discussão até este ponto ilustra a visão tradicional dos esquemas representacionais de IA, uma visão que frequentemente inclui uma base de conhecimento global de estruturas de linguagem que refletem um “viés pré-interpretado” e estático de um “mundo real”. Mais recentemente em robótica (Brooks, 1991a; Lewis e Luger, 2000), a cognição situada (Agre e Chapman, 1987; Lakoff e Johnson, 1999), a solução de problemas baseada em agentes (Jennings et al., 1998; Wooldridge, 2000; Fatima et al., 2005, 2006; Vieira et al., 2007) e a filosofia (Clark, 1997) têm desafiado essa abordagem tradicional. Esses domínios de problema requerem conhecimento distribuído, um mundo que pode ele próprio ser usado como uma estrutura parcial de conhecimento, capacidade de raciocinar com informação parcial e mesmo representações que evoluem conforme elas experimentam as invariâncias no domínio do problema. Essas abordagens são introduzidas nas seções 7.3 e 7.4.

## 7.1 Uma breve história dos esquemas representacionais de IA

### 7.1.1 Teorias associacionistas do significado

As representações lógicas surgiram dos esforços de filósofos e matemáticos em caracterizar os princípios do raciocínio correto. A principal preocupação da lógica é o desenvolvimento de linguagens de representação formais com regras de inferência consistentes e completas. Como resultado, a semântica do cálculo de predicados enfatiza operações que *preservam a verdade* em expressões bem formadas. Uma linha alternativa de pesquisa se desenvolveu dos esforços de psicólogos e linguistas em caracterizar a natureza da compreensão humana. Esse trabalho está menos preocupado em estabelecer uma ciência do raciocínio correto do que em descrever o modo como os humanos realmente adquirem, associam e usam o conhecimento do seu mundo. Essa abordagem tem se mostrado particularmente útil em áreas de aplicação da IA, como compreensão de linguagem natural e raciocínio de senso comum.

Existem muitos problemas que surgem ao se mapear o raciocínio de senso comum para a lógica formal. Por exemplo, é comum se considerar os operadores  $\vee$  e  $\rightarrow$  como correspondentes ao “ou” e ao “se... então...” do português. Entretanto, esses operadores em lógica estão preocupados apenas com valores verdade e ignoram o fato de que, em português, “se... então...” sugere um relacionamento específico (quase sempre, mas correlacional do que causal) entre suas premissas e sua conclusão. Por exemplo, a sentença “Se um pássaro é um cardeal, então ele é vermelho” (associando o pássaro cardeal à cor vermelha) pode ser escrita em cálculo de predicados como:

$$\forall X (\text{cardeal}(X) \rightarrow \text{vermelho}(X)).$$

Essa expressão pode ser modificada por meio de uma série de operações que preservem o valor verdade (Capítulo 2) na expressão logicamente equivalente:

$$\forall X (\neg \text{vermelho}(X) \rightarrow \neg \text{cardeal}(X)).$$

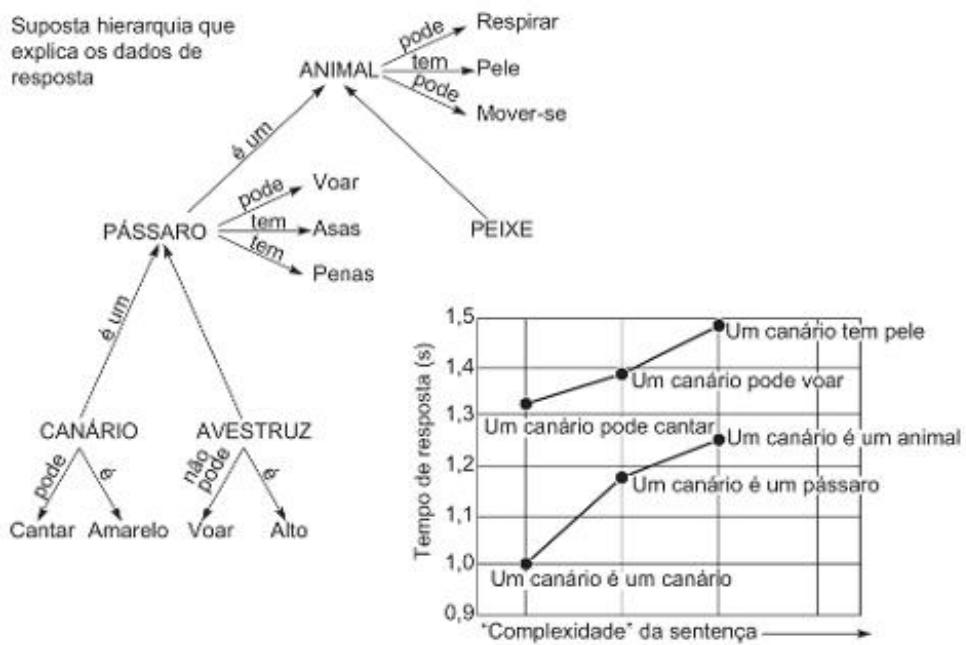
Essas duas expressões têm o mesmo valor verdade; isto é, a segunda é verdadeira se, e somente se, a primeira for verdadeira. Entretanto, a equivalência do valor verdade é inapropriada nessa situação. Se considerássemos a evidência física da verdade dessas declarações, o fato de essa folha de papel não ser vermelha, e também de não ser um cardeal, seria uma evidência para a verdade da segunda expressão. Como as duas expressões são logicamente equivalentes, resulta que também são uma evidência para a verdade da primeira declaração. Isso nos leva à conclusão que a brancura da folha de papel é uma evidência de que cardais são vermelhos.

Essa linha de raciocínio soa sem sentido e um tanto boba para nós. A razão para essa incongruência é que a implicação lógica expressa apenas uma relação entre os valores verdade de seus operandos, enquanto a sentença em português implica uma correlação positiva entre a pertinência a uma classe e a posse de propriedades dessa classe. De fato, a disposição genética de um pássaro faz com que ele tenha uma certa cor. Essa relação é perdida na segunda versão da expressão. Embora o fato de o papel não ser vermelho ser consistente com a verdade em ambas as sentenças, ele é irrelevante para a natureza causal da cor dos pássaros.

As teorias *associacionistas*, seguindo a tradição empirista na filosofia, definem o significado de um objeto em termos de uma rede de associações com outros objetos. Para os associacionistas, quando os seres humanos percebem um objeto, essa percepção é mapeada primeiro em um conceito. Esse conceito é parte do nosso conhecimento completo do mundo e está conectado através de relacionamentos apropriados a outros conceitos. Esses relacionamentos formam uma compreensão das propriedades e do comportamento de objetos, tais como a neve. Por exemplo, a partir da experiência, associamos o conceito neve a outros conceitos, tais como frio, branco, boneco de neve, escorregadio e gelo. A nossa compreensão de neve e a verdade de declarações como “a neve é branca” e “o boneco de neve é branco” se manifestam diretamente a partir dessa rede de associações.

Há evidências psicológicas de que, além da sua habilidade para associar conceitos, os seres humanos também organizam hierarquicamente o seu conhecimento, de forma que a informação seja mantida nos níveis apropriados mais altos da taxonomia. Collins e Quillian (1969) modelaram o armazenamento e o gerenciamento da informação pelo homem usando uma rede semântica (Figura 7.1). A estrutura dessa hierarquia foi derivada a partir de testes de laboratório com seres humanos. Perguntava-se às pessoas sobre as diferentes propriedades dos pássaros, tais como “Um canário é um pássaro?”, “Um canário pode cantar?” ou “Um canário pode voar?”

**Figura 7.1** Rede semântica desenvolvida por Collins e Quillian em sua pesquisa sobre o armazenamento de informação pelo homem e seu tempo de resposta (Harmon e King, 1985).



Tão óbvias quanto possam parecer essas questões, estudos sobre o tempo de reação indicaram que demorava mais para que as pessoas respondessem a “Um canário pode voar?” do que a “Um canário pode cantar?”. Collins e Quillian explicaram essas diferenças em tempo de resposta argumentando que as pessoas armazenam informação no seu nível mais abstrato. Em vez de tentar recordar que canários voam, que pardais voam e que andorinhas voam, tudo armazenado com o pássaro específico, o homem recorda que canários são pássaros e que pássaros têm (normalmente) a propriedade de voar. Propriedades ainda mais gerais, como alimentar-se, respirar e movimentar-se, são armazenadas no nível do “animal” e, assim, tentar recordar se um canário pode respirar deve levar mais tempo do que recordar se um canário pode voar. Claro que isso ocorre porque o homem tem que subir mais na hierarquia das estruturas da memória para encontrar a resposta.

A recordação mais rápida se dava no caso dos traços específicos do pássaro, ou seja, que ele sabia cantar ou que era amarelo. O tratamento de exceções parece ocorrer também no nível mais específico. Quando se perguntava às pessoas se um avestruz poderia voar, a resposta era produzida mais rapidamente do que quando se perguntava se um avestruz poderia respirar. Assim, a hierarquia avestruz → pássaro → animal parece não ser percorrida para se chegar à informação sobre a exceção: ela está armazenada diretamente no nível do avestruz. Essa organização do conhecimento foi formalizada em sistemas de herança.

Sistemas baseados em herança nos permitem armazenar informação nos níveis mais altos de abstrações, o que reduz o tamanho das bases de conhecimento e ajuda a prevenir inconsistências na sua atualização. Por exemplo, se estivermos construindo uma base de conhecimento sobre pássaros, podemos definir os traços comuns a todos os pássaros, tais como poder voar e ter penas, para a classe geral pássaro e permitir que uma espécie em particular herde essas propriedades. Isso reduz o tamanho da base de conhecimento, pois é necessário que definamos esses traços apenas uma vez, em vez de requerer a sua declaração para cada indivíduo. A herança também nos ajuda a manter a consistência da base de conhecimento quando acrescentamos novas classes e indivíduos. Suponha que acrescentemos a espécie sabiá a uma base de conhecimento. Quando declararmos que sabiá é uma subclasse de pássaro\_canoro, sabiá herda todas as propriedades comuns tanto de pássaro\_canoro como de pássaro. Não cabe ao programador se lembrar (ou se esquecer!) de acrescentar essa informação.

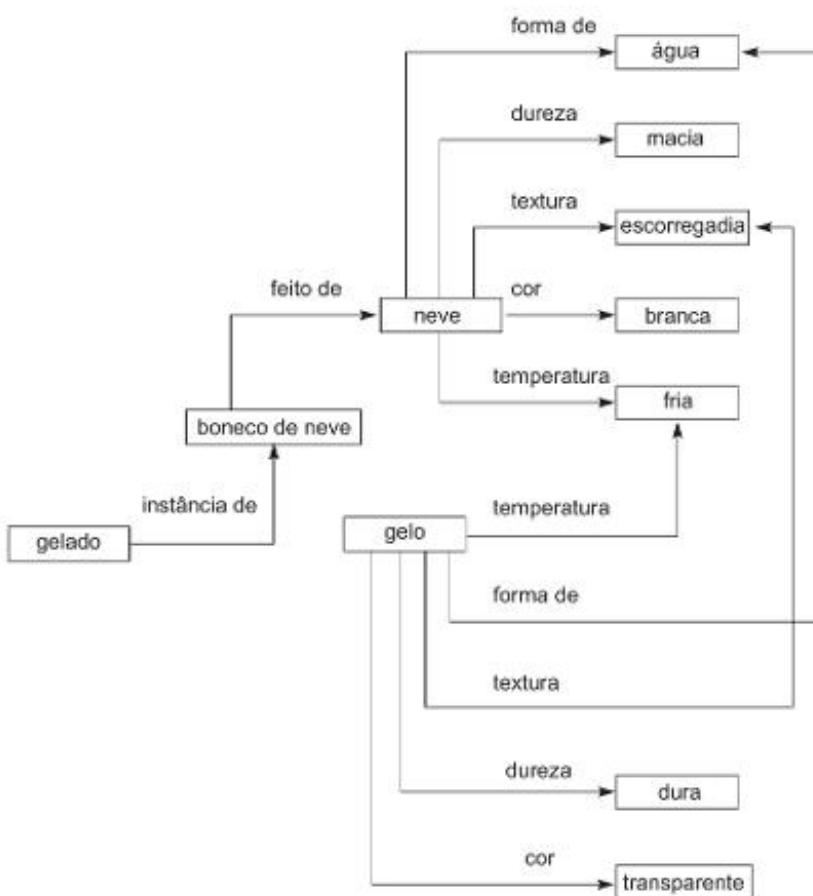
Os grafos, por proverem um meio de representar explicitamente relações usando arcos e nós, têm se mostrado um veículo ideal para formalizar as teorias associacionistas do conhecimento. Uma *rede semântica* representa o conhecimento como um grafo, com os nós que correspondem a fatos ou conceitos e os arcos como relações ou associações entre conceitos. Tanto os nós como os arcos são normalmente rotulados. Por exemplo, a Figura 7.2 mostra uma rede semântica que define as propriedades de neve e gelo. Essa rede poderia ser usada (com regras de inferência apropriadas) para responder uma série de questões sobre neve, gelo e boneco de neve. Chega-se a essas conclusões seguindo-se os arcos para os conceitos relacionados. As redes semânticas também implementam herança; por exemplo, gelado herda todas as propriedades de boneco\_de\_neve.

O termo “rede semântica” abrange uma família de representações baseadas em grafos. Essas representações diferem, sobretudo, nos nomes que podem ser usados para os nós e arcos e nas inferências que podem vir dessas estruturas. Entretanto, todas as linguagens de representação por redes compartilham um conjunto comum de suposições e interesses; isso é ilustrado a partir de uma discussão da história das representações por redes. Na Seção 7.2, examinamos os *grafos conceituais* (Sowa, 1984), uma linguagem de representação em redes mais moderna, que integra muitas dessas ideias.

### 7.1.2 Trabalhos iniciais em redes semânticas

As representações por redes têm uma história quase tão longa quanto a lógica. O filósofo grego Porfírio criou hierarquias de tipos baseadas em árvore — com suas raízes no topo — para descrever as categorias de Aristóteles (Porfírio, 1887). Frege desenvolveu uma notação de árvore para expressões lógicas. Talvez o trabalho mais antigo a ter uma influência direta sobre as redes semânticas contemporâneas foi o sistema de grafos existenciais de Char-

**Figura 7.2** Representação em rede das propriedades de neve e gelo.



les S. Peirce, desenvolvido no século XIX (Roberts, 1973). A teoria de Peirce tinha todo o poder expressivo do cálculo de predicados de primeira ordem, com uma base axiomática e regras de inferência formais.

Em psicologia, os grafos têm sido usados há muito tempo para representar estruturas de conceitos e associações. Selz (1913, 1922) foi o pioneiro nesse trabalho, usando grafos para representar hierarquias de conceitos e a herança de propriedades. Ele desenvolveu, também, uma teoria de antecipação esquemática que influenciou o trabalho em IA sobre quadros e esquemas. Anderson, Norman, Rumelhart e outros usaram redes para modelar a memória e o desempenho intelectual dos seres humanos (Anderson e Bower, 1973; Norman et al., 1975).

Muito da pesquisa em representações em redes tem sido feito na área de compreensão de linguagem natural. Em geral, a compreensão da linguagem natural requer a compreensão do senso comum, os modos como os objetos físicos se comportam, as interações que ocorrem entre os seres humanos e os modos como as instituições humanas estão organizadas. Um programa de linguagem natural precisa compreender intenções, crenças, raciocínio hipotético, planos e objetivos. Devido a esses requisitos, a compreensão de linguagem natural tem sido sempre uma força propulsora para as pesquisas em representação de conhecimento.

As primeiras implementações de redes semânticas em computador foram desenvolvidas no início dos anos 1960 para uso em tradução de máquina. Masterman (1961) definiu um conjunto de 100 tipos de conceitos primitivos e os usou para definir um dicionário de 15.000 conceitos. Wilks (1972) deu continuidade ao desenvolvimento do trabalho de Masterman em sistemas de linguagem natural baseados em redes semânticas. O programa MIND de Shapiro (1971) foi a primeira implementação de uma rede semântica baseada em cálculo proposicional. Entre os primeiros pesquisadores em IA que exploraram as representações em redes estão, entre outros, Ceccato (1961), Raphael (1968), Reitman (1965) e Simmons (1966).

Um programa muito influente, que ilustra muitas das características das redes semânticas iniciais, foi escrito por Quillian no final dos anos 1960 (Quillian, 1967). Esse programa definia as palavras em inglês de forma semelhante aos dicionários: uma palavra é definida em termos de outras palavras e os componentes da definição são definidos da mesma forma. Em vez de definir as palavras formalmente em termos de axiomas básicos, cada definição simplesmente conduz a outras definições em uma forma desestruturada e, possivelmente, circular. Ao procurar por uma palavra, percorremos essa “rede” até que estejamos satisfeitos com o que compreendemos da palavra original.

Cada nó na rede de Quillian corresponde a um *conceito de palavra*, com elos associados a outros conceitos de palavras que formaram a sua definição. A base de conhecimento era organizada em *planos*, onde cada plano era um grafo que definia uma única palavra. A Figura 7.3, tirada (e traduzida para o português) de um artigo de Quillian (1967), ilustra três planos que capturam três definições diferentes da palavra *planta*: um organismo vivo (planta 1), um lugar onde pessoas trabalham (planta 2) e o ato de colocar uma semente no solo (planta 3).

O programa usou essa base de conhecimento para encontrar relacionamentos entre pares de palavras. Dadas duas palavras, ele realiza uma busca em amplitude nos grafos que se originam de cada palavra, buscando um conceito comum ou um *nó de interseção*. Os caminhos para esse nó representam um relacionamento entre os conceitos de palavra. Por exemplo, a Figura 7.4, do mesmo artigo, mostra os *caminhos de interseção* entre chorar e confortar.

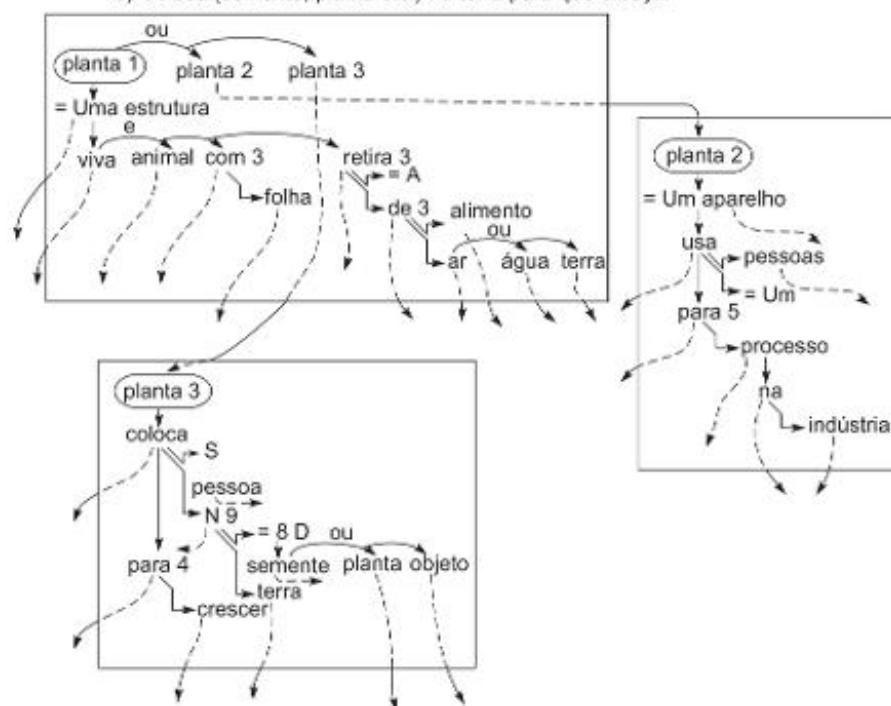
Usando esse caminho de interseção, o programa era capaz de concluir:

chorar 2 é, entre outras coisas, fazer um som triste. Confortar 3 pode ser tornar 2 um pouco menos triste (Quillian, 1967).

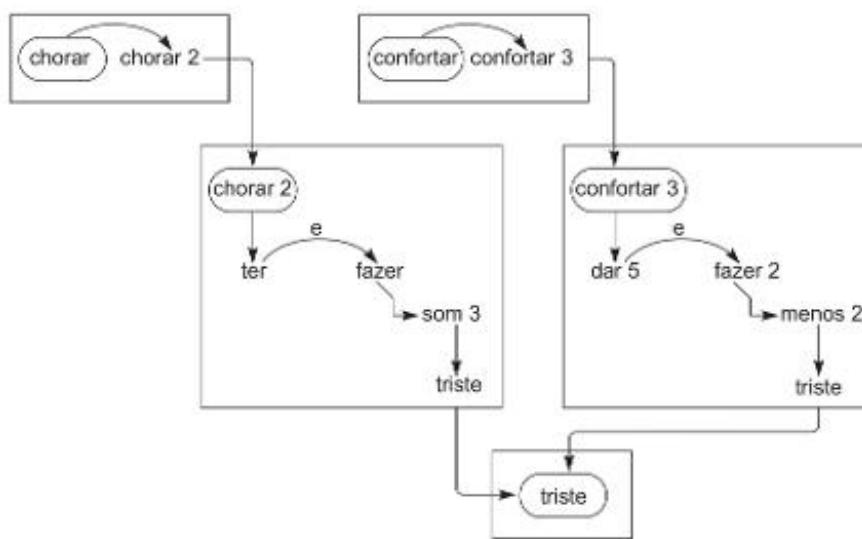
Os números na resposta indicam que o programa realizou uma seleção entre diferentes significados das palavras.

**Figura 7.3** Três planos representando três definições da palavra “*planta*” (Quillian, 1967).

- Planta:*
- 1) Estrutura viva que não é animal, frequentemente com folhas,  
retira seu alimento do ar, da água ou da terra.
  - 2) Aparelho usado em qualquer processo industrial.
  - 3) Coloca (semente, planta etc.) na terra para que cresça.



**Figura 7.4** Interseção entre “chorar” e “confortar” (adaptado de Quillian, 1967).



Quillian (1967) sugeriu que essa abordagem para semântica poderia dotar um sistema de compreensão da linguagem natural com as capacidades de:

1. Determinar o significado de um corpo de texto por meio da construção de coleções desses nós de interseção.
2. Escolher entre múltiplos significados de palavras procurando o significado com o menor caminho de interseção para outras palavras da sentença. Por exemplo, ele poderia selecionar um significado para “planta” em “Tom foi para casa regar sua nova planta” com base na interseção dos conceitos das palavras “regar” e “planta”.
3. Responder a uma gama flexível de consultas com base em associações entre os conceitos das palavras das consultas e os conceitos do sistema.

Embora esse e outros trabalhos iniciais demonstrassem o poder dos grafos na modelagem do significado associativo, eles estavam limitados pela extrema generalidade do formalismo. O conhecimento é, geralmente, estruturado em termos de relacionamentos específicos, tais como objeto/propriedade, classe/subclasse e agente/verbo/objeto.

### 7.1.3 Padronização dos relacionamentos em redes

Por si mesma, uma notação de relacionamento por grafo leva pouca vantagem sobre a lógica; ela é apenas mais uma notação para relacionamentos entre objetos. De fato, o SNePS, do inglês *Semantic Net Processing System* (Shapiro, 1979; Shapiro et al., 2006), foi um provedor de teoremas no cálculo de predicados de primeira ordem. O poder de representações por rede advém da definição de elos e regras de inferência associadas, tal como a herança.

Embora o trabalho inicial de Quillian tenha estabelecido a maioria das características significativas do formalismo das redes semânticas, como os arcos e os elos rotulados, a herança hierárquica e as inferências ao longo de elos associativos, ele se mostrou limitado em sua habilidade de tratar as complexidades de muitos domínios. Uma das principais razões para essa deficiência se encontra na pobreza dos relacionamentos (elos) que captavam os aspectos semânticos mais profundos do conhecimento. A maioria dos elos representava associações extremamente genéricas entre os nós e não fornecia uma base real para a estruturação dos relacionamentos semânticos. O mesmo problema é encontrado nos esforços para usar o cálculo de predicados para captar significado semântico. Embora o formalismo

seja altamente expressivo e possa representar quase qualquer tipo de conhecimento, ele é irrestrito demais e transfere ao programador o fardo correspondente à construção de conjuntos apropriados de fatos e regras.

Muito do trabalho em representações por redes que se seguiu ao de Quillian se concentrou na definição de um conjunto mais rico de rótulos dos elos (relacionamentos) que pudesse modelar mais completamente a semântica da linguagem natural. Ao implementar os relacionamentos semânticos fundamentais da linguagem natural como parte do *formalismo*, em vez de ser parte do *conhecimento do domínio* acrescentado pelo construtor do sistema, as bases de conhecimento deixam de ser tão artesanais e alcançam maior generalidade e consistência.

Brachman (1979) afirmou:

A questão primordial aqui é isolar as *primitivas* para as linguagens das redes semânticas. As primitivas de uma linguagem de rede são aquilo que o interpretador está programado para compreender e que não são normalmente representadas na própria linguagem de rede.

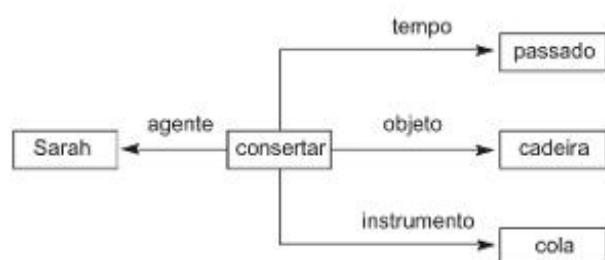
Simmons (1973) abordou essa necessidade para relacionamentos-padrão focando na *estrutura de casos* dos verbos em inglês. Nessa abordagem orientada a verbos, baseada no trabalho anterior de Fillmore (1968), os elos definem os papéis assumidos por nomes e frases nominais na ação da sentença. Entre os relacionamentos de casos estão *agente*, *objeto*, *instrumento*, *localização* e *tempo*. Uma sentença é representada como um nó verbal, com vários elos de caso ligando nós que representam outros participantes da ação. Essa estrutura é chamada de *quadro de caso*. Ao analisar uma sentença, o programa encontra o verbo e recupera o quadro de caso para este verbo da sua base de conhecimento. Ele então liga os valores do agente, objeto etc. aos nós apropriados no quadro de caso. Usando essa abordagem, a sentença “Sarah consertou a cadeira com cola” poderia ser representada pela rede da Figura 7.5.

Assim, a própria linguagem de representação capta muito da estrutura mais profunda da linguagem natural, tal como a relação entre um verbo e o seu sujeito (a relação agente) ou aquela entre um verbo e seu objeto. O conhecimento da estrutura de caso da língua portuguesa é parte do próprio formalismo da rede. Quando a sentença individual é analisada, esses relacionamentos embutidos indicam que Sarah é a pessoa que executa o conserto e que é usada cola para montar a cadeira. Note que *esses relacionamentos linguísticos estão armazenados de uma forma que é independente da sentença real ou mesmo da linguagem na qual a sentença é expressa*. Uma abordagem similar é adotada também em linguagens de rede propostas por Norman (1972) e Rumelhart et al. (1972, 1973).

Vários esforços importantes foram realizados para tentar padronizar ainda mais os nomes de elos (Masterman, 1961; Wilks, 1972; Schank e Colby, 1973; Schank e Nash-Webber, 1975). Cada um desses esforços visou estabelecer um conjunto completo de primitivas que pudesse ser usado para representar, de uma maneira uniforme, a estrutura semântica de expressões em linguagem natural. Esses conjuntos visavam auxiliar no raciocínio com esquemas de linguagem, sendo independentes das idiossincrasias das linguagens individuais ou do estilo linguístico.

Talvez a tentativa mais ambiciosa para modelar formalmente a estrutura semântica profunda da linguagem natural seja a teoria da *dependência conceitual* de Roger Schank (Schank e Rieger, 1974). A teoria da dependência conceitual fornece um conjunto de quatro conceitualizações primitivas das quais o mundo dos significados é construído. Esses conceitos são iguais e independentes. São eles:

**Figura 7.5** Representação por quadro de caso da sentença “Sarah consertou a cadeira com cola”.



ATOs	ações
PFs	objetos (produtores de figuras)
AAs	modificadores de ações (auxiliares de ações)
AFs	modificadores de objetos (auxiliares de figuras)

Por exemplo, supõe-se que todas as ações se reduzam a um ou mais ATOs primitivos. Essas primitivas, listadas a seguir, são tomadas como os componentes básicos das ações, com verbos mais específicos sendo formados a partir da sua modificação e combinação.

ATRANS	transferir um relacionamento (dar)
FTRANS	transferir a localização física de um objeto (ir)
PROPUL	aplicar força física a um objeto (empurrar)
MOVER	mover parte do corpo do sujeito (chutar)
PEGAR	pegar um objeto, por um agente (agarrar)
INGERIR	ingerir um objeto, por um animal (comer)
EXPELIR	expelir do corpo de um animal (gritar)
MTRANS	transferir informação mental (contar)
MCONST	construir mentalmente informação nova (decidir)
CONC	conceitualizar ou refletir sobre uma ideia (pensar)
FALAR	produzir som (dizer)
ATENTAR	focar órgão sensorial (escutar)

Essas primitivas são usadas para definir os *relacionamentos de dependência conceitual* que descrevem as estruturas de significado como relações de caso ou a associação de objetos e valores. Relacionamentos de dependência conceitual são *regras de sintaxe conceituais* e constituem uma gramática de relacionamentos semânticos significativos. Esses relacionamentos podem ser usados para construir uma representação interna de uma sentença em inglês. Uma lista de dependências conceituais básicas (Schank e Rieger, 1974) aparece na Figura 7.6. Elas

**Figura 7.6** Dependências conceituais (Schank e Rieger, 1974).

PF ↔ ATO	indica que um agente age.
PF ↔ AF	indica que um objeto tem um certo atributo.
ATO $\overset{O}{\leftarrow}$ PF	indica o objeto de uma ação.
ATO $\overset{R}{\leftarrow}$ PF ATO $\overset{D}{\leftarrow}$ PF	indica o receptor e o doador de um objeto dentro de uma ação.
ATO $\overset{1}{\leftarrow} \emptyset$	indica a direção de um objeto dentro de uma ação.
X ↑ Y	indica que a conceitualização X causou a conceitualização Y. Quando escrito com P denota que X PODERIA causar Y.
PF $\leftarrow$ AF2 PF $\leftarrow$ AF1	indica uma mudança de estado de um objeto.
PF1 $\leftarrow$ PF2	indica que PF2 é PARTE DE ou o POSSUIDOR DE PF1.

captam, segundo seus criadores, as estruturas semânticas fundamentais da linguagem natural. Por exemplo, a primeira dependência conceitual da Figura 7.6 descreve a relação entre um sujeito e seu verbo, e a terceira descreve a relação verbo-objeto. Elas podem ser combinadas para representar uma sentença transitiva simples como “John jogou a bola” (Figura 7.7).

Por fim, pode ser adicionada ao conjunto de conceitualizações a informação de tempo e de modo verbais. Schank fornece uma lista de anexos ou modificadores das relações. Uma lista parcial é:

p	passado
f	futuro
t	transição
k	continuação
t <sub>i</sub>	transição inicial
?	interrogativo
t <sub>f</sub>	transição final
c	condicional
/	negativo
nil	presente
delta?	atemporal

Essas relações são as construções de primeiro nível da teoria, os relacionamentos semânticos mais simples, a partir dos quais estruturas mais complexas podem ser construídas. Outros exemplos de como essas dependências conceituais básicas podem ser compostas para representar o significado de sentenças simples aparecem na Figura 7.8.

Com base nessas primitivas, a sentença “John comeu um ovo” é representada como mostra a Figura 7.9, onde os símbolos têm os seguintes significados:

←	indica a direção da dependência
↔	indica a relação agente-verbo
p	indica tempo verbal passado
INGERIR	é um ato primitivo da teoria
O	relação objeto
D	indica a direção do objeto na ação

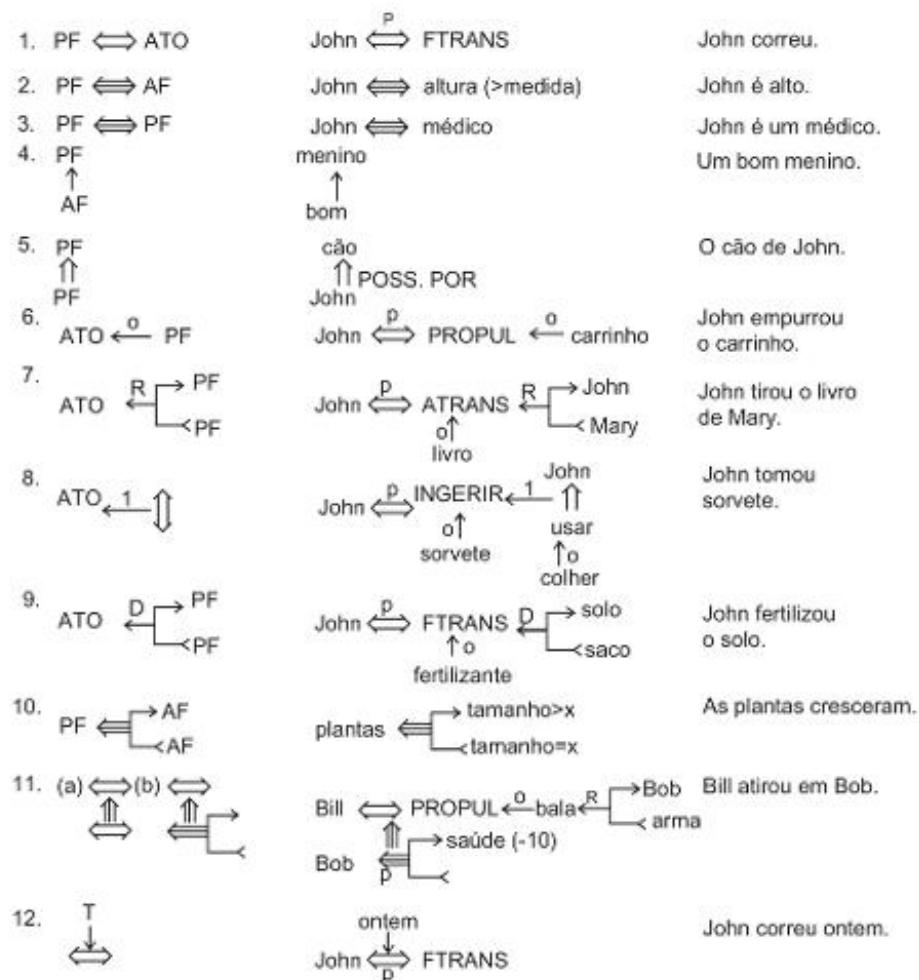
Outro exemplo de estruturas que podem ser construídas usando dependências conceituais é o grafo que representa a sentença “John impediu Mary de dar um livro a Bill” (Figura 7.10). Esse exemplo em particular é interessante porque demonstra como a causalidade pode ser representada.

A teoria da dependência conceitual oferece uma série de benefícios importantes. Por fornecer uma teoria formal para a semântica da linguagem natural, ela reduz problemas de ambiguidade. Em segundo lugar, a representação em si captura diretamente muito da semântica da linguagem natural, por procurar fornecer uma *forma canônica* para o significado de sentenças. Isto é, todas as sentenças que têm o mesmo significado serão representadas internamente por grafos *sintaticamente idênticos*, não apenas equivalentes do ponto de vista semântico. Essa representação canônica é um esforço para simplificar as inferências necessárias para a compreensão. Por exemplo, podemos de-

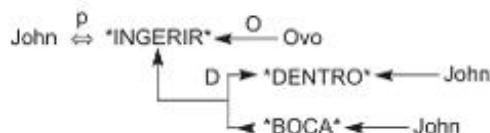
**Figura 7.7** Representação por dependência conceitual da sentença “John jogou a bola”.



**Figura 7.8** Algumas dependências conceituais básicas e seu uso na representação de sentenças mais complexas; adaptado de Schank e Colby (1973).



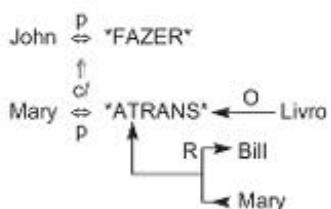
**Figura 7.9** Dependência conceitual representando “John comeu o ovo” (adaptado de Schank e Rieger, 1974).



monstrar que duas sentenças significam o mesmo por meio de um simples casamento de grafos de dependência conceitual; uma representação que não fornece uma forma canônica pode necessitar de operações extensivas sobre grafos diferentemente estruturados.

Infelizmente, é questionável a possibilidade de se escrever um programa para reduzir, de forma confiável, sentenças à forma canônica. Como foi salientado por Woods (1985) e outros, a redução à forma canônica é provavelmente incomputável para monoides, um tipo de grupo algébrico que é bem mais simples que a linguagem natural. Além disso, não há evidência de que o homem armazene o seu conhecimento em algum tipo de forma canônica.

**Figura 7.10** Representação da dependência conceitual da sentença “John impediu Mary de dar um livro a Bill” (adaptado de Schank e Rieger, 1974).



Outras críticas a esse ponto de vista, além de questionar o preço computacional pago ao reduzir tudo a tais primitivas de baixo nível, sugerem que as próprias primitivas não são adequadas para capturar vários dos conceitos mais sutis que são importantes em linguagem natural. Por exemplo, a representação de “alto” na segunda sentença da Figura 7.8 não aborda a ambiguidade desse termo de forma tão cuidadosa como é feito em sistemas como o da lógica difusa (Zadeh, 1983, e Seção 9.2.2).

Entretanto, ninguém pode dizer que o modelo de dependência conceitual não tenha sido estudado extensivamente e nem bem compreendido. Mais de uma década de pesquisas guiadas por Schank se concentraram em refinar e estender o modelo. Extensões importantes das dependências conceituais incluem pesquisas em *roteiros* e em *pacotes de organização de memória*, ou MOPs (do inglês *Memory Organization Packets*). A pesquisa em roteiros examina a organização do conhecimento na memória e o papel que essa organização desempenha no raciocínio (Seção 7.1.4). Os MOPs representam uma das áreas de pesquisa que dão suporte ao projeto de sistemas de raciocínio baseado em casos (Seção 8.3). A teoria da dependência conceitual é um modelo totalmente desenvolvido da semântica de linguagem natural com consistência de propósitos e ampla aplicabilidade.

#### 7.1.4 Roteiros

Um programa para a compreensão de linguagem natural precisa usar uma grande quantidade de conhecimento prévio até mesmo para compreender uma simples conversação (Seção 15.0). Há evidências de que o ser humano organiza esse conhecimento em estruturas correspondentes a situações típicas (Bartlett, 1932). Se estivermos lendo uma história sobre restaurantes, futebol ou política, resolvemos qualquer ambiguidade do texto de uma forma consistente com restaurantes, futebol ou política. Se o tema de uma história muda de forma brusca, há evidências de que as pessoas interrompem brevemente a leitura, presumivelmente para trocar as estruturas de conhecimento. É difícil entender uma história organizada ou estruturada de maneira pobre, possivelmente porque não podemos adaptá-la com facilidade a uma de nossas estruturas de conhecimento preexistentes. Podem ocorrer, também, erros de compreensão quando o assunto de uma conversação muda de forma brusca, presumivelmente porque nos confundimos sobre qual contexto usar para resolver referências pronominais e outras ambiguidades na conversação.

Um *roteiro* é uma representação estruturada que descreve uma sequência estereotipada de eventos em um contexto particular. O modelo de roteiro foi originalmente concebido por Schank e seu grupo de pesquisa (Schank e Abelson, 1977) como um meio de organizar estruturas de *dependência conceitual* formando descrições de situações típicas. Os roteiros são usados em sistemas de compreensão de linguagem natural para organizar uma base de conhecimento em termos das situações que o sistema deve compreender.

A maioria dos adultos se sente confortável (isto é, eles sabem o que os espera e como agir) em um restaurante. Eles são recepcionados na entrada ou recebem alguma indicação para que sigam em frente até encontrarem uma mesa livre. Se o cardápio não estiver sobre a mesa, ou então se não for entregue pelo garçom, o cliente deverá solicitar um. De modo semelhante, conhecem as rotinas para fazer o pedido de pratos, bem como para comer, pagar e sair.

Na verdade, o roteiro para o restaurante é bem diferente de outros roteiros para refeições, como o modelo para *fast-food*, ou para uma “refeição familiar formal”. No modelo de *fast-food*, o cliente entra, pega uma fila para fazer o pedido, paga pela refeição (antes de comê-la), espera por uma bandeja com a comida, pega a bandeja e tenta encontrar uma mesa livre, e assim por diante. Essas são duas sequências estereotipadas diferentes de eventos e cada uma tem um roteiro potencial.

Os componentes de um roteiro são:

*Condições de entrada* ou descriptores do mundo que devem ser verdadeiros para que o roteiro seja chamado. No nosso exemplo de roteiro, fazem parte um restaurante aberto e um cliente com fome que tenha algum dinheiro.

*Resultados* ou fatos que sejam verdadeiros quando o roteiro tiver terminado; por exemplo, o cliente está satisfeito e mais pobre, o dono do restaurante tem mais dinheiro.

*Acessórios* ou “coisas” que suportam o conteúdo do roteiro. No exemplo, poderiam ser mesas, garçons e cardápios. O conjunto de acessórios permite que se faça pressuposições razoáveis sobre a situação: supõe-se que um restaurante tenha mesas e cadeiras, a menos que se declare o contrário.

*Papéis* são as ações que os participantes individuais desempenham. O garçom recebe pedidos, entrega a comida e apresenta a conta. O cliente faz o pedido, come e paga.

*Cenas*. Schank divide o roteiro em uma sequência de cenas em que cada uma delas apresenta um aspecto temporal do roteiro. No restaurante há as cenas correspondentes a entrar, pedir, comer etc.

Os elementos do roteiro, as “peças” básicas de significado semântico, são representados usando relações de dependência conceitual. Dispostos em uma estrutura análoga a um quadro, eles representam uma sequência de significados, ou uma sequência de eventos. O roteiro do restaurante retirado dessa pesquisa é apresentado na Figura 7.11.

O programa lê uma pequena história sobre restaurantes e a analisa segundo uma representação interna de dependência conceitual. Como os conceitos-chave nessa descrição interna casam com as condições de entrada do roteiro, o programa liga as pessoas e as coisas mencionadas na história aos papéis e acessórios mencionados no roteiro. O resultado é uma representação expandida do conteúdo da história, usando o roteiro para completar qualquer informação faltante e para fazer pressuposições. O programa, então, responde questões sobre a história referenciando o roteiro. O roteiro permite que se faça pressuposições razoáveis que são essenciais à compreensão da linguagem natural. Por exemplo:

#### Exemplo 7.1.1

*John foi a um restaurante ontem à noite. Ele pediu um bife. Quando pagou a conta, percebeu que não tinha mais dinheiro. Ele correu para casa, pois tinha começado a chover.*

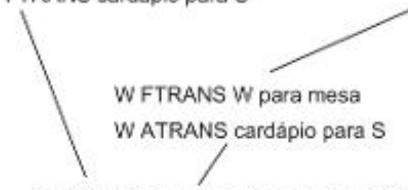
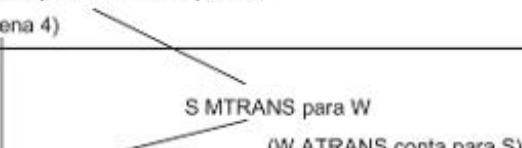
Usando o roteiro, o sistema pode responder corretamente questões como: John jantou ontem à noite (a história não explicita isso)? John pagou com dinheiro ou com cartão de crédito? Como John poderia ter acesso a um cardápio? O que John comprou?

#### Exemplo 7.1.2

*Sue saiu para almoçar. Ela se sentou à mesa e chamou a garçonete, que lhe trouxe o cardápio. Ela pediu um sanduíche.*

Entre as questões que poderiam ser razoáveis para essa história estão as seguintes: Por que a garçonete trouxe um cardápio para Sue? Sue estava em um restaurante (o exemplo não esclarece esse ponto)? Quem pagou a conta? Quem é “ela” que pediu um sanduíche? A última questão é difícil. A garçonete foi a última pessoa do sexo feminino mencionada, o que nos levaria a uma conclusão incorreta. Os *papéis* do roteiro ajudam a resolver referências pronominais e outras ambiguidades.

Figura 7.11 Um roteiro para restaurante (Schank e Abelson, 1977).

<p>Roteiro: RESTAURANTE          Trilha: Cafeteria          Acessórios: Mesas              Cardápio              F = Comida              Conta              Dinheiro</p>	<p>Cena 1: Entrando          S FTRANS S no restaurante          S ATENTAR olhos nas mesas          S MCONST onde sentar          S FTRANS para a mesa          S MOVER S para posição sentada</p>
<p>Papéis: S = Cliente              W = Garçom              C = Cozinheiro              M = Caixa              P = Proprietário</p>	<p>Cena 2: Pedindo          (Cardápio sobre a mesa) (W traz o cardápio)          S FTRANS cardápio para S            W FTRANS W para mesa          W ATRANS cardápio para S            S MTRANS lista de pratos para PC (S)          * S MCONST escolha de F          S MTRANS faz sinal para W          W FTRANS W para mesa          S MTRANS 'Eu quero F' para W            W FTRANS W para C          W MTRANS (ATRANS F) para C            C MTRANS 'não tem F' para W          W FTRANS W para S          W MTRANS 'não tem F' para S          (voltar para *) ou          (ir para a Cena 4, para o caminho não pagar)</p>
<p>Condições de entrada: S está com fome.          S tem dinheiro.</p> <p>Resultados: S tem menos dinheiro          P tem mais dinheiro          S não está com fome          S está satisfeita (opcional)</p>	<p>Cena 3: Comendo          C ATRANS F para W          W ATRANS F para S          S INGERIR F            (Opção: retorne para Cena 2 para fazer novo pedido;          caso contrário, vá para Cena 4)</p> <p>Cena 4: Saindo            S MTRANS para W          (W ATRANS conta para S)            W MOVER (escreve a conta)          W FTRANS W para S          W ATRANS conta para S          S ATRANS gorjeta para W          S FTRANS S para M          S ATRANS dinheiro para M          S FTRANS S para saída do restaurante            (Caminho não pagar)</p>

Roteiros podem, também, ser usados para interpretar resultados imprevistos ou interrupções na atividade representada. Assim, na Cena 2 da Figura 7.11, há o ponto de escolha entre “tem comida” ou “não tem comida” para ser entregue ao cliente. Isso permite que se compreenda o exemplo a seguir.

#### Exemplo 7.1.3

*Kate foi a um restaurante. Ela foi conduzida a uma mesa e pediu um sushi para a garçonete. Ela se sentou e esperou por um longo tempo. Ao final, ela ficou aborrecida e saiu.*

Para essa história poderiam ser formuladas as seguintes questões, usando o roteiro do restaurante: Quem é “ela” que se sentou e esperou? Por que ela esperou? Quem é “ela” que ficou aborrecida e saiu? Por que ela ficou aborrecida? Note que há outras questões que os roteiros não podem responder, como por que as pessoas ficam aborrecidas quando o garçom demora a trazer a comida. Como qualquer sistema baseado em conhecimento, os roteiros requerem que o engenheiro de conhecimento antecipe corretamente o conhecimento necessário.

Os roteiros, assim como os quadros e outras representações estruturadas, estão sujeitos a certos problemas, incluindo o problema do *casamento* de roteiros e o problema das *entrelinhas*. Considere o Exemplo 7.1.4, que poderia executar tanto o roteiro para o *restaurante* quanto para o *concerto*. A escolha é crítica porque “lugar” pode se referir tanto a uma mesa no restaurante como a uma poltrona no teatro.

#### Exemplo 7.1.4

*John visitou seu restaurante favorito a caminho do concerto. Ele estava contente por ter conseguido um lugar, pois gosta muito de Mozart.*

Como a seleção de roteiro é normalmente baseada no casamento de palavras-chave, muitas vezes é difícil determinar qual roteiro deve ser usado, entre duas ou mais possibilidades. O problema do casamento de roteiro é “profundo”, pois não existe um algoritmo que garanta a escolha correta. Nesse caso, é necessário conhecimento heurístico sobre a organização do mundo, e talvez até algum retrocesso, mas os roteiros auxiliam apenas na organização daquele conhecimento específico.

O problema das *entrelinhas* é igualmente difícil: não é possível saber de antemão as possíveis ocorrências que podem interromper um roteiro. Por exemplo:

#### Exemplo 7.1.5

*Melissa jantava em seu restaurante favorito quando um grande pedaço de reboco caiu do teto e atingiu o seu broto.*

Questões: Melissa estava comendo uma salada de broto de feijão? O namorado de Melissa ficou coberto de gesso? O que ela fez a seguir? Como ilustra esse exemplo, as representações estruturadas podem ser inflexíveis. O raciocínio pode ficar preso a um único roteiro, muito embora isso possa ser inapropriado.

Os pacotes de organização da memória tratam o problema da inflexibilidade dos roteiros usando representação do conhecimento como componentes menores acompanhados de regras para combiná-los dinamicamente para formar um esquema que é apropriado à situação atual (Schank, 1982). A organização do conhecimento na memória é particularmente importante para implementações de raciocínio baseado em casos, no qual o sistema deve recuperar eficientemente na memória uma solução prévia relevante para o problema (Kolodner, 1988a; Seção 8.3).

Os problemas de organizar e recuperar conhecimento são difíceis e inerentes à modelagem do significado semântico. Eugene Charniak (1972) exemplificou a quantidade de conhecimento necessária para compreender até mesmo histórias infantis simples. Considere uma declaração sobre uma festa de aniversário: “Mary recebeu duas pipas de aniversário e por isso ela devolveu uma à loja”. Precisamos conhecer a tradição de dar pre-

sentem em uma festa; precisamos saber o que é uma pipa e por que Mary não desejaria ficar com duas; precisamos ter conhecimento sobre lojas e suas políticas de troca. Apesar desses problemas, programas usando roteiros e outras representações semânticas podem compreender a linguagem natural em domínios limitados. Um exemplo disso são programas que interpretam mensagens que chegam de várias agências de notícias. Usando roteiros para desastres naturais, golpes, ou outras histórias estereotipadas, tais programas têm obtido um sucesso notável nesse domínio limitado, mas realístico (Schank e Riesbeck, 1981).

### 7.1.5 Quadros

Outro esquema representacional, similar em muitos aspectos aos roteiros, que foi concebido para capturar as conexões implícitas da informação em um domínio de problema em estruturas de dados explicitamente organizadas são os *quadros* (*frames*). Essa representação possibilita a organização do conhecimento em unidades mais complexas que refletem a organização de objetos no domínio.

Em um artigo de 1975, Minsky descreve um quadro:

Eis aqui a essência da teoria dos quadros: quando alguém encontra uma nova situação (ou modifica substancialmente o seu entendimento sobre um problema), recupera da memória uma estrutura chamada “quadro”. Essa estrutura é um arcabouço memorizado que deve ser adaptado para se adequar à realidade, alterando detalhes, conforme a necessidade (Minsky, 1975).

De acordo com Minsky, um quadro pode ser visto como uma estrutura de dados estática usada para representar situações estereotipadas bem compreendidas. Parece que estruturas similares a quadros organizam o nosso próprio conhecimento sobre o mundo. Utilizamos informação estruturada por experiências passadas para nos ajustarmos a cada nova situação. Então, moldamos ou revisamos os detalhes dessas experiências passadas para representar as diferenças individuais para a nova situação.

Qualquer um que tenha se hospedado em um ou dois hotéis não encontra problemas em lidar com hotéis totalmente diferentes e seus quartos. Esperamos encontrar uma cama, um banheiro, um lugar para abrir uma mala, um telefone, informação sobre preços e procedimentos de emergência atrás da porta, e assim por diante. Os detalhes de cada quarto podem ser fornecidos quando necessário: as cores das cortinas, localização e uso de interruptores de luz etc. O quadro de quartos de hotel fornece, também, informação que corresponde a pressuposições básicas: se não houver lençóis, chame a arrumadeira; se precisar de gelo, peça na recepção; e assim por diante. Não precisamos reconstruir o nosso entendimento para cada novo quarto de hotel que ocuparmos. Todas as peças de um quarto de hotel genérico estão organizadas em uma estrutura conceitual que acessamos quando nos registramos em um hotel; as particularidades de um quarto individual serão fornecidas quando necessário.

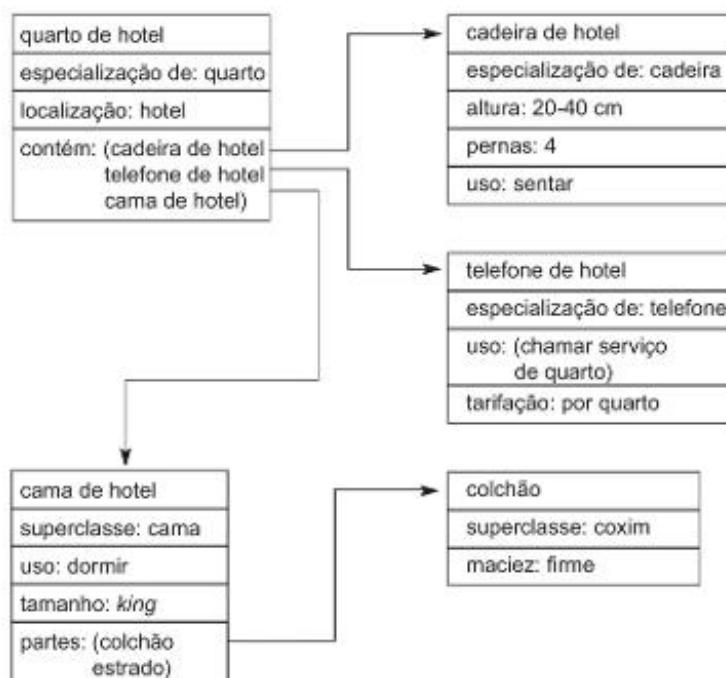
Poderíamos representar essas estruturas de alto nível diretamente em uma rede semântica, organizando-a como uma coleção de redes separadas, cada uma representando uma situação estereotipada. Os quadros, assim como os *sistemas orientados a objetos*, fornecem um veículo para essa organização, representando entidades como objetos estruturados com partições rotuladas (*slots*) e valores atribuídos. Assim, um quadro, ou esquema, é visto como uma única entidade complexa.

Por exemplo, o quarto de hotel e seus componentes podem ser descritos a partir de vários quadros individuais. Além da cama, um quadro poderia representar uma cadeira: altura esperada entre 20 e 40 cm, o número de pernas é 4, que é um valor padrão (*default*), é projetada para que uma pessoa possa se sentar. Outro quadro representa o telefone do hotel: é uma especialização de um telefone comum, exceto que a tarifa se dá por quarto, há uma telefonista do hotel (padrão) e pode-se usar o telefone do hotel para pedir que refeições sejam servidas no quarto, fazer ligações externas e receber outros serviços. A Figura 7.12 apresenta um quadro representando o quarto de hotel.

Cada quadro individual pode ser visto como uma estrutura de dados, similar em vários aspectos ao “registro” tradicional, que contém informação relevante acerca de entidades estereotipadas. As partições do quadro contêm informações como:

1. *Informação sobre a identificação do quadro.*
2. *Relação desse quadro com outros quadros.* O “telefone de hotel” poderia ser um caso especial de “telefone”, o qual poderia ser uma instância de “dispositivo de comunicação”.

**Figura 7.12** Parte de uma descrição por quadro de um quarto de hotel. “Especialização” indica um ponteiro para uma superclasse.



3. *Descritores de requisitos para um quadro.* Uma cadeira, por exemplo, tem o seu assento entre 20 e 40 cm do chão, o seu encosto tem mais que 60 cm etc. Esses requisitos podem ser usados para determinar quando novos objetos satisfazem o estereótipo definido pelo quadro.
4. *Informação procedural sobre o uso da estrutura descrita.* Uma característica importante dos quadros é a habilidade de se vincular códigos de procedimentos a uma partição.
5. *Informação padrão do quadro.* São valores de partições que são tomados como verdadeiros, enquanto não for encontrada evidência do contrário. Por exemplo, as cadeiras têm quatro pernas, telefones utilizam teclado, ou camas de hotel são arrumadas pelo serviço de camareiras.
6. *Nova informação de instância.* Muitas partições podem ser deixadas sem especificação até que seja dado um valor específico para uma instância particular, ou até que elas sejam necessárias por um certo aspecto de um processo de solução de problema. Por exemplo, a cor da colcha pode não ser especificada.

Os quadros estendem as redes semânticas de várias formas. Embora a descrição do quadro das camas de hotel da Figura 7.12 possa ser equivalente a uma descrição por rede, a versão por quadro torna muito mais claro o fato de estarmos descrevendo uma cama com seus vários atributos. Na versão por rede, há simplesmente uma coleção de nós, e dependemos mais da nossa interpretação da estrutura para vermos a cama de hotel como o objeto primordial que está sendo descrito. Essa habilidade de organizar o nosso conhecimento nessas estruturas é um atributo importante de uma base de conhecimento.

Os quadros tornam mais fácil a organização hierárquica de nosso conhecimento. Em uma rede, cada conceito é representado por nós e elos no mesmo nível de especificação. Entretanto, muito frequentemente podemos querer considerar um objeto como uma entidade única para certos propósitos e considerar detalhes de sua estrutura interna apenas para outros propósitos. Por exemplo, normalmente não nos damos conta da organização mecânica de um carro até que haja uma pane; somente então resgatamos nosso “esquema do motor do carro” e tentamos encontrar o problema.

A vinculação de procedimentos é uma característica importante dos quadros porque ela permite a conexão de trechos específicos de código a entidades apropriadas em sua representação. Por exemplo, poderíamos desejar incluir a habilidade de gerar imagens gráficas em uma base de conhecimento. Uma linguagem gráfica é mais

apropriada para isso do que uma linguagem de rede. Usamos vinculação procedural para criar demons. Um demon é um procedimento que é invocado como um efeito colateral de outra ação da base de conhecimento. Por exemplo, poderíamos desejar que o sistema realizasse verificações de tipo ou que executasse testes de consistência sempre que certo valor de partição fosse alterado.

Os sistemas de quadro suportam herança de classe. As partições e valores padrão de uma classe de quadro são herdados a partir da hierarquia de classe/subclasse e classe/membro. Por exemplo, um telefone de hotel poderia ser uma subclasse de um telefone comum, exceto que (1) todas as ligações para fora do prédio passariam pela central telefônica do hotel (para tarifação) e (2) os serviços do hotel poderiam ser discados diretamente. Os valores padrão são atribuídos a partições selecionadas para serem usados apenas se não houver outra informação disponível: suponha que os quartos de hotel tenham camas e sejam, portanto, lugares apropriados para ir se você desejar dormir; se você não souber como chamar a recepção, tente “zero”; podemos supor que o telefone seja de teclas (não havendo evidências do contrário).

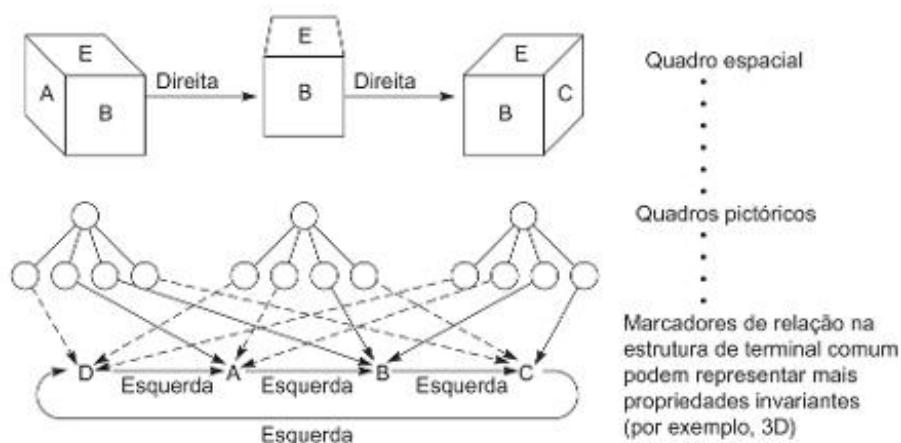
Quando uma instância da classe do quadro é criada, o sistema tenta preencher as suas partições perguntando ao usuário, aceitando o valor padrão da classe do quadro, ou executando um procedimento, ou demon, para obter o valor para a instância. Como no caso das redes semânticas, partições e valores padrão são herdados por hierarquia de classe/subclasse. Claro que a informação padrão pode fazer com que a descrição dos dados do problema seja não monotônica, o que acarreta em suposições nem sempre corretas sobre valores padrão (ver Seção 9.1).

Os próprios trabalhos de Minsky sobre a visão fornecem um exemplo de quadros e seu uso no raciocínio padrão: o problema de reconhecer que vistas diferentes de um objeto na realidade representam o mesmo objeto. Por exemplo, as três perspectivas do cubo da Figura 7.13 realmente parecem ser muito diferentes. Minsky (1975) propôs um sistema de quadro que reconhece essas vistas de um único objeto inferindo as faces ocultas como suposições padrão.

O sistema de quadro da Figura 7.13 representa quatro faces de um cubo. A linhas tracejadas indicam que uma determinada face não é visível por aquela perspectiva. As ligações entre os quadros indicam as relações entre as vistas representadas por eles. Os nós, é claro, poderiam ser mais complexos se as faces contivessem cores ou padrões. De fato, cada partição em um quadro poderia ser um apontador para um outro quadro completo. Como a informação dada pode preencher várias partições diferentes (face E na Figura 7.13), não há necessidade de redundância na informação que é armazenada.

Os quadros superam o poder das redes semânticas por permitir que objetos complexos sejam representados como um único quadro em vez de uma grande estrutura de rede. Isso fornece, também, um meio natural de representar entidades estereotipadas, classes, herança e valores padrão. Embora os quadros, como as representações por lógica e por rede, sejam uma ferramenta poderosa, muitos dos problemas de adquirir e organizar uma base de conhecimento complicada devem ser ainda resolvidos por meio da habilidade e da intuição do programador. Fi-

**Figura 7.13** Quadro espacial para visualizar um cubo (Minsky, 1975).



nalmente, essa pesquisa do MIT dos anos 1970 e o trabalho similar da Xerox Palo Alto Research Center levaram à filosofia de programação “orientada a objetos”, bem como à construção de importantes linguagens de implementação, incluindo Smalltalk, C++ e Java.

## 7.2 Grafos conceituais: uma linguagem de rede

Dando sequência ao trabalho inicial em desenvolvimento de esquemas representacionais para IA (Seção 7.1), várias linguagens de rede foram desenvolvidas para modelar a semântica de linguagem natural e outros domínios. Nesta seção, examinamos em detalhe um determinado formalismo para mostrar como foram abordados, nessa situação, os problemas de representar o significado. Os *grafos conceituais* de John Sowa (Sowa, 1984) são um exemplo de linguagem de representação por rede. Definimos as regras para formar e manipular grafos conceituais e as convenções para representar classes, indivíduos e relacionamentos. Na Seção 15.3.2, mostramos como esse formalismo pode ser usado para representar significado na compreensão de linguagem natural.

### 7.2.1 Introdução aos grafos conceituais

Um *grafo conceitual* é um grafo finito, conectado e bipartido. Os nós do grafo são *conceitos* ou, então, *relações conceituais*. Os grafos conceituais não usam arcos rotulados; em vez disso, os nós de relações conceituais representam relações entre conceitos. Como os grafos conceituais são bipartidos, os conceitos podem ter apenas arcos para relações e vice-versa. Na Figura 7.14, cão e marrom são nós conceituais, e cor é uma relação conceitual. Para distinguir esses tipos de nós, representamos conceitos como caixas e relações conceituais como elipses.

Nos grafos conceituais, os nós de conceitos representam objetos concretos ou abstratos no mundo do discurso. Conceitos concretos, como gato, telefone ou restaurante, são caracterizados por nossa habilidade de formar uma imagem deles em nossa mente. Note que os conceitos concretos incluem conceitos genéricos, como gato e restaurante, e também conceitos de gatos e restaurantes específicos. Podemos, ainda, formar uma imagem de um gato genérico. Conceitos abstratos incluem coisas como amor, beleza e lealdade, que não correspondem a imagens em nossa mente.

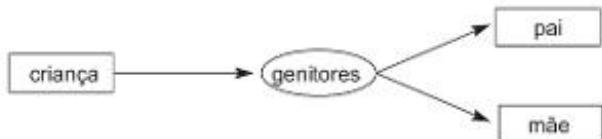
**Figura 7.14** Relações conceituais de diferentes aridades.



Voa é uma relação unária.



Cor é uma relação binária.

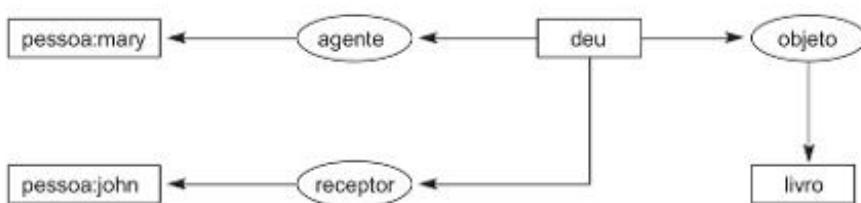


Genitores é uma relação ternária.

Os nós de relações conceituais indicam uma relação que envolve um ou mais conceitos. Uma vantagem de se formular grafos conceituais como grafos bipartidos, em vez de usar arcos rotulados, é que isso simplifica a representação de relações de qualquer aridade. Uma relação de aridade  $n$  é representada por um nó de relação conceitual que tem  $n$  arcos, como mostra a Figura 7.14.

Cada grafo conceitual representa uma única proposição. Uma base de conhecimento típica contém vários desses grafos. Os grafos podem ser arbitrariamente complexos, mas devem ser finitos. Por exemplo, um grafo na Figura 7.14 representa a proposição “um cão é da cor marrom”. A Figura 7.15 é um grafo de complexidade um pouco maior, que representa a sentença “Mary deu o livro a John”. Esse grafo usa relações conceituais para representar os casos do verbo “dar” e indica de qual maneira os grafos conceituais são usados para modelar a semântica da linguagem natural.

**Figura 7.15** Grafo de “Mary deu o livro a John”.



## 7.2.2 Tipos, indivíduos e nomes

Muitos dos primeiros projetistas de redes semânticas não foram cuidadosos ao definir as relações de classe/membro e classe/subclasse, o que resultou em confusão semântica. Por exemplo, a relação entre um indivíduo e sua classe é diferente da relação entre uma classe (tal como um cão) e a sua superclasse (carnívoro). Da mesma forma, certas propriedades pertencem a indivíduos e outras pertencem à própria classe; a representação deveria fornecer um meio de se fazer essa distinção. As propriedades de ter pelo e gostar de ossos pertencem a cães individuais; a classe “cão” não tem pelo nem come coisa alguma. Entre as propriedades apropriadas para a classe estão incluídos seu nome e sua pertinência a uma taxonomia zoológica.

Em grafos conceituais, cada conceito é um indivíduo único de um tipo particular. Cada bloco de conceito é rotulado com um rótulo de *tipo*, que indica a classe ou o tipo de indivíduo representado por aquele nó. Assim, um nó rotulado cão representa um indivíduo daquele tipo. Os tipos são organizados em uma hierarquia. O tipo cão é um subtipo de carnívoro, que é um subtipo de mamífero etc. Blocos com o mesmo rótulo de tipo representam conceitos do mesmo tipo; entretanto, esses blocos podem ou não representar o mesmo conceito individual.

Cada caixa de conceito é rotulada com os nomes do tipo e do indivíduo. Os rótulos de tipo e indivíduo são separados por um sinal de dois-pontos, “:”. O grafo da Figura 7.16 indica que o cão “Emma” é marrom. O grafo da Figura 7.17 afirma que uma entidade não especificada do tipo cão tem cor marrom. Se o indivíduo não for indicado, o conceito representará um indivíduo não especificado daquele tipo.

**Figura 7.16** Grafo conceitual indicando que o cão de nome Emma é marrom.



**Figura 7.17** Grafo conceitual indicando que um determinado (ainda não identificado) cão é marrom.



Grafos conceituais nos permitem também indicar indivíduos específicos, mas não identificados. Um símbolo único chamado *marcador* indica cada indivíduo do mundo do discurso. Esse marcador é escrito como um número precedido de um #. Os marcadores se diferenciam dos nomes por serem únicos: indivíduos podem ter um nome, muitos nomes, ou nenhum nome, mas eles possuem exatamente um marcador. De forma análoga, indivíduos diferentes podem ter o mesmo nome, mas não o mesmo marcador. Essa distinção nos dá uma base para lidar com as ambiguidades semânticas que surgem quando damos nomes a objetos. O grafo da Figura 7.17 afirma que um determinado cão, #1352, é marrom.

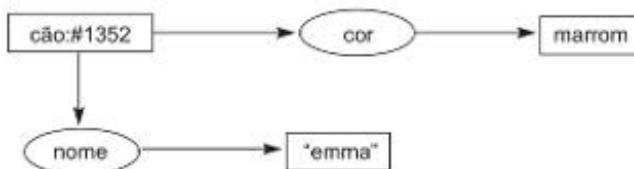
Os marcadores nos permitem separar um indivíduo do seu nome. Se o cão #1352 tiver o nome "Emma", podemos usar uma relação conceitual chamada nome para acrescentar isso ao grafo. O resultado é o grafo da Figura 7.18. O nome é escrito entre aspas duplas para indicar que é uma sequência de caracteres. Onde não houver perigo de ambiguidade, podemos simplificar o grafo e referenciar um indivíduo diretamente pelo nome. Por essa convenção, o grafo da Figura 7.18 é equivalente ao grafo da Figura 7.16.

Embora frequentemente ignoremos, tanto na conversação casual como nas representações formais, essa distinção entre um indivíduo e seu nome, ela é importante e deve ser suportada por uma linguagem de representação. Por exemplo, ao dizermos que "John" é um nome comum entre os homens, afirmamos uma propriedade do próprio nome em vez de uma propriedade de um indivíduo chamado "John". Isso nos permite representar sentenças como "chimpanzé" é o nome de uma espécie de primatas". De forma semelhante, podemos representar o fato de um indivíduo ter vários nomes diferentes. O grafo da Figura 7.19 representa a situação descrita em uma canção de Lennon e McCartney (1968): "Seu nome é McGill, e ela chamava a si própria de Lil, mas era conhecida por todos como Nancy".

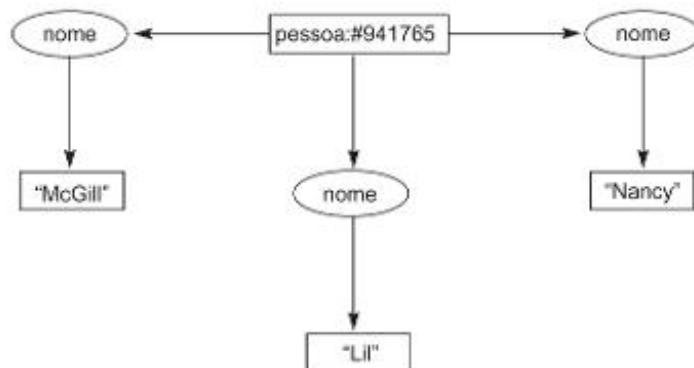
Como alternativa para a indicação de um indivíduo por seu marcador ou nome, podemos também usar o marcador genérico \* para indicar um indivíduo não especificado. Por convenção, isso é frequentemente omitido dos rótulos de conceito; um nó que tenha recebido apenas um rótulo de tipo, cão, é equivalente a um nó rotulado cão:\*. Além do marcador genérico, os grafos conceituais permitem o uso de variáveis com nomes. Essas variáveis são representadas por um asterisco seguido pelo nome da variável (por exemplo, \*X ou \*foo). Isso é útil se dois nós separados indicarem o mesmo indivíduo não especificado. O grafo da Figura 7.20 representa a afirmação "o cão coça sua orelha com sua pata". Embora não saibamos qual cão está coçando a orelha, a variável \*X indica que a pata e a orelha pertencem ao mesmo cão que está coçando.

Em resumo, cada nó conceitual pode indicar um indivíduo de um tipo especificado. Esse indivíduo é o *referente* do conceito. Essa referência é indicada individualmente ou, então, genericamente. Se o referente usa um marcador individual, o conceito é *individual*; se o referente usa o marcador genérico, então o conceito é *genérico*. Na Seção 7.2.3, apresentamos a hierarquia de tipo de grafos conceituais.

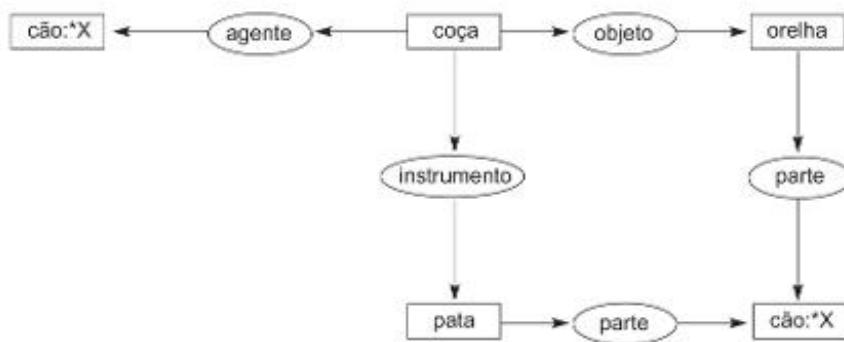
**Figura 7.18** Grafo conceitual indicando que um cão de nome Emma é marrom.



**Figura 7.19** Grafo conceitual de uma pessoa com três nomes.



**Figura 7.20** Grafo conceitual da sentença “O cão coça sua orelha com sua pata”.

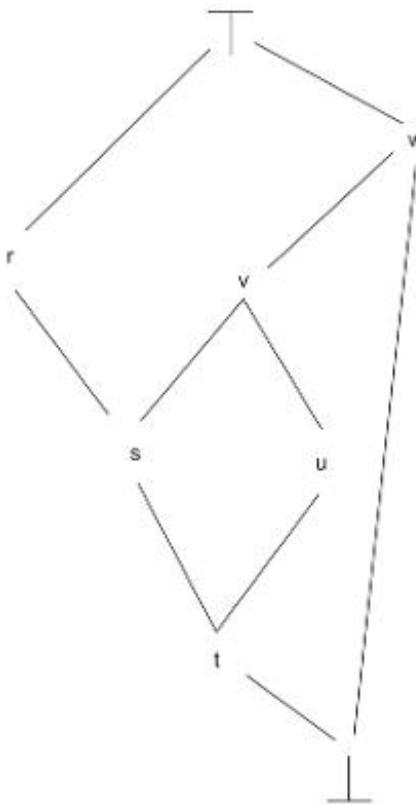


### 7.2.3 Hierarquia de tipo

A hierarquia de tipo, como ilustra a Figura 7.21, é uma ordenação parcial sobre o conjunto de tipos, indicada pelo símbolo  $\leq$ . Se  $s$  e  $t$  forem tipos e  $t \leq s$ , então dizemos que  $t$  é um *subtipo* de  $s$  e  $s$  é um *supertipo* de  $t$ . Como se trata de uma ordenação parcial, um tipo pode ter um ou mais supertipos, bem como um ou mais subtipos. Se  $s$ ,  $t$  e  $u$  forem tipos, com  $t \leq s$  e  $t \leq u$ , diz-se que  $t$  é um *subtipo comum* de  $s$  e de  $u$ . De modo semelhante, se  $s \leq v$  e  $u \leq v$ , então  $v$  é um *supertipo comum* de  $s$  e de  $u$ .

A hierarquia de tipo de grafos conceituais é um reticulado (*lattice*), uma forma comum de sistemas de múltiplas heranças. Em um reticulado, os tipos podem ter múltiplos pais e filhos. Entretanto, cada par de tipos deve ter um *supertipo comum minimal* e um *subtipo comum maximal*. Para os tipos  $s$  e  $u$ ,  $v$  é um supertipo comum minimal se  $s \leq v$ ,  $u \leq v$ , e, para qualquer  $w$ , um supertipo comum de  $s$  e  $u$ ,  $v \leq w$ . O subtipo comum maximal tem uma definição correspondente. O supertipo comum minimal de uma coleção de tipos é o lugar apropriado para se definir propriedades comuns apenas àqueles tipos. Como muitos tipos, tais como *emoção* e *rocha*, não têm supertipos ou subtipos comuns óbvios, é necessário acrescentar tipos que desempenhem esses papéis. Para que a hierarquia de tipo se torne um reticulado verdadeiro, os grafos conceituais incluem dois tipos especiais. O *tipo universal*, indicado por  $T$ , é um supertipo de todos os tipos. O *tipo absurdo*, indicado por  $\perp$ , é um subtipo de todos os tipos.

**Figura 7.21** Um reticulado de tipo ilustrando subtipos, supertipos, o tipo universal e o tipo absurdo. Os arcos representam a relação.



#### 7.2.4 Generalização e especialização

A teoria dos grafos conceituais inclui várias operações que criam novos grafos a partir de grafos existentes. Essas operações permitem a geração de um grafo novo pela especialização ou pela generalização de um grafo existente, operações que são importantes para representar a semântica da linguagem natural. As quatro operações são *copiar*, *restringir*, *juntar* e *simplificar*, como visto na Figura 7.22. Suponha que  $g_1$  e  $g_2$  sejam grafos conceituais. Então:

A regra *copiar* nos permite formar um novo grafo  $g$ , que é a cópia exata de  $g_1$ .

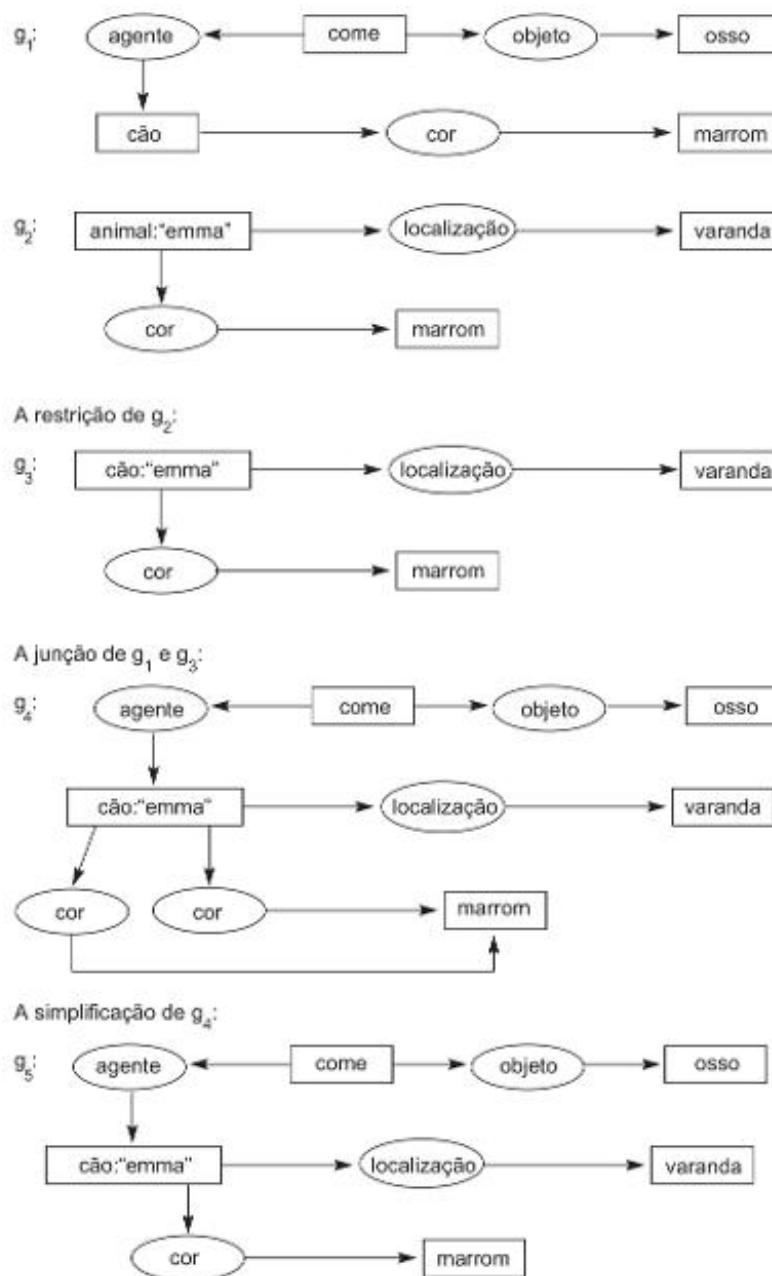
A operação *restringir* permite que nós de conceito de um grafo sejam substituídos por um nó que represente a sua especialização. Há dois casos:

1. Se um conceito é rotulado com um marcador genérico, este pode ser substituído por um marcador individual.
2. Um rótulo de tipo pode ser substituído por um de seus subtipos se isso for consistente com o referente do conceito. Na Figura 7.22, podemos substituir animal por cão.

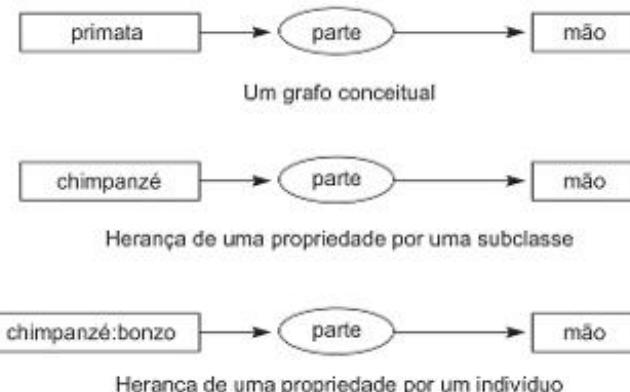
A regra *juntar* nos permite combinar dois grafos em um único grafo. Se houver um nó de conceito  $c_1$  no grafo  $s_1$  que seja idêntico a um nó de conceito  $c_2$  em  $s_2$ , então podemos formar um novo grafo excluindo  $c_2$  e ligando todas as relações incidentes em  $c_2$  a  $c_1$ . *Unir* é uma regra de especialização, porque o grafo resultante é menos geral que qualquer um de seus componentes.

Se um grafo contém duas relações duplicadas, então uma delas pode ser excluída com todos os seus arcos. Essa é a regra *simplificar*. Relações duplicadas ocorrem frequentemente como resultado de uma operação de *unir*, como no grafo  $g_4$  da Figura 7.22.

**Figura 7.22** Exemplos de operações de restringir, unir e simplificar.



Um uso para a regra de *restrição* é fazer com que dois conceitos casem entre si, de modo que uma *junção* possa ser realizada. Juntas, a *junção* e a *restrição* permitem a implementação de herança. Por exemplo, a substituição de um marcador genérico por um individual implementa a herança das propriedades de um tipo por um indivíduo. A substituição de um rótulo de tipo por um rótulo de subtipo define a herança entre uma classe e uma superclasse. Juntando um grafo a outro e restringindo certos nós de conceito, podemos implementar a herança de várias propriedades. A Figura 7.23 mostra como chimpanzés herdam a propriedade de ter uma mão da classe primatas pela substituição de um rótulo de tipo por seu subtipo. A figura mostra, também, como o indivíduo Bonzo herda essa propriedade pela instânciação de um conceito genérico.

**Figura 7.23** Herança em grafos conceituais.

De modo semelhante, podemos usar junções e restrições para implementar as suposições plausíveis que desempenham um papel específico na compreensão de linguagem comum. Se nos dizem que “Maria e Tom sairam para comer uma pizza juntos”, automaticamente fazemos uma série de suposições: eles comeram um pão sírio redondo coberto com queijo e molho de tomate. Eles a comeram em uma pizzaria e devem ter arranjado um jeito de pagar por isso. Podemos raciocinar desse modo usando junções e restrições. Formamos um grafo conceitual da sentença e o *juntamos* com os grafos conceituais (de nossa base de conhecimento) para pizzas e restaurantes. O grafo resultante nos leva a supor que eles comeram molho de tomate e pagaram a conta.

*Junção* e *restrição* são regras de especialização. Elas definem uma ordenação parcial no conjunto de grafos deriváveis. Se um grafo  $g_1$  é uma especialização de  $g_2$ , então podemos dizer que  $g_2$  é uma generalização de  $g_1$ . Hierarquias de generalização são importantes em representação de conhecimento. Além de fornecerem a base para a herança e outros esquemas de raciocínio de senso comum, as hierarquias de generalização são usadas em vários métodos de aprendizado, permitindo, por exemplo, que se construa uma asserção generalizada de uma instância de treinamento particular.

Essas regras não são de inferência. Elas não garantem que a partir de grafos verdadeiros sejam derivados grafos verdadeiros. Por exemplo, na restrição do grafo da Figura 7.19, o resultado pode não ser verdadeiro; Emma pode ser um gato. De modo semelhante, o exemplo de junção da Figura 7.19 também não preserva necessariamente a verdade: o cão na varanda e o cão que come ossos podem ser animais diferentes. Essas operações são *regras de formação canônicas* e, embora elas não preservem a verdade, elas têm a propriedade sutil, mas importante, de preservar o “sentido”. Isso é uma garantia importante quando usamos grafos conceituais para implementar compreensão de linguagem natural. Considere as três sentenças:

Albert Einstein formulou a teoria da relatividade.

Albert Einstein joga como pivô no Los Angeles Lakers.

Grafos conceituais são picolés amarelos voadores.

A primeira dessas sentenças é verdadeira, e a segunda é falsa. A terceira, entretanto, não tem significado; embora gramaticalmente correta, ela não faz sentido. A segunda sentença, embora falsa, tem significado ou faz sentido. Eu posso imaginar Albert Einstein em uma quadra de basquete. As regras de formação canônica impõem restrições sobre o significado semântico; isto é, elas não nos permitem formar grafos sem sentido a partir de grafos que fazem sentido. Embora elas não sejam regras de inferência consistentes, as regras de formação canônica formam a base para vários raciocínios plausíveis que são usados em compreensão de linguagem natural e no raciocínio de senso comum. Discutimos a abordagem com mais detalhes na Seção 15.5.

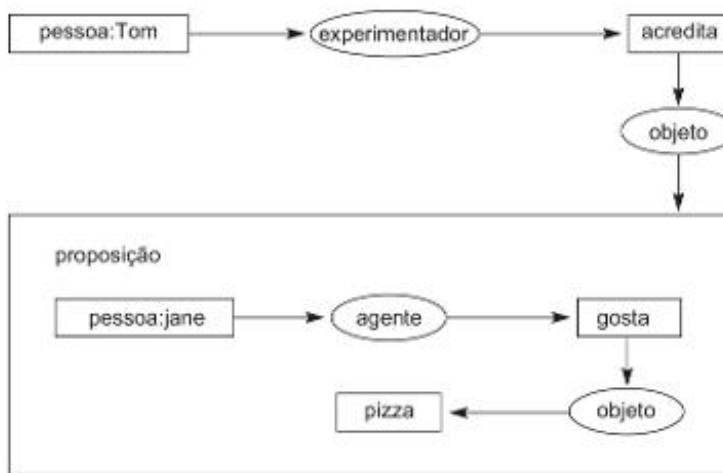
### 7.2.5 Nós proposicionais

Além de usar grafos para definir relações entre objetos do mundo, podemos também querer definir relações entre proposições. Considere, por exemplo, a afirmação “Tom acredita que Jane gosta de pizza”. “Acredita” é uma relação que tem proposições como seus argumentos.

Grafos conceituais incluem um tipo de conceito, *proposição*, que toma um conjunto de grafos conceituais como seu referente e nos permite definir relações envolvendo proposições. Conceitos proposicionais são indicados como uma caixa que contém outro grafo conceitual. Esses conceitos proposicionais podem ser usados com relações apropriadas para representar conhecimento sobre proposições. A Figura 7.24 mostra o grafo conceitual para a afirmação anterior sobre Tom, Jane e pizza. A relação do experimentador é relativamente análoga à relação de agente, já que ela liga um sujeito e um verbo. O elo de experimentador é usado com estados de crença com base na noção que eles são algo que alguém experimenta, em vez de fazer.

A Figura 7.24 mostra como grafos conceituais com nós proposicionais podem ser usados para expressar conceitos *modais* de conhecimento e crença. As *lógicas modais* estão preocupadas com os vários modos como uma proposição é considerada: acredita, afirmada como possível, provavelmente ou necessariamente verdadeira, intencionada como um resultado de uma ação ou contrafeita (Turner, 1984).

**Figura 7.24** Grafo conceitual da declaração “Tom acredita que Jane gosta de pizza”, mostrando uso de um conceito proposicional.

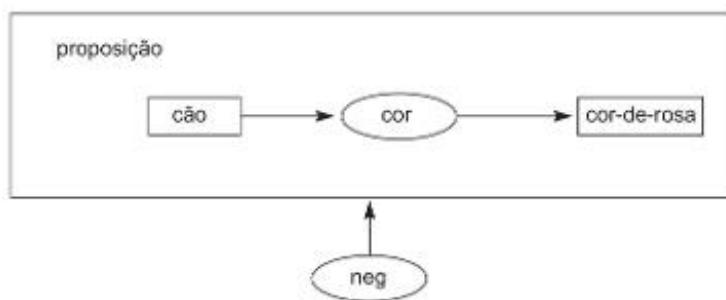


### 7.2.6 Grafos conceituais e lógica

Usando grafos conceituais, podemos representar facilmente conceitos conjuntivos como “o cão é grande e está faminto”, mas não estabelecemos um modo de representar a negação ou a disjunção. Não tratamos, também, da questão da quantificação de variáveis.

Podemos implementar a negação usando conceitos proposicionais e uma operação unária chamada neg. neg toma como argumento um conceito proposicional e afirma que o conceito é falso. O grafo conceitual da Figura 7.25 usa neg para representar a declaração “não existem cães cor-de-rosa”.

**Figura 7.25** Grafo conceitual da proposição “não existem cães cor-de-rosa”.



Usando negação e conjunção, podemos formar grafos que representam declarações disjuntivas de acordo com as regras da lógica. Para simplificar, podemos também definir uma relação ou, que toma duas proposições e representa a sua disjunção.

Em grafos conceituais, supõe-se que conceitos genéricos sejam quantificados existencialmente. Por exemplo, o conceito genérico cão no grafo da Figura 7.14 representa, na realidade, uma variável quantificada existencialmente. Esse grafo corresponde à expressão lógica:

$$\exists X \exists Y (\text{cão}(X) \wedge \text{cor}(X,Y) \wedge \text{marrom}(Y)).$$

Usando a negação e a quantificação existencial, Seção 2.2.2, podemos também representar a quantificação universal. Por exemplo, o grafo da Figura 7.25 poderia ser visto como representante da declaração lógica:

$$\forall X \forall Y (\neg (\text{cão}(X) \wedge \text{cor}(X,Y) \wedge \text{cor_de_rosa}(Y))).$$

Os grafos conceituais são equivalentes ao cálculo de predicados no seu poder expressivo. Como esses exemplos sugerem, existe um mapeamento direto dos grafos conceituais para a notação do cálculo de predicados. O algoritmo, tomado de Sowa (1984), para transformar um grafo conceitual  $g$  em uma expressão do cálculo de predicados é o seguinte:

1. Atribuir uma única variável,  $x_1, x_2, \dots, x_n$ , a cada um dos  $n$  conceitos genéricos em  $g$ .
2. Atribuir uma única constante a cada conceito de indivíduo em  $g$ . Essa constante pode ser simplesmente o nome ou o marcador usado para indicar o referente do conceito.
3. Representar cada nó conceitual com um predicado unário com o mesmo nome que o tipo daquele nó e cujo argumento seja a variável ou a constante dada àquele nó.
4. Representar cada relação conceitual  $n$ -ária de  $g$  como um predicado  $n$ -ário, cujo nome seja o mesmo da relação. Considere que cada argumento do predicado seja a variável, ou a constante, que é atribuída ao nó conceitual correspondente ligado a essa relação.
5. Fazer a conjunção de todas as sentenças atômicas formadas por 3 e 4. Esse é o corpo da expressão do cálculo de predicados. Todas as variáveis na expressão são quantificadas existencialmente.

Por exemplo, o grafo da Figura 7.16 pode ser transformado na expressão do cálculo de predicados:

$$\exists X_1 (\text{cão}(\text{emma}) \wedge \text{cor}(\text{emma}, X_1) \wedge \text{marrom}(X_1))$$

Embora possamos reformular os grafos conceituais na sintaxe do cálculo de predicados, os grafos conceituais suportam vários mecanismos de inferência de propósito especial, tais como a *junção* e a *restrição*, apresentadas na Seção 7.2.4, que normalmente não fazem parte do cálculo de predicados.

Apresentamos a sintaxe dos grafos conceituais e definimos a operação *restrição* como um meio de implementar a herança. Ainda não examinamos toda a gama de operações e inferências que podem ser realizadas

com relação a esses grafos, nem tratamos do problema de definir os conceitos e relações necessários para domínios como a linguagem natural. Abordamos essas questões novamente na Seção 15.3.2, onde usamos grafos conceituais para implementar uma base de conhecimento para um programa simples de compreensão de linguagem natural.

## 7.3 Alternativas para representações e ontologias

Em anos recentes, pesquisadores de IA continuaram a questionar o papel da representação explícita na inteligência. Além das abordagens conexionistas e emergentes dos capítulos 11 e 12, um desafio mais direto ao papel da representação explícita veio do trabalho de Rodney Brooks no MIT (Brooks, 1991a). Brooks questiona, no projeto de um robô explorador, a necessidade de *qualquer* esquema representacional central e, com a sua *arquitetura de subsunção* (*subsumption architecture*), mostra como inteligência geral pode evoluir de formas inferiores e básicas de inteligência.

Uma segunda abordagem para o problema das representações explícitas e estáticas vem do trabalho de Melanie Mitchell e Douglas Hofstadter da Indiana University. A arquitetura *Copycat* é uma rede evolutiva que se ajusta às relações de significado que ela descobre por meio de experimentação com o mundo externo.

Por fim, ao criar resolvedores de problemas em ambientes complexos, quase sempre é necessário empregar vários esquemas representativos diferentes. Cada esquema representativo pode ser considerado uma *ontologia*. Depois, torna-se necessário criar a comunicação e outros elos entre essas diferentes representações para dar suporte ao gerenciamento de conhecimento, comunicação, backup e outros aspectos da resolução do problema. Introduzimos essas questões e as possíveis soluções sob o título geral de *tecnologia de gerenciamento de conhecimento*. Essa tecnologia também pode dar suporte à resolução de problemas guiada por agente, conforme apresentada na Seção 7.4.

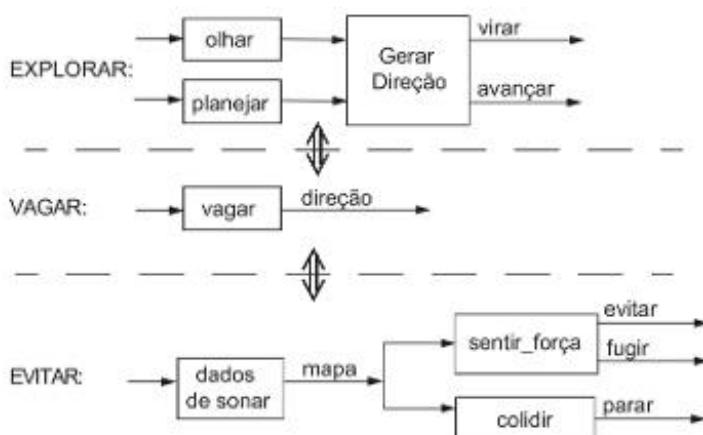
### 7.3.1 Arquitetura de subsunção de Brooks

Brooks conjectura e oferece exemplos a partir de suas criações robóticas que o comportamento inteligente, racional, não resulta de sistemas incorpóreos como provadores de teoremas, ou mesmo de sistemas especialistas tradicionais (Seção 8.2). A inteligência, afirma Brooks, é o produto da interação entre um sistema apropriadamente projetado e seu ambiente. Além disso, Brooks expõe a visão de que o comportamento inteligente *emerge* das interações de arquiteturas de comportamentos organizados mais simples: sua *arquitetura de subsunção*.

A arquitetura de subsunção é uma coleção em camadas de tratadores de tarefas. Cada tarefa é realizada por uma máquina de estados finitos, que mapeia continuamente uma entrada baseada em percepção em uma saída orientada à ação. Esse mapeamento é realizado por conjuntos simples de regras de produção *condição → ação* (Seção 6.2). Essas regras determinam, de um modo bastante cego, isto é, sem conhecimento do estado global, quais ações são apropriadas para o estado atual daquele subsistema. Brooks permite alguma realimentação dos sistemas de mais baixo nível.

A Figura 7.26, adaptada de Brooks (1991a), mostra uma arquitetura de subsunção de três camadas. Cada camada é composta por uma rede com topologia fixa de máquinas de estados finitos simples, cada uma tendo poucos estados, um ou dois registradores, um ou dois relógios internos e acesso a dispositivos computacionais simples, por exemplo, para calcular somas de vetores. Essas máquinas de estados finitos funcionam de modo assíncrono, enviando e recebendo mensagens de comprimento fixo por meio de linhas de dados. Não há um bloco central de controle. Em vez disso, cada máquina de estados finitos é guiada pelos dados das mensagens que recebe. A chegada de uma mensagem ou a expiração de um período de tempo causa a mudança de estado das máquinas. Não há acesso a dados globais ou a qualquer elo de comunicação criado dinamicamente. Assim, não há a possibilidade de controle global.

**Figura 7.26** As funções da arquitetura de subsunção de três camadas de Brooks (1991a). As camadas são descritas pelos comportamentos EVITAR, VAGAR e EXPLORAR.



A Figura 7.26 apresenta um subconjunto das funções da arquitetura de três camadas que suportava um dos primeiros robôs (Brooks, 1991a). O robô tinha um anel com doze sensores de sonar ao seu redor. A cada segundo, esses sensores forneciam doze medidas de profundidade radial. A camada do nível mais baixo da arquitetura de subsunção, EVITAR, implementa um comportamento em que o robô evita colidir com objetos, quer eles sejam estáticos, quer sejam móveis. A máquina rotulada como dados de sonar emite um mapa instantâneo que é passado para colidir e sentir\_força, que, por sua vez, são capazes de gerar mensagens de parar para a máquina de estados finitos que está encarregada de fazer o robô avançar. Quando sentir\_força está ativada, ela é capaz de gerar instruções de fugir ou evitar para se afastar de objetos e ameaças.

A rede do nível mais baixo de máquinas de estados finitos da arquitetura gera todas as instruções de parar e evitar para todo o sistema. A próxima camada, VAGAR, faz com que o sistema vagueie pela geração de uma direção aleatória para o movimento do robô a cada dez segundos. A máquina (do nível mais baixo) EVITAR recebe a direção gerada por VAGAR e a acopla com as forças calculadas pela arquitetura EVITAR. VAGAR usa o resultado para reprimir o comportamento de mais baixo nível, forçando o robô a se mover em uma direção próxima àquela que VAGAR decidiu, mas ao mesmo tempo evitar todos os obstáculos. Finalmente, se as máquinas virar e avançar (no nível mais alto da arquitetura) forem ativadas, elas reprimirão qualquer impulso novo enviado por VAGAR.

O nível mais alto, EXPLORAR, faz com que o robô tente explorar o seu ambiente, procurando por lugares distantes e tentando alcançá-los por meio do planejamento de um caminho. Essa camada é capaz de reprimir as instruções de vagar e observa como a camada mais baixa desvia o robô devido aos obstáculos. Ela corrige esses desvios e mantém o robô focado no seu objetivo, inibindo o comportamento de vagar, mas permitindo que o desvio de objetos no nível mais baixo continue com a sua função. Quando ocorrem os desvios gerados pelo nível mais baixo, o nível EXPLORAR chama planejar para manter o sistema orientado ao objetivo. O aspecto principal da arquitetura de subsunção de Brooks é que o sistema não necessita de um raciocínio simbólico centralizado e executa todas as suas ações sem buscar através de próximos estados possíveis. Embora a máquina de estados finitos baseada em comportamento gere sugestões para ações com base no seu próprio estado atual, o sistema global age com base nas interações dos sistemas nas camadas abaixo dele.

A arquitetura em três camadas apresentada se originou no projeto inicial de Brooks para um robô que vagasse e buscasse um objetivo. Mais recentemente, o seu grupo de pesquisa construiu sistemas complexos com mais camadas (Brooks, 1991a; Brooks e Stein, 1994). Um sistema é capaz de vagar pelo Laboratório de Robótica do MIT procurando por latas de bebida de alumínio vazias sobre as mesas. Esse sistema requer camadas para descobrir escritórios, procurar mesas e reconhecer latas de bebida. Outras camadas são capazes de guiar o braço do robô para coletar essas latas para fins de reciclagem.

Brooks insiste que o comportamento de alto nível emerge como resultado do projeto e do teste das camadas individuais dos níveis mais baixos da arquitetura. O projeto de comportamentos finais coerentes, que requer comunicação tanto entre camadas como dentro delas, é descoberto por meio de experimentação. Apesar dessa simplicidade de projeto, contudo, a arquitetura de subsunção tem obtido sucesso em várias aplicações (Brooks, 1989, 1991a, 1997).

Entretanto, continua em aberto uma série de questões importantes envolvendo a arquitetura de subsunção e outras abordagens relacionadas ao projeto de sistemas de controle (ver também a Seção 16.2):

1. Existe um problema de suficiência de informação local em cada nível do sistema. Como em cada nível as máquinas de estado puramente reativas tomam decisões com base em informação local, isto é, sobre dados locais, é difícil se imaginar como tais tomadas de decisão podem levar em conta qualquer informação que não esteja naquele nível local. Por definição, elas serão míopes.
2. Se não existir absolutamente nenhum “conhecimento” ou “modelo” do ambiente total, como as entradas limitadas à situação local podem ser suficientes para determinar ações globalmente aceitáveis? Como pode ser possível que isso resulte em coerência no nível mais alto?
3. Como um componente puramente reativo com estados muito limitados pode *aprender* a partir de seu ambiente? Em algum nível do sistema deve haver estados suficientes para a criação de mecanismos de aprendizado, se desejarmos que o agente global seja chamado de inteligente.
4. Há um problema de escala. Embora Brooks e seus associados afirmem que têm construído arquiteturas de subsunção de seis e até mesmo de dez camadas, quais são os princípios de projeto que permitem aumentar a escala para se obter comportamentos mais interessantes, isto é, para sistemas grandes e complexos?

Finalmente, devemos perguntar: O que é “emergência”? Ela é mágica? Nesse estágio de evolução da ciência, “emergência” parece ser uma palavra para abranger aqueles fenômenos que ainda não podemos esclarecer. Dizem que devemos construir sistemas e testá-los no mundo, e, assim, eles mostrarão inteligência. Infelizmente, sem outras instruções de projeto, a emergência se torna apenas uma palavra para descrever o que ainda não compreendemos. Como resultado, é muito difícil determinar como podemos *usar* essa tecnologia para construir sistemas cada vez mais complexos. Na próxima seção, descrevemos *copycat*, uma arquitetura híbrida que é capaz de, por exploração, descobrir e usar invariâncias encontradas em um domínio de problema.

### 7.3.2 *Copycat*

Uma crítica muito frequente aos esquemas tradicionais de representação de IA é que eles são estáticos e, possivelmente, não conseguirão refletir a natureza dinâmica dos processos do pensamento e da inteligência. Quando uma pessoa percebe uma situação nova, por exemplo, ela normalmente é bombardeada por relações com situações já conhecidas ou análogas. Na verdade, frequentemente é verificado que a percepção humana se dá tanto de baixo para cima, isto é, estimulada por novos padrões do ambiente, como de cima para baixo, mediada por aquilo que o agente espera perceber.

*Copycat* é uma arquitetura para resolver problemas, construída por Melanie Mitchell (1993) como tese de doutorado, sob a orientação de Douglas Hofstadter (1995) na Indiana University. A *copycat* se baseia em muitas das técnicas representacionais que a precederam, incluindo a arquitetura de quadro-negro (Seção 6.3), as redes semânticas (Seção 7.2), as redes conexionistas (Capítulo 11) e os sistemas de classificação (Holland, 1986). Ela também segue a abordagem de Brooks para a solução de problemas por intervenção ativa no seu domínio do problema. Entretanto, diferentemente de Brooks e das redes conexionistas, a *copycat* requer que um “estado” global faça parte do resolvidor de problemas. Em segundo lugar, a representação é uma característica evolutiva desse estado. A *copycat* admite um mecanismo semelhante a uma rede semântica que cresce e se modifica com a experiência continuada com o seu ambiente.

O domínio de problemas original da *copycat* foi a percepção e a construção de analogias simples. Nesse sentido, ele se baseia no trabalho anterior de Evans (1968) e de Reitman (1965). Alguns exemplos desse domínio são

o complemento de padrões: quente está para frio assim como alto está para {parede, baixo, molhado, rodapé}, ou urso está para porco assim como cadeira está para {pé, mesa, café, morango}. A *copycat* também foi usada para descobrir complementos adequados para padrões de caracteres alfabéticos como: abc está para abd assim como ijk está para ?, ou, ainda, abc está para abd assim como ijkk está para ?, na Figura 7.27.

A *copycat* é constituída por três componentes principais: o *espaço de trabalho*, a *rede deslizante* e a *estante de códigos*. Esses três componentes são mediados em suas interações por uma medida de *temperatura*. O mecanismo de temperatura captura o grau de organização perceptiva do sistema, bem como controla o grau de aleatoriedade usado nas tomadas de decisão. Temperaturas mais altas refletem o fato de que há pouca informação que possa servir de base para decisões, e assim elas são mais aleatórias. Uma queda de temperatura indica que o sistema está próximo a um consenso, e uma temperatura baixa indica que uma resposta está surgindo e reflete a “confiança” do programa na solução que produz.

O *espaço de trabalho* é uma estrutura global similar ao quadro-negro, da Seção 6.3, para criar estruturas que os outros componentes do sistema podem inspecionar. Nesse sentido, ele é muito parecido com uma área de mensagem do sistema classificador de Holland (1986). O espaço de trabalho é o lugar onde as estruturas perceptivas são construídas hierarquicamente sobre as entradas (as três cadeias de símbolos alfabéticos da Figura 7.27) e apresenta os estados possíveis para o espaço de trabalho, com ligações (as setas) construídas entre componentes relacionados da cadeia.

A *rede deslizante* reflete a rede de conceitos ou associações potenciais para os componentes da analogia. Um modo de ver a rede deslizante é como uma rede semântica dinamicamente deformável, cada um dos nós tendo um nível de ativação. Os elos da rede podem ser rotulados por outros nós. Com base nos níveis de ativação dos nós de rótulo, os nós conectados aumentam ou diminuem. Desse modo, o sistema modifica o grau de associação entre os nós em função do contexto. O espalhamento de ativação entre nós é encorajado mais entre aqueles que estejam mais proximamente relacionados no contexto atual.

A *estante de códigos* é uma fila probabilística controlada por prioridade contendo *codeletas*. Codeletas são pequenas partes de código executável, concebidas para interagir com os objetos do espaço de trabalho e para favorecer uma pequena parte da solução que é desenvolvida, ou, mais simplesmente, para explorar aspectos diferentes do espaço do problema. As codeletas são muito parecidas com os classificadores individuais do sistema de Holland (1986).

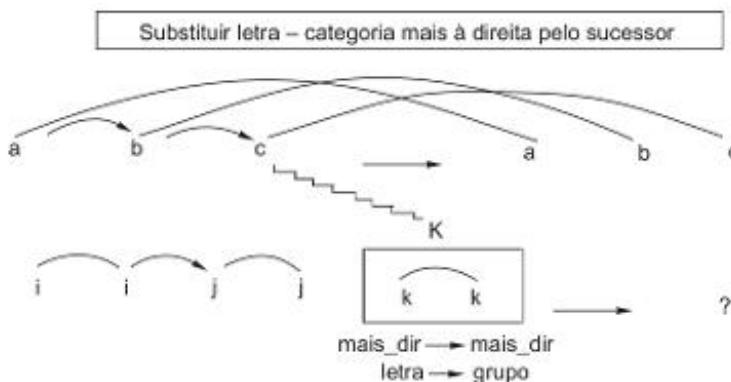
Os nós que estão na rede deslizante geram codeletas e as transmitem para a estante de código, onde elas têm uma chance probabilística de execução. Assim, o sistema mantém partes de código em paralelo que competem pela oportunidade de encontrar e construir estruturas no espaço de trabalho. Esses códigos correspondem aos nós de onde eles vieram. Essa é uma atividade de cima para baixo, que procura por mais exemplos de coisas que já geraram interesse. As codeletas podem também funcionar de baixo para cima, tentando identificar e construir a partir de relações já existentes entre os objetos do espaço de trabalho.

Ao longo da construção de estruturas no espaço de trabalho, é adicionada uma medida de *ativação* aos nós que geraram as codeletas que construíram as estruturas. Isso permite que o comportamento do sistema no contexto do estado atual do problema/da solução afete o seu comportamento futuro.

Por fim, a *temperatura* serve como um mecanismo de realimentação, calculando a “coesão” entre as estruturas construídas no espaço de trabalho. Com pouca coesão, isto é, poucas estruturas promissoras para uma solução, as influências nas escolhas probabilísticas feitas pelo sistema são menos importantes: uma escolha é tão útil como outra qualquer. Quando a coesão é alta, com uma solução internamente consistente que evolui, as influências se tornam muito importantes nas escolhas probabilísticas: as escolhas que são feitas são mais intimamente ligadas à solução evolutiva.

Há vários novos projetos no mundo de *copycat* para testar a arquitetura em configurações mais ricas. Hofstadter e seus estudantes (Marshall, 1999) continuam a modelar relações analógicas. Lewis e Luger (2000) expandiram a arquitetura *copycat* para ser usada como um sistema de controle para um robô móvel. O domínio robótico dá à arquitetura *copycat* um ambiente concreto, com suporte para interações orientadas ao aprendizado. A evolução da interação do robô com seu “mundo” faz com que seja gerado um mapa do espaço e que caminhos sejam planejados e replanejados.

**Figura 7.27** Um estado possível do espaço de trabalho da *copycat*. São mostrados vários exemplos de ligações e elos entre as letras; adaptado de Mitchell (1993).



### 7.3.3 Múltiplas representações, ontologias e serviços de conhecimento

Muitas aplicações complexas de IA exigem que vários esquemas de representação sejam projetados e que cada um deles seja configurado para cumprir tarefas específicas e depois integrado em uma plataforma de software orientada ao usuário. Um exemplo simples disso poderia ser a criação de um espaço de encontro virtual. Os diversos atendentes, de seus locais remotos, interagiriam entre si nesse “espaço” virtual. Além da tecnologia de voz, o espaço de encontro virtual exigiria *streaming* (fluxo de dados) de vídeo, compartilhamento de texto e apresentação, acesso a registros, e assim por diante. Uma técnica para essa tarefa seria adaptar diversos componentes já prontos e depois integrá-los em um serviço de conhecimento com suporte para as interações exigidas.

O serviço automatizado de reunião virtual recém-descrito é um exemplo de integração de vários esquemas de representação para fornecer um serviço. Normalmente, usamos o termo *ontologia* para caracterizar o esquema de representação de um componente em particular desse serviço. O serviço de conhecimento resultante requer a integração desses componentes. A ontologia é derivada de uma palavra grega, uma forma do verbo “ser”, que significa *ser* ou *existência*. Assim, o termo se refere aos componentes de um módulo de software e à estrutura que dá suporte às interações desses componentes, ou seja, à sua função ou “ser constituinte” como parte do módulo.

Quando os módulos são compostos para montar o serviço de conhecimento, cada um deles precisa ser vinculado de modo apropriado para formar o serviço transparente. Dizemos que as ontologias de cada módulo são mescladas para criar uma nova ontologia mais complexa, que dê suporte ao serviço completo. Para criar um novo serviço de conhecimento, como o software para dar suporte à reunião virtual, as ontologias de cada um de seus componentes precisam ser compreendidas e frequentemente expostas para a integração. Atualmente, existem diversas linguagens de ontologia projetadas explicitamente para a criação de serviços de conhecimento e outros produtos de software relacionados. Estas incluem a família Owl de linguagens (Mika et al., 2004), bem como a API Java de código fonte aberto SOFA (do inglês *Simple Ontology Framework*) (ver <<http://sopha.projects.sem-webcentral.org/>>).

De fato, a informação ontológica é usada de diversas maneiras na computação moderna; algumas simples, como o acesso otimizado ao conhecimento armazenado em um banco de dados tradicional, e outras bastante originais. Com a importância do WWW, sofisticados coletores de páginas *web* e máquinas de busca são ferramentas de suporte críticas. Embora a maioria dos coletores de páginas *web* específicas atuais aplique sua busca a arquivos de texto vinculados diretamente, um coletor mais sofisticado pode ser preparado para atravessar as classes e as ligações de representações complexas particulares, além de “interpretar” entidades mais úteis, incluindo grafos e imagens.

Embora a “interpretação de imagem” completa continue além do estado atual de nossas capacidades, sua criação ajudará a dar suporte aos nossos desejos de uma *web* mais “semântica”. Podemos querer vasculhar nossas fotografias de férias para achar todas as que exibem uma pessoa em particular, ou algum animal, por exemplo.

No contexto maior da web, podemos querer não apenas rastrear e achar quaisquer histórias de jornal que descrevam desastres naturais, mas também localizar quaisquer imagens que apresentem esses desastres. Atualmente, com frequência temos nos limitado à busca das legendas de uma figura ou outro material descritivo (textual) para obter um indicio de seu conteúdo.

Além do objetivo de interpretar dados gráficos e de imagem, também se deseja localizar dados de texto sobre tópicos específicos e produzir um resumo de seu conteúdo. Por exemplo, poderíamos querer procurar artigos de pesquisa on-line em certa área e informar seu conteúdo. Embora esse seja outro assunto de pesquisa atual na área de compreensão da linguagem natural (Capítulo 15), sua implementação continua no limite de nossas habilidades “semânticas” baseadas em computador. Atualmente, fazemos coisas como procurar algumas palavras-chave que poderiam indicar o conteúdo de um artigo e talvez imprimir seu resumo, mas, fora isso, nossas habilidades de resumos são limitadas (Ciravegna e Wilks, 2004; Wilks, 2004).

Outro serviço da web poderia ser necessário para procurar anúncios de emprego. Ele poderia ser disparado para procurar profissionais de software, por exemplo, com habilidades em Java e C++ e localizados em uma cidade específica. Esse coletor de páginas web precisaria saber algo sobre esse tipo de anúncio, incluindo os fatos de que o nome da empresa e sua localização normalmente são incluídos. Além das habilidades exigidas para a função, esses anúncios normalmente descrevem a experiência necessária com possíveis salários e benefícios. Assim, o serviço de emprego pela web precisa ter o conhecimento e a capacidade para acessos às estruturas ontológicas usadas no projeto de “anúncios” de emprego. Apresentamos um exemplo desse serviço de conhecimento na Seção 15.5.

No entanto, talvez a aplicação mais importante para os serviços de conhecimento baseados em ontologia seja a da solução de problemas orientada a agentes. Por definição, os agentes deverão ser autônomos, independentes e flexíveis ao endereçar tarefas, localizados em situações específicas e capazes de se comunicar de modo apropriado. A solução de problemas baseada em agentes também é altamente distribuída entre ambientes, como poderia ser exigido para a construção de um serviço web inteligente. A criação de habilidades específicas do agente pode ser vista como uma questão de criação e vínculo de ontologias. Na próxima seção, sobre solução de problemas baseada em agentes, consideraremos mais a visão da representação distribuída e orientada a componentes.

## 7.4 Solução de problemas distribuída e baseada em agentes

Havia dois pontos de vista na comunidade de pesquisadores em IA, nos anos 1980, que tiveram consequências importantes para a pesquisa futura na análise do papel da representação na solução inteligente de problemas. O primeiro era a tradição em pesquisas da “Inteligência Artificial Distribuída”, ou comunidade de “IAD”. O primeiro encontro de IAD aconteceu no MIT, em 1980, para discutir questões relacionadas com a solução inteligente de problemas com sistemas consistindo de múltiplos resolvedores de problemas. Naquela ocasião, foi decidido pela comunidade de IAD que eles *não* estavam interessados em questões de paralelismo de baixo nível, por exemplo, como distribuir o processamento para diferentes máquinas ou como paralelizar algoritmos complexos. Em vez disso, o seu objetivo era compreender como resolvedores de problemas distribuídos poderiam ser efetivamente coordenados para a solução inteligente de problemas. Na verdade, já havia uma história mais antiga do processamento distribuído em inteligência artificial, com o uso e coordenação de *atores* e *demons* e o projeto de sistemas de quadro-negro, como visto na Seção 6.3.

O segundo ponto de vista das pesquisas em IA dos anos 1980, já apresentado na Seção 7.3, foi aquele de Rodney Brooks e seu grupo no MIT. O desafio de Brooks para a visão da comunidade de IA sobre representação e raciocínio teve várias consequências importantes (Seção 7.3.1). Primeiro, a noção de que a solução inteligente de problemas não requer um armazenamento central de conhecimento manipulado por um esquema de inferência de propósito geral levou à noção de modelos distribuídos e cooperativos de inteligência, em que cada elemento da representação distribuída era responsável pelo seu próprio componente do processo de solução do

problema. Segundo, o fato de a inteligência ser localizada e ativa no contexto de tarefas particulares permite que o resolvedor de problemas transfira aspectos do processo de solução no próprio ambiente. Isso permite, por exemplo, que um resolvedor individual execute uma determinada tarefa e, ao mesmo tempo, não tenha conhecimento sobre o progresso da solução dentro do domínio geral do problema. Assim, um agente da *web*, por exemplo, pode verificar uma informação de estoque, enquanto outro verifica a aprovação de crédito de um cliente, sem que nenhum dos agentes saiba da tomada de decisão do nível mais alto, por exemplo, se uma compra será permitida ou não. Essas duas ênfases dos anos 1980 ocasionaram o interesse atual *no projeto e no uso de agentes inteligentes*.

### 7.4.1 Solução de problemas orientada a agentes: definição

Antes de prosseguir com a discussão sobre pesquisa em agentes, definiremos o que entendemos por “agente”, “sistema baseado em agentes” e “sistemas multiagentes”. Entretanto, existem problemas com essas definições, pois vários grupos diferentes da comunidade de pesquisa em agentes têm opiniões diferentes sobre o que exatamente significa a solução de problemas baseada em agentes. Nossa definição e discussão são baseadas em uma extensão do trabalho de Jennings, Sycara e Wooldridge (1998), Wooldridge (2000), Wooldridge et al. (2006, 2007) e Lewis e Luger (2000).

Para nós, um sistema multiagente é um programa de computador com resolvedores de problemas localizados em ambientes interativos, que são capazes de ações flexíveis, autônomas e, ainda, socialmente organizadas que podem, mas não necessariamente, ser dirigidas para objetivos ou metas predeterminadas. Assim, os quatro critérios para um sistema de agentes inteligentes incluem resolvedores de problemas que são *situados, autônomos, flexíveis e sociais*.

Um agente inteligente ser *situado*, o que significa que ele recebe entradas do ambiente no qual está ativo e pode, também, efetuar modificações dentro desse ambiente. Exemplos de ambientes para agentes situados incluem a Internet, jogos ou uma aplicação da robótica. Um exemplo concreto poderia ser o de um jogador de futebol em uma competição da ROBOCUP (Veloso et al., 2000) que deve interagir apropriadamente com a bola e um oponente sem total conhecimento das localizações, dos desafios e dos sucessos de todos os outros jogadores. Essa situação pode ser contrastada com resolvedores de problemas de IA mais tradicionais, como o planejador STRIPS, que veremos na Seção 8.4, ou o sistema especialista MYCIN, na Seção 8.3, que mantém conhecimento exaustivo e centralizado dos seus domínios de aplicação.

Um sistema *autônomo* é aquele que pode interagir com o seu ambiente sem a intervenção direta de outros agentes. Para tanto, ele deve ter controle sobre suas próprias ações e seu estado interno. Alguns agentes autônomos podem, também, aprender com a sua experiência a fim de melhorar o seu desempenho ao longo do tempo (veja aprendizado de máquina na Parte IV). Por exemplo, na Internet, um agente autônomo poderia fazer uma verificação de autenticação de cartão de crédito, independentemente de outras questões da transação de compra. No exemplo da ROBOCUP, um agente poderia passar a bola para outro jogador de seu time ou chutar a gol, dependendo de sua situação individual.

Um agente *flexível* é tanto intelligentemente *responsivo* como *proativo* dependendo de sua situação atual. Um agente responsivo recebe estímulos de seu ambiente e responde a eles de forma apropriada e no momento oportuno. Um agente proativo não apenas responde a situações que ocorrem no seu ambiente, mas é também capaz de planejar, ser oportunista, direcionado a objetivo e ter alternativas apropriadas para diversas situações. Um agente de crédito, por exemplo, seria capaz de retornar ao usuário com resultados ambíguos ou encontrar outra agência de crédito se uma alternativa não fosse suficiente. O agente jogador de futebol poderia mudar o seu drible dependendo do padrão de desafio de um oponente.

Por fim, um agente é *social* quando ele pode interagir, de modo apropriado, com outro software ou com agentes humanos. Afinal, um agente é apenas parte de um processo complexo de solução de um problema. As interações do agente social são orientadas aos objetivos do sistema multiagente maior. Essa dimensão social do sistema de agentes deve lidar com muitas situações difíceis. Aí se incluem: Como agentes diferentes devem participar de uma subtarefa na solução de um problema? Como os agentes podem se comunicar

entre si para facilitar a realização da tarefa de nível mais alto do sistema?; no exemplo da ROBOCUP, isso poderia ser marcar um gol. Como um agente pode ajudar nos objetivos de outro agente, como, por exemplo, tratar as questões de segurança de uma tarefa de Internet? Todas essas questões relacionadas com a dimensão social são o assunto de pesquisas atuais.

Descrevemos a base para criar sistemas multiagentes, que são ideais para representar problemas que incluem muitos métodos de solução de problemas, múltiplos pontos de vista e múltiplas entidades. Nesses domínios, os sistemas multiagentes oferecem as vantagens da solução de problemas concorrente e distribuída, juntamente com as vantagens dos esquemas sofisticados de interação. Exemplos de interação incluem a cooperação no trabalho para alcançar um objetivo comum, a coordenação na organização das atividades da solução do problema, de modo que interações prejudiciais sejam evitadas e as possibilidades benéficas sejam exploradas, e a negociação de restrições de subproblemas, de modo que se alcance um desempenho aceitável. É a flexibilidade dessas interações sociais que distingue os sistemas multiagentes de software mais tradicionais e que confere poder e entusiasmo ao paradigma de agentes.

Nos anos mais recentes, o termo *sistema multiagente* se refere a todos os tipos de sistemas de software compostos de múltiplos componentes semiautônomos. O sistema de agentes distribuído considera como um problema em particular pode ser solucionado por diversos módulos (agentes) que cooperam dividindo e compartilhando o conhecimento sobre o problema e a solução que evolui daí. As pesquisas em sistemas multiagentes estão focadas nos comportamentos de coleções de agentes autônomos, algumas vezes já existentes, com o objetivo de solucionar um determinado problema. Um sistema multiagente pode ser visto, também, como uma rede fracamente acoplada de resolvidores de problemas que trabalham juntos em problemas que podem estar além da capacidade individual de qualquer um dos agentes (Durfee e Lesser, 1989). Veja também o projeto de serviços de conhecimento e ontologias (Seção 7.3.3).

Os resolvidores de problemas de um sistema multiagente, além de serem autônomos, podem ter, também, um projeto heterogêneo. Com base na análise de Jennings, Sycara e Wooldridge (1998) e Wooldridge et al. (2006, 2007), existem quatro características importantes da solução de problemas por sistemas multiagentes. Em primeiro lugar, cada agente tem informação incompleta e capacidades insuficientes para solucionar o problema completo, e assim pode sofrer de um ponto de vista limitado. Em segundo lugar, não há um controlador global do sistema para a solução completa do problema. Em terceiro lugar, o conhecimento e os dados de entrada para o problema são também descentralizados e, em quarto lugar, os processos de raciocínio são frequentemente assíncronos.

É interessante que os programadores tradicionais que usam o paradigma orientado a objetos frequentemente não conseguem ver algo de novo em um sistema baseado em agentes. Considerando-se as propriedades relativas de agentes e de objetos, isso pode ser compreensível. Objetos são definidos como sistemas computacionais com estado encapsulado, tendo métodos associados com esse estado que permitem interações em um ambiente e que se comunicam enviando mensagens.

As diferenças entre objetos e agentes incluem o fato de que objetos raramente exibem controle sobre seu próprio comportamento. Não vemos agentes invocando métodos entre si, mas sim requisitando ações que devem ser executadas. Além disso, os agentes são projetados para terem comportamento flexível, isto é, reativo, proativo e social. Finalmente, agentes interativos são vistos frequentemente como tendo suas próprias linhas de controle. Entretanto, todas essas diferenças não devem servir para indicar que linguagens de programação orientadas a objetos, como C++, Java ou CLOS, não possam oferecer um meio adequado para a construção de sistemas de agentes; muito pelo contrário, seu poder e flexibilidade as tornam ideais para essa tarefa.

#### 7.4.2 Exemplos de um paradigma orientado a agentes e seus desafios

Para tornar as ideias da seção anterior mais concretas, descrevemos, a seguir, uma série de domínios de aplicação em que a solução de problemas baseada em agentes é apropriada. Incluímos, também, referências a pesquisas nessas áreas de problema.

**Fabricação** O domínio da fabricação pode ser modelado como uma hierarquia de áreas de trabalho. Pode haver áreas de trabalho em moagem, tornearia, pintura, montagem, e assim por diante. Essas áreas de trabalho são agrupadas em subsistemas de fabricação, sendo cada subsistema uma função dentro do processo maior de fabricação. Esses subsistemas de fabricação podem, então, ser agrupados em uma única fábrica. Uma entidade maior, a empresa, pode controlar aspectos de cada uma dessas fábricas, por exemplo, gerenciar pedidos, estoques, níveis de duplicação, lucros etc. Referências sobre fabricação baseada em agentes incluem trabalhos em sequenciamento de produção (Chung e Wu, 1997), operações de fabricação (Oliveira et al., 1997) e projeto colaborativo de produtos (Cutosky et al., 1993; Darr e Birmingham, 1996; e Woldridge et al., 2007).

**Controle automatizado** Como os controladores de processos são sistemas autônomos, reativos e frequentemente distribuídos, não é de surpreender que modelos de agentes possam ser importantes. Existem pesquisas em sistemas de controle de transporte (Corera et al., 1996), controle espacial (Schwuttke e Quan, 1993), aceleradores de feixes de partículas (Perriolat et al., 1996; Klein et al., 2000), controle de tráfego aéreo (Ljunberg e Lucas, 1992) e outros.

**Telecomunicações** Sistemas de telecomunicações são grandes redes distribuídas de componentes interativos que requerem monitoração e gerenciamento em tempo real. Sistemas baseados em agentes têm sido usados para controle e gerenciamento de redes (Schoonderwoerd et al., 1997; Adler et al., 1989; Fatima et al., 2006), transmissão e comutação (Nishibe et al., 1993) e serviços (Busuoic e Griffiths, 1994). Para ter uma visão geral abrangente, consulte Veloso et al., 2000.

**Sistemas de transporte** Sistemas de tráfego são quase que, por definição, distribuídos, situados e autônomos. Entre as aplicações se incluem a coordenação de comutadores e de carros compartilhados (Burmeister et al., 1997) e o planejamento de transporte cooperativo (Fischer et al., 1996).

**Gerenciamento de informação** A riqueza, a diversidade e a complexidade da informação disponível para a sociedade atual é quase esmagadora. Sistemas de agentes possibilitam o gerenciamento inteligente da informação, especialmente na Internet. Tanto fatores humanos como a organização da informação parecem conspirar contra o acesso confortável à informação. Duas tarefas críticas de agentes são a filtragem de informações, apenas uma pequena porção da informação a que temos acesso faz o que realmente desejamos, e a coleta de informação, a tarefa de coletar e priorizar partes da informação que realmente queremos. Entre as aplicações estão incluídas WEBMATE (Chen e Sycara, 1998), a filtragem de mensagens eletrônicas (Maes, 1994), um assistente de navegação na Web (Lieberman, 1995) e um agente especialista locador (Kautz et al., 1997). Veja também os serviços de conhecimento (Seção 7.3.3).

**Comércio eletrônico (e-commerce)** Atualmente, o comércio parece ser guiado pela atividade humana: nós é que decidimos quando comprar ou vender, as quantidades e o preço apropriados, até mesmo qual informação pode ser apropriada em um dado instante. Certamente o comércio é um domínio adequado para modelos de agentes. Embora o total desenvolvimento de agentes para comércio eletrônico ainda seja algo para o futuro, muitos sistemas já estão disponíveis. Por exemplo, existem programas que podem tomar várias decisões de compra e venda no mercado de ações baseados em muitas partes distintas de informação distribuída. Vários sistemas de agentes vêm sendo desenvolvidos para gerenciamento de portfólio (Sycara et al., 1996), auxílio em compras (Doorenbos et al., 1997; Krulwich, 1996) e catálogos interativos (Schrooten e van de Velde, 1997; Takahashi et al., 1997).

**Jogos e teatro interativos** Personagens de jogos ou de teatro fornecem um ambiente simulado muito rico. Esses agentes podem nos desafiar em jogos de guerra, cenários de gerenciamento financeiro ou mesmo em esportes. Agentes de teatro desempenham papéis semelhantes aos seus correspondentes humanos e podem oferecer a ilusão de vida para trabalhar com situações emocionais, simulando emergências médicas ou treinando para tarefas diversas. As pesquisas nessa área incluem jogos de computador (Wavish e Graham, 1996) e personalidades interativas (Hayes-Roth, 1995; Trapp e Petta, 1997) e negociação (Fatima et al., 2006).

Há muitos outros domínios, é claro, em que abordagens baseadas em agentes são apropriadas.

Muito embora a tecnologia de agentes ofereça muitas vantagens potenciais para a solução inteligente de problemas, ela ainda tem uma série de desafios a superar. As seguintes questões investigativas são baseadas em ideias de Jennings et al. (1998) e de Bond e Gasser (1988).

Como podemos sistematicamente formalizar, decompor e alocar problemas para agentes? Além disso, como podemos sintetizar apropriadamente os seus resultados?

Como podemos capacitar os agentes a se comunicarem e interagirem? Quais linguagens de comunicação e protocolos estão disponíveis? O que e quando é apropriada a comunicação?

Como podemos assegurar que os agentes ajam coerentemente ao empreender ações ou tomar decisões? Como eles podem tratar efeitos não locais e evitar interações prejudiciais entre agentes?

Como agentes individuais podem representar e raciocinar sobre ações, planos e conhecimento de outros agentes, de modo a se coordenarem com eles? Como agentes podem raciocinar sobre o estado de seus processos coordenados?

Como pontos de vista divergentes e intenções conflitantes entre agentes podem ser reconhecidos e coordenados?

Como um comportamento prejudicial do sistema global, como uma ação caótica ou oscilatória, pode ser reconhecido e evitado?

Como recursos limitados, tanto dos agentes individuais como do sistema como um todo, podem ser alocados e gerenciados?

Por fim, quais são as melhores plataformas de hardware e tecnologias de software para suporte e desenvolvimento de sistemas de agentes?

Ao longo de todo este livro encontram-se habilidades de projeto de software inteligentes necessárias para suportar a tecnologia de solução de problemas por agentes. Em primeiro lugar, os requisitos representacionais para a solução inteligente de problemas constituem um tema constante em nossa apresentação. Em segundo lugar, questões de busca, especialmente da busca heurística, podem ser encontradas na Parte II. Em terceiro lugar, a área de *planejamento*, apresentada na Seção 8.4, fornece uma metodologia para ordenar e coordenar subobjetivos no processo de organizar uma solução de problema. Em quarto lugar, apresentamos, na Seção 9.3, a ideia de raciocínio estocástico de agentes sob condições de incerteza. Finalmente, as questões de aprendizado, raciocínio automatizado e compreensão de linguagem natural são tratadas nas partes IV e V. Essas subáreas da IA tradicional têm seu papel na criação de arquiteturas de agentes.

Existem diversas outras questões de projeto do modelo de agente que vão além do escopo deste livro, por exemplo, as linguagens de comunicação de agentes, esquemas de leilão e técnicas para controle distribuído. Essas questões são abordadas na literatura de agentes (Jennings, 1995; Jennings et al., 1998; Wooldridge, 1998; Wooldridge et al., 2006, 2007; Fatima et al., 2005, 2006) e, em particular, nos anais de conferências apropriadas (AAAI, IJCAI e DAI).

## 7.5 Epílogo e referências

Neste capítulo, examinamos muitas das principais alternativas para representação de conhecimento, incluindo o uso de lógica, regras, redes semânticas e quadros. Consideramos, também, sistemas sem uma base de conhecimento centralizada ou um esquema de raciocínio de propósito geral. Finalmente, consideramos a solução de problemas distribuída com agentes. Como resultado desse estudo cuidadoso, pudemos entender melhor as vantagens e as limitações de cada uma dessas abordagens de representação. Todavia, continua o debate sobre a relativa naturalidade, eficiência e adequação de cada abordagem. Encerramos este capítulo com uma breve discussão de várias questões importantes da área de representação de conhecimento.

A primeira delas é a *seleção e a granularidade de símbolos atómicos* para representar o conhecimento. Os objetos do mundo constituem o domínio do mapeamento; objetos computacionais da base de conhecimento formam o contradomínio. A natureza dos elementos atómicos da linguagem determina enormemente o que pode ser

dito sobre o mundo. Por exemplo, se um “automóvel” for o menor átomo da representação, então o sistema não pode raciocinar acerca de motores, rodas ou qualquer outra parte componente de um automóvel. Entretanto, se os átomos corresponderem a essas partes, então pode ser exigida uma estrutura maior para representar “automóvel” como um conceito único, possivelmente introduzindo um custo em eficiência na manipulação dessa estrutura maior.

Outro exemplo de compromisso na escolha de símbolos vem do trabalho em compreensão de linguagem natural. Programas que usam palavras isoladas como elementos de significado podem ter dificuldade em representar conceitos complexos que não têm uma denotação por uma única palavra. Existem, também, dificuldades em distinguir significados diferentes da mesma palavra ou de palavras diferentes com o mesmo significado. Uma abordagem para esse problema é usar primitivas semânticas, unidades conceituais independentes da linguagem, como base para representar o significado da linguagem natural. Embora esse ponto de vista evite o problema de usar palavras isoladas como unidades de significado, ele envolve outros compromissos: muitas palavras exigem estruturas complexas para sua definição; por outro lado, ao se basear em um conjunto pequeno de primitivas, muitas distinções sutis, como empurrar e impulsionar ou gritar e bradar, são difíceis de expressar.

A *exaustividade* é uma propriedade de uma base de conhecimento que é assistida por uma representação adequada. Um mapeamento é *exaustivo* em relação a uma propriedade ou classe de objetos se todas as ocorrências corresponderem a um elemento explícito da representação. Supõe-se que mapas geográficos sejam exaustivos a um certo nível de detalhe; um mapa em que falte uma cidade ou um rio não seria bem aceito como ferramenta navegacional. Embora a maioria das bases de conhecimento não seja exaustiva, a exaustividade em relação a certas propriedades ou objetos é um objetivo desejável. Por exemplo, a habilidade de supor que uma representação é exaustiva pode permitir a um planejador ignorar possíveis efeitos do *problema de enquadramento*.

Quando descrevemos problemas como um estado do mundo que é modificado por uma série de ações ou eventos, essas ações ou eventos geralmente modificam apenas alguns poucos componentes da descrição; o programa deve ser capaz de inferir efeitos colaterais e mudanças implícitas da descrição do mundo. O problema de representar os efeitos e as ações colaterais é chamado de *problema de enquadramento*. Por exemplo, um robô que empilha caixas pesadas sobre um caminhão deve compensar o abaixamento da carroceria devido ao peso de cada nova caixa. Se uma representação for exaustiva, não haverá efeitos colaterais não especificados e o problema de enquadramento efetivamente desaparecerá. A dificuldade do problema de enquadramento resulta do fato de que é impossível construir uma base de conhecimento completamente exaustiva para a maioria dos domínios. Uma linguagem de representação deveria dar assistência ao programador para decidir qual conhecimento pode ser omitido de forma segura e ajudar a lidar com as consequências dessa omissão. (A Seção 8.4 discute o problema de enquadramento no planejamento.)

A *plasticidade*, ou a capacidade de modificação, da representação está relacionada com a exaustividade: a adição de conhecimento em resposta a deficiências é a solução primária para a falta de exaustividade. Como a maioria das bases de conhecimento não é exaustiva, é mais fácil modificar ou atualizar essas bases. Além de oferecer facilidade sintática para acréscimo de conhecimento, uma representação deveria ajudar a garantir a consistência de uma base de conhecimento ao acrescentar ou retirar informação. Por permitir que propriedades de uma classe sejam herdadas por novas instâncias, a herança é um exemplo de como um esquema representacional pode ajudar a assegurar a consistência.

Vários sistemas, incluindo a *Copycat* (Mitchell, 1993) e o robô de Brooks (ver Seção 7.3), abordaram a questão da plasticidade projetando estruturas em rede que se modificam e evoluem conforme encontram as restrições do mundo natural. Nesses sistemas, a representação é o resultado da aquisição de baixo para cima de novos dados, restrita pela expectativa do sistema perceptivo. Uma aplicação desse tipo de sistema é o raciocínio por analogia.

Outra propriedade útil de representações diz respeito ao grau com que o mapeamento entre o mundo e a base de conhecimento é *homomórfico*. Aqui, homomorfismo implica uma correspondência um para um entre objetos e ações do mundo e os objetos computacionais e as operações da linguagem. Em um mapeamento homomórfico, a base de conhecimento reflete a organização percebida do domínio e pode ser organizada de um modo mais natural e intuitivo.

Além da naturalidade, da objetividade e da facilidade de uso, os esquemas representacionais podem ser avaliados também por sua *eficiência computacional*. Levesque e Brachman (1985) discutem os compromissos entre expressividade e eficiência. A lógica, quando usada como um esquema representacional, é altamente expressiva como resultado de sua plenitude; entretanto, sistemas baseados em lógica irrestrita pagam um preço considerável em eficiência (ver Capítulo 14).

A maioria das questões de representação apresentada até aqui está relacionada a qualquer informação que deva ser capturada e usada por um computador. Existem outras questões que os projetistas de sistemas distribuídos e de agentes devem tratar. Muitas dessas questões estão relacionadas a tomadas de decisão com informação parcial (local), distribuição de responsabilidade para realizar tarefas, linguagens de comunicação entre agentes e desenvolvimentos de algoritmos para a cooperação e para o compartilhamento de informação entre agentes. Muitas dessas questões são apresentadas na Seção 7.4.

Por fim, se quisermos realizar as abordagens filosóficas de uma inteligência distribuída baseada no ambiente, propostas por Clark (1997), Haugeland (1997) e Dennett (1991, 1995, 2006), em que o chamado sistema “vazado” utiliza tanto o ambiente quanto outros agentes como meio crítico para armazenamento e uso de conhecimento, pode ser que tenhamos que esperar por linguagens representacionais inteiramente novas. Onde estão as ferramentas representacionais e o suporte para “usar o mundo como o seu próprio modelo”, como Brooks propõe (1991a)?

Concluímos com outras referências para o material apresentado no Capítulo 7. Teorias associacionistas têm sido estudadas como modelos tanto de memória humana e de computador como de raciocínio (Selz, 1913, 1922; Anderson e Bower, 1973; Sowa, 1984; Collins e Quillian, 1969).

SNePS (Shapiro, 1979; Shapiro et al., 2007), uma das primeiras representações baseadas em associação/lógica (ver Seção 7.1), tem sido estendida recentemente como uma ferramenta para a metacognição, em que os predicados podem tomar outros predicados como argumentos, admitindo assim a capacidade de raciocinar sobre as estruturas de uma base de conhecimento.

Entre os trabalhos importantes em linguagens de representação de conhecimento estruturadas estão: a linguagem de representação KRL (Bobrow e Winograd, 1977), e a linguagem representacional KL-ONE (Brachman, 1979), que dá atenção particular aos fundamentos semânticos das representações estruturadas.

Nossa visão geral dos grafos conceituais se baseia, fundamentalmente, no livro de John Sowa, *Conceptual Structures* (1984). O leitor deverá consultar esse livro para obter detalhes que tenham sido omitidos. No seu tratamento completo, os grafos conceituais combinam o poder expressivo tanto do cálculo de predicados quanto da lógica modal e de ordem mais elevada, com um rico conjunto de conceitos e relações embutidos, derivados da epistemologia, da psicologia e da linguística.

Há uma série de outras abordagens interessantes para o problema de representação. Por exemplo, Brachman, Fikes e Levesque propuseram uma representação que enfatiza especificações *funcionais*, isto é, que informação pode ser consultada a partir de uma base de conhecimento ou informada para ela (Brachman, 1985; Brachman et al., 1985; Levesque, 1984).

Diversos livros podem ajudar com estudos avançados sobre essas questões. *Readings in Knowledge Representation*, de Brachman e Levesque (1985), é uma compilação de artigos importantes nessa área. Muitos dos artigos referenciados neste capítulo podem ser ai encontrados, embora tenham sido referenciados na sua fonte original. *Representation and Understanding*, de Bobrow e Collins (1975); *Representations of Commonsense Knowledge*, de Davis (1990); *Readings in Qualitative Reasoning about Physical Systems*, de Weld e deKleer (1990), são fontes importantes. *Principles of Knowledge Representation and Reasoning* (Brachman et al., 1990), *An Overview of Knowledge Representation* (Mylopoulos e Levesque, 1984) e os anais de qualquer uma das conferências anuais sobre inteligência artificial são fontes úteis.

Há, agora, um número considerável de artigos que seguem as orientações propostas por Brooks com a sua arquitetura de subsunção; veja especialmente Brooks (1991a), Brooks e Stein (1994), Maes (1994) e Veloso et al. (2000). Há também posições contrastantes, como McGonigle (1990, 1998) e Lewis e Luger (2000). Para uma visão filosófica da natureza corporificada e distribuída do conhecimento e da inteligência, consulte *Being There*, de Clark (1997).

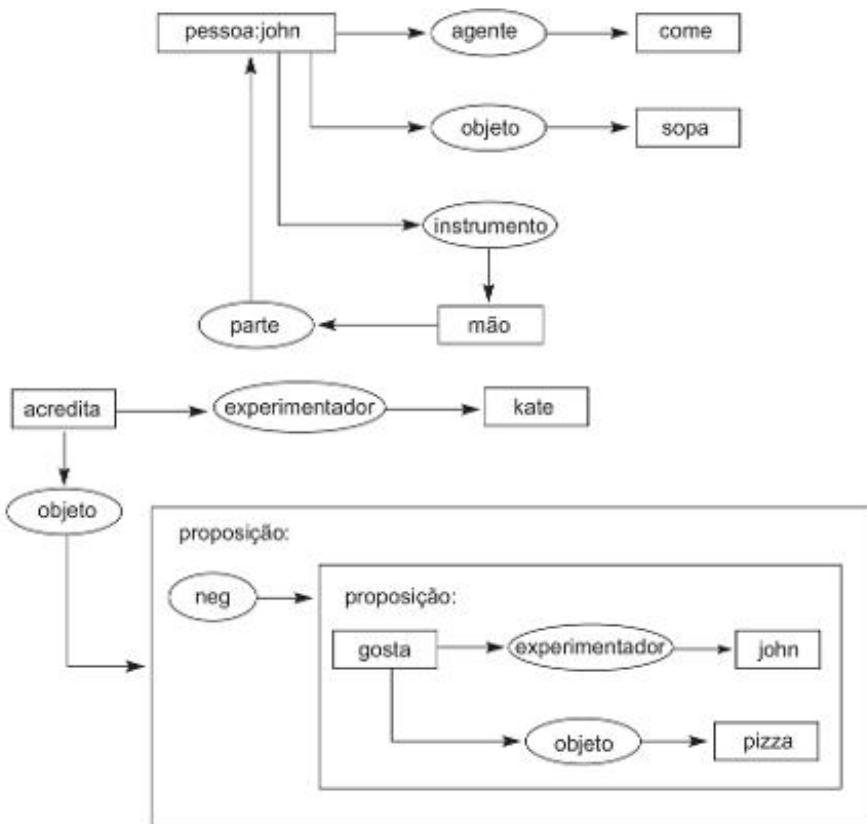
A pesquisa baseada em agentes é bastante ampla na IA moderna. Veja em Jennings et al. (1998) uma introdução a essa área. Há, agora, seções inteiras nas conferências anuais de IA (IAAI e IJCAI) dedicadas à pesquisa sobre agentes. Recomendamos a leitura dos anais recentes dessas conferências para obter discussões mais atualizadas das questões de pesquisa atuais. Recomendamos, também, os anais de conferências e leituras na área de inteligência artificial distribuída, ou IAD. Na Seção 7.4, foram apresentados um resumo e referências de importantes áreas de aplicação para a pesquisa baseada em agentes.

Questões envolvendo representação de conhecimento estão, também, no terreno entre IA e ciências cognitivas; veja *Cognitive Science: The Science of Intelligent Systems* (Luger, 1994) e *Being There* (Clark, 1997). *Computation and Intelligence*, editado por George Luger (1995), é uma coletânea de artigos clássicos que enfatiza o desenvolvimento de muitos esquemas diferentes de representação de conhecimento.

## 7.6 Exercícios

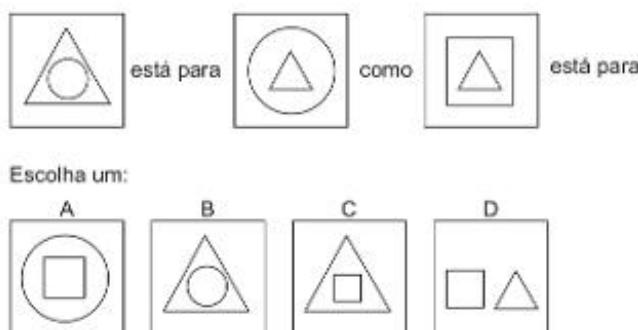
- O raciocínio de senso comum emprega noções como causalidade, analogia e equivalência, mas as usa de uma forma diferente das linguagens formais. Por exemplo, se dissermos “A inflação motivou Jane a pedir um aumento”, estamos sugerindo um relacionamento causal mais complicado que o encontrado nas leis físicas simples. Se dissermos “Use uma faca ou um formão para desbastar a madeira”, estamos sugerindo uma noção importante de equivalência. Discuta os problemas de traduzir esses e outros conceitos em uma linguagem formal.
- Na Seção 7.2.1, apresentamos alguns dos argumentos contra o uso de lógica para representar o raciocínio de senso comum. Formule um argumento a favor do uso de lógica para representar esse conhecimento. McCarthy e Hayes (1969) têm uma discussão interessante sobre isso.
- Traduza cada uma das sentenças a seguir em cálculo de predicados, dependências conceituais e grafos conceituais:
  - “Jane deu uma casquinha de sorvete para Tom.”
  - “Jogadores de basquete são altos.”
  - “Paul derrubou a árvore com um machado.”
  - “Coloque todos os ingredientes em uma tigela e misture bem.”
- Leia “What's in a Link”, de Woods (1985). A Seção IV desse artigo lista uma série de problemas em representação de conhecimento. Sugira uma solução para cada um desses problemas usando as notações de lógica, grafos conceituais e quadros.
- Traduza os grafos conceituais da Figura 7.28 para sentenças em português.
- As operações *juntar* e *restringir* definem uma ordem de generalização nos grafos conceituais. Mostre que a relação de generalização é transitiva.

**Figura 7.28** Dois grafos conceituais para serem traduzidos para o português.



7. A especialização dos grafos conceituais usando *juntar* e *restringir* não é uma operação que preserva a verdade. Dê um exemplo que demonstre que a restrição de um grafo verdadeiro não é necessariamente verdadeira. Entretanto, a generalização de um grafo verdadeiro é sempre verdadeira; prove essa afirmação.
8. Defina uma linguagem de representação especializada para descrever as atividades de uma biblioteca pública. Essa linguagem será um conjunto de conceitos e relações usando grafos conceituais. Faça o mesmo para um negócio de varejo. Que conceitos e relações essas duas linguagens teriam em comum? Que conceitos e relações existiriam em ambas as linguagens, mas com significados diferentes?
9. Traduza os grafos conceituais da Figura 7.28 para o cálculo de predicados.
10. Traduza a base de conhecimento para o consultor financeiro, Seção 2.4, na forma de grafo conceitual.
11. Dê uma evidência de sua própria experiência que sugira uma organização da memória humana parecida com roteiros ou quadros.
12. Usando dependências conceituais, defina um roteiro para:
  - a. uma lanchonete.
  - b. interagir com um vendedor de carros usados.
  - c. ir a uma ópera.
13. Construa uma hierarquia de subtipos para o conceito veículo; por exemplo, subtipos de veículo poderiam ser veículo\_terrestre ou veículo\_marítimo. Estes, por sua vez, teriam outros subtipos. Isso seria melhor representado como uma árvore, um reticulado ou um grafo genérico? Faça o mesmo para o conceito mover e para o conceito zangado.
14. Construa uma hierarquia de tipos na qual alguns tipos não tenham um supertipo comum. Acrescente tipos para torná-la um reticulado. Essa hierarquia poderia ser expressa usando a herança de árvore? Quais seriam os problemas que surgiriam ao se fazer isso?
15. Cada uma das sequências de caracteres a seguir foi gerada de acordo com uma regra geral. Descreva uma representação que poderia ser usada para representar as regras e as relações necessárias para continuar cada sequência:
  - a. 2, 4, 6, 8, ...
  - b. 1, 2, 4, 8, 16, ...
  - c. 1, 1, 2, 3, 5, 8, ...
  - d. 1, a, 2, c, 3, f, 4, ...
  - e. o, t, t, f, f, s, s, ...
16. Na Seção 7.3.2 foram apresentados exemplos de raciocínio por analogia. Descreva uma representação apropriada e uma estratégia de busca que permitiria a identificação da melhor resposta em cada situação. Crie mais dois exemplos de analogias que funcionariam com a representação que você propôs. Os dois exemplos a seguir poderiam ser resolvidos?
  - a. quente está para frio assim como alto está para {parede, baixo, molhado, rodapé}.
  - b. urso está para porco assim como cadeira está para {pé, mesa, café, morango}.
17. Descreva uma representação que poderia ser usada em um programa para resolver problemas de analogia como aquele da Figura 7.29. Essa classe de problemas foi tratada por T. G. Evans (1968). A representação deve ser capaz de representar as características essenciais de tamanho, forma e posição relativa.

**Figura 7.29** Exemplo de um problema de teste de analogia.



18. O artigo de Brooks (1991a) oferece uma discussão importante sobre o papel da representação na IA tradicional. Leia esse artigo e comente as limitações de esquemas representacionais explícitos de propósito geral.
19. No final da Seção 7.3.1, há cinco questões potenciais que a arquitetura de subsunção de Brooks (1991a) deve tratar para oferecer uma abordagem de propósito geral bem-sucedida para a solução de problemas. Escolha uma ou mais dessas questões e comente.
20. Identifique cinco propriedades que uma linguagem de agentes deveria ter para implementar um serviço de Internet orientado a agentes. Comente o papel da linguagem Java como uma linguagem de agentes de propósito geral para construir serviços de Internet. Você vê um papel similar para a linguagem CLOS? Por quê? Há muita informação sobre esse tópico na própria Internet.
21. No final da Seção 7.4, é apresentada uma série de questões importantes relativa à criação de uma solução de problemas orientada a agentes. Escolha uma delas e discuta a questão.
22. Suponha que você esteja projetando um sistema de agentes para representar um time de futebol. Para que os agentes cooperem em uma manobra defensiva ou para marcar um gol, eles devem ter alguma ideia dos planos dos outros agentes e as respostas possíveis às situações. Como você construiria um modelo de metas e planos de um outro agente cooperativo?
23. Escolha uma das áreas de aplicação para arquiteturas de agentes resumidas na Seção 7.4. Escolha um artigo de pesquisa ou uma aplicação nessa área. Projete uma organização de agentes que possa tratar o problema. Divida o problema para especificar questões de responsabilidade de cada agente. Liste os procedimentos de cooperação apropriados.

# Solução de problemas por método forte

*O primeiro princípio da engenharia de conhecimento é que o poder de solução de problemas exibido por um agente inteligente é, fundamentalmente, a consequência de sua base de conhecimento, e, apenas de forma secundária, uma consequência do método de inferência empregado. Os sistemas especialistas precisam ser ricos em conhecimento se forem pobres em método. Esse é um resultado importante e que apenas recentemente passou a ser bem entendido em IA. Por um longo período de tempo, a IA focou sua atenção quase que exclusivamente no desenvolvimento de métodos de inferência engenhosos; quase todos os métodos de inferência são adequados. O poder reside no conhecimento.*

—EDWARD FEIGENBAUM, Universidade de Stanford

*Nam et ipsa scientia potestas est (conhecimento é poder).*

—FRANCIS BACON (1620)

## 8.0 Introdução

Continuaremos a estudar questões de representação e inteligência considerando um componente importante da IA: a solução de problemas *utilizando intensivamente o conhecimento ou o método forte*.

Os especialistas humanos são capazes de obter desempenhos de alto nível, porque eles conhecem muito suas áreas de especialidade. Essa observação simples é a razão subjacente do projeto de resolvedores de problemas por método forte ou baseado em conhecimento (ver introdução da Parte III). Um *sistema especialista*, por exemplo, usa o conhecimento específico de um domínio de problema para conseguir um desempenho com “qualidade de especialista” naquela área de aplicação. De modo geral, os projetistas de sistemas especialistas adquirem esse conhecimento com a ajuda de peritos humanos no domínio, e o sistema emula a metodologia e a atuação do especialista humano. Os sistemas especialistas, assim como as pessoas experientes, tendem a ser especializados e focam um conjunto reduzido de problemas. Assim como também ocorre com as pessoas, o seu conhecimento é tanto teórico como prático: o especialista humano que fornece o conhecimento do sistema geralmente ampliou o seu próprio entendimento teórico do domínio do problema com artifícios, atalhos e heurísticas para *usar* o conhecimento que ganhou por meio de experiência em resolução de problemas.

Por causa da sua natureza heurística, intensiva em conhecimento, os sistemas especialistas geralmente:

1. Suportam inspeção de seus processos de raciocínio, apresentando os passos intermediários e respondendo a questões sobre o processo de solução.
2. Permitem a fácil modificação na adição ou exclusão de habilidades da base de conhecimento.
3. Raciocinam heuristicamente usando conhecimento (frequentemente imperfeito) para obter soluções úteis.

O raciocínio de um sistema especialista deve ser aberto para inspeção, fornecendo informações sobre o estado de sua solução para o problema e explicações sobre as escolhas e decisões que o programa faz. As expli-

cações são importantes para que um perito humano, como um médico ou um engenheiro, aceite as recomendações feitas por um computador. Na verdade, poucos especialistas humanos aceitariam conselhos de outra pessoa, quanto mais de um computador, sem compreender as justificativas para isso.

A natureza exploratória da IA e da programação de sistemas especialistas exige que os programas sejam facilmente prototipados, testados e modificados. As linguagens de programação e os ambientes de IA são projetados para dar suporte a essa metodologia iterativa de desenvolvimento. Em um sistema de produção puro, por exemplo, a modificação de uma única regra não tem efeitos sintáticos colaterais globais. Podemos adicionar ou remover regras sem a necessidade de outras modificações do programa maior. Os projetistas de sistemas especialistas frequentemente comentam que a facilidade de modificação da base de conhecimento é um fator fundamental na produção de um programa bem-sucedido.

Outra característica dos sistemas especialistas é o uso de métodos heurísticos para a solução de problemas. Os projetistas de sistemas especialistas descobriram que “truques do negócio” e “regras práticas” informais são um complemento essencial à teoria-padrão apresentada nos livros-texto e nos cursos. Algumas vezes, essas regras ampliam o conhecimento teórico de modo compreensível; outras vezes, são simplesmente atalhos que empiricamente mostraram que funcionam.

Os sistemas especialistas são construídos para resolver uma grande variedade de problemas em domínios como medicina, matemática, engenharia, química, geologia, ciência da computação, economia, direito, defesa e educação. Esses programas tratam de uma série de problemas; a lista a seguir, de Waterman (1986), é um resumo útil das categorias gerais de problemas para sistemas especialistas.

*Interpretação* — formar conclusões de alto nível de coleções de dados brutos.

*Predição* — projetar consequências prováveis de situações indicadas.

*Diagnóstico* — determinar a causa de mau funcionamento em situações complexas com base em sintomas observáveis.

*Projeto* — encontrar uma configuração de componentes do sistema que alcance os objetivos de desempenho e, simultaneamente, satisfaça um conjunto de restrições de projeto.

*Planejamento* — estabelecer uma sequência de ações que alcançarão um conjunto de objetivos, dadas certas condições iniciais e restrições em tempo de execução.

*Monitoramento* — comparar o comportamento observado de um sistema com o seu comportamento esperado.

*InSTRUÇÃO* — dar assistência ao processo de educação em domínios técnicos.

*Controle* — governar o comportamento de um ambiente complexo.

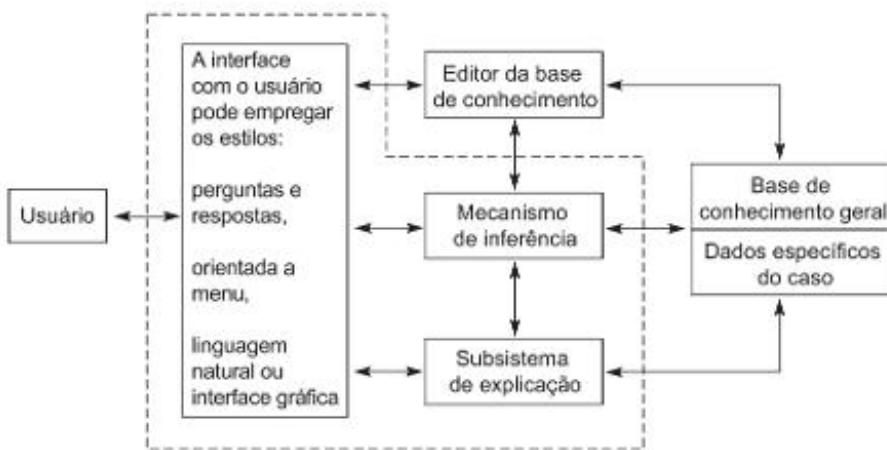
Neste capítulo, examinamos, primeiro, a tecnologia que torna possível a solução de problemas baseada em conhecimento. A *engenharia do conhecimento* bem-sucedida deve tratar de uma série de problemas, desde a escolha de um domínio de aplicação adequado até a aquisição e a formalização do conhecimento para a solução do problema. Na Seção 8.2, introduzimos os sistemas baseados em regras e apresentamos o sistema de produção como uma arquitetura de software para os processos de solução e explicação. A Seção 8.3 examina técnicas para o raciocínio baseado em modelos e em casos. A Seção 8.4 apresenta o *planejamento*, um processo de organizar partes do conhecimento em uma sequência consistente de ações que alcançarão um objetivo. O Capítulo 9 apresenta técnicas para raciocínio em situações de incerteza, um componente importante para os resolvedores de problemas por método forte.

## 8.1 Visão geral da tecnologia de sistemas especialistas

### 8.1.1 Projeto de sistemas especialistas baseados em regras

A Figura 8.1 mostra os módulos que compõem um sistema especialista típico. O usuário interage com o sistema por meio de uma *interface com o usuário*, que simplifica a comunicação e oculta boa parte da complexidade,

**Figura 8.1** Arquitetura de um sistema especialista típico para um domínio de problema particular.



como a estrutura interna da base de regras. As interfaces dos sistemas especialistas empregam uma série de estilos de usuários, como as interfaces do tipo pergunta e resposta, orientadas a menu ou gráficas. A decisão final sobre o tipo de interface é um compromisso entre as necessidades do usuário e os requisitos da base de conhecimento e do sistema de inferência.

O coração do sistema especialista é a *base de conhecimento*, que contém o conhecimento de um domínio particular de aplicação. Em um sistema especialista baseado em regras, esse conhecimento é representado em forma de regras *se... então...*, como nos nossos exemplos da Seção 8.2. A base de conhecimento contém tanto o *conhecimento geral* como a informação *específica do caso*.

O *mecanismo de inferência* aplica o conhecimento à solução de problemas reais. Ele é essencialmente um interpretador para a base de conhecimento. No sistema de produção, o mecanismo de inferência executa o ciclo de controle reconhece-atua. Os procedimentos que implementam o ciclo de controle são separados das regras de produção propriamente ditas. É importante manter essa separação entre a base de conhecimento e o mecanismo de inferência por várias razões:

1. Essa separação torna possível representar o conhecimento em uma forma mais natural. As regras *se... então...*, por exemplo, são mais próximas do modo como as pessoas descrevem as suas habilidade para resolver problemas do que o código de computador em baixo nível.
2. Como a base de conhecimento está separada das estruturas de controle de nível mais baixo do programa, os construtores de sistemas especialistas podem se concentrar em capturar e organizar o conhecimento para a solução do problema em vez de se preocupar com os detalhes da sua implementação no computador.
3. Idealmente, a separação entre conhecimento e controle permite que sejam feitas modificações em uma parte da base de conhecimento sem criar efeitos colaterais em outras partes.
4. A separação entre elementos de conhecimento e de controle do programa permite que o mesmo software de controle e de interface seja usado em diversos sistemas. O *ambiente de sistemas especialistas* tem todos os componentes da Figura 8.1, exceto que a base de conhecimento e os dados específicos do caso estão vazios e podem ser acrescentados em uma nova aplicação. As linhas tracejadas da Figura 8.1 indicam os módulos do ambiente.

O sistema especialista deve manter as informações sobre os *dados específicos do caso*: os fatos, as conclusões e outras informações relevantes ao caso em consideração. Ai se incluem os dados fornecidos em uma instância do problema, as conclusões parciais, as medidas de confiança das conclusões e os becos sem saída encontrados no processo de busca. Essa informação é separada da base de conhecimento geral.

O *subsistema de explicação* permite que o programa explique seu raciocínio ao usuário. Essas explicações incluem justificativas para as conclusões do sistema em resposta a *consultas do tipo como* (Seção 8.2), explicações

sobre as razões pelas quais o sistema necessita de um dado em particular, *consultas do tipo por que* (Seção 8.2) e, se for útil, explicações didáticas ou justificativas teóricas mais aprofundadas sobre as ações do programa.

Muitos sistemas incluem, também, um *editor da base de conhecimento*, que ajuda o programador a localizar e corrigir erros na execução do programa, frequentemente acessando a informação fornecida pelo subsistema de explicação. Além disso, pode ajudar na adição de conhecimento novo, a manter correta a sintaxe das regras e a realizar verificações de consistência sobre a base de conhecimento atualizada.

Uma razão importante para o declínio nos tempos de projeto e de entrada dos sistemas especialistas atuais é a disponibilidade dos *ambientes de sistemas especialistas*. A NASA criou CLIPS, JESS está disponível na Sandia National Laboratories, e oferecemos ambientes em Lisp e Prolog na Sala Virtual deste livro. Infelizmente, programas de ambiente não resolvem todos os problemas envolvidos na construção de sistemas especialistas. Apesar da separação entre conhecimento e controle, a modularidade da arquitetura do sistema de produção e o uso de uma linguagem de representação de conhecimento adequada ajudam na construção de um sistema especialista, a aquisição e a formalização do conhecimento do domínio ainda continuam a ser tarefas muito difíceis.

### 8.1.2 Seleção de um problema e o processo de engenharia do conhecimento

Os sistemas especialistas envolvem um investimento considerável de dinheiro e de esforço humano. O esforço para resolver um problema que é complexo demais, muito pouco compreendido, ou ainda inadequado para a tecnologia disponível, pode levar a fracassos desagradáveis e muito custosos. Pesquisadores desenvolveram diretrizes para determinar se um problema é apropriado para a solução por um sistema especialista:

- 1. A necessidade de uma solução justifica o custo e o esforço de construir um sistema especialista.** Muitos sistemas especialistas foram construídos em domínios como a exploração mineral, negócios, defesa e medicina, em que existe um grande potencial para a economia de tempo, dinheiro e vidas humanas.
- 2. A perícia humana não está disponível em todas as situações em que ela é necessária.** Na geologia, por exemplo, há necessidade de peritos em locais remotos de mineração e perfuração. Frequentemente, geólogos e engenheiros devem se deslocar por grandes distâncias para visitar esses locais, e isso resulta em despesas e perda de tempo. Por meio do emprego de sistemas especialistas, muitos problemas podem ser resolvidos sem a necessidade de visitas.
- 3. O problema pode ser resolvido usando o raciocínio simbólico.** As soluções de problemas não devem requerer destreza ou habilidade perceptiva. Aos robôs e sistemas de visão atuais faltam a sofisticação e a flexibilidade dos seres humanos.
- 4. O domínio do problema é bem estruturado e não requer raciocínio de senso comum.** Campos altamente técnicos têm a vantagem de ser bem estudados e formalizados: os termos são bem definidos e os domínios têm modelos conceituais claros e específicos. O raciocínio de senso comum, por outro lado, é difícil de automatizar.
- 5. O problema não pode ser resolvido usando métodos de computação tradicionais.** A tecnologia de sistemas especialistas não deve ser usada onde não for necessária. Se um problema puder ser solucionado satisfatoriamente usando técnicas mais tradicionais, então ela não deve ser usada.
- 6. Existem especialistas cooperativos e articulados.** O conhecimento usado por sistemas especialistas vem da experiência e do julgamento de seres humanos que trabalham em um domínio. É importante que esses especialistas tenham boa vontade e capacidade para compartilhar o seu conhecimento.
- 7. O problema é do tamanho e do escopo adequados.** Um programa que, por exemplo, tenta captar toda a experiência de um médico não é viável; um programa que aconselha médicos no uso de um equipamento de diagnóstico em particular, ou de um conjunto específico de diagnósticos, é mais apropriado.

As principais pessoas envolvidas na construção de um sistema especialista são o *engenheiro de conhecimento*, o *especialista no domínio* e o *usuário final*. O engenheiro de conhecimento é o especialista em linguagens de IA e em representação. Sua tarefa principal é selecionar as ferramentas de software e hardware para o projeto, ajudar o especialista no domínio a articular o conhecimento necessário e implementar esse conhecimento em uma base de conhecimento correta e eficiente. No inicio, frequentemente, o engenheiro de conhecimento desconhece o domínio de aplicação.

O especialista no domínio fornece o conhecimento na área do problema. Ele geralmente é alguém que trabalhou na área do domínio e comprehende as técnicas de solução de problemas, como atalhos, tratamento de dados imprecisos, avaliação de soluções parciais e todas as outras habilidades que caracterizam um especialista na solução do problema. O especialista no domínio é fundamentalmente responsável por expressar essas habilidades para o engenheiro de conhecimento.

Como na maioria das aplicações, é o usuário final quem determina as principais restrições de projeto. Se o usuário não ficar satisfeito, o esforço de desenvolvimento será desperdiçado. As habilidades e necessidades do usuário devem ser consideradas ao longo de todo o ciclo de projeto: O programa fará com que o trabalho do usuário seja mais fácil, mais rápido ou mais confortável? Qual o nível de explicação que o usuário necessita? O usuário pode fornecer informações corretas ao sistema? A interface é apropriada? O ambiente de trabalho do usuário coloca restrições ao uso do programa? Uma interface que exija digitação, por exemplo, não seria apropriada para uma cabine de um avião caça.

Assim como a maior parte da programação em IA, a construção de sistemas especialistas requer um ciclo de desenvolvimento não tradicional baseado em prototipação precoce e em revisão incremental de código. Geralmente, o trabalho no sistema começa com a tentativa do engenheiro de conhecimento de ganhar um pouco de familiaridade com o domínio do problema. Isso ajuda na comunicação com o especialista no domínio. Isso é feito em entrevistas iniciais com o especialista e pela observação de especialistas durante a execução de seu trabalho. A seguir, o engenheiro de conhecimento e o especialista iniciam o processo de extração do conhecimento do especialista em resolver problemas. Normalmente, isso é feito dando ao especialista no domínio uma série de amostras de problemas, deixando-o explicar as técnicas usadas na sua solução. Nesse processo, trechos de vídeo e/ou áudio quase sempre são essenciais.

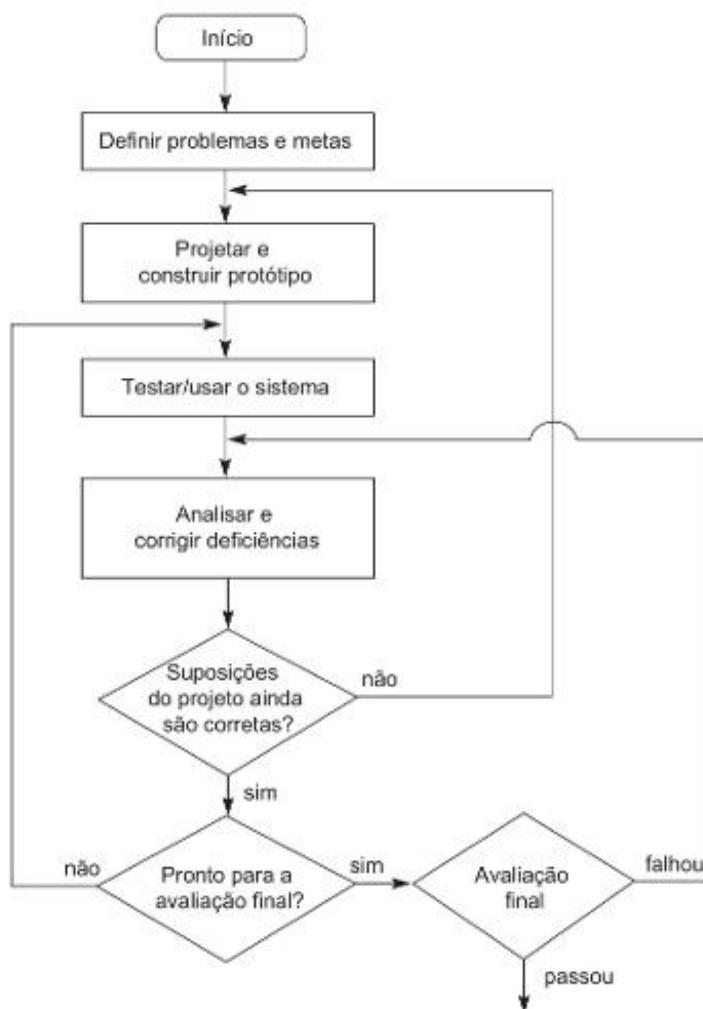
Normalmente é útil que o engenheiro de conhecimento seja um novato no domínio do problema. Os especialistas notoriamente não são confiáveis ao explanar exatamente o que acontece na solução de um problema complexo. Frequentemente, eles se esquecem de mencionar passos que se tornaram óbvios ou mesmo automáticos para eles após anos de trabalho na área. Os engenheiros de conhecimento, em virtude de sua relativa ingenuidade no domínio, podem localizar esses saltos conceituais e pedir ajuda.

Uma vez que o engenheiro de conhecimento tenha obtido uma visão geral do domínio do problema e tenha participado de várias sessões de solução de problemas com o especialista, ele estará pronto para começar o projeto real do sistema: selecionar uma maneira de representar o conhecimento como regras ou quadros, determinar a estratégia de busca, para a frente, para trás, em profundidade, melhor escolha etc. e projetar a interface com o usuário. Após estabelecer esses compromissos de projeto, o engenheiro de conhecimento constrói um protótipo.

Esse protótipo deve ser capaz de resolver problemas em uma pequena área do domínio e deve servir de bancada de testes para as suposições preliminares do projeto. Uma vez o protótipo implementado, o engenheiro de conhecimento e o especialista no domínio testam e refinam seu conhecimento, fornecendo problemas para ele solucionar e corrigindo seus defeitos. Caso as suposições feitas no projeto do protótipo se mostrem corretas, o protótipo pode ser estendido de forma incremental até que se torne um sistema final.

Os sistemas especialistas são construídos por aproximações sucessivas, com os erros do programa levando a correções ou acréscimos à base de conhecimento. Nesse sentido, a base de conhecimento é “acrescida” em vez de construída. A Figura 8.2 apresenta um fluxograma que descreve o ciclo de desenvolvimento exploratório de programação. Essa abordagem para a programação foi investigada por Seymour Papert com sua linguagem LOGO (Papert, 1980), bem como pelo trabalho de Alan Kay com Smalltalk na Xerox PARC. A filosofia de LOGO argumenta que observar o computador responder a ideias formuladas impropriamente representadas pelo código leva à correção (ser *debugado*) e ao esclarecimento, com um código mais preciso. Esse processo de tentativa e de correção

**Figura 8.2** Ciclo de desenvolvimento exploratório.



de projetos candidatos é comum ao desenvolvimento de sistemas especialistas e contrasta com processos perfeitamente hierárquicos, como o projeto de cima para baixo.

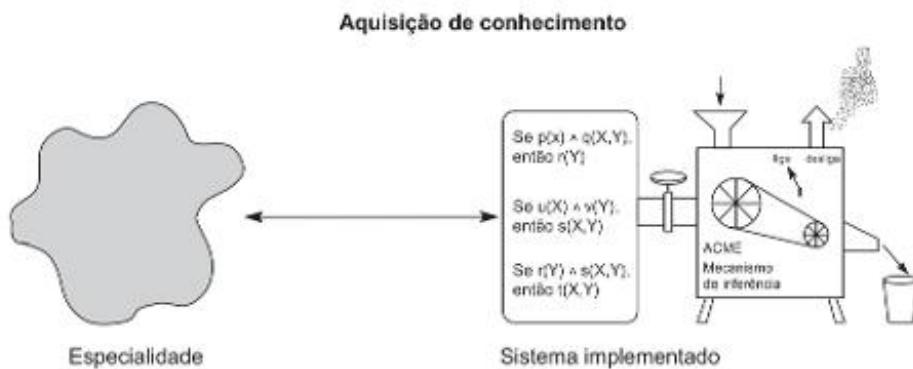
Entende-se também que o protótipo pode ser descartado se ele ficar muito sobrecarregado ou se os projetistas decidirem trocar a sua abordagem básica para o problema. Por meio do protótipo, os construtores podem explorar o problema e os seus relacionamentos importantes construindo realmente um programa para resolvê-lo. Uma vez que esses esclarecimentos estiverem completos, os projetistas poderão escrever uma versão mais limpa do programa, usualmente com menos regras.

A segunda característica importante da programação de sistemas especialistas é que o programa nunca precisa ser considerado “acabado”. Uma grande base de conhecimento heurístico sempre terá limitações. A regularidade do sistema de produção faz com que seja natural, a qualquer momento, acrescentar regras novas ou fazer ajustes devido a deficiências da base de regras atual.

### 8.1.3 Modelos conceituais e seu papel na aquisição de conhecimento

A Figura 8.3 apresenta um modelo simplificado do processo de aquisição de conhecimento que servirá como uma “primeira aproximação” da compreensão dos problemas envolvidos na aquisição e na formalização da atua-

**Figura 8.3** Visão-padrão da construção de um sistema especialista.



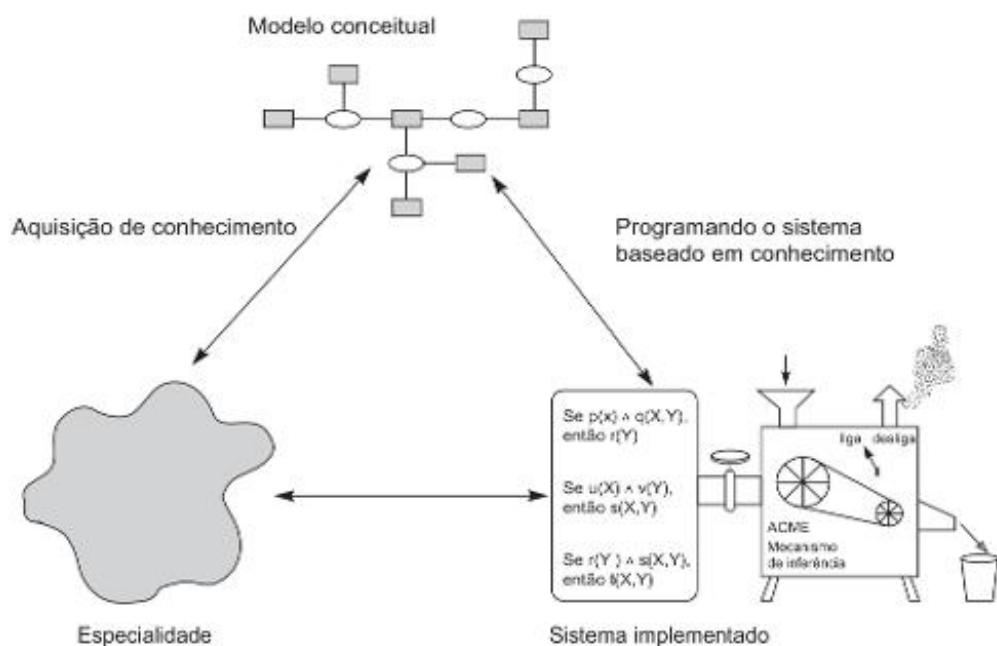
ção humana. O especialista humano, ao trabalhar em uma área de aplicação, opera em um domínio de conhecimento, de habilidades e prática. Esse conhecimento é, muitas vezes, vago, impreciso e apenas parcialmente verbalizado. O engenheiro do conhecimento deve traduzir essa especialidade informal em uma linguagem formal adequada para um sistema de computação. No processo de formalizar a atuação do especialista humano, surge uma série de questões importantes:

1. A habilidade humana é, muitas vezes, inacessível à mente consciente. Como Aristóteles observou em sua obra *Ethics*, “o que precisamos aprender a fazer, aprendemos fazendo”. As habilidades que um médico possui, por exemplo, são aprendidas tanto em vários anos de residência, com o foco constante nos pacientes, quanto em aulas de fisiologia, em que a ênfase está em experimentos e na teoria. O fornecimento de cuidados médicos é, em grande parte, orientado pela prática. Após anos de atuação, essas habilidades se tornam altamente integradas e funcionam em um nível largamente inconsciente. Pode ser difícil para especialistas descrever exatamente o que fazem ao resolver um problema.
  2. A especialidade humana muitas vezes assume a forma de saber *como* agir em uma situação, em vez de saber *o que* poderia ser uma caracterização racional da situação, de desenvolver mecanismos para atuação de perícia, em vez de compreender o que são esses mecanismos. Um exemplo óbvio disso é andar em um monociclo: andar com sucesso, em tempo real, não significa resolver conscientemente múltiplos conjuntos de equações diferenciais simultâneas para manter o equilíbrio; em vez disso, usamos uma combinação intuitiva de sentimentos de “gravidade”, “momento” e “inércia” para formar um procedimento de controle útil.
  3. Muitas vezes, vemos a aquisição de conhecimento como um ganho de conhecimento factual de uma realidade objetiva, chamada de “mundo real”. Como mostram tanto a teoria quanto a prática, a especialidade humana representa um *modelo* do mundo de um indivíduo ou de uma comunidade. Tais modelos são influenciados tanto por convenção, processos sociais e planos ocultos quanto por metodologias empíricas.
  4. A especialidade se modifica. Não apenas os especialistas ganham novos conhecimentos, mas também o conhecimento existente pode estar sujeito a reformulações radicais, como evidenciam as controvérsias atuais nos campos científico e social.

Consequentemente, a engenharia do conhecimento é difícil e deveria ser vista como algo que abrange o ciclo de vida de qualquer sistema especialista. Para simplificar essa tarefa, é útil considerar, como vemos na Figura 8.4, um *modelo conceitual* que se encontra entre a especialidade humana e o programa implementado. Por modelo conceitual entendemos a concepção evolutiva do engenheiro de conhecimento sobre o conhecimento do domínio. Embora, sem dúvida, ele seja diferente do modelo do especialista do domínio, é esse modelo que realmente determina a construção da base de conhecimento formal.

Por causa da complexidade da maioria dos problemas interessantes, não podemos considerar esse estágio intermediário como garantido. Os engenheiros do conhecimento devem documentar e tornar públicas as suas suposições sobre o domínio por meio de metodologias comuns de engenharia de software. Um sistema especialista deve

**Figura 8.4** O papel de modelos mentais ou conceituais na solução de problemas.



incluir um documento de requisitos; entretanto, por causa das restrições da programação exploratória, os requisitos do sistema especialista devem ser tratados como coevolutivos com o protótipo. Dicionários de dados, representações gráficas de espaços de estados e comentários no próprio código fazem parte desse modelo. A publicação dessas decisões de projeto nos ajuda a reduzir erros tanto de implementação quanto de manutenção do programa.

Os engenheiros do conhecimento devem guardar registros de entrevistas com especialistas do domínio. Muitas vezes, à medida que cresce a compreensão do domínio pelo engenheiro do conhecimento, ele forma novas interpretações ou descobre novas informações sobre o domínio. Os registros e a documentação da interpretação dada a eles têm um papel importante na revisão das decisões de projeto e no teste de protótipos. Por fim, esse modelo tem um papel intermediário na formalização do conhecimento. A escolha de uma linguagem de representação exerce uma forte influência sobre o modelo do domínio do engenheiro de conhecimento.

O modelo conceitual não é formal ou diretamente executável em um computador. Ele é uma construção de projeto intermediária, um molde para começar a restringir e codificar a habilidade humana. Se o engenheiro do conhecimento usar um modelo de cálculo de predicados, ele pode começar com um número de redes simples representando estados de raciocínio por meio de situações típicas da solução do problema. Só depois de outros refinamentos essa rede se torna um conjunto de regras explícitas *se... então...*

Entre as questões frequentes no contexto de um modelo conceitual se incluem: A solução para o problema é determinística ou baseada em busca? O raciocínio é guiado por dados, talvez no estilo de geração e teste, ou guiado por objetivos, baseado em um pequeno conjunto de hipóteses prováveis sobre situações? O raciocínio é realizado em estágios? O domínio é bem entendido e capaz de fornecer modelos preditivos aprofundados ou todo o conhecimento para a solução do problema é basicamente heurístico? Podemos usar exemplos de problemas passados e suas soluções para resolver diretamente problemas futuros ou precisamos primeiro converter esses exemplos em regras mais gerais? O conhecimento é exato ou ele é “nebuloso” e aproximado, prestando-se a avaliações numéricas de certeza (Capítulo 9)? Nossas estratégias de raciocínio nos permitirão deduzir fatos estáveis sobre o domínio ou mudanças e incertezas dentro do sistema exigem raciocínio não monotônico, que permite que assentimentos acerca do domínio sejam mais tarde modificados ou revogados (Seção 9.1)? Por fim, a estrutura do conhecimento do domínio requer que abandonemos a inferência baseada em regras, privilegiando esquemas alternativos como redes neurais ou algoritmos genéticos (Parte IV)?

Deve-se, também, abordar as necessidades eventuais do usuário no contexto do modelo conceitual: Quais são as suas expectativas sobre o programa? Qual é o seu nível de especialidade: novato, intermediário ou especialista? Quais são os níveis de explicação apropriados? Qual interface serve melhor às suas necessidades?

Com base nas respostas a essas e outras questões relacionadas, no conhecimento obtido de especialistas do domínio e no modelo conceitual resultante, podemos iniciar o desenvolvimento do sistema especialista. Como o sistema de produção, apresentado inicialmente no Capítulo 6, oferece uma série de vantagens inerentes para organizar e aplicar o conhecimento, quase sempre ele é usado como base para a representação em sistemas especialistas baseados em regras.

## 8.2 Sistemas especialistas baseados em regras

Sistemas especialistas baseados em regras representam o conhecimento para resolver o problema como regras *se... então...*. Essa abordagem é apropriada para a arquitetura da Figura 8.1 e é uma das técnicas mais antigas para representar o conhecimento do domínio em um sistema especialista. Ela é também uma das mais naturais e permanece sendo amplamente usada em sistemas especialistas práticos e experimentais.

### 8.2.1 Sistema de produção e a solução de problemas guiada por objetivos

A arquitetura dos sistemas especialistas baseados em regras pode ser entendida em termos do modelo de sistema de produção para a solução de problemas apresentado na Parte II. O paralelo entre os dois é mais que uma simples analogia: o sistema de produção foi o precursor intelectual da arquitetura moderna dos sistemas especialistas, onde a aplicação de regras de produção leva a refinamentos da compreensão de uma situação de problema particular. Quando Newell e Simon desenvolveram o sistema de produção, o seu objetivo era modelar o desempenho humano na solução de problemas.

Se considerarmos a arquitetura de sistema especialista da Figura 8.1 como um sistema de produção, a base de conhecimento específica do domínio é o conjunto de regras de produção. Em um sistema baseado em regras, esses pares condição-ação são representados como regras *se... então...*, com as premissas das regras, a parte *se*, correspondendo à condição, e a conclusão, a parte *então*, correspondendo à ação: quando a condição é satisfeita, o sistema especialista executa a ação de declarar a conclusão verdadeira. Os dados específicos do caso podem ser mantidos na memória de trabalho. O mecanismo de inferência implementa o ciclo reconhece-atua do sistema de produção; esse controle pode ser tanto guiado por dados como por objetivo.

Muitos domínios de problemas parecem ser naturalmente adequados para a busca para a frente. Em um problema de interpretação, por exemplo, a maior parte dos dados para o problema é fornecida no início e, muitas vezes, é difícil formular uma hipótese ou objetivo. Isso sugere um processo de raciocínio progressivo no qual os fatos são colocados na memória de trabalho e o sistema busca uma interpretação, conforme apresentado inicialmente na Seção 3.2.

Em um sistema especialista guiado por objetivo, a expressão-objetivo é colocada inicialmente na memória de trabalho. O sistema tenta casar as *conclusões* das regras com o objetivo, selecionando uma regra e colocando as suas *premissas* na memória de trabalho. Isso corresponde a uma decomposição do objetivo do problema em subobjetivos mais simples. O processo continua na próxima iteração do sistema de produção, e essas premissas se tornam os novos objetivos a serem casados com as conclusões das regras. Dessa forma, o sistema trabalha retroativamente a partir do objetivo inicial até que se prove que todos os subobjetivos na memória de trabalho são verdadeiros, indicando que a hipótese foi verificada. Assim, a busca para trás em um sistema especialista corresponde, grosso modo, ao processo de teste de hipóteses na solução humana de problemas, o que também foi apresentado inicialmente na Seção 3.2.

Em um sistema especialista, os subobjetivos podem ser resolvidos por meio de pedidos de informação ao usuário. Alguns sistemas especialistas permitem que o projetista do sistema especifique quais subobjetivos podem ser

resolvidos por meio de perguntas ao usuário. Outros simplesmente perguntam sobre todos os subobjetivos que não casam com as regras da base de conhecimento; isto é, se o programa não consegue inferir a verdade de um subobjetivo, ele pergunta ao usuário.

Como exemplo de solução de problema guiada por objetivo com consultas ao usuário, apresentamos a seguir um pequeno sistema especialista para análise de problemas automotivos. Esse não é um sistema de diagnóstico completo, pois ele contém apenas quatro regras bem simples. Ele serve como exemplo para que demonstremos o encadeamento de regras orientado a objetivos, a integração de dados novos e o uso de facilidades de explicação:

Regra 1: se  
o motor recebe combustível e  
o motor tenta pegar,  
então  
o problema é vela.

Regra 2: se  
o motor não pega e  
as luzes não acendem,  
então  
o problema é bateria ou cabo.

Regra 3: se  
o motor não pega e  
as luzes acendem,  
então  
o problema é o motor de partida.

Regra 4: se  
há combustível no tanque de combustível e  
há combustível no carburador,  
então  
o motor está recebendo combustível.

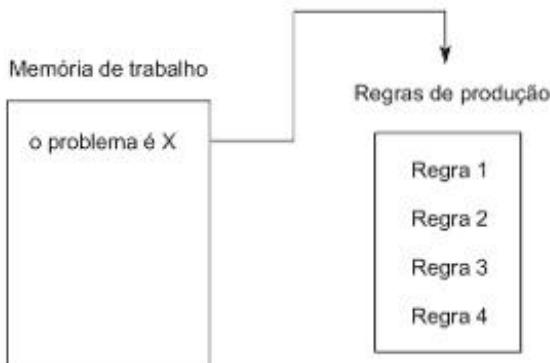
Para utilizar essa base de conhecimento sob um regime de controle guiado por objetivo, coloque o objetivo de nível mais alto, o problema é X, na memória de trabalho, como mostra a Figura 8.5. X é uma variável que pode casar com qualquer frase, por exemplo, o problema é bateria ou cabo; ela será ligada à solução quando o problema é resolvido.

Três regras casam com a expressão na memória de trabalho: Regra 1, Regra 2 e Regra 3. Se resolvemos conflitos em favor da regra com o menor número, então a Regra 1 disparará. Isso faz com que X seja ligado ao valor vela e as premissas da Regra 1 sejam colocadas na memória de trabalho, como na Figura 8.6. Assim, o sistema escolheu explorar a hipótese possível de que as velas estejam falhando. Podemos, também, considerar que o sistema tenha selecionado um ramo ou em um grafo e/ou (ver Capítulo 3).

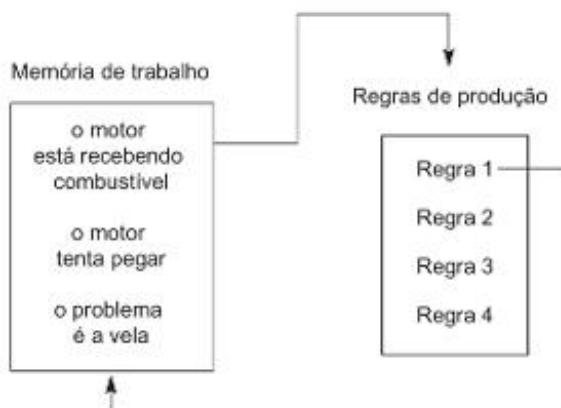
Note que há duas premissas para a Regra 1 e que ambas devem ser satisfeitas para que a conclusão seja verdadeira. Essas premissas são ramos e do grafo de busca que representa uma decomposição do problema (descobrir se o problema é vela) em dois subproblemas (descobrir se o motor está recebendo combustível e se o motor tenta pegar). Podemos, então, disparar a Regra 4, cuja conclusão casa com o motor está recebendo combustível, fazendo com que sua premissa seja colocada na memória de trabalho, como na Figura 8.7.

Nesse ponto, há três entradas na memória de trabalho que não casam com nenhuma conclusão de regra. Nessa situação, nosso sistema especialista pergunta diretamente ao usuário sobre esses subobjetivos. Se o usuário confirmar que os três são verdadeiros, o sistema especialista terá determinado com sucesso que o carro não funciona por causa das velas. Para chegar a essa solução, o sistema buscou o ramo mais à esquerda do grafo e/ou, apresentado na Figura 8.8.

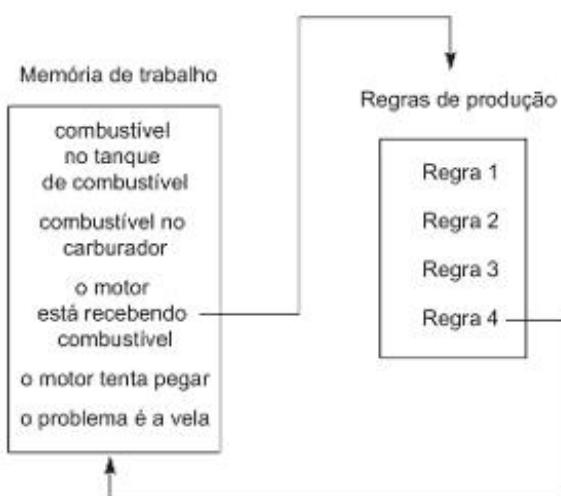
**Figura 8.5** O sistema de produção no inicio de uma consulta no exemplo do diagnóstico automotivo.



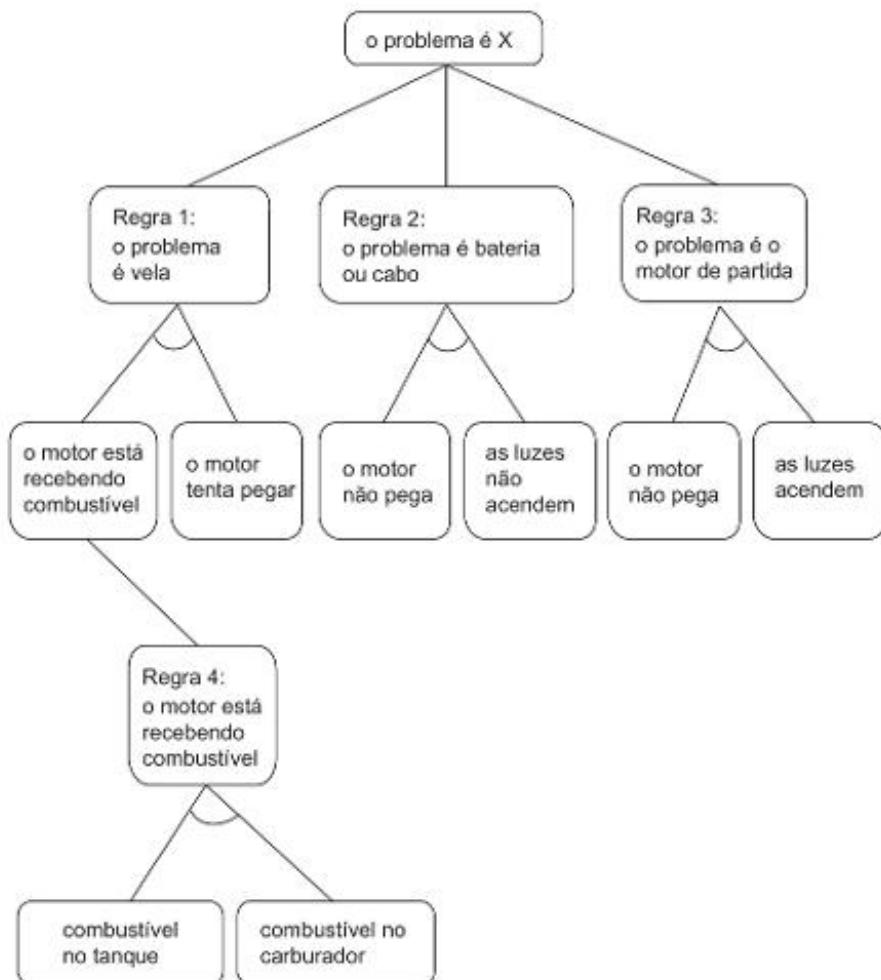
**Figura 8.6** O sistema de produção depois de a Regra 1 ter disparado.



**Figura 8.7** O sistema depois de a Regra 4 ter disparado. Note a abordagem baseada em pilha para a redução do objetivo.



**Figura 8.8** O grafo e/ou buscado no exemplo do diagnóstico automotivo com a conclusão da Regra 4 casando com a primeira premissa da Regra 1.



Claro que esse exemplo é muito simples. Não apenas o conhecimento automotivo é limitado ao extremo, mas ele também ignora uma série de aspectos importantes de implementações reais. As regras são escritas em português, em vez de em linguagem formal. Ao encontrar uma solução, um sistema especialista real dará ao usuário o seu diagnóstico (nossa modelo simplesmente para). Deveríamos, também, manter registro suficiente do raciocínio para permitir um retrocesso se necessário. No exemplo, se tivéssemos falhado em determinar que havia um problema com as velas, precisaríamos retroceder ao nível mais alto e tentar a Regra 2. Note que essa informação está implícita na ordenação dos subobjetivos na memória de trabalho da Figura 8.7 e no grafo da Figura 8.8. Contudo, apesar da sua simplicidade, esse exemplo enfatiza a importância da busca baseada em sistema de produção e a sua representação por grafo e/ou como fundamentação para sistemas especialistas baseados em regras.

Enfatizamos, anteriormente, que um sistema especialista precisa ser aberto para inspeção, facilmente modificável e de natureza heurística. A arquitetura do sistema de produção é um fator importante em todos esses requisitos. A facilidade de modificação, por exemplo, é suportada pela independência sintática das regras de produção: cada regra é uma parte do conhecimento que pode ser modificada independentemente. Contudo, existem restrições semânticas, já que os significados das regras individuais são associados. Esses significados devem ser reconciliados em cada processo de edição ou modificação. Discutimos, a seguir, a geração de explicações e o monitoramento da inferência.

### 8.2.2 Explicação e transparência em raciocínio guiado por objetivo

O sistema de produção implementa uma busca em grafo. As rotinas de explicação da Figura 8.1 acompanham e armazenam o processo de busca e usam a trajetória da busca para responder às consultas do usuário. As próprias regras de produção documentam cada passo do processo de raciocínio.

Em geral, as duas questões que são respondidas pelos sistemas especialistas baseados em regras são **por quê?** e **como?**. Uma consulta por **que** ocorre quando o programa solicita ao usuário uma informação e ele responde com **por que**, o que significa “por que você está solicitando essa informação?”. A resposta é a regra atual que o sistema de produção tenta disparar. A resposta a “**como** você chegou a esse resultado?” é a sequência de regras que foram usadas para concluir um objetivo.

Suponha que o nosso sistema especialista automotivo tenha recursos de explicação e que a consulta tenha alcançado o estágio representado pela configuração de memória de trabalho da Figura 8.7. O diálogo a seguir começa com o computador perguntando ao usuário sobre os objetivos que estão presentes na memória de trabalho. As respostas do usuário estão em negrito.

combustível no tanque de combustível?

**sim**

combustível no carburador?

**sim**

o motor tenta pegar?

**por quê?**

Aqui, o usuário faz uma consulta **por que**, isto é, por que você quer saber se o motor tenta pegar? O sistema responde com a apresentação literal da regra em questão:

Foi estabelecido que:

1. o motor está recebendo combustível,  
portanto, se
2. o motor tenta pegar,  
então o problema é vela.

Como o usuário respondeu **sim** às duas primeiras questões, a Regra 4 pode ser disparada, comprovando a primeira premissa da Regra 1. Quando a consulta **por que** foi realizada, o sistema tentava resolver a segunda premissa da Regra 1. Note que a explicação para o comportamento do sistema é apenas a declaração da regra que é considerada no momento. Contudo, isso fornece uma explicação adequada do porquê de o programa querer saber se o motor tenta pegar. Na resposta à consulta **por que**, o sistema declara que ele concluiu que o motor está recebendo combustível. Suponha que o usuário pergunta então:

**como o motor está recebendo combustível**

O sistema responderá como ele concluiu que o motor está recebendo combustível. A resposta é o caminho do raciocínio que levou a essa conclusão, percorrendo retroativamente, a partir do objetivo, ao longo das regras que o suportam, até às respostas do usuário:

Isso resulta da regra 4:

se

combustível no tanque de combustível e

combustível no carburador

então

o motor está recebendo combustível.

combustível no tanque foi declarado pelo usuário

combustível no carburador foi declarado pelo usuário

Mais uma vez, a arquitetura do sistema de produção fornece a base essencial para essas explicações. Cada ciclo do laço de controle seleciona e dispara outra regra. O programa pode ser parado após cada ciclo e inspecionado. Como cada regra representa uma porção completa de conhecimento para a solução do problema, a regra atual fornece um contexto para a explicação. Observe o contraste dessa abordagem por sistema de produção com arquiteturas mais tradicionais de programação: se pararmos um programa C, C++ ou Java no meio da execução, é bastante provável que o comando atual não tenha muito significado.

Em resumo, o sistema baseado em conhecimento responde a consultas por que mostrando a regra atual que ele está tentando disparar; ele responde a consultas como apresentando o caminho de raciocínio que o levou até um objetivo ou subobjetivo. Embora os mecanismos sejam conceitualmente simples, eles podem exibir um poder explanatório notável se a base de conhecimento estiver organizada de uma forma lógica. O material na Sala Virtual sobre Java, Lisp e Prolog demonstra o uso de pilhas de regras e árvores de prova para implementar essas explicações.

Se desejarmos que as explicações se comportem logicamente, é importante não apenas que a base de conhecimento encontre a resposta correta, mas também que cada regra corresponda a um único passo lógico no processo de solução do problema. Se uma base de conhecimento combinar vários passos em uma única regra, ou se ela quebrar as regras de uma maneira arbitrária, ela poderá encontrar respostas corretas, mas que são vagas, arbitrárias ou ilógicas às consultas como e por quê. Isso não apenas abala a confiança do usuário no sistema, mas também torna mais difícil para o projetista entender e modificar o programa.

### 8.2.3 Usando o sistema de produção para raciocínio guiado por dados

A demonstração do diagnóstico automotivo da Seção 8.2.1 ilustra o uso de um sistema de produção para implementar busca guiada por objetivo. O processo de busca se dava em profundidade, já que ele buscava exaustivamente cada subobjetivo encontrado na base de regras antes de se mover para qualquer outro objetivo irmão. Entretanto, como vimos na Seção 6.2, o sistema de produção é, também, uma arquitetura ideal para o raciocínio guiado por dados. O Exemplo 6.2.1 demonstrou esse processo com o quebra-cabeça dos 8 e os exemplos 6.2.2 e 6.2.3, com o “percurso do cavalo”. Em cada um desses problemas, resolvemos conflitos escolhendo a primeira regra encontrada na base de conhecimento e, então, seguimos os resultados dessa regra. Isso dava ao processo um caráter de busca em profundidade, embora não houvesse um mecanismo, como o retrocesso, para tratar o problema de “becos sem saída” no espaço de busca.

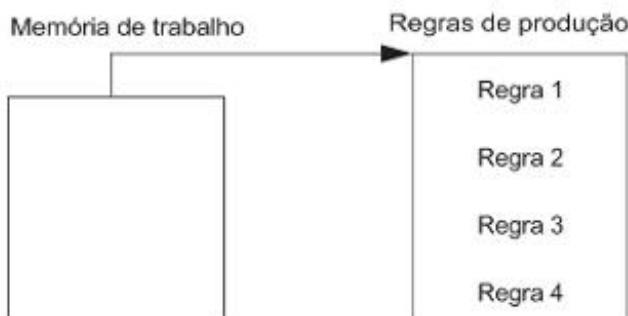
A busca em amplitude é ainda mais comum em raciocínio guiado por dados. O algoritmo para isso é simples: compare o conteúdo da memória de trabalho com as condições de cada regra na base de regras usando a ordenação da base de regras. Se os dados na memória de trabalho permitem o disparo de uma regra, o resultado é colocado na memória de trabalho e, então, o controle se move para a próxima regra. Uma vez que todas as regras tenham sido consideradas, a busca recomeça no início do conjunto de regras.

Considere, por exemplo, o problema do diagnóstico automotivo e as regras da Seção 8.2.1. Se uma parte de informação que compõe (é parte de) a premissa de uma regra não for a conclusão de alguma outra regra, então esse fato é considerado “perguntável” quando o controle encontrar a situação (regra) em que aquela informação é necessária. Por exemplo, o motor está recebendo combustível não é questionável na premissa da Regra 1, porque esse fato é uma conclusão de uma outra regra, a Regra 4.

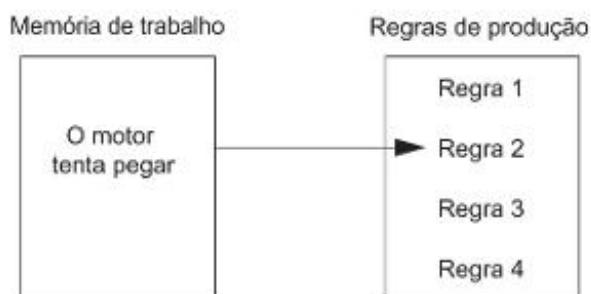
O exemplo guiado por dados, com busca em amplitude, começa como na Figura 8.5, exceto que não há informação na memória de trabalho, como na Figura 8.9. Primeiro, examinamos as premissas das quatro regras para verificar que informação é “perguntável”. A premissa o motor está recebendo combustível não é questionável, fazendo com que a Regra 1 falhe e o controle se move para a Regra 2. O motor não tenta pegar é questionável. Suponha que a resposta a essa consulta seja falsa, de modo que o motor tenta pegar é colocado na memória de trabalho, como vemos na Figura 8.10.

Mas a Regra 2 falha, já que a primeira das duas premissas é falsa, e a Regra 3 passa a ser considerada, onde novamente a primeira premissa falha. Na Regra 4, as duas premissas são questionáveis. Suponha que a resposta às duas perguntas seja verdadeiro. Então, há combustível no tanque de combustível e há combustível

**Figura 8.9** Sistema de produção no início de uma consulta para o raciocínio guiado por dados.



**Figura 8.10** Sistema de produção após a avaliação da primeira premissa da Regra 2, que falha.



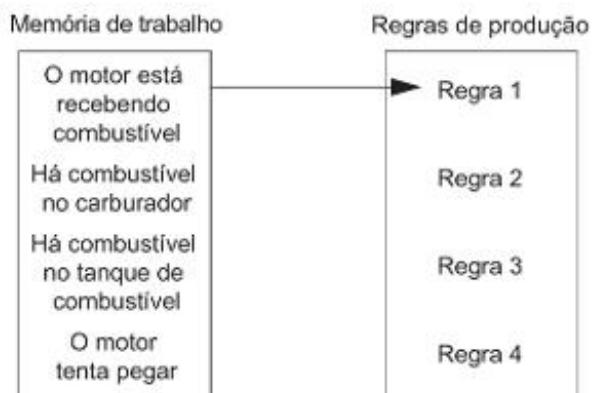
no carburador são colocados na memória de trabalho, assim como a conclusão da regra, o motor está recebendo combustível.

Nesse ponto, todas as regras foram consideradas de modo que a busca agora, com o novo conteúdo da memória de trabalho, volta a considerar as regras na ordem pela segunda vez. Como é visto na Figura 8.11, quando a memória de trabalho é casada com a Regra 1, a sua conclusão, o problema é vela, é colocada na memória de trabalho. Nesse exemplo, nenhuma outra regra casará e disparará, e a sessão de resolução do problema é encerrada. A Figura 8.12 apresenta um grafo do processo de busca, com o conteúdo de informação da memória de trabalho (MT) como os nós do grafo.

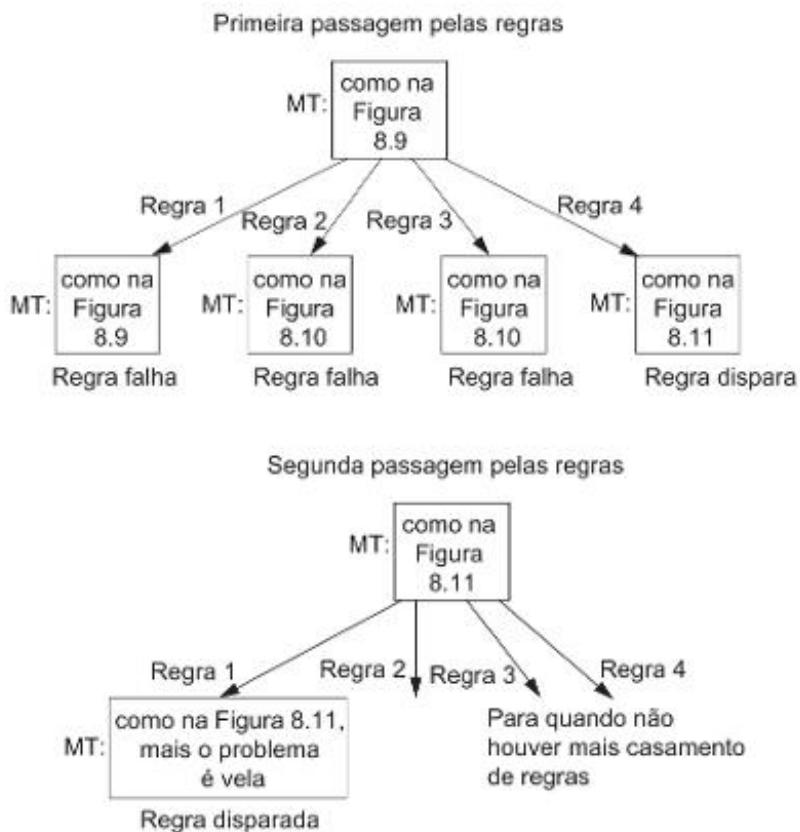
Um refinamento importante da estratégia de busca em amplitude usada no exemplo anterior é a chamada *busca oportunista*. Essa estratégia de busca é simples: sempre que uma regra dispara para concluir informações novas, o controle se desloca para considerar as regras que têm essa informação nova como premissa. Isso faz com que qualquer nova informação *concluída* (a busca não é alterada em consequência de premissas “questionáveis”) seja a força que controla a busca pelas próximas regras a disparar. A busca é chamada de *oportunista* porque cada conclusão de informação nova guia a busca. Por causa da ordenação casual das regras, o exemplo simples anterior também é oportunista.

Concluímos essa seção sobre raciocínio guiado por dados com uma série de comentários sobre explicação e transparência em sistemas com encadeamento para a frente. Em primeiro lugar, em comparação com sistemas guiados por objetivo, que vimos nas seções 8.2.1 e 8.2.2, o raciocínio guiado por dados é muito menos “focado” na sua busca. A razão para isso é óbvia: em um sistema guiado por objetivo, o raciocínio persegue um objetivo particular; esse objetivo é dividido em subobjetivos que dão suporte ao objetivo de alto nível, sendo que esses subobjetivos podem ser também divididos. Como resultado, a busca é sempre direcionada a partir dessa hierarquia de objetivo e subobjetivos. Em sistemas guiados por dados, essa orientação a objetivo não existe. Em vez

**Figura 8.11** Sistema de produção guiado por dados após a consideração da Regra 4 iniciando a sua segunda passagem pelas regras.



**Figura 8.12** Grafo de busca como descrito pelos conteúdos da memória de trabalho (MT) para a busca em amplitude guiada por dados do conjunto de regras da Seção 8.2.1.



disso, a busca se desloca pela árvore dependendo apenas da ordenação das regras e da descoberta de informação nova. Como resultado, o progresso da busca muitas vezes pode parecer muito difuso e desfocado.

Em segundo lugar, e como resultado direto do primeiro comentário, a explicação disponível ao usuário a qualquer tempo durante a busca é bastante limitada. Há uma atribuição de responsabilidade ao nível das regras

pela qual, quando o usuário pergunta por que uma informação está sendo solicitada, a consulta por que da Seção 8.2.2, a regra em consideração é apresentada. Contudo, a explicação não pode ir muito adiante, a menos que seja acrescentado ao sistema algum mecanismo de rastreamento explícito de regras, digamos, com a busca oportunista. A natureza difusa da busca guiada por dados torna isso difícil de ser feito. Por fim, quando um objetivo é alcançado, é também difícil apresentar uma explicação como completa. A única coisa que pode ser usada como uma explicação parcial e incompleta é a apresentação do conteúdo da memória de trabalho ou da lista de regras disparadas. Contudo, isso também não oferecerá a atribuição de responsabilidade consistente e focada que temos com o raciocínio guiado por objetivo.

### 8.2.4 Heurística e controle em sistemas especialistas

Por causa da separação entre a base de conhecimento e o mecanismo de inferência, e pelo regime de controle fixo que o mecanismo de inferência fornece, o programador pode usar um método importante para controlar a busca que funciona por meio da estruturação e da ordenação das regras na base de conhecimento. Esse microgerenciamento do conjunto de regras fornece oportunidades importantes, especialmente quando as estratégias de controle necessárias para a solução de problemas no nível de especialista tendem a ser específicas para o domínio e intensivas em conhecimento. Embora uma regra na forma se  $p, q \text{ e } r$  então  $s$  se pareça com uma expressão lógica, ela também pode ser interpretada como uma série de procedimentos ou passos para resolver um problema: para fazer  $s$ , primeiro faça  $p$ , então faça  $q$  e então faça  $r$ . O papel da ordenação das regras e das premissas se encontrava implícito nos exemplos que acabamos de apresentar na Seção 8.2.

Esse método procedural de interpretação de regras é um componente essencial de uso prático do conhecimento e, muitas vezes, reflete a estratégia de solução do especialista humano. Podemos, por exemplo, ordenar as premissas de uma regra de modo que, em primeiro lugar, seja testado aquilo que é mais provável de não ser válido ou, então, mais fácil de se confirmar. Isso possibilita eliminar uma regra (e com isso uma parte do espaço de busca) o mais cedo possível. A Regra 1 do exemplo automotivo tenta determinar se o motor está recebendo combustível antes de perguntar se o motor tenta pegar. Isso é ineficiente, pois, ao tentar determinar se o motor está recebendo combustível, invoca-se uma outra regra que acaba fazendo duas perguntas ao usuário. Se invertermos a ordem das premissas, uma resposta negativa à consulta “o motor tenta pegar?” exclui essa regra das considerações antes que as condições mais complexas sejam examinadas, tornando o sistema mais eficiente.

Além disso, faz mais sentido determinar se o motor está tentando pegar antes de verificar se ele está recebendo combustível; se o motor não tenta pegar, não importa se ele está ou não recebendo combustível! Na Regra 4, solicita-se que o usuário verifique se há combustível no tanque de combustível antes de verificar se está entrando combustível no carburador. Nessa situação, a verificação mais fácil ocorre primeiro. Há um ponto importante aqui, se o sistema global deve ser mais eficiente, todos os aspectos devem ser considerados: a ordem das regras, a organização das premissas, o custo de testes, a quantidade de busca eliminada por meio das respostas a testes, a maneira como maioria dos casos ocorre etc.

Assim, o planejamento da ordem das regras, a organização das premissas de uma regra e os custos de diferentes testes são todos fundamentalmente de natureza heurística. Na maioria dos sistemas especialistas baseados em regras, essas escolhas heurísticas refletem as abordagens seguidas pelo especialista humano e que podem causar resultados errados. No nosso exemplo, se o motor estiver recebendo combustível e tentando pegar, o problema pode ser o distribuidor, e não as velas.

O raciocínio guiado por dados fornece problemas e possibilidades adicionais para o controle do raciocínio. Entre eles se encontram as heurísticas de alto nível, como refração, atualidade (busca oportunista) e especificidade, apresentadas na Seção 6.2.3. Uma abordagem mais específica do domínio agrupa conjuntos de regras de acordo com os estágios do processo de solução. No diagnóstico de problemas automotivos, por exemplo, poderíamos criar quatro estágios distintos de (1) organizar situação, (2) coletar dados, (3) fazer a análise (pode haver mais de um problema com o carro) e, finalmente, (4) relatar as conclusões e sugerir reparos.

Essa solução de problema em estágios pode ser realizada pela criação de descritores para cada estágio da solução, colocando essa descrição como a primeira premissa de todas as regras que pertencem a esse estágio.

Poderíamos começar, por exemplo, colocando a declaração organizar situação na memória de trabalho. Se não ocorrerem outras descrições de estágios na memória de trabalho, apenas as regras que tiverem organizar situação no seu conjunto de premissas dispararão. É claro que essa deve ser a primeira premissa de cada uma dessas regras. Passamos para o próximo estágio, quando a última regra a disparar no estágio organizacional remover (retratar) o fato que o estágio é organizar solução e declarar o novo fato que o estágio é coletar dados. Todas as regras do estágio coletar dados teriam então como primeira premissa SE estágio é coletar dados e.... Quando o estágio coletar dados é encerrado, a última regra retrata esse fato e insere o fato analisar dados na memória de trabalho, para que apenas aquelas regras, cujas premissas começem com o fato SE o estágio é analisar dados..., possam disparar.

Até agora em nossa discussão, descrevemos o comportamento do sistema de produção em termos de considerações exaustivas da base de regras. Embora seja custoso, isso capta a semântica pretendida do sistema de produção. Contudo, existe uma série de algoritmos, como RETE (Forgy, 1982), que podem ser usados para otimizar a busca por todas as regras potencialmente úteis. Basicamente, o algoritmo RETE compila regras em uma estrutura de rede que permite que o sistema realize o casamento de regras com dados, seguindo diretamente um ponteiro para a regra. Esse algoritmo acelera consideravelmente a execução, especialmente em grandes conjuntos de regras, retendo o comportamento semântico que descrevemos nesta seção.

Em resumo, regras formam a abordagem mais antiga para a representação de conhecimento em sistemas especialistas e continuam sendo uma técnica importante para a construção de resolvedores de problemas baseados intensivamente em conhecimento. As regras de um sistema especialista captam o conhecimento do especialista humano como ele é usado na prática; consequentemente, com frequência eles são uma mistura de conhecimento teórico, de heurísticas derivadas da experiência e de regras de propósito especial para tratar casos estranhos e outras exceções à prática normal. Em muitas situações, essa abordagem se mostrou efetiva. Apesar disso, sistemas fortemente heurísticos podem falhar, ou por encontrar um problema que não se encaixa nas regras disponíveis, ou por aplicar erroneamente uma regra heurística a uma situação não apropriada. Os especialistas humanos não sofrem desses problemas, porque eles têm um entendimento teórico mais profundo do domínio do problema que permite que apliquem as regras heurísticas intelligentemente ou recorram ao raciocínio a partir de “princípios elementares” em novas situações. As abordagens *baseadas em modelo*, descritas a seguir, na Seção 8.3.1, tentam conferir ao sistema especialista esse poder e essa flexibilidade.

A habilidade de aprender a partir de exemplos é outra capacidade humana que os resolvedores de problema baseados intensivamente em conhecimento emulam. Os sistemas para raciocínio *baseado em casos*, que vemos Seção 8.3.3, mantêm uma base de conhecimento de exemplos de soluções do problema, ou *casos*. Quando confrontado com um novo problema, o sistema seleciona desse conjunto de casos armazenados um caso que seja parecido com o problema apresentado e, então, tenta aplicar uma forma de sua estratégia de solução para esse problema. No raciocínio legal, o argumento por meio de precedentes é um exemplo comum de raciocínio baseado em casos.

## 8.3 Sistemas baseados em modelo, em casos e híbridos

### 8.3.1 Introdução ao raciocínio baseado em modelo

A perícia humana é um mistura extremamente complexa de conhecimento teórico, heurísticas para solucionar problemas baseados na experiência, exemplos de problemas passados e suas soluções, habilidades interpretativas e perceptivas e outras habilidades tão pouco compreendidas que podemos apenas descrevê-las como intuitivas. Os anos de experiência fazem com que os especialistas humanos desenvolvam regras muito poderosas para lidar com situações encontradas comumente. Com frequência essas regras são altamente “compiladas”, assumindo a forma de associações diretas entre sintomas observáveis e diagnóstico final, e ocultando suas fundamentações explicatórias mais profundadas.

O sistema especialista MYCIN, por exemplo, poderia propor um diagnóstico baseado em sintomas observáveis, como “dor de cabeça”, “náusea” ou “febre alta”. Embora esses parâmetros possam ser indicativos de uma doença, as regras que os ligam diretamente a um diagnóstico não refletem nenhum entendimento causal mais profundo da fisiologia humana. As regras de MYCIN indicam os resultados de uma infecção, mas não explicam as suas causas. Uma abordagem explanatória mais aprofundada detectaria a presença de agentes infecciosos, notaria a inflamação resultante de tecidos celulares, a presença de pressão intracraniana e inferiria a conexão causal aos sintomas observados de dor de cabeça, temperatura elevada e náusea.

Em um exemplo de sistema especialista baseado em regras para análise de falhas em semicondutores, uma abordagem descritiva poderia basear um diagnóstico de falha de um circuito em sintomas como a descoloração de componentes (uma possível indicação de que há um componente queimado), a história de falhas em dispositivos similares, ou mesmo em observações de interiores de componentes usando um microscópio eletrônico. Entretanto, as abordagens que usam regras para ligar observações a diagnósticos não oferecem os benefícios de uma análise mais profunda da estrutura e da função do dispositivo. Uma abordagem explanatória mais aprofundada e mais robusta começaria com um modelo detalhado da estrutura física do circuito e equações que descrevem o comportamento esperado de cada componente e suas interações. Ela basearia o seu diagnóstico em leituras de sinais de várias partes do dispositivo, usando esses dados e o seu modelo do circuito para determinar os pontos exatos de falha.

Como a primeira geração de sistemas especialistas se baseava em regras heurísticas obtidas da descrição do especialista humano das técnicas para resolver o problema, esses sistemas exibiam uma série de limitações fundamentais (Clancy, 1985). Se uma ocorrência de problema não casasse com as suas heurísticas, elas simplesmente falhavam, muito embora uma análise mais teórica pudesse encontrar uma solução. Frequentemente, os sistemas especialistas aplicavam heurísticas em situações inapropriadas, onde um entendimento mais profundo do problema indicaria outro caminho. Essas são as limitações que as abordagens *baseadas em modelo* procuram tratar. Um sistema de raciocínio baseado em conhecimento, cuja análise esteja fundamentada diretamente na especificação e na funcionalidade de um sistema físico, é chamado de *sistema baseado em modelo*. Em seu projeto e uso, um sistema de raciocínio baseado em modelo cria uma simulação, frequentemente referida como “qualitativa”, da função do que está sendo compreendido ou reparado. (Claro que há outros tipos de sistemas baseados em modelos, em particular, os baseados em lógica e os estocásticos, que apresentaremos no Capítulo 9.)

Os primeiros sistemas de raciocínio baseados em modelo surgiram em meados dos anos 1970 e continuaram a amadurecer até os anos 1980 (Davis e Hamscher, 1992). É interessante notar que alguns dos primeiros trabalhos tinham a intenção de criar modelos de software de vários dispositivos físicos, como circuitos eletrônicos, para fins de instrução (de Kleer, 1976; Brown et al., 1982). Nesses primeiros tutores inteligentes, as especificações para um dispositivo ou circuito se refletiam em conjuntos de regras, por exemplo, as leis de Kirchoff e de Ohm. O sistema tutor tanto testava o conhecimento do estudante sobre o dispositivo, ou circuito, quanto transmitia ao estudante o conhecimento que lhe faltasse. As regras eram a representação da funcionalidade do hardware e o meio para transmitir esse conhecimento ao estudante.

Desses primeiros sistemas tutores, em que a tarefa era modelar e ensinar a funcionalidade de um sistema, os sistemas de raciocínio baseados em modelos se deslocaram para sistemas de teste de falhas. Em um teste de falhas de um sistema físico, o modelo leva a conjuntos de comportamentos previstos. Uma falha se manifesta na discrepância entre os comportamentos previsto e observado. O sistema baseado em modelo informa ao usuário o que ele deve esperar, quando as observações diferem dessas expectativas e como essas discrepâncias levam à identificação de falhas.

O raciocínio qualitativo baseado em modelo inclui:

1. Uma descrição de cada componente do dispositivo. Essas descrições podem simular o comportamento do componente.
2. Uma descrição da estrutura interna do dispositivo. Geralmente, isso é uma representação de seus componentes e de suas interconexões com a habilidade de simular interações dos componentes. A extensão do conhecimento da estrutura interna necessária depende dos níveis de abstração aplicados e do diagnóstico desejado.

3. O diagnóstico de um problema particular requer observações do desempenho real do dispositivo, geralmente medidas de suas entradas e saídas. Essas medidas de entrada/saída são fáceis de serem obtidas, mas, na realidade, qualquer medida poderia ser usada.

A tarefa é, então, determinar quais desses componentes poderiam ter falhado, de modo que os comportamentos observados sejam explicados. Isso requer regras adicionais que descrevam modos conhecidos de falhas para os diferentes componentes e suas interconexões. O sistema de raciocínio deve encontrar as falhas mais prováveis que podem explicar o comportamento observado do sistema.

Uma série de estruturas de dados pode ser usada para representar a informação causal e estrutural em modelos. Muitos projetistas de programas baseados em modelo usam regras para refletir a causalidade e a funcionalidade de um dispositivo. Regras podem ser usadas, também, para captar as relações entre componentes. Um sistema orientado a objetos também oferece uma ferramenta representacional excelente para refletir a estrutura de dispositivos e componentes dentro de um modelo, com os atributos de um objeto representando um estado do dispositivo, ou componente, e seus métodos definindo a sua funcionalidade.

Para sermos mais concretos no projeto e na avaliação de um modelo, consideramos, agora, vários exemplos de análise de dispositivos e circuitos (Davis e Hamscher, 1992). O comportamento do dispositivo é representado por um conjunto de expressões que captam as relações entre os valores nos terminais do dispositivo. Para o somador da Figura 8.13, há três expressões:

Se conhecemos os valores em A e B, o valor em C é  $A + B$  (linha contínua).

Se conhecemos C e A, o valor em B é  $C - A$  (a linha tracejada).

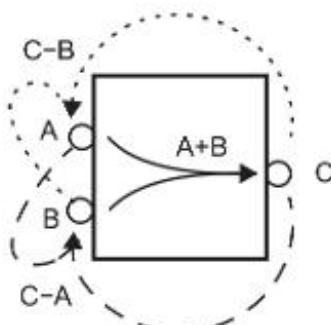
Se conhecemos C e B, o valor de A é  $C - B$  (a linha pontilhada).

Não precisamos usar uma forma algébrica para representar essas relações. Poderíamos igualmente ter usado tuplas relacionais, ou ainda ter representado as restrições com funções Lisp. O objetivo no raciocínio baseado em modelo é representar o conhecimento que capta a funcionalidade do somador.

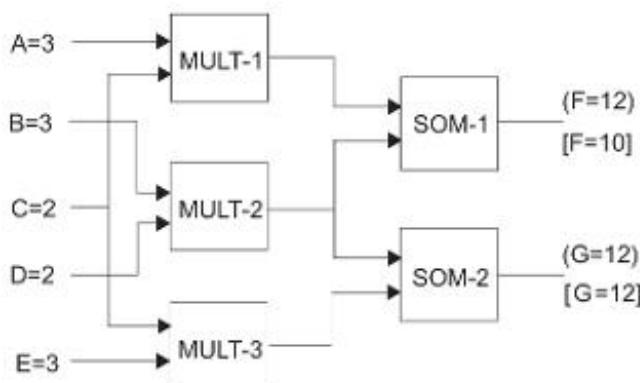
Em um segundo exemplo, considere o circuito de três multiplicadores e dois somadores ligados como na Figura 8.14. Nesse exemplo, os valores de entrada são apresentados em A a E e os valores de saída em F e G. Os valores de saída esperados são dados entre parênteses, e as saídas reais, entre colchetes. A tarefa é determinar onde se encontra a falha que explica a discrepância. Em F, temos um conflito, esperamos 12 e recebemos 10. Verificamos as dependências nesse ponto e determinamos que o valor em F é uma função de SOM-1 o que, por sua vez, depende das saídas de MULT-1 e MULT-2. Um desses três dispositivos deve ter falhado, e assim temos três hipóteses a considerar: ou o comportamento do somador é inadequado, ou uma de suas entradas era incorreta e o problema está mais para trás no circuito.

Raciocinando a partir do resultado (10) em F e supondo que o comportamento de SOM-1 e de uma de suas entradas X (6) seja correto, então a entrada Y do SOM-1 deve ser 4. Mas isso conflita com a expectativa de 6, que é o comportamento correto de MULT-2 com as entradas B e D. Observamos essas entradas e sabemos que elas são

**Figura 8.13** Descrição de comportamento de um somador, de Davis e Hamscher (1992).



**Figura 8.14** Tirando proveito da direção do fluxo de informação, adaptado de Davis e Hamscher (1988).



corretas, assim MULT-2 deve ter falhado. Em um argumento paralelo, nossa segunda hipótese é que SOM-1 está correto e que MULT-1 falhou.

Continuando nosso raciocínio, se a primeira entrada X de SOM-1 estiver correta e o próprio SOM-1 estiver correto, então a segunda entrada Y deve ser 4. Se ela fosse 4, G seria 10, em vez de 12, e assim a saída de MULT-2 deveria ser 6 e correta. Ficamos, assim, com as hipóteses de que a falha está em MULT-1 ou em SOM-1, e continuamos a restringir esses dispositivos com outros testes.

No nosso raciocínio sobre a situação da Figura 8.14, realizamos três tarefas:

1. Geração de hipóteses, pela qual, dada uma discrepância, conjecturamos quais componentes do dispositivo poderiam tê-la causado.
2. Teste de hipóteses, no qual determinamos quais componentes, dada uma coleção de componentes potencialmente defeituosos, poderiam explicar o comportamento observado.
3. Discriminação da hipótese, na qual, quando mais de uma hipótese sobrevive à fase de teste, como aconteceu no caso da Figura 8.14, devemos determinar que informação adicional deve ser coletada para continuar a busca pela falha.

Por fim, devemos observar que, no exemplo da Figura 8.14, assumimos que apenas um dispositivo falhou. O mundo não é sempre tão simples assim, embora a suposição de falha única seja uma heurística útil e frequentemente correta.

Por se basearem em uma compreensão teórica dos dispositivos em questão, as técnicas qualitativas baseadas em modelos evitam muitas das limitações das abordagens mais heurísticas. Em vez de raciocinarem diretamente a partir de fenômenos observados buscando explicações causais, as abordagens baseadas em modelo tentam representar dispositivos e configurações de dispositivos em um nível causal ou funcional. O código do programa reflete tanto a função dos dispositivos como, também, as dependências dentro de um sistema de dispositivos. Tais modelos são frequentemente mais robustos que as técnicas heurísticas. Entretanto, o lado adverso dessa modelagem explícita de funções é que o estágio de aquisição de conhecimento pode ser bastante complexo, e o programa resultante, grande, desajeitado e lento. Como as abordagens heurísticas “compilam” casos típicos em uma única regra, elas quase sempre são mais eficientes contanto que as outras restrições do sistema sejam apropriadas.

Entretanto, essa abordagem apresenta problemas mais profundos. Como no caso do raciocínio baseado em regras, o modelo de um sistema é apenas isso, um modelo. Ele é, necessariamente, uma abstração do sistema e, em um determinado nível de detalhe, deve ser incorreto. Por exemplo, considere os fios de entrada da Figura 8.14. Em nossa discussão, consideramos esses valores como fornecidos e corretos. Não examinamos o estado do fio propriamente dito e, em particular, a outra ponta em que ele se conecta aos multiplicadores. O que aconteceria se o fio estivesse quebrado, ou tivesse uma conexão defeituosa com o multiplicador? Se o usuário não conseguisse detectar essa falha, o modelo não seria coerente com o dispositivo real.

Qualquer modelo tenta descrever a situação ideal, o que é esperado do sistema, mas isso não é necessariamente o que o sistema faz. Uma falha de “ponte” é um ponto de contato no sistema em que dois fios, ou dispositivos, estão inadvertidamente ligados, como quando um ponto de solda ruim conecta dois fios que não deveriam estar em contato. Na maioria dos casos, o raciocínio baseado em modelo tem dificuldade em levantar hipóteses sobre falhas desse tipo por causa das suposições *a priori* subjacentes ao modelo e dos métodos de busca para determinar anomalias. Falhas de ponte são simplesmente “novos” fios que não são parte do projeto original. Há uma “hipótese de mundo fechado” (Seção 9.1) implícita pela qual se supõe que a descrição da estrutura do modelo é completa e que tudo o que não está no modelo simplesmente não existe.

Apesar desses defeitos, o raciocínio baseado em modelo é um adicional importante ao conjunto de ferramentas do engenheiro do conhecimento. Os pesquisadores continuam a expandir a nossa compreensão do diagnóstico, tanto em relação a como os especialistas fazem isso tão eficientemente quanto a como melhores algoritmos podem ser implementados em máquinas (Stern e Luger, 1997; Pless et al., 2006).

### 8.3.2 Raciocínio baseado em modelo: um exemplo da NASA (Williams e Nayak)

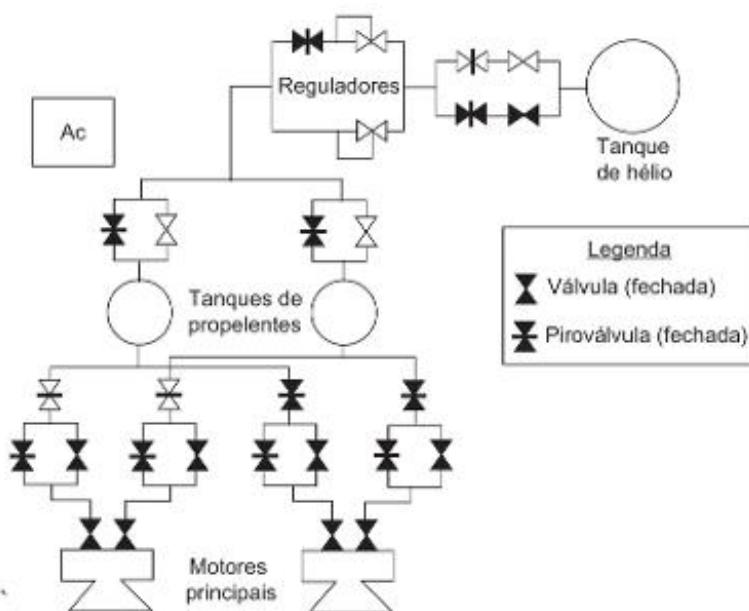
A NASA atualmente vem se mantendo presente no espaço por meio do desenvolvimento de uma frota de sondas espaciais inteligentes que exploram autonomamente o sistema solar (Williams e Nayak, 1996a; Bernard et al., 1998). Esse esforço começou com o software para a primeira sonda, em 1997, e o lançamento do Deep Space 1 em 1998. Para obter sucesso no decorrer dos anos e nas condições adversas de uma viagem espacial, uma nave precisa ser capaz de alterar radicalmente o seu regime de controle em resposta a falhas e, então, planejar considerando essas falhas durante o resto do voo. Para obter custos aceitáveis e uma rápida alteração de configuração, módulos de um tipo apropriado deverão ser combinados rapidamente para gerar, de forma automática, um software de voo. Por fim, a NASA espera que o conjunto de potenciais cenários de falhas e possíveis respostas seja grande demais para o uso de um software que suporte uma enumeração anterior ao voo de todas as contingências. Em vez disso, a nave espacial deverá considerar, de modo reativo, todas as consequências de suas opções de reconfiguração.

*Livingstone* (Williams e Nayak, 1996b) é um núcleo previamente implementado de um sistema autônomo reativo, autoconfigurável, baseado em modelo. A linguagem de representação para o raciocínio baseado em modelo do Livingstone é o cálculo proposicional, um desvio do cálculo de predicados de primeira ordem (Capítulo 2), a linguagem de representação tradicional para diagnóstico baseado em modelo. A partir de suas pesquisas sobre a construção de algoritmos de diagnóstico rápidos em lógica proposicional baseados em conflito (de Kleer e Williams, 1989), Williams e Nayak perceberam que era possível construir um sistema reativo rápido que realizasse deduções significativas no ciclo de percepção/resposta.

O uso de um único modelo centralizado para dar suporte a uma variedade de tarefas de engenharia tem sido uma visão almejada há bastante tempo para o raciocínio baseado em modelo. Para sistemas autônomos baseados em modelo, isso significa usar um único modelo para dar suporte a uma diversidade de tarefas de execução. Aqui se incluem as tarefas de manter o rastreamento de planos de desenvolvimento (Seção 8.4), confirmar os modos de hardware, alterar sua configuração, detectar anomalias, diagnóstico e recuperação de falhas. O Livingstone automatiza todas essas tarefas usando um único modelo e um único algoritmo central, constituindo, com isso, um progresso significativo no sentido de alcançar a visão desejada para resolvedores de problemas baseados em modelo.

A Figura 8.15 mostra um esquema idealizado do módulo do motor principal do Cassini, a nave espacial mais complexa construída até aquele momento. Ele consiste em um tanque de hélio, um tanque de combustível, um tanque oxidante, um par de motores principais, reguladores, válvulas de segurança, piroválvulas e tubos. O tanque de hélio pressuriza os dois tanques de combustível, com os reguladores agindo para reduzir a alta pressão do hélio a uma pressão de trabalho mais baixa. Quando as vias de combustível para um motor principal estão abertas (o ícone da válvula não está preenchido), o tanque pressurizado força o combustível e o oxidante a entrar no motor principal, onde eles combinam, resultando em ignição espontânea e produzindo impulso. As piroválvulas podem

**Figura 8.15** Esquema simplificado do sistema de propulsão do Livingstone, de Williams e Nayak (1996b).



ser disparadas apenas uma vez, isto é, elas podem mudar de estado exatamente uma vez, ou passando de aberta para fechada ou vice-versa. Sua função é isolar partes do subsistema do motor principal até que elas sejam necessárias, ou isolar permanentemente componentes que tenham falhado. As válvulas de segurança são controladas usando acionadores de válvulas (não incluídos na Figura 8.15), e o Ac (acelerômetro) detecta o impulso gerado pelos motores principais.

Partindo da configuração mostrada na Figura 8.15, o objetivo de alto-nível de se produzir impulso pode ser alcançado usando uma série de configurações diferentes: o impulso pode ser gerado por qualquer um dos motores principais, e há várias formas de abrir vias de combustível para cada motor principal. Pode-se gerar impulso, por exemplo, abrindo a válvula de segurança que conduz ao motor da esquerda, ou disparando um par de piroválvulas e abrindo um conjunto de válvulas de segurança que levam ao motor da direita. Outras configurações correspondem a várias combinações de disparos de piroválvulas. As diferentes configurações têm características distintas, pois esses disparos são ações irreversíveis e que requerem muito mais potência do que abrir ou fechar válvulas de segurança.

Suponha que o subsistema dos motores principais tenha sido configurado para fornecer impulso a partir do motor principal da esquerda, abrindo as válvulas de segurança que conduzem a ele. Suponha que esse motor falhe, por exemplo, por superaquecimento, de modo que ele não consiga fornecer impulso. Para assegurar que o impulso desejado seja fornecido mesmo nessa situação, a espaçonave deve passar para uma nova configuração, onde o impulso seja agora fornecido pelo motor principal da direita. Idealmente, isso é alcançado pelo disparo das duas piroválvulas que levam ao lado direito e pela abertura das válvulas de segurança restantes, em vez do disparo de outras piroválvulas.

Um gerenciador de configurações tenta constantemente mover a espaçonave para configurações de custo mais baixo que alcancem um conjunto de objetivos de alto nível, dinamicamente variável. Quando a espaçonave se afasta, por motivo de falhas, da configuração escolhida, o gerenciador analisa os dados dos sensores para identificar a configuração atual da espaçonave, movendo-a para uma nova configuração, a qual, mais uma vez, alcança os objetivos de configuração desejados. Um gerenciador de configurações é um sistema de controle discreto que assegura que a configuração da espaçonave sempre alcança algum ponto definido pelos objetivos de configuração. Os algoritmos de planejamento que suportam o gerenciador de configurações são apresentados na Seção 8.4.4.

O raciocínio sobre as configurações (e reconfiguração autônoma) de um sistema requer os conceitos de modos de operação e de falha, de falhas reparáveis e de mudanças de configuração. A NASA expressa esses conceitos em um diagrama de espaço de estados: falhas reparáveis são transições de um estado de falha para um estado operante; mudanças de configuração ocorrem entre estados operantes; e falhas são transições de um estado operante para um estado de falha.

Williams e Nayak (1997) veem um sistema autônomo como uma combinação de um gerenciador de configurações reativo e um planejador de alto nível. O planejador gera uma sequência de objetivos de configuração de hardware (veja detalhes na Seção 8.4). O gerenciador de configurações evolui o sistema de transição do domínio de aplicação — aqui, o sistema de propulsão da espaçonave — ao longo da trajetória desejada. O gerenciamento de configurações ocorre, então, pela percepção e controle do estado de um sistema de transição.

Um gerenciador de configurações baseado em modelo é um gerenciador de configurações que usa uma especificação do sistema de transição para calcular a sequência desejada de válvulas de controle. No exemplo da Figura 8.15, cada componente de hardware é modelado por um sistema de transição de componente. A comunicação de componentes, indicada por fios (arcos) na figura, é modelada por termos proposicionais compartilhados entre os sistemas de transição de componentes correspondentes.

O gerenciador de configurações faz uso extensivo do modelo para inferir o estado atual do sistema e para selecionar as ações de controle ótimas para atingir os objetivos de configuração. Isso é essencial em situações em que erros podem levar a desastres, excluindo métodos simples de tentativa e erro. O gerenciador de configurações *baseado em modelo* usa um modelo para determinar a sequência de controle desejada em dois estágios: estimativa de modo e reconfiguração de modo (EM e RM na Figura 8.16). EM gera, de modo incremental, o conjunto de todas as trajetórias consistentes com o modelo de transição do mecanismo e a sequência de controle do sistema e de valores percebidos. RM usa o modelo de transição do mecanismo e as trajetórias parciais geradas por EM até o estado atual, para determinar um conjunto de valores de controle, tal que todas as trajetórias previstas alcancem o objetivo de configuração do próximo estado. Tanto EM como RM são reativos. EM infere o estado atual a partir do conhecimento do estado anterior e de observações feitas no estado atual. RM considera apenas ações que alcancem objetivos de configuração contidos no próximo estado.

Na próxima seção, consideramos o raciocínio baseado em casos, uma técnica intensiva em conhecimento que nos permite reusar uma experiência passada em um domínio de problema para tratar de situações novas. Na Seção 8.4, apresentamos o planejamento e retornamos ao exemplo de controle da NASA.

**Figura 8.16** Um sistema de gerenciamento de configuração baseado em modelo, de Williams e Nayak (1996b).



### 8.3.3 Introdução ao raciocínio baseado em casos

Regras heurísticas e modelos teóricos são dois tipos de informação que especialistas humanos usam para resolver problemas. Outra estratégia poderosa que os especialistas usam é raciocinar a partir de casos, exemplos

de problemas passados e suas soluções. O *raciocínio baseado em casos* (RBC) usa uma base de dados explícita de soluções de problema para tratar novas situações de solução de problema. Essas soluções podem ser coletadas de especialistas humanos a partir do processo de engenharia do conhecimento ou podem refletir os resultados anteriores de sucessos e fracassos baseados em busca. A formação médica, por exemplo, não se baseia apenas em modelos teóricos de anatomia, fisiologia e doenças; ela também depende muito do histórico de casos e da experiência do médico residente com outros pacientes e com o tratamento de cada um deles. CASEY (Koton, 1988a, b) e PROTOs (Bareiss et al., 1988) são exemplos de raciocínio baseado em casos aplicado à medicina.

Os advogados selecionam casos jurídicos passados, que são similares aos do seu cliente e que sugerem uma decisão favorável, e tentam convencer o tribunal que essas semelhanças merecem veredictos similares. Embora as leis genéricas sejam feitas por processos democráticos, sua interpretação é normalmente baseada em precedentes legais. O modo como uma lei foi interpretada em uma situação similar anterior é um fator crítico para a sua interpretação atual. Assim, um componente importante do raciocínio jurídico é identificar, a partir de casos, precedentes legais para decisões em um caso particular. Rissland (1983) e Rissland e Ashley (1987) projetaram sistemas de raciocínio baseado em casos para dar suporte a argumentos legais.

Os programadores de computadores frequentemente reúsam o seu código, adaptando um programa antigo para adequá-lo a uma nova situação com estrutura similar. Os arquitetos contam com o seu conhecimento de prédios esteticamente agradáveis e úteis do passado para projetar novos prédios que as pessoas achem agradáveis e confortáveis. Os historiadores usam histórias do passado para ajudar estadistas, burocratas e cidadãos a compreender eventos passados e a planejar o futuro. A capacidade de raciocinar a partir de casos é fundamental para a inteligência humana.

Outras áreas óbvias para o raciocínio a partir de casos são o desenho industrial, onde os aspectos de um artefato executado com sucesso podem ser apropriados para uma nova situação, e o diagnóstico, onde as falhas do passado frequentemente ocorrem de novo. O diagnóstico de hardware é um bom exemplo disso. Um especialista nessa área, além de usar o conhecimento teórico extensivo de sistemas eletrônicos e mecânicos, agrega ao diagnóstico experiências passadas de sucessos e fracassos que se relacionam com o problema atual. O RBC tem sido um componente importante de muitos sistemas de diagnóstico de hardware, incluindo tarefas de manutenção de fontes de sinal e baterias em satélites que orbitam a Terra (Skinner e Luger, 1992) e a análise de falhas de componentes discretos de semicondutores (Stern e Luger, 1997).

O RBC apresenta uma série de vantagens na construção de sistemas especialistas. A aquisição de conhecimento pode ser simplificada se registrarmos soluções de um especialista humano para uma série de problemas e fizermos com que o sistema de raciocínio baseado em casos selecione e raciocine a partir do caso apropriado. Isso pouparia o engenheiro de conhecimento da dificuldade de construir regras gerais a partir de exemplos; em vez disso, o sistema generalizaria automaticamente as regras aplicando-as a novas situações.

As abordagens baseadas em casos também podem permitir que um sistema especialista aprenda a partir da sua experiência. Após encontrar uma solução baseada em busca para um problema, o sistema pode armazenar essa solução, de modo que, da próxima vez em que uma situação similar ocorra, não seja mais necessário realizar a busca. Pode ser importante, também, armazenar, na base de casos, informações sobre o sucesso ou o fracasso das tentativas anteriores de solução; assim, o RBC oferece um modelo poderoso de aprendizado. Um exemplo antigo disso é o programa de jogo de damas de Samuel (1959, Seção 4.1.1), onde as posições do tabuleiro, que por busca ou experiência se mostravam importantes, eram armazenadas na esperança de que pudessem ocorrer novamente em um jogo posterior.

Os sistemas de RBC compartilham uma estrutura comum. Para cada novo problema, eles:

- 1. Recuperam casos apropriados da memória.** Um caso é apropriado se a sua solução pode ser aplicada com sucesso à nova situação. Como os sistemas de RBC não têm como saber isso previamente, eles geralmente usam a heurística de escolher casos que sejam similares à ocorrência do problema. Tanto as pessoas como os sistemas de raciocínio artificiais determinam similaridades com base em características comuns: por exemplo, se dois pacientes compartilham uma série de características comuns em seus sintomas e históricos médicos, há uma boa probabilidade que eles tenham a mesma doença e que respondam ao mesmo tratamento. Recuperar casos eficientemente também requer que a memória de casos seja organizada de

modo a ajudar essa recuperação. Tipicamente, os casos são indexados pelas suas características significativas, permitindo a recuperação eficiente de casos que tenham a maioria das características em comum com o problema atual. A identificação das características salientes é altamente dependente da situação.

2. **Modificam um caso recuperado de modo que ele seja aplicável à situação atual.** Geralmente, um caso recomenda uma sequência de operações que transformam um estado inicial em um estado objetivo. O sistema de raciocínio tem que transformar a solução armazenada em operações adequadas para o problema atual. Métodos analíticos, como o ajuste de curvas aos parâmetros comuns aos casos armazenados e aos novos problemas, podem ser muito úteis; por exemplo, no caso da determinação das temperaturas e dos materiais adequados para uma tarefa de soldagem automática. Quando não se dispõe das relações analíticas entre os casos, talvez o uso de métodos heurísticos, como no caso de centrais de assistência para diagnóstico de hardware, seja o mais apropriado.
3. **Aplicam o caso transformado.** O passo dois modifica um caso armazenado, o qual, quando aplicado, pode não garantir uma solução satisfatória do problema. Para tanto, podem ser necessárias modificações na solução, com outras iterações dos primeiros três passos.
4. **Armazenam a solução, com um registro de sucesso ou fracasso, para uso futuro.** O armazenamento do novo caso requer atualização da estrutura de índices. Existem métodos que podem ser usados para gerenciar índices, como os algoritmos para formação de agrupamentos (Fisher, 1987) e outras técnicas de aprendizado de máquina (Stubblefield e Luger, 1996).

As estruturas de dados para o raciocínio baseado em casos podem ser bastante variadas. Na situação mais simples, os casos são armazenados como tuplas relacionais, onde um subconjunto dos argumentos registra as características a serem casadas e outros argumentos apontam para os passos da solução. Os casos podem ser também representados como estruturas mais complexas, como árvores de prova. Um mecanismo bastante comum no armazenamento de casos é a representação dos casos como um conjunto de grandes regras de situação-ação. Os fatos que descrevem a situação da regra são as características salientes do caso armazenado, e os operadores que constituem a ação da regra são as transformações a serem usadas na nova situação. Quando esse tipo de representação de regra é usado, podem ser usados algoritmos como RETE (Forgy, 1982) para organizar e otimizar a busca por casos apropriados.

A questão mais difícil na solução de problemas baseada em casos, independentemente da estrutura de dados selecionada para a representação dos casos, é a seleção das características salientes para a indexação e a recuperação de casos. Kolodner (1993) e outros ativamente envolvidos na solução de problemas baseada em casos estabeleceram como regra de ouro que os casos sejam organizados pelos objetivos e pelas necessidades do resolvedor de problemas. Isto é, que seja feita uma análise cuidadosa dos descritores de caso no contexto de como esses casos serão usados no processo de solução.

Por exemplo, suponha um problema de *sinal de comunicação fraco* em um satélite às 10:24:35 HMG (Hora Média de Greenwich). A partir de uma análise, verifica-se que o sistema de energia está em um nível baixo de potência, que pode ocorrer porque os painéis solares não estão orientados apropriadamente em relação ao Sol. Os controladores na Terra fazem ajustes na orientação do satélite, a potência aumenta e o sinal de comunicação fica forte novamente. Há uma série de características salientes que poderiam ser usadas para registrar esse caso, sendo a mais óbvia a que existe um sinal de comunicação fraco ou que o fornecimento de energia está baixo. Outra característica para descrever esse caso é que a hora do problema era 10:24:35 HMG. Os objetivos e as necessidades do resolvedor de problemas nesse caso sugerem que as características salientes são o sinal de comunicação fraco e o baixo fornecimento de energia; o tempo em que tudo isso aconteceu pode ser considerado irrelevante, a não ser, é claro, que a falha ocorra justamente após o Sol desaparecer sob o horizonte (Skinner e Luger, 1995).

Outro problema essencial a ser considerado pelo RBC é o da representação de noções como sinal fraco e energia baixa. Como é provável que uma situação nunca case precisamente com outra, por exemplo, um número real exato que descreve a intensidade do sinal, o sistema de raciocínio provavelmente representará os valores como intervalos de números reais, por exemplo, os níveis bom, razoavelmente bom, fraco e alertas de perigo.

Kolodner (1993) propõe um conjunto de heurísticas de preferências para ajudar a organizar o armazenamento e a recuperação de casos. Entre elas estão:

1. *Preferência orientada a objetivo.* Organizar os casos, pelo menos em parte, pelas descrições do objetivo. Recuperar casos que tenham o mesmo objetivo que a situação atual.
2. *Preferência por características salientes.* Preferir casos que casem com as características mais importantes ou aqueles que casem com o maior número de características importantes.
3. *Preferência por maior especificidade.* Procurar por casamentos de características, tão exatos quanto possível, antes de considerar casamentos mais genéricos.
4. *Preferência por ocorrências frequentes.* Verificar primeiro os casos que foram mais frequentemente casados.
5. *Preferência por atualidade.* Preferir casos usados mais recentemente.
6. *Preferência por facilidade de adaptação.* Usar primeiro os casos mais fáceis de serem adaptados à situação atual.

O raciocínio baseado em casos tem uma série de vantagens para o projeto de sistemas especialistas. Uma vez que os engenheiros do conhecimento chegaram à representação apropriada para os casos, a aquisição continuada de conhecimento segue diretamente: simplesmente coletar e armazenar mais casos. Frequentemente, a aquisição de casos pode ser feita a partir de registros históricos ou monitorando-se as operações atuais, minimizando, com isso, a demanda por tempo do especialista humano.

Além disso, o RBC levanta uma série de questões teóricas importantes relacionadas com o aprendizado e o raciocínio humanos. Uma das questões mais sutis e críticas levantadas pelo RBC é a questão de definir *similaridade*. Embora a noção de que similaridade é uma função do número de características que dois casos têm em comum seja razoável, ela mascara uma série de sutilezas profundas. Por exemplo, a maioria dos objetos e das situações tem um grande número de propriedades descritivas potenciais; os sistemas de raciocínio baseados em casos em geral selecionam casos com base em um vocabulário mínimo de recuperação. Geralmente, os raciocinadores baseados em casos requerem que o engenheiro do conhecimento defina um vocabulário apropriado de características altamente relevantes. Embora existam trabalhos que visam permitir que um raciocinador determine características relevantes pela sua própria experiência (Stubblefield, 1995), a determinação da relevância continua a ser um problema difícil.

Outro problema importante no raciocínio baseado em casos trata do compromisso entre armazenamento e computação. Conforme um raciocinador baseado em casos adquire mais casos, ele se torna mais inteligente, mais capaz de resolver uma série de problemas-alvo. De fato, conforme acrescentamos casos ao raciocinador, o seu desempenho aumenta — até um certo ponto. O problema é que, conforme a base de casos aumenta, o tempo necessário para recuperar e processar um caso apropriado também aumenta. Uma série de outros fatores pode causar o declínio da eficiência em grandes bases de casos, como a superposição de casos, ruído e a distribuição de tipos de problemas. Uma solução para esse problema é armazenar apenas os “melhores” casos ou “protótipos”, excluindo aqueles que são redundantes ou que não são usados com frequência; isto é, esquecer aqueles casos que não se mostraram úteis. Como um exemplo importante disso, veja o algoritmo de retenção de Samuel (1959) para o armazenamento de posições do tabuleiro de damas. Entretanto, em geral, não fica claro como podemos automatizar essas decisões; esta continua sendo uma área de pesquisa ativa (Kolodner, 1993).

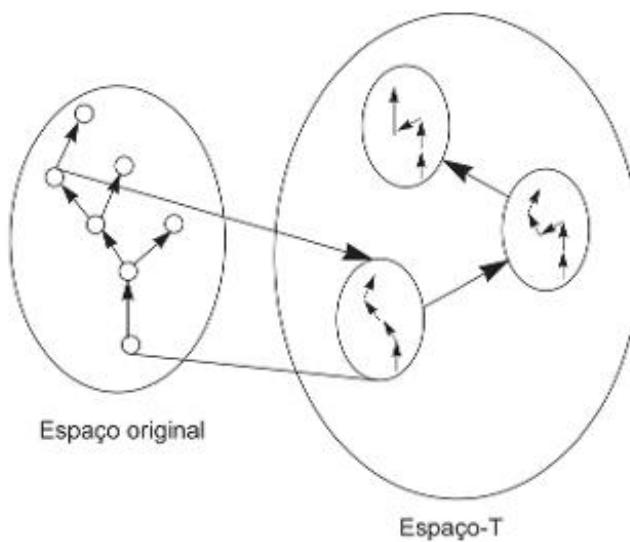
Uma explicação automatizada para o fato de por que uma solução é recomendada também é difícil para um raciocinador baseado em casos. Quando questionado sobre a razão pela qual uma solução foi selecionada para remediar uma situação atual, a única explicação que o sistema pode fornecer é a de que essa solução já funcionou anteriormente. Pode haver, também, explicações fracas baseadas em semelhanças na descrição de alto nível do objetivo entre a situação atual e o conjunto de casos armazenados. No exemplo do problema de comunicação de um satélite, o caso relevante foi escolhido em função do fraco sinal de comunicação. Esse modo de raciocinar não permite uma explicação mais profunda do que aquela em que se afirma que ele já funcionou em uma situação semelhante. Porém, como dito anteriormente, essa pode ser uma explicação suficiente em muitas situações.

Entretanto, muitos pesquisadores perceberam que a simples repetição de objetivos de alto nível e do caso a ser aplicado oferece uma explicação insuficiente (Leake, 1992; Stern e Luger, 1992), especialmente quando precisamos de uma explicação de por que algumas soluções não funcionam. Tomemos novamente a situação do satélite. Suponha que a reorientação do painel solar funcione, mas que três horas mais tarde o sinal se torne fraco novamente. Usando as heurísticas de frequência e de atualidade, reorientamos novamente o painel solar. Em três horas, o sinal fraco retorna. E, novamente, três horas depois; sempre aplicamos a mesma solução. Esse exemplo se baseia em uma situação de um satélite real na qual se verificou que existia um problema mais complexo; nesse caso, um giroscópio estava com problemas de superaquecimento e fornecia uma leitura distorcida que desorientava o satélite. O sistema que finalmente solucionou o problema usava um raciocinador baseado em modelos para simular o comportamento do satélite, visando determinar a causa primária do fraco sinal de comunicação (Skinner e Luger, 1995).

O raciocínio baseado em casos está relacionado com o problema de aprender a partir de analogia. Para reusar experiências passadas, devemos não apenas reconhecer as características salientes do passado, mas também construir um mapeamento de como essa experiência pode ser usada na situação atual. A *analogia transformacional* (Carbonell, 1983) é um exemplo de uma abordagem baseada em casos para a solução de problemas. Ela soluciona novos problemas modificando soluções existentes até que elas possam ser aplicadas à nova ocorrência. Os operadores que modificam soluções completas de um problema definem um nível mais alto de abstração, ou espaço-T, no qual estados são soluções do problema e operadores transformam essas soluções, como na Figura 8.17. O objetivo é transformar uma solução original em uma solução possível para o problema-alvo. Os operadores modificam soluções inserindo ou excluindo passos em um caminho de solução, reordenando passos numa solução, dividindo novas soluções em uma parte de uma solução antiga, ou modificando as ligações de parâmetros na solução atual.

A analogia transformacional tipifica a abordagem usada pela solução de problemas baseada em casos. Trabalhos posteriores refinaram essa abordagem, considerando questões como representação de casos, estratégias para organizar uma memória de casos anteriores, recuperação de casos anteriores relevantes e uso de casos na solução de novos problemas. Para obter mais informações sobre o raciocínio baseado em casos, veja Hammond (1989) e Kolodner (1988a, b). O raciocínio por analogia é discutido mais adiante no contexto do aprendizado simbólico de máquina (ver Seção 10.5.4).

**Figura 8.17** Analogia transformacional, adaptada de Carbonell (1983).



### 8.3.4 Projeto híbrido: pontos fortes e fracos dos sistemas de método forte

Os sucessos na construção de sistemas especialistas que resolvem problemas práticos dificeis têm demonstrado a veracidade da ideia central por trás dos sistemas baseados em conhecimento: que o poder de um raciocinador está no seu conhecimento do domínio, e não na sofisticação de seus métodos de raciocínio. Entretanto, essa observação levanta uma das questões centrais em inteligência artificial: a representação do conhecimento. Em um nível prático, todo engenheiro do conhecimento deve fazer escolhas sobre como representar o conhecimento do domínio de uma forma que será a mais adequada para um dado domínio. A representação do conhecimento levanta, também, uma série de questões teoricamente importantes e intelectualmente dificeis, como o tratamento de informação faltante e incerta, a medida da expressividade de uma representação, a relação entre uma linguagem de representação e questões como aprendizado, aquisição de conhecimento e a eficiência de um raciocinador.

Neste capítulo, consideraremos uma série de abordagens básicas para a representação do conhecimento: sistemas especialistas baseados em regras, raciocinadores baseados em modelos e raciocinadores baseados em casos. Como um auxílio à prática da engenharia do conhecimento, resumimos, a seguir, os pontos fortes e fracos de cada abordagem baseada em conhecimento intensivo para a solução de problemas.

#### Raciocínio baseado em regras

Como vantagens de uma abordagem baseada em regras, podemos citar:

1. Capacidade de usar, de uma forma muito direta, o conhecimento experimental adquirido de especialistas humanos. Isso é particularmente importante em domínios que se baseiam pesadamente em heurísticas para gerenciar a complexidade e/ou informação faltante.
2. As regras são adequadas para a busca em espaço de estados. Os recursos de explicação dão suporte ao processo de correção.
3. A separação entre conhecimento e controle simplifica o desenvolvimento de sistemas especialistas, permitindo um processo de desenvolvimento iterativo, onde o engenheiro adquire, implementa e testa regras individuais.
4. Um bom desempenho em domínios limitados é possível. Por causa das grandes quantidades de conhecimento necessárias para a solução inteligente de problemas, os sistemas especialistas estão limitados a domínios estreitos. Entretanto, há muitos domínios em que o projeto de um sistema apropriado tem se mostrado extremamente útil.
5. Bons recursos de explicação. Embora a estrutura baseada em regras básicas suporte explicações flexíveis, específicas do problema, devemos mencionar que a qualidade final dessas explicações depende da estrutura e do conteúdo das regras. Os recursos de explicação diferem bastante entre sistemas guiados por objetivo e por dados.

Desvantagens do raciocínio baseado em regras:

1. Frequentemente as regras obtidas de especialistas humanos são de natureza altamente heurística e não captam o conhecimento funcional ou baseado em modelo do domínio.
2. As regras heurísticas tendem a ser “frágeis” e não são capazes de lidar com informação faltante ou valores de dados inesperados.
3. Outro aspecto da fragilidade das regras é a tendência a se degradarem rapidamente próximo a “bordas” do conhecimento do domínio. Diferentemente das pessoas, os sistemas baseados em regras são normalmente incapazes de recorrer a princípios fundamentais do raciocínio quando confrontados com problemas inusitados.
4. As explicações funcionam apenas no nível descritivo, omitindo explicações teóricas. Isso advém do fato de que as regras heurísticas devem muito do seu poder à associação direta dos sintomas do problema com as soluções, sem requerer (ou possuir) um raciocínio mais profundo.
5. O conhecimento tende a ser muito dependente da tarefa. O conhecimento formal do domínio tende a ser muito específico em sua aplicabilidade. Atualmente, as linguagens de representação de conhecimento não contemplam a flexibilidade do raciocínio humano.

### Raciocínio baseado em casos

Entre as vantagens do raciocínio baseado em casos, incluímos:

1. A capacidade de codificar diretamente o conhecimento histórico. Em muitos domínios, os casos podem ser obtidos de histórias de casos disponíveis, descrições de reparos, ou outras fontes, eliminando a necessidade de aquisição de conhecimento intensivo de um especialista humano.
2. Permite atalhos no raciocínio. Se um caso apropriado puder ser encontrado, novos problemas poderão ser resolvidos em muito menos tempo do que seria necessário para gerar uma solução por regras ou por modelos e busca.
3. Permite que um sistema evite erros do passado e explore sucessos do passado. O RBC fornece um modelo de aprendizado que é tanto teoricamente interessante como suficientemente prático para ser aplicado em problemas complexos.
4. Não é necessária uma análise extensiva do conhecimento do domínio. Diferentemente de um sistema baseado em regras, onde o engenheiro do conhecimento deve antecipar as interações das regras, o RBC permite um modelo aditivo simples para a aquisição de conhecimento. Isso requer uma representação apropriada para os casos, um índice de recuperação útil e uma estratégia de adaptação de casos.
5. Estratégias de indexação apropriadas aumentam a capacidade de compreensão e o poder de solução de problemas. A habilidade de distinguir diferenças em problemas-alvo e selecionar um caso apropriado é uma fonte importante de poder de um raciocinador baseado em casos; frequentemente, os algoritmos de indexação podem fornecer essa funcionalidade automaticamente.

Entre as desvantagens do raciocínio baseado em casos estão:

1. Os casos frequentemente não incluem um conhecimento profundo do domínio. Isso causa deficiências nos recursos de explicação e, em muitas situações, permite que os casos possam ser mal aplicados, levando a conselhos errados ou de baixa qualidade.
2. Uma grande base de casos pode sofrer problemas pelo compromisso entre armazenamento e computação.
3. É difícil determinar bons critérios para indexar e fazer casamento de casos. Atualmente, os vocabulários de recuperação e os algoritmos de casamento por similaridade precisam ser projetados cuidadosamente à mão; isso pode neutralizar muitas das vantagens que o RBC oferece para a aquisição de conhecimento.

### Raciocínio baseado em modelo

Entre as vantagens do raciocínio baseado em modelo, incluímos:

1. A capacidade de usar conhecimento funcional ou estrutural do domínio na solução do problema. Isso aumenta a capacidade do raciocinador de tratar uma variedade de problemas, incluindo aqueles que não puderam ser antecipados pelos projetistas do sistema.
2. Os raciocinadores baseados em modelo tendem a ser muito robustos. Pelas mesmas razões que as pessoas frequentemente recorrem a princípios fundamentais quando confrontadas com um problema inusitado, os raciocinadores baseados em modelo tendem a ser resolvedores de problemas robustos e flexíveis.
3. Alguns conhecimentos podem ser transferidos entre tarefas. Os raciocinadores baseados em modelo são construídos, frequentemente, usando conhecimento científico, teórico. Como a ciência se esforça para encontrar teorias aplicáveis genericamente, essa generalidade muitas vezes se estende aos raciocinadores baseados em modelo.
4. Os raciocinadores baseados em modelo podem, muitas vezes, fornecer explicações causais. Elas podem transmitir ao usuário humano uma compreensão mais profunda da falha e podem também desempenhar um papel instrutivo importante.

Entre as desvantagens do raciocínio baseado em modelo estão:

1. Uma falta de conhecimento experimental (descritivo) do domínio. Os métodos heurísticos usados pelas abordagens baseadas em regras refletem uma classe valiosa de especialidade.

2. Ele requer um modelo explícito do domínio. Muitos domínios, como o de diagnóstico de falhas em circuitos eletrônicos, têm uma forte base teórica que dá suporte a abordagens baseadas em modelo. Entretanto, muitos domínios, como as especialidades médicas, a maioria dos problemas de projeto, ou muitas aplicações financeiras, carecem de uma teoria científica bem definida. Nesses casos, abordagens baseadas em modelo não podem ser usadas.
3. Alta complexidade. O raciocínio baseado em modelo geralmente opera em um nível de detalhe que resulta em uma complexidade significativa; afinal, essa é uma das razões principais para os especialistas humanos darem preferência ao desenvolvimento de heurísticas.
4. Situações excepcionais. Circunstâncias não usuais, por exemplo, falhas de ponte ou a interação de múltiplas falhas em componentes eletrônicos, podem alterar a funcionalidade de um sistema de uma forma difícil de ser prevista utilizando um modelo *a priori*.

### **Projeto híbrido**

Uma importante área de pesquisa e aplicação é a da combinação de diferentes modelos de raciocínio. Com uma arquitetura híbrida, dois ou mais paradigmas são integrados para produzir um efeito cooperativo, onde as vantagens de um sistema podem compensar as fraquezas de outro sistema. Por meio da combinação, podemos lidar com as desvantagens mencionadas na discussão anterior.

Por exemplo, a combinação de um sistema baseado em regras com outro baseado em casos pode:

1. Oferecer uma primeira verificação natural de casos conhecidos antes de proceder com o raciocínio baseado em regras, com os seus custos de busca associados.
2. Fornecer um registro de exemplos e exceções para soluções por meio da retenção na base de casos.
3. Armazenar resultados baseados em busca como casos para uso futuro. Pelo armazenamento de casos apropriados, um raciocinador pode evitar a duplicação de buscas custosas.

A combinação de um sistema baseado em regras com um sistema baseado em modelos pode:

1. Melhorar as explicações por meio de conhecimento funcional. Isso pode ser particularmente útil em aplicações tutoriais.
2. Melhorar a robustez quando as regras falham. Se não houver regras heurísticas que se apliquem a determinada ocorrência do problema, o raciocinador pode lançar mão do raciocínio a partir dos princípios fundamentais.
3. Acrescentar a busca heurística à busca baseada em modelo. Isso pode ajudar a gerenciar a complexidade do raciocínio baseado em modelo e permitir que o raciocinador escolha inteligentemente entre as alternativas possíveis.

A combinação de um sistema baseado em modelos com um sistema baseado em casos pode:

1. Dar explicações mais maduras às situações armazenadas nos casos.
2. Oferecer uma primeira verificação natural dos casos armazenados antes de começar a busca mais extensiva requerida pelo raciocínio baseado em modelo.
3. Fornecer um registro de exemplos e exceções em uma base de casos que podem ser usados para guiar a inferência baseada em modelos.
4. Armazenar os resultados da inferência baseada em modelos para uso futuro.

Os métodos híbridos merecem a atenção tanto de pesquisadores como de programadores de aplicações. Entretanto, a construção de tais sistemas não é uma tarefa fácil e exige a solução de problemas, como determinar qual método de raciocínio aplicar em uma dada situação, decidir quando trocar o método de raciocínio, resolver diferenças entre os métodos de raciocínio e projetar representações que permitam que o conhecimento seja compartilhado.

A seguir, consideraremos o planejamento, ou a organização de procedimentos em potenciais soluções.

## 8.4 Planejamento

A tarefa de um planejador é encontrar uma sequência de ações que permita que o resolvedor de problemas, um sistema de controle, por exemplo, realize uma tarefa específica. O planejamento tradicional utiliza muito conhecimento, pois a criação de planos requer a organização de porções de conhecimento e planos parciais em um procedimento de solução. O planejamento desempenha um papel importante em sistemas especialistas, raciocinando sobre eventos que ocorrem ao longo do tempo. O planejamento tem muitas aplicações em fabricação, como, por exemplo, em controle de processos. Ele é importante, também, no projeto de um controlador de robô.

### 8.4.1 Introdução ao planejamento: robótica

Para começar, consideremos exemplos da robótica tradicional. (Muito da robótica moderna usa controle reativo em vez de planejamento deliberativo, como apresentam as seções 7.3.1 e 8.4.3.) Os passos de um plano de robô tradicional são compostos pelas *ações atômicas* de um robô. Em nosso exemplo, não descrevemos as ações atômica em micronível, por exemplo: “gire o sexto motor de passo uma volta para a frente”. Em vez disso, especificamos ações em um nível mais alto, por exemplo, por seus efeitos sobre um mundo. Por exemplo, um robô do mundo de blocos poderia incluir ações como “pegue o objeto a” ou “vá para a posição x”. O microcontrole está incorporado nessas ações de nível mais alto.

Assim, uma sequência de ações para “vá pegar o bloco a na sala b” poderia ser:

1. largue o que estiver segurando no momento
2. vá para a sala b
3. se aproxime do bloco a
4. pegue o bloco a
5. saia da sala b
6. retorne para a posição inicial

Os planos são criados pela realização de uma busca por meio do espaço de ações possíveis até que a sequência necessária para realizar a tarefa seja descoberta. Esse espaço representa estados do mundo que são modificados pela aplicação de cada uma dessas ações. A busca termina quando o estado objetivo (a descrição apropriada do mundo) é produzido. Assim, muitas das questões da busca heurística, incluindo encontrar algoritmos A\*, são também apropriadas no planejamento.

Entretanto, o ato de planejar não depende da existência de um robô real para executar os planos. Nos anos iniciais do planejamento por computador (década de 1960), planos inteiros eram formulados antes de o robô executar a sua primeira ação. Assim, os planos eram concebidos sem a presença de um robô! Mais recentemente, com a implementação de dispositivos sensores e reativos sofisticados, a pesquisa concentrou-se no sequenciamento mais integrado de planos e ações.

O planejamento tradicional se baseia em técnicas de busca e levanta uma série de questões interessantes. Por exemplo, a descrição dos estados do mundo pode ser consideravelmente mais complexa do que nos exemplos anteriores de busca. Considere o número de predicados necessários para descrever salas, corredores e objetos no ambiente do robô. Não apenas devemos representar o mundo do robô; devemos, também, representar o efeito das ações atômicas sobre esse mundo. A descrição completa de cada estado do espaço do problema pode ser bastante grande.

Outra diferença no planejamento é a necessidade de caracterizar o que *não* é modificado por uma ação particular. Pegar um objeto muda (a) a posição do objeto e (b) o fato de que a mão do robô agora segura o objeto. Essa ação não muda (a) as posições das portas e salas e (b) as posições de outros objetos. A especificação do que é verdadeiro em um estado do mundo e *exatamente* o que é mudado pela execução de uma ação se tornou conhecida como *problema do enquadramento* (McCarthy, 1980; McCarthy e Hayes, 1969). Conforme aumenta a complexidade do espaço do problema, a questão de rastrear as mudanças que ocorrem devido à cada ação e às características de uma

descrição de estado que permanecem imutáveis se torna mais importante. Apresentamos duas soluções para lidar com o problema do enquadramento, mas, como veremos, nenhuma delas é totalmente satisfatória.

Outras questões importantes são a geração de planos, o armazenamento e a generalização de bons planos, a recuperação de falhas imprevistas em planos (parte do mundo pode não ser como o esperado, talvez por ter sido acidentalmente deslocado de sua posição antecipada) e a manutenção da consistência entre o mundo e um modelo do mundo interno do programa.

Nos exemplos desta seção, limitamos o mundo do nosso robô a um conjunto de blocos sobre uma mesa e as ações do robô a um braço que pode empilhar, desempilhar ou mover os blocos sobre a mesa. Na Figura 8.18, temos cinco blocos, rotulados como **a**, **b**, **c**, **d**, **e**, sobre uma mesa. Todos os blocos são cubos de mesmo tamanho, e pilhas de blocos, como na figura, têm um bloco diretamente sobre outro. O braço do robô tem uma garra que pode agarrar qualquer bloco livre (que não tenha um bloco sobre ele) e movê-lo para qualquer posição sobre a mesa, ou colocá-lo sobre qualquer outro bloco (sem outro bloco sobre ele).

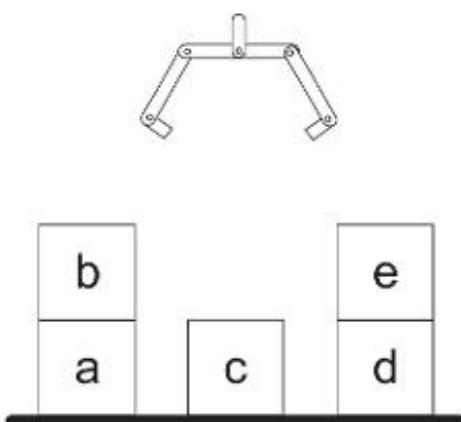
O braço do robô pode realizar as seguintes tarefas (**U**, **V**, **W**, **X**, **Y** e **Z** são variáveis):

<b>ir_para(X,Y,Z)</b>	Ir para a posição descrita pelas coordenadas <b>X</b> , <b>Y</b> e <b>Z</b> . Essa posição pode estar implícita no comando <b>pega(W)</b> , onde o bloco <b>W</b> está na posição <b>X</b> , <b>Y</b> , <b>Z</b> .
<b>pega(W)</b>	Pegar o bloco <b>W</b> na sua posição atual. Assume-se que o bloco <b>W</b> esteja livre, que a garra esteja vazia nesse momento e que o computador saiba a posição do bloco <b>W</b> .
<b>larga(W)</b>	Colocar o bloco <b>W</b> em uma posição sobre a mesa e registrar a nova posição de <b>W</b> . Nesse momento, a garra deve estar segurando <b>W</b> .
<b>empilha(U,V)</b>	Colocar o bloco <b>U</b> sobre o bloco <b>V</b> . A garra deve estar segurando <b>U</b> , e <b>V</b> deve estar livre.
<b>desempilha(U,V)</b>	Remover o bloco <b>U</b> de cima de <b>V</b> . <b>U</b> deve estar livre, <b>V</b> deve ter o bloco <b>U</b> sobre ele e a garra deve estar vazia antes que esse comando seja executado.

O estado do mundo é descrito por um conjunto de predicados e relacionamentos entre predicados:

<b>posição(W,X,Y,Z)</b>	O bloco <b>W</b> está nas coordenadas <b>X</b> , <b>Y</b> , <b>Z</b> .
<b>sobre(X,Y)</b>	O bloco <b>X</b> está imediatamente sobre o bloco <b>Y</b> .
<b>livre(Y)</b>	O bloco <b>X</b> não tem nenhum bloco sobre ele.
<b>agarra(X)</b>	O braço do robô segura o bloco <b>X</b> .
<b>agarra( )</b>	A garra do robô está vazia.
<b>sobre_a_mesa(W)</b>	O bloco <b>W</b> está sobre a mesa.

**Figura 8.18** O mundo de blocos.



`sobre_a_mesa(W)` é uma forma concisa do predicado `posição(W,X,Y,Z)`, onde `Z` é o nível da mesa. Da mesma forma, `sobre(X,Y)` indica que o bloco `X` está localizado com o seu fundo coincidente com o topo do bloco `Y`. Podemos simplificar bastante as descrições do mundo fazendo com que o computador registre a posição(`X,Y,Z`) atual de cada bloco, e rastreando os seus movimentos para novas posições. Com essa suposição de posicionamento, o comando `ir_para` se torna desnecessário; um comando como `pega(X)` ou `empilha(X)` implicitamente contém a posição de `X`.

O mundo de blocos da Figura 8.18 pode ser agora representado pelo conjunto de predicados a seguir. Chamamos essa coleção de predicados **ESTADO 1** no exemplo a seguir. Como os predicados que descrevem o estado do mundo para a Figura 8.18 são todos verdadeiros ao mesmo tempo, a descrição total do estado é a conjunção ( $\wedge$ ) de todos esses predicados.

#### ESTADO 1

<code>sobre_a_mesa(a).</code>	<code>sobre(b, a).</code>	<code>livre(b).</code>
<code>sobre_a_mesa(c).</code>	<code>sobre(e, d).</code>	<code>livre(c).</code>
<code>sobre_a_mesa(d).</code>	<code>agarra( ).</code>	<code>livre(e).</code>

A seguir, uma série de relações de verdade (no sentido declarativo) ou regras para execução (no sentido procedural) é criada para `livre(X)`, `sobre_a_mesa(X)` e `agarra( )`:

1.  $(\forall X) (\text{livre}(X) \leftarrow \neg(\exists Y) (\text{sobre}(Y, X)))$
2.  $(\forall Y) (\forall X) \neg(\text{sobre}(Y, X) \leftarrow (\text{sobre\_a\_mesa}(Y)))$
3.  $(\forall Y) \text{agarra}( ) \leftrightarrow \neg(\text{agarra}(Y))$

A primeira declaração afirma que, se o bloco `X` estiver livre, não existe qualquer bloco `Y` tal que `Y` esteja sobre `X`. Interpretado de modo procedural, é o mesmo que dizer “para tornar o bloco `X` livre remova qualquer bloco `Y` que esteja sobre `X`”.

Agora, projetamos regras para operar sobre estados e produzir novos estados. Fazendo isso, impomos novamente uma semântica procedural a uma representação similar à lógica de predicados. Os operadores (`pega`, `larga`, `empilha`, `desempilha`) são:

4.  $(\forall X) (\text{pega}(X) \rightarrow (\text{agarra}(X) \leftarrow (\text{agarra}( ) \wedge \text{livre}(X) \wedge \text{sobre\_a\_mesa}(X))))$ .
5.  $(\forall X) (\text{larga}(X) \rightarrow ((\text{agarra}( ) \wedge \text{sobre\_a\_mesa}(X) \wedge \text{livre}(X)) \leftarrow \text{agarra}(X)))$ .
6.  $(\forall X) (\forall Y) (\text{empilha}(X, Y) \rightarrow ((\text{sobre}(X, Y) \wedge \text{agarra}( ) \wedge \text{livre}(X)) \leftarrow (\text{livre}(Y) \wedge \text{agarra}(X))))$ .
7.  $(\forall X) (\forall Y) (\text{desempilha}(X, Y) \rightarrow ((\text{livre}(Y) \wedge \text{agarra}(X)) \leftarrow (\text{sobre}(X, Y) \wedge \text{livre}(X) \wedge \text{agarra}( ))))$ .

Considere a quarta regra: para todo bloco `X`, `pega(X)` significa `agarra(X)` se a mão estiver vazia e `X` estiver livre. Note a forma dessas regras:  $A \rightarrow (B \leftarrow C)$ . Isso diz que o operador `A` produz um novo predicado `B` quando a condição `C` for verdadeira. Usamos essas regras para gerar novos estados em um espaço. Isto é, se o predicado `C` é verdadeiro em um estado, então `B` é verdadeiro no seu estado filho. Em outras palavras, o operador `A` pode ser usado para criar um novo estado descrito pelo predicado `B` quando o predicado `C` for verdadeiro. Como abordagens alternativas para se gerar esses operadores estão STRIPS (Nilsson, 1980, e Seção 8.4.2) e a função faça de Rich e Knight (1991).

Mas, primeiro, precisamos tratar do *problema do enquadramento* antes de podermos usar essas relações de regras para gerar novos estados do mundo de blocos. Os *axiomas de enquadramento* são regras para dizer quais predicados que descrevem um estado *não* são modificados por aplicações de regras e são, por isso, deixados intacdos para ajudar a descrever o novo estado do mundo. Por exemplo, se aplicarmos o operador `pega` bloco `b` na Figura 8.18, então todos os predicados relativos aos demais blocos permanecem verdadeiros no estado filho. Para o nosso mundo de blocos, podemos especificar várias dessas regras de enquadramento:

8.  $(\forall X) (\forall Y) (\forall Z) (\text{desempilha}(Y, Z) \rightarrow (\text{sobre\_a\_mesa}(X) \leftarrow \text{sobre\_a\_mesa}(X)))$ .
9.  $(\forall X) (\forall Y) (\forall Z) (\text{empilha}(Y, Z) \rightarrow (\text{sobre\_a\_mesa}(X) \leftarrow \text{sobre\_a\_mesa}(X)))$ .

Essas duas regras afirmam que `sobre_a_mesa` não é afetado pelos operadores `empilha` e `desempilha`. Isso é verdadeiro mesmo quando `X` e `Z` forem idênticos; se `Y = Z`, então 6 ou 7 não será verdadeiro.

Outros axiomas de enquadramento afirmam que `sobre` e `livre` são afetados pelos operadores `empilha` e `desempilha` apenas quando essa relação `sobre` em particular está desempilhada ou quando uma relação `livre` está empilhada. Assim, no nosso exemplo, `sobre(b,a)` não é afetado por `desempilha(c,d)`.

De modo semelhante, os axiomas de enquadramento afirmam que as relações `livre(X)` não são afetadas por `agarra(Y)` mesmo quando `X = Y` ou `agarra( )` são verdadeiras. Outros axiomas declaram que `agarra` não afeta relações `sobre(X,Y)`, mas afeta apenas relações `sobre_a_mesa(X)` onde `X` encontra-se agarrado. Vários outros axiomas precisam ser especificados para o nosso exemplo.

Juntos, esses operadores e axiomas de enquadramento definem um espaço de estados, como ilustrado pelo operador `desempilha`. `desempilha(X,Y)` requer que três condições sejam verdadeiras simultaneamente: `sobre(X,Y)`, `agarra( )` e `livre(X)`. Quando essas condições são satisfeitas, são produzidos os novos predicados `agarra(X)` e `livre(Y)` pela aplicação do operador `desempilha`. Vários outros predicados também verdadeiros para ESTADO 1 permanecerão verdadeiros em ESTADO 2. Esses estados são preservados para o ESTADO 2, pelos axiomas de enquadramento. Produzimos, agora, os nove predicados que descrevem o ESTADO 2 pela aplicação do operador `desempilha`, e os axiomas de enquadramento para os nove predicados de ESTADO 1, cujo resultado final `desempilha` o bloco e:

#### ESTADO 2

<code>sobre_a_mesa(a).</code>	<code>sobre(b, a).</code>	<code>livre(b).</code>
<code>sobre_a_mesa(c).</code>	<code>livre(c).</code>	<code>livre(d).</code>
<code>sobre_a_mesa(d).</code>	<code>agarra(e).</code>	<code>livre(e).</code>

Resumindo:

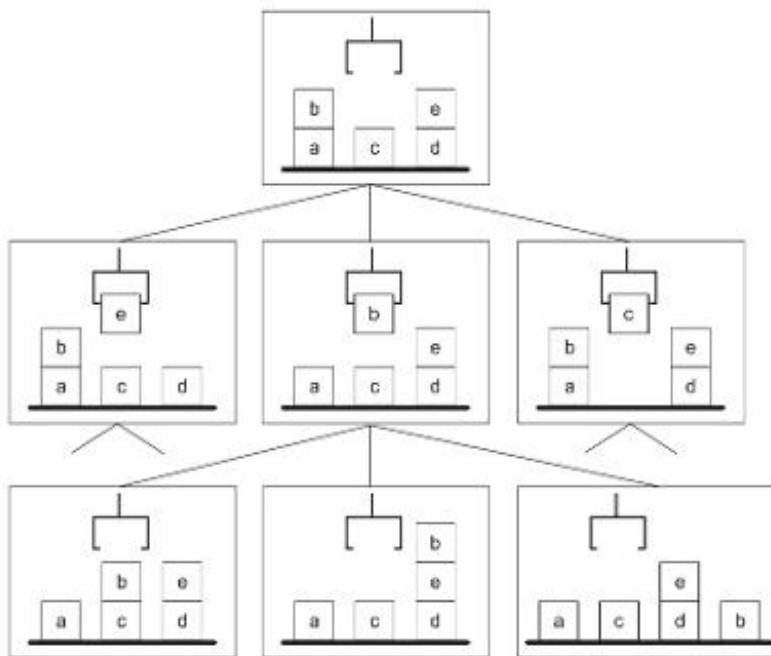
1. O planejamento pode ser visto como uma busca em espaço de estados.
2. Novos estados são produzidos por operadores gerais, como `empilha` e `desempilha`, além de regras de enquadramento.
3. As técnicas de busca em grafo podem ser aplicadas para encontrar um caminho que vá do estado inicial até o estado objetivo. As operações sobre esse caminho constituem um plano.

A Figura 8.19 mostra um exemplo de um espaço de estados em que a busca é realizada pela aplicação de operadores como aqueles descritos anteriormente. Se uma descrição de objetivo for acrescentada a esse processo de solução de problema, então um plano pode ser visto como um conjunto de operadores que produz um caminho que leva do estado atual desse grafo até o objetivo (ver Seção 3.1.2).

Essa caracterização do problema de planejamento define as suas raízes teóricas na busca de espaço de estados, bem como na representação por cálculo de predicados e em inferência. Entretanto, é importante notar quão complexa pode ser essa forma de solução. Em particular, o uso de regras de enquadramento para calcular o que permanece imutável entre estados pode aumentar exponencialmente o processo de busca, como pode ser visto pela complexidade do problema de blocos muito simples. Na verdade, quando um novo descritor de predicho é introduzido, para cor, forma ou tamanho, devem ser definidas novas regras de enquadramento para relacioná-lo com todas as ações apropriadas!

Essa discussão assume, também, que os subproblemas que constituem uma tarefa são independentes e podem, por isso, ser resolvidos em uma ordem arbitrária. Entretanto, isso é muito raro em domínios de problema interessantes e/ou complexos, onde as pré-condições e ações necessárias para alcançar um subobjetivo muitas vezes podem conflitar com as pré-condições e ações necessárias para alcançar outro subobjetivo. A seguir, ilustramos esses problemas e discutimos uma abordagem para o planejamento que nos auxilia muito a lidar com a complexidade.

**Figura 8.19** Parte do espaço de estados para uma parte do mundo de blocos.



#### 8.4.2 Usando macros para planejamento: STRIPS

O *STRIPS*, desenvolvido pelo atual SRI International, é uma sigla para STanford Research Institute Planning System, ou Sistema de Planejamento do Instituto de Pesquisa de Stanford (Fikes e Nilsson, 1971, Fikes et al., 1972). Esse controlador foi usado para guiar o robô *SHAKY* no início dos anos 1970. O STRIPS trata o problema de se representar e implementar eficientemente as operações de um planejador. Ele aborda o problema de subobjetivos conflitantes e fornece um modelo inicial de aprendizado; os planos bem-sucedidos são armazenados e generalizados como *macro-operadores*, que podem ser usados em situações futuras similares. No restante desta seção, apresentamos uma versão do estilo de planejamento do STRIPS e as *tabelas triangulares*, estrutura de dados usada para organizar e armazenar as macro-operações.

Usando o exemplo de blocos, os quatro operadores, pega, larga, empilha e desempilha, são representados como triplas de descrições. O primeiro elemento da tripla é o conjunto de *pré-condições* (P), ou condições que o mundo deve satisfazer para que um operador seja aplicado. O segundo elemento da tripla é a lista *adiciona* (A), ou as adições à descrição de estado que são resultado da aplicação do operador. Finalmente, existe uma lista *exclui* (E), ou os itens que são removidos de uma descrição de estado para criar o novo estado, quando o operador é aplicado. Essas listas eliminam a necessidade de axiomas de enquadramentos separados. Podemos representar os quatro operadores da seguinte forma:

pega(X)                    P: agarra( )  $\wedge$  livre(X)  $\wedge$  sobre\_a\_mesa(X)

A: agarra(X)

E: sobre\_a\_mesa(X)  $\wedge$  agarra( )

larga(X)                    P: agarra(X)

A: sobre\_a\_mesa(X)  $\wedge$  agarra( )  $\wedge$  livre(X)

E: agarra(X)

	P: livre(Y) $\wedge$ agarra(X)
empilha(X,Y)	A: sobre(X,Y) $\wedge$ agarra( ) $\wedge$ livre(X)
	E: livre(Y) $\wedge$ agarra(X)
	P: livre(X) $\wedge$ agarra( ) $\wedge$ sobre(X,Y)
desempilha(X,Y)	A: agarra(X) $\wedge$ livre(Y)
	E: agarra( ) $\wedge$ sobre(X,Y)

O importante nas listas adiciona e exclui é que elas especificam *tudo* o que é necessário para satisfazer os axiomas de enquadramento! Existe certa redundância na abordagem das listas adiciona e exclui. Por exemplo, em desempilha, a *adição* de agarra(X) poderia implicar a *exclusão* de agarra( ). Mas o ganho dessa redundância é que todo o descritor de estado que não seja mencionado por *adiciona* ou *exclui* permanece o mesmo na descrição do novo estado.

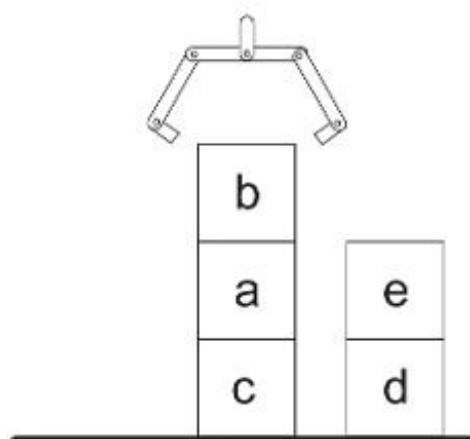
Um ponto fraco relacionado com a abordagem das listas pré-condição-adiciona-exclui é que não usamos mais um processo de prova de teorema para produzir (por inferência) os novos estados. Entretanto, isso não é um problema sério, pois, a partir de provas da equivalência das duas abordagens, podemos garantir a correção do método da pré-condição-adiciona-exclui. A abordagem por listas de pré-condição-adiciona-exclui pode ser usada para produzir os mesmos resultados que produzimos no exemplo anterior com regras de inferência e axiomas de enquadramento. A busca em espaço de estados, como na Figura 8.19, seria idêntica em ambas as abordagens.

Uma série de outros problemas inerentes ao planejamento não é resolvida por nenhuma das duas abordagens apresentadas até aqui. Para resolver um objetivo, muitas vezes o dividimos em subproblemas, por exemplo, desempilha(e,d) e desempilha(b,a). A tentativa de resolver esses subobjetivos independentemente pode causar problemas se as ações necessárias para alcançar um objetivo desfizerem o outro. Se a suposição de *linearidade* (independência) de subobjetivos for falsa, os subobjetivos podem ser incompatíveis. A não linearidade de um espaço de planejamento/ação pode tornar as buscas de solução desnecessariamente difíceis e até impossíveis.

Mostramos, agora, um exemplo muito simples de um subobjetivo incompatível usando o estado inicial ESTADO 1 da Figura 8.18. Suponha que o objetivo do plano seja o ESTADO G, como mostra a Figura 8.20, com sobre(b,a)  $\wedge$  sobre(a,c) e os blocos d e e permanecendo como no ESTADO 1. Podemos notar que uma das partes do objetivo conjunto sobre(b,a)  $\wedge$  sobre(a,c) é verdadeira no ESTADO 1, isto é, sobre(b,a). Essa parte já satisfeita do objetivo deve ser desfeita antes que o segundo subobjetivo, sobre(a,c), possa ser realizado.

A representação por *tabela triangular* (Fikes e Nilsson, 1971; Nilsson, 1980) visa aliviar algumas dessas anomalias. Uma tabela triangular é uma estrutura de dados para organizar sequências de ações, incluindo subob-

**Figura 8.20** Estado objetivo do mundo de blocos.



jetivos potencialmente incompatíveis, dentro de um plano. Ela aborda o problema de subobjetivos conflitantes dentro de macroações, representando a interação global de sequências de operações. Uma tabela triangular relaciona as pré-condições de uma ação com as pós-condições, as listas combinadas de adição e exclusão, de ações que a precedem.

As tabelas triangulares são usadas para determinar quando um macro-operador pode ser usado na construção de um plano. Armazenando esses macro-operadores e reutilizando-os, o STRIPS aumenta a eficiência de sua busca de planejamento. Na verdade, podemos generalizar um macro-operador usando nomes de variáveis para substituir os nomes de blocos em um exemplo em particular. Então, podemos chamar a nova macro generalizada para podar a busca. No Capítulo 10, com a nossa apresentação sobre aprendizado em ambientes simbólicos, discutimos técnicas para generalizar macro-operações.

O reuso de macro-operadores ajuda também a resolver o problema de subobjetivos conflitantes. Como ilustra o próximo exemplo, uma vez que o planejador tenha desenvolvido um plano para objetivos da forma  $\text{empilha}(X,Y) \wedge \text{empilha}(Y,Z)$ , ele pode armazenar e reusar esse plano. Isso elimina a necessidade de quebrar o objetivo em subobjetivos e evita complicações decorrentes.

A Figura 8.21 apresenta uma tabela triangular para a macroação  $\text{empilha}(X,Y) \wedge \text{empilha}(Y,Z)$ . Essa macroação pode ser aplicada a estados em que  $\text{sobre}(X,Y) \wedge \text{livre}(X) \wedge \text{livre}(Z)$  seja verdadeiro. Essa tabela triangular é adequada para iniciar ESTADO 1 com  $X = b$ ,  $Y = a$  e  $Z = c$ .

As ações atômicas do plano são registradas ao longo da diagonal. Elas são as quatro ações, pegar, largar, empilhar e desempilhar, discutidas anteriormente nesta seção. O conjunto de pré-condições de cada uma dessas ações se encontra na linha que precede essa ação, e as pós-condições de cada ação estão na coluna abaixo dela. Por exemplo, a linha 5 lista as pré-condições para pegar( $X$ ) e a coluna 6 lista as pós-condições (as listas adiciona e exclui) de pegar( $X$ ). Essas pós-condições são colocadas na linha da ação que as usa como pré-condições, organizando-as de uma maneira relevante para outras ações. O propósito da tabela triangular é intercalar adequadamente as pré-condições e as pós-condições de cada uma das ações menores que compõem o objetivo maior. Dessa forma, as tabelas triangulares tratam questões envolvendo não linearidade no planejamento, no nível dos macro-operadores; os Planejadores de Ordem Parcial (Russell e Norvig, 1995) e outras abordagens também tratam dessas questões.

**Figura 8.21** Tabela triangular, adaptada de Nilsson (1971).

	1	agarra( ) livre(X) sobre(X,Y)	desempilha(X,Y)				
2		agarra(X)	larga(X)				
3	sobre_a_mesa(Y)	livre(Y)	agarra( )	pega(Y)			
4	livre(Z)			agarra(Y)	empilha(Y,Z)		
5		livre(X) sobre_a_mesa(X)		agarra( )	pega(X)		
6				livre(Y)	agarra(X)	empilha(X,Y)	
7				sobre(Y,Z)		sobre(X,Y) livre(X) agarra( )	
	1	2	3	4	5	6	7

Uma vantagem das tabelas triangulares é que elas podem oferecer auxílio na tentativa de se recuperar de acontecimentos inesperados, como quando um bloco está ligeiramente fora do lugar, ou no caso de acidentes, como a derrubada de um bloco. Muitas vezes, um acidente pode requerer que se retorne vários passos antes que o plano possa ser retomado. Quando algo errado acontece com uma solução, o planejador pode andar para trás nas linhas e nas colunas da tabela triangular para verificar o que ainda é verdadeiro. Uma vez que o planejador tenha percebido o que ainda é verdadeiro nas linhas e colunas, ele sabe, então, qual deve ser o próximo passo quando a solução maior for reiniciada. Isso é formalizado com a noção de um *núcleo* (ou *kernel*).

O *n*-ésimo núcleo é a interseção de todas as linhas abaixo da *n*-ésima linha e todas as colunas à esquerda da *n*-ésima coluna, incluindo essa coluna. Na Figura 8.21, ressaltamos o terceiro núcleo em negrito. Ao executar um plano representado em uma tabela triangular, a *i*-ésima operação (isto é, a operação na linha *i*) pode ser realizada apenas se todos os predicados contidos no *i*-ésimo núcleo forem verdadeiros. Isso oferece uma maneira direta de se verificar se um passo pode ser executado e, também, permite que o sistema se recupere sistematicamente de qualquer ruptura de um plano. Dada uma tabela triangular, encontramos e executamos a ação de número mais alto cujo núcleo esteja habilitado. Com isso, não apenas podemos voltar em um plano, mas também podemos pular para a frente em um plano, devido a um evento inesperado.

As condições na coluna mais à esquerda são as pré-condições para a macroação como um todo. As condições na linha inferior são as condições acrescentadas ao mundo pelo macro-operador. Assim, uma tabela triangular pode ser armazenada como um *macro-operador* com seu próprio conjunto de pré-condições e pós-condições, ou listas de adição e de exclusão.

É claro que a abordagem por tabela triangular perde um pouco da semântica dos modelos anteriores de planejamento. Note, por exemplo, que retemos apenas aquelas pós-condições de um ato que são também pré-condições de outros atos mais adiante. Assim, se quisermos garantir a correção do processo, é desejável realizarmos uma verificação adicional das tabelas triangulares, talvez com informação adicional que possa permitir que sejam compostas sequências de tabelas triangulares.

O uso de macro-operadores no planejamento causa também outros problemas. Conforme aumenta o seu número, o planejador tem um número maior de operações mais poderosas para usar, diminuindo o tamanho do espaço de estados que deve ser buscado. Infelizmente, a cada passo da busca, todos esses operadores devem ser examinados. O reconhecimento de padrões necessário para determinar quando uma macro-operação pode ser aplicada pode aumentar consideravelmente o tempo do processo de busca, muitas vezes neutralizando os ganhos conseguidos ao armazenar macro-operações. Os problemas de determinar quando uma macro-operação deve ser armazenada e a melhor forma de determinar o próximo operador a ser usado permanece sendo um tema de muita pesquisa. Na próxima seção, descrevemos um algoritmo por meio do qual muitos subobjetivos podem ser simultaneamente satisfeitos: o planejamento teleorreativo (do inglês *teleo-reactive*).

#### 8.4.3 Planejamento teleorreativo (Nilsson, 1994; Benson, 1995)

Desde o trabalho inicial em planejamento descrito na seção anterior (Fikes e Nilsson, 1971), tem havido uma série de avanços significativos. Grande parte desse trabalho tem sido feita em planejamento independente de domínio, mas recentemente tem se tornado importante o planejamento específico para um domínio, onde mecanismos distribuídos de percepção-ação são empregados. Nas próximas seções, descrevemos um sistema independente de domínio, o planejamento teleorreativo, bem como um planejador dependente de domínio, o Burton, da NASA. Para ver um resumo das primeiras duas décadas de pesquisa em planejamento, recomendamos (Allen et al., 1990).

O planejamento *teleorreativo* (TR) foi proposto inicialmente por Nilsson (1994) e Benson (1995) na Universidade de Stanford. O planejamento TR oferece uma arquitetura de propósito geral, com aplicação em vários domínios, onde o controle de subsistemas complexos deve ser coordenado para alcançar um objetivo de nível mais alto. Assim, ele combina a abordagem de cima para baixo das arquiteturas de controle hierárquicas com uma assistência “baseada em agentes” (ver Seção 7.4) ou de baixo para cima. O resultado é um sistema que pode resolver problemas complexos por meio da coordenação de agentes simples, específicos para uma tarefa. A justificativa para

essa abordagem é que agentes simples têm a vantagem de trabalhar em espaços de problema menores e mais restritos. O controlador de mais alto nível, por outro lado, pode tomar decisões mais globais sobre o sistema como um todo, por exemplo, como o resultado atual de uma decisão puramente local pode afetar o resultado da tarefa geral de solução do problema.

O controle telereativo combina aspectos do controle realimentado e do planejamento de ações discretas. Os programas telereativos sequenciam a execução de ações que foram montadas formando um plano orientado a objetivo. Diferentemente dos ambientes de planejamento baseados em IA mais tradicionais (Weld, 1994), não são feitas suposições de que as ações são discretas e não podem ser interrompidas e que todos os efeitos de uma ação são totalmente previsíveis. Ao contrário, as teleações podem ser sustentadas por extensos períodos de tempo, isto é, elas são executadas enquanto as pré-condições das ações são satisfeitas e o objetivo associado ainda não tiver sido alcançado. Nilsson (1994) se refere a esse tipo de ação como *duradoura*. Ações duradouras podem ser interrompidas quando uma outra ação mais próxima do objetivo de alto nível for ativada. Um ciclo curto de perceber-reagir assegura que, quando o ambiente muda, as ações de controle rapidamente mudam para refletir o novo estado da solução do problema.

As sequências de ações telereativas são representadas por uma estrutura de dados chamada de árvore TR, como vemos na Figura 8.22. Uma árvore TR é descrita por um conjunto de pares de condição → ação (ou regras de produção, Seção 6.3). Por exemplo:

$$C_0 \rightarrow A_0$$

$$C_1 \rightarrow A_1$$

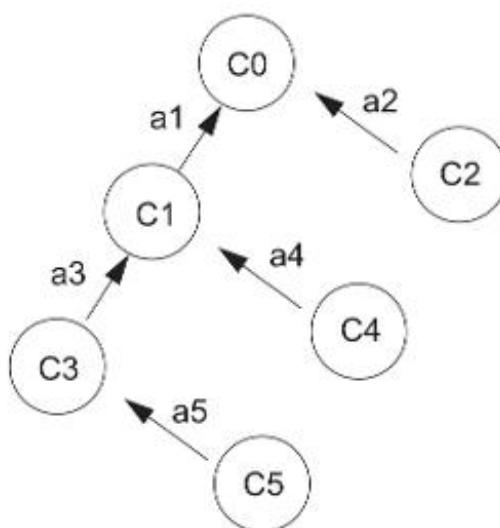
$$C_2 \rightarrow A_2$$

$$\dots$$

$$C_n \rightarrow A_n$$

onde  $C_i$  são as condições e  $A_i$  são as ações associadas. Nos referimos a  $C_0$  como o objetivo de alto nível da árvore e  $A_0$  como a ação nula, indicando que não precisa ser feito mais nada, uma vez que o objetivo de alto nível foi alcançado. A cada execução do sistema telereativo, cada  $C_i$  é avaliada, partindo do topo das regras e seguindo em direção à base ( $C_0, C_1, \dots, C_n$ ), até que a primeira condição verdadeira seja encontrada. A ação associada a essa condição verdadeira é, então, executada. O ciclo de avaliação é repetido a uma frequência que se aproxima da reatividade do controle baseado em circuito.

**Figura 8.22** Uma árvore TR simples mostrando regras de condição-ação que suportam um objetivo de alto nível, de Klein et al. (2000).



As produções  $C_i \rightarrow A_i$  são organizadas de tal forma que cada ação  $A_i$ , se executada continuamente sob condições normais, em algum momento, tornará verdadeira uma condição localizada mais acima na árvore de regras TR (Figura 8.22). A execução da árvore TR pode ser vista como adaptativa, pois se um evento inesperado no ambiente de controle inverter o efeito de ações anteriores, a execução TR recuará para a condição da regra de nível mais baixa que reflete essa condição. A partir desse ponto, a execução será reiniciada procurando satisfazer todos os objetivos de nível mais alto. De modo semelhante, se inadvertidamente acontecer algo de bom, a execução TR será oportunista no sentido de que o controle automaticamente mudará para a ação dessa condição verdadeira.

As árvores TR podem ser construídas com um algoritmo de planejamento que emprega métodos comuns de IA para a redução de objetivos. Partindo do objetivo de nível mais alto, o planejador realiza uma busca sobre as ações cujos efeitos incluem a satisfação desse objetivo. As pré-condições dessas ações geram um novo conjunto de subobjetivos e esse procedimento é recorrente. Esse processo é encerrado quando as pré-condições dos nós folhas forem satisfeitas pelo estado atual do ambiente. Assim, o algoritmo de planejamento regressa do objetivo de nível mais alto, por meio da redução de objetivos, para o estado atual. É claro que as ações podem ter efeitos colaterais e o planejador deve ter o cuidado de verificar que uma ação, em qualquer nível, não altere condições que sejam necessárias como pré-condições de ações de um nível mais alto na árvore TR. A redução de objetivos está acomplida com a satisfação de restrições, onde é usada uma série de estratégias para reordenar ações visando eliminar possíveis violações de restrições.

Assim, os algoritmos de planejamento TR são usados para construir planos cujos nós folhas são satisfeitos pelo estado atual do ambiente do problema. Eles normalmente não constroem planos completos, isto é, planos que podem iniciar em qualquer estado do mundo, porque esses planos são em geral grandes demais para serem armazenados e executados eficientemente. Este último ponto é importante porque, algumas vezes, um evento ambiental inesperado pode levar o mundo para um estado no qual nenhuma pré-condição de ação da árvore TR pode ser satisfeita, sendo necessária alguma forma de replanejamento. Isso normalmente é feito pela reativação do planejador TR.

Benson (1995) e Nilsson (1994) usaram o planejamento telereativo em uma série de domínios de aplicação, incluindo o controle de agentes robóticos distribuídos e na construção de um simulador de voo. Klein et al. (1999, 2000) usaram um planejador telereativo para construir e testar uma arquitetura de controle portátil para a aceleração de um feixe de partículas carregadas. Esses últimos pesquisadores deram uma série de razões para o uso de um controlador telereativo no domínio dos controladores de feixe de partículas:

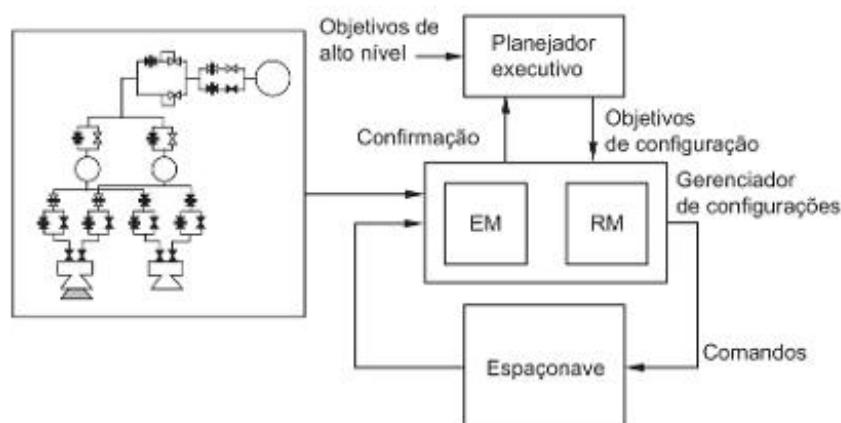
1. Os feixes de aceleradores e seu diagnóstico associado são geralmente dinâmicos e ruidosos.
2. A satisfação de objetivos de sintonia de um acelerador é, muitas vezes, afetada por processos estocásticos, ruptura RF ou por oscilações na fonte de feixe.
3. Muitas das ações exigidas para a sintonia são duradouras. Isso é particularmente verdadeiro para as operações de adaptação e otimização que precisam ser realizadas de forma contínua, até que certos critérios específicos sejam satisfeitos.
4. As árvores TR oferecem uma estrutura intuitiva para codificar planos de sintonia adquiridos de físicos especialistas em aceleradores. De fato, com muito pouca ajuda, os próprios físicos são capazes de desenvolver suas próprias árvores TR.

Mais detalhes dessas aplicações podem ser encontrados em uma consulta às referências. A seguir, revemos o exemplo da NASA de raciocínio baseado em modelo da Seção 8.3 para descrever algoritmos de controle/planejamento para o sistema de propulsão de veículos espaciais.

#### 8.4.4 Planejamento: um exemplo da NASA (Williams e Nayak)

Nesta seção, descrevemos como um planejador pode ser implementado no contexto de um raciocinador baseado em modelo. Continuamos com o exemplo da NASA de Williams e Nayak (1996b) introduzido na Seção 8.3.2. Livingstone é um gerenciador de configuração reativo que usa um modelo baseado em componentes do sistema de propulsão de uma espaçonave para determinar as ações de configuração, como mostra a Figura 8.23.

**Figura 8.23** Gerenciamento de configuração reativo baseado em modelo, de Williams e Nayak (1996b).

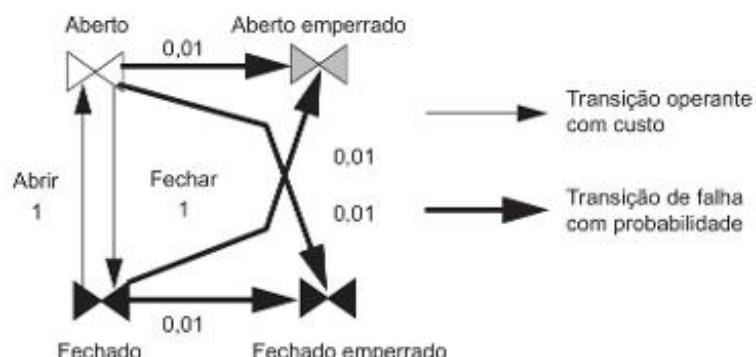


Cada componente de propulsão é modelado como um sistema de transição que especifica os comportamentos dos modos de operação e falha dos componentes, as transições operantes e de falha entre os modos e os custos e probabilidades de transições, como na Figura 8.24. Nela, aberto e fechado são os modos de operação normais, mas aberto emperrado e fechado emperrado são modos de falha. O comando abrir tem custo unitário e causa uma transição de modo de fechado para aberto, assim como o comando fechar tem transição e custo semelhantes. As transições de falha movimentam a válvula dos modos normais de operação para um dos modos de falha com probabilidade 0,01.

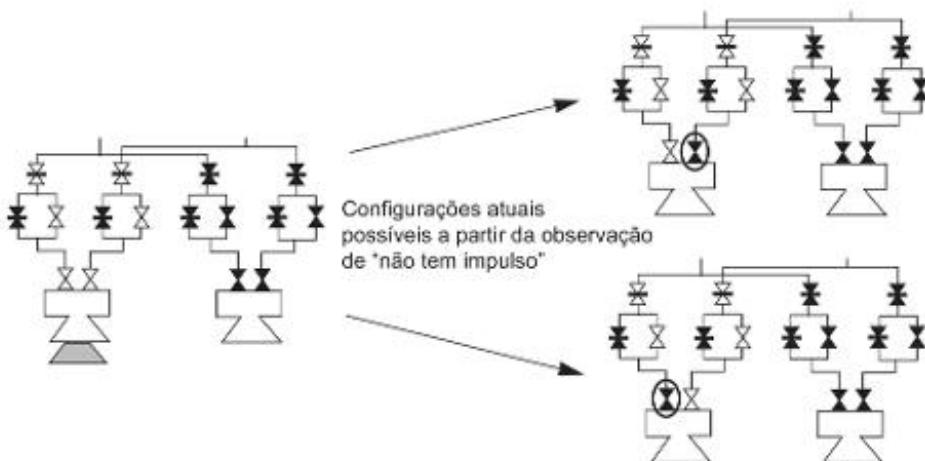
Os comportamentos dos modos são especificados usando fórmulas da lógica proposicional, mas as transições entre modos são especificadas usando fórmulas em uma lógica proposicional restrita, temporal. A lógica proposicional restrita, temporal, é adequada para modelar hardware digital, analógico, usando abstrações qualitativas (de Kleer e Williams, 1989; Weld e de Kleer, 1990) e software de tempo real usando os modelos de sistemas concorrentes reativos (Manna e Pnueli, 1992). O modelo do sistema de transição da espaçonave é uma composição dos seus sistemas de transição de componentes no qual o conjunto de configurações da espaçonave é o produto cruzado dos conjuntos de modos dos componentes. Supomos que os sistemas de transição de componentes operam de modo síncrono; ou seja, para cada transição da aeronave, cada um dos componentes realiza uma transição.

Um gerenciador de configuração baseado em modelos usa o seu modelo de sistema de transição tanto para identificar a configuração atual da espaçonave, chamada de *estimativa de modo EM*, quanto para movimentar a espaçonave para uma nova configuração que alcance os objetivos de configuração desejados, chamada de *recon-*

**Figura 8.24** O modelo do sistema de transição de uma válvula, de Williams e Nayak (1996a).



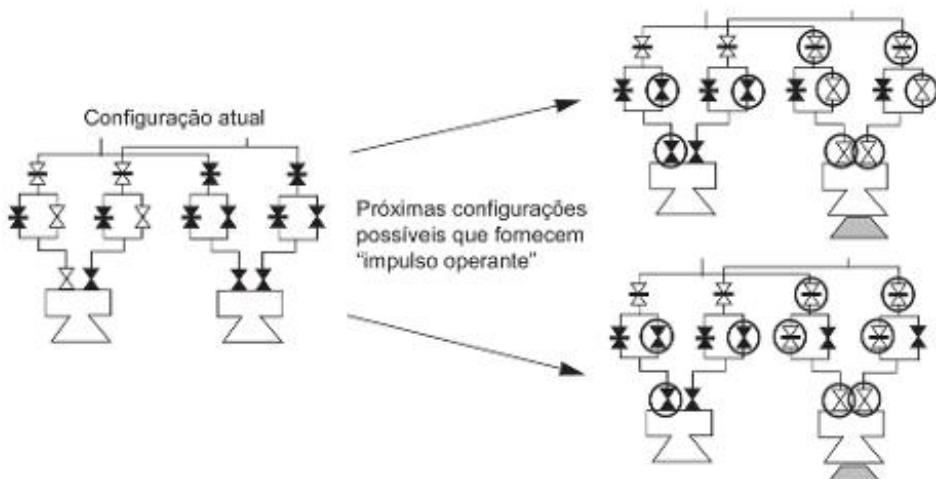
**Figura 8.25** Estimativa de modo (EM), de Williams e Nayak (1996a).



*figuração de modo, RM.* EM gera, de modo incremental, todas as transições da espaçonave a partir da configuração anterior, de modo que os modelos das configurações resultantes sejam consistentes com as observações atuais. Assim, na Figura 8.25, é mostrada uma situação na qual o motor esquerdo dispara normalmente no estado anterior, mas não é observado impulso no estado atual. A tarefa de EM é identificar as configurações resultantes da transição da espaçonave que explicam essa observação. A figura mostra duas transições possíveis que correspondem à falha de uma das válvulas do motor principal. As válvulas defeituosas, ou que estão no modo fechado emperrado, aparecem circuladas. Muitas outras transições, incluindo as pouco prováveis falhas duplas, podem também explicar essas observações.

O módulo RM determina os comandos que devem ser enviados para a espaçonave de modo que as transições resultantes coloquem a espaçonave em uma configuração que alcance no próximo estado a meta de configuração, como mostra a Figura 8.26. Essa figura mostra uma situação na qual o processo de identificação de modo identificou uma válvula defeituosa que leva ao propulsor esquerdo do motor principal. O módulo RM conclui, por raciocínio, que o impulso normal pode ser restaurado no próximo estado se um conjunto apropriado de válvulas que levem ao propulsor direito for aberto. A figura mostra duas das várias configurações que podem alcançar o obje-

**Figura 8.26** Reconfiguração de modo (RM), de Williams e Nayak (1996a).



tivo desejado, quando as válvulas assinaladas receberem o comando de mudar de estado. A transição para a configuração de cima tem um custo mais baixo, porque apenas as piroválvulas necessárias são disparadas (veja a discussão na Seção 8.3.2). As válvulas que levam ao propulsor esquerdo são desligadas para satisfazer a restrição que, no máximo, um propulsor pode disparar a cada vez. O uso de um modelo de espaçonave tanto em EM como em RM assegura que os objetivos de configuração sejam alcançados corretamente.

Tanto EM quanto RM são reativos (veja a Seção 6.4). EM infere a configuração atual a partir do conhecimento da configuração anterior e das observações atuais. RM considera apenas os comandos que alcancem o objetivo de configuração no próximo estado. Dados esses comprometimentos, é fundamental a decisão de modelar as transições dos componentes como sendo síncronas. Uma alternativa é modelar múltiplas transições de componentes por intercalação. Entretanto, a intercalação pode colocar uma distância arbitrária entre a configuração atual e uma configuração alvo, frustrando o desejo de limitar a inferência a um número de estados fixo e pequeno. Por isso, modelamos as transições dos componentes como sendo síncronas. Se as transições dos componentes de hardware e software subjacentes não forem síncronas, a suposição da modelagem do Livingstone é que todas as intercalações de transições são corretas e permitem alcançar a configuração desejada. Essa suposição é removida em Burton, um sucessor do Livingstone, cujo planejador determina uma sequência de ações de controle que produzem todas as transições desejadas (Williams e Nayak, 1997). A NASA vem usando a arquitetura Burton em uma missão chamada Tech Sat 21.

Para Livingstone, EM e RM não precisam gerar todas as transições e comandos de controle, respectivamente. Em vez disso, apenas as transições mais prováveis e um comando de controle ótimo são necessários. Isso pode ser gerado eficientemente considerando EM e RM como problemas de otimização combinatória. Nessa reformulação, EM rastreia, de forma incremental, as prováveis trajetórias da espaçonave, estendendo sempre as trajetórias que levam às configurações atuais com as transições mais prováveis. RM identifica, então, o comando com o menor custo esperado, que realiza a transição das prováveis configurações atuais para uma configuração que alcance o objetivo desejado. Resolvemos eficientemente esses problemas de otimização combinatória usando um algoritmo de busca pela melhor escolha, orientado por conflito. Veja Williams e Nayak (1996a, 1996b, 1997) para obter uma caracterização mais formal de EM e RM, bem como uma descrição detalhada dos algoritmos de busca e planejamento.

## 8.5 Epílogo e referências

A arquitetura para os sistemas especialistas baseados em regras é o sistema de produção. Se o produto final for guiado por dados ou por objetivo, o modelo para o software é a busca em grafo gerado por sistema de produção, conforme apresentado no Capítulo 6. Implementamos ambientes de desenvolvimento de sistemas especialistas baseados em sistema de produção em Java, Prolog e Lisp na Sala Virtual deste livro. As regras desses sistemas podem incluir medidas de certeza em uma forma limitada para o projeto de uma busca baseada em heurística.

Diversas referências complementam o material apresentado neste capítulo; recomendamos, especialmente, uma coleção de publicações originais sobre o MYCIN de Stanford intitulada *Rule-Based Expert Systems*, de Buchanan e Shortliffe (1984). Outros dos primeiros trabalhos sobre sistemas especialistas são descritos em Waterman (1968), Michie (1979), Patil et al. (1981), Clancy (1983) e Clancy e Shortliffe (1984a, 1984b). Para uma implementação robusta de busca baseada em regras, guiada por dados, recomendamos o software CLIPS, distribuído pela NASA. Uma excelente referência para o CLIPS é Giarratano e Riley (1989, 2005).

Outros livros importantes sobre a engenharia de conhecimento em geral são *Building Expert Systems*, de Hayes-Roth et al. (1984); *A Guide to Expert Systems*, de Waterman (1986); *Expert Systems: Concepts and Examples*, de Alty e Coombs (1984); *Expert Systems Technology: A Guide*, de Johnson e Keravnou (1985); *Expert Systems and Fuzzy Systems*, de Negoita (1985), *Introduction to Expert Systems*, de Jackson (1986), e *Expert Systems: Tools and Applications*, de Harmon et al. (1988). Veja também *Introduction to Expert Systems*, de Ignizio (1991);

*An Introduction to Expert Systems*, de Mockler e Dologite (1992); e *Expert Systems: Design and Development*, de John Durkin (1994), e os anais de conferências sobre aplicações de IA e sistemas especialistas, IEA/AIE (2004-2007).

Por causa da especificidade de domínio das soluções por sistemas especialistas, os estudos de casos são uma fonte importante de conhecimento na área. Entre os livros dessa categoria se incluem *Expert Systems: Techniques, Tools and Applications*, de Klahr e Waterman (1986); *Competent Expert Systems: A Case Study in Fault Diagnosis*, de Keravnou e Johnson (1986); *The CRI Directory of Expert Systems*, de Smart e Langeland-Knudsen (1986); *Developments in Expert Systems*, de Coombs (1984); e *Developing and Managing Expert Systems*, de Prerau (1990). Recomendamos, especialmente, *Expert Systems: Design and Development*, de Durkin (1994), por seu vasto número de sugestões práticas sobre a construção de sistemas.

Foram desenvolvidas, também, diversas técnicas de aquisição de conhecimento. Para obter mais informações sobre metodologias específicas veja: *Knowledge Acquisition: Principles and Guidelines*, de McGraw e Harbison-Briggs (1989), e *Knowledge Engineering*, de Chorafas (1990), bem como *An Introduction to Expert Systems*, de Mockler e Dologite (1992), e livros e anais de conferências sobre sistemas especialistas IEA/AIE (2004-2007).

O raciocínio baseado em casos é um ramo da pesquisa anterior do grupo de Schank em Yale e seu trabalho sobre roteiros (ver seções 7.1.3 e 7.1.4). *Case-Based Reasoning*, de Kolodner (1993), é uma excelente introdução à tecnologia de RBC. Outro trabalho relacionado a RBC, de Kolodner e seu grupo de pesquisa, é Kolodner (1987, 1991). Leake (1992, 1996) oferece uma discussão importante sobre explicações em sistemas baseados em casos. Atualmente, estão disponíveis vários produtos de software de RBC comerciais que dão suporte à tecnologia baseada em casos.

O raciocínio baseado em modelo tem o seu inicio na representação explícita da medicina, circuitos lógicos e outros domínios matemáticos, quase sempre usados para fins de instrução (de Kleer, 1975; Weiss et al., 1977; Brown e Burton, 1978; Brown e VanLehn, 1980; Genesereth, 1982; Brown et al., 1982). Outras questões de pesquisas modernas se encontram em *Readings in Model-based Diagnosis* (editado por Hamscher et al., 1992) e *Diagnosis Based on Description of Structure and Function* (Davis et al., 1982) e diagnóstico no contexto de falhas múltiplas (de Kleer e Williams, 1987, 1989). Skinner e Luger (1995) e outros escritores sobre arquiteturas de agentes (Russell e Norvig, 1995, 2003) descrevem os sistemas especialistas híbridos, onde as interações de múltiplas abordagens para a solução de problemas podem criar um efeito de sinergia com as vantagens de um sistema compensando as desvantagens de outros.

A seção sobre planejamento demonstra algumas das estruturas de dados e técnicas de busca usadas para planejadores de propósito geral. Outras referências incluem ABSTRIPS, ou especificação abstrata para relações do gerador STRIPS (Sacerdotti, 1974), o trabalho de Warren (1976) e NOAH, para o planejamento não linear ou hierárquico (Sacerdotti, 1975, 1977). Para obter informações sobre um planejador mais moderno, veja o planejamento *teleorreativo*, Seção 15.3 (Benson e Nilsson, 1995).

O metaplanejamento é uma técnica para raciocinar não apenas sobre o plano, mas também sobre o processo de planejamento. Isso pode ser importante em soluções por sistemas especialistas. Como referência, recomendamos Meta-DENDRAL para soluções DENDRAL (Lindsay et al., 1980) e Teiresias para soluções MYCIN (Davis, 1982). Sobre planos que interagem continuamente com o mundo, isto é, modelam um ambiente em transformação, veja McDermott (1978).

Outra linha de pesquisa é o *planejamento oportunista* usando quadros negros e o planejamento baseado em uma especificação orientada a objetos (Smoliar, 1985). Há várias revisões sobre pesquisas em planejamento no *Handbook of Artificial Intelligence* (Barr e Feigenbaum, 1989; Cohen e Feigenbaum, 1982) e na *Encyclopedia of Artificial Intelligence* (Shapiro, 1992). Em Russell e Norvig (1995), são apresentadas questões sobre não linearidades e o planejamento de ordem parcial. *Readings in Planning* (Allen et al., 1990) é relevante para este capítulo.

Nossa descrição e nossa análise do raciocínio baseado em modelo e algoritmos de planejamento da NASA foram tiradas de Williams e Nayak (1996a, 1996b, 1997). Agradecemos Williams e Nayak, bem como a AAAI Press, por permitirem citar suas pesquisas.

## 8.6 Exercícios

1. Na Seção 8.2, apresentamos um conjunto de regras para diagnosticar problemas em automóveis. Identifique possíveis engenheiros de conhecimento, especialistas no domínio e potenciais usuários finais para tal aplicação. Discuta as expectativas, habilidades e necessidades de cada um desses grupos.
2. A partir do Exercício 1, crie, em português ou em pseudocódigo, 15 regras *se... então...* (além daquelas descritas na Seção 8.2) para descrever relações nesse domínio. Crie um grafo para representar as relações entre essas 15 regras.
3. Considere o grafo do Exercício 2. Você recomendaria a busca guiada por dados ou guiada por objetivo? Busca em amplitude ou em profundidade? De que maneira as heurísticas poderiam ajudar na busca? Justifique suas respostas.
4. Tome outra área de interesse para projetar um sistema especialista. Repita os exercícios 1-3 para essa aplicação.
5. Implemente um sistema especialista usando um ambiente de desenvolvimento comercial. Existem vários desses ambientes disponíveis para computadores pessoais e máquinas maiores. Recomendamos, especialmente, o CLIPS da NASA (Giarratano e Riley, 1989) ou JESS, de Sandia National Laboratories.
6. Faça uma crítica ao ambiente que você usou para resolver o Exercício 5. Quais são seus pontos fortes e fracos? O que você melhoraria nele? Ele foi apropriado para o seu problema? Quais problemas são mais adequados para essa ferramenta?
7. Crie um sistema de raciocínio baseado em modelo para um dispositivo eletrônico simples. Combine vários desses dispositivos para compor um sistema maior. Você pode usar regras *se... então...* para caracterizar a funcionalidade do sistema.
8. Leia e comente o artigo *Diagnosis based on description of structure and function* (Davis et al., 1982).
9. Leia um dos primeiros artigos que usam raciocínio baseado em modelo para ensinar aritmética a crianças (Brown e Burton, 1978) ou técnicas de eletrônica (Brown e VanLehn, 1980). Comente essa abordagem.
10. Construa um raciocinador baseado em casos para uma aplicação de sua escolha. Uma das áreas poderia corresponder à seleção de cursos de ciências da computação e de engenharia para completar um curso de graduação ou de mestrado.
11. Use um software comercial (verifique na web) para construir o sistema de raciocínio baseado em casos do Exercício 10. Se não houver um software disponível, construa um sistema em Prolog, Lisp ou Java.
12. Leia e comente o artigo *Improving Human Decision Making through Case-Based Decision Aiding* (Kolodner, 1991).
13. Crie os *axiomas de enquadramento* restantes, necessários para os quatro operadores pegar, largar, empilhar e desempilhar, descritos nas regras 4 a 7 da Seção 8.4.
14. Use os operadores e axiomas de enquadramento da questão anterior para gerar o espaço de busca da Figura 8.19.
15. Mostre como as listas adiciona e exclui podem ser usadas para substituir os axiomas de enquadramento na geração do ESTADO 2 a partir do ESTADO 1 da Seção 8.4.
16. Use as listas *adiciona* e *exclui* para gerar o espaço de busca da Figura 8.19.
17. Projete um controlador automático que possa usar as listas *adiciona* e *exclui* para gerar uma busca em grafo semelhante àquela da Figura 8.19.
18. Mostre mais dois subobjetivos (pré-condições) incompatíveis nos operadores do mundo de blocos da Figura 8.19.
19. Leia o artigo sobre ABSTRIPS (Sacerdotti, 1974) e mostre como ele trata o problema da linearidade (ou subobjetivo incompatível) no planejamento.
20. Na Seção 8.4.3, apresentamos um planejador criado por Nilsson e seus estudantes em Stanford (Benson e Nilsson, 1995). O planejamento *teleorreativo* permite ações descritas como *duradouras*, isto é, que precisam continuar a ser verdadeiras ao longo de períodos de tempo. Por que o planejamento teleorreativo seria preferível a um planejador do tipo STRIPS? Construa um planejador teleorreativo em Prolog, Lisp ou Java.

21. Leia Williams e Nayak (1996, 1996a) para ver uma discussão mais completa do seu sistema de planejamento baseado em modelo.
22. Expanda o esquema de representação por cálculo proposicional introduzido por Williams e Nayak (1996a, p. 973) para descrever transições de estado para o seu sistema de propulsão.

# Raciocinando em situações incertas

*Toda a lógica tradicional habitualmente supõe que estão sendo empregados símbolos precisos. Entretanto, isso não se aplica a esta vida terrena, mas apenas a uma experiência celestial imaginada.*

— BERTRAND RUSSELL

*A característica marcante de uma mente instruída é ficar satisfeita com o grau de precisão que a natureza do tema admite e não procurar a exatidão onde apenas uma aproximação da verdade é possível.*

— ARISTÓTELES

*Na medida em que as leis da matemática se referem à realidade, elas não são certas. E na medida em que elas são certas, elas não se referem à realidade.*

— ALBERT EINSTEIN

## 9.0 Introdução

No decorrer das partes I, II e III, nossos procedimentos de inferência seguiram o modelo de raciocínio usado no cálculo de predicados: a partir de premissas corretas, regras de inferência consistentes garantem que as novas conclusões produzidas sejam corretas. Porém, como vimos nos capítulos 5 e 7, há muitas situações que não se enquadram nessa abordagem; isto é, devemos tirar conclusões úteis de evidências malformadas e incertas usando regras de inferência inconsistentes.

Tirar conclusões úteis, a partir de dados incompletos e imprecisos, com raciocínio inconsistente, não é uma tarefa impossível; fazemos isso com sucesso em quase todos os aspectos de nossa vida diária. Fazemos diagnósticos médicos corretos e recomendamos tratamentos a partir de sintomas ambíguos; analisamos problemas com carros e equipamentos de som; compreendemos sentenças que são, muitas vezes, ambíguas ou incompletas; reconhecemos amigos por suas vozes ou por seus gestos; e assim por diante.

Para demonstrar o problema de raciocinar em situações ambíguas, considere a Regra 2 do sistema especialista para o diagnóstico automotivo da Seção 8.2:

se  
o motor não pega e  
as luzes não acendem  
então  
o problema é bateria ou cabo.

Vista superficialmente, essa regra parece uma relação normal de predicados para ser usada por um método de inferência consistente (*modus ponens*). Entretanto, ela não é assim; essa regra é de natureza heurística. Seria pos-

sível, embora muito improvável, que a bateria ou os cabos estivessem em ordem, mas que o carro simplesmente tivesse um motor de partida com problemas e faróis queimados. As falhas do motor em pegar e das luzes em acender não necessariamente implicam que a bateria e os cabos estejam com problemas.

É interessante notar que o *inverso* da regra é verdadeiro:

se  
o problema é bateria ou cabo  
então  
o motor não pega e  
as luzes não acendem.

Excetuando-se o sobrenatural, com uma bateria descarregada, nem as luzes acendem nem o motor de partida funcional!

Nosso sistema especialista oferece um exemplo de raciocínio *abolutivo*. Formalmente, a abdução declara que, a partir de  $P \rightarrow Q$  e  $Q$ , é possível deduzir  $P$ . A abdução é uma regra *inconsistente* de inferência, significando que a conclusão não é necessariamente verdadeira para toda interpretação na qual as premissas sejam verdadeiras (Seção 2.3).

Embora a abdução seja inconsistente, ela é, muitas vezes, essencial para resolver problemas. A versão “lógica-correta” da regra da bateria não é muito útil para o diagnóstico de problemas com automóveis, pois a sua premissa, o problema da bateria, é o nosso objetivo e as suas conclusões são os sintomas observáveis com os quais devemos trabalhar. Entretanto, a regra pode ser usada de uma forma abolutiva, como acontece com as regras em muitos sistemas especialistas para diagnósticos. Falhas ou doenças causam (implicam) sintomas, não o contrário; mas o diagnóstico precisa funcionar a partir dos sintomas agindo retroativamente até as suas causas.

Nos sistemas baseados em conhecimento, normalmente conectamos um fator de certeza à regra para medir a crença em sua conclusão. Por exemplo, a regra  $P \rightarrow Q$  (0,9) expressa a crença de que “se você acredita que  $P$  é verdadeiro, então acredita que  $Q$  acontecerá 90% do tempo”. Assim, as regras heurísticas podem expressar uma política explícita para a crença.

Outra questão importante para o raciocínio em sistemas especialistas é como chegar a resultados úteis a partir de dados com informação faltante, incompleta ou incorreta. Podemos usar medidas de certeza para refletir a nossa crença na qualidade dos dados; por exemplo, afirmar que as luzes têm potência máxima (0,2) pode indicar que os faróis acendem, mas são fracos e pouco visíveis. Crenças e dados imperfeitos podem ser propagados por meio de regras a fim de restringir as conclusões.

Neste capítulo, discutimos várias formas de gerenciar a inferência abolutiva e as incertezas, especialmente no que diz respeito à resolução de problemas que utilizam intensivamente o conhecimento. Na Seção 9.1, mostramos como os formalismos baseados em lógica podem ser estendidos para tratar a tarefa abolutiva, incluindo o uso de sistemas monotônicos suportados por algoritmos de manutenção da verdade. Na Seção 9.2, consideramos várias alternativas à lógica, incluindo a álgebra de fatores de certezas, de Stanford, o raciocínio “nebuloso” (*fuzzy*) e a teoria da evidência de Dempster-Shafer. Esses cálculos simples são concebidos para tratar algumas das questões de complexidade de utilizar a abordagem Bayesiana completa para construir sistemas especialistas.

Na Seção 9.3, introduzimos as abordagens estocásticas para o raciocínio incerto. Essas técnicas são baseadas no teorema de Bayes para raciocinar sobre a frequência de eventos, que usa a informação prévia sobre esses eventos. Concluímos a Seção 9.3 apresentando modelos gráficos, incluindo as redes Bayesianas de crença e os modelos de Markov observáveis e ocultos. Apresentamos as abordagens estocásticas de aprendizado no Capítulo 13.

## 9.1 Inferência abolutiva baseada em lógica

Primeiro, apresentamos as abordagens de abdução baseadas em lógica. A partir da lógica, partes de conhecimento são usadas explicitamente no raciocínio e, como vimos no Capítulo 8, podem ser parte das explicações das

conclusões que são derivadas. Mas a lógica tradicional também tem suas limitações, especialmente em áreas em que falta informação ou em que esta é inconstante ou incerta; nessas situações, os procedimentos de inferência tradicionais podem não ser utilizáveis. Na Seção 9.1, apresentamos várias extensões à lógica tradicional que permitem suportar a inferência abdutiva.

Na Seção 9.1.1, estendemos a lógica tradicional para permitir que ela descreva um mundo em que informação e crenças estão se modificando. A lógica matemática tradicional é *monotônica*: ela começa com um conjunto de axiomas, considerados verdadeiros, e deduz suas consequências. Se inserirmos nova informação a esse sistema, isso pode fazer com que o conjunto de declarações verdadeiras aumente. A adição de conhecimento nunca causará a diminuição do conjunto de declarações verdadeiras. Essa propriedade monotônica leva a problemas quando tentamos modelar o raciocínio com base em crenças e suposições. Ao raciocinar com incertezas, os seres humanos tiram conclusões com base no seu conjunto atual de crenças; entretanto, diferentemente dos axiomas matemáticos, essas crenças, juntamente com as suas consequências, podem mudar ao longo do tempo, à medida que mais informações se tornam disponíveis.

O *raciocínio não monotônico* aborda o problema de crenças que podem ser modificadas. Um sistema de raciocínio não monotônico trata a incerteza fazendo as suposições mais razoáveis à luz de informação incerta. Ele prossegue, então, com o seu raciocínio como se essas suposições fossem verdadeiras. Mais tarde, uma crença pode mudar, necessitando um reexame das conclusões derivadas daquela crença. Os algoritmos de manutenção da verdade (Seção 9.1.2) podem, então, ser empregados para manter nossa base de conhecimento consistente. Outras extensões abdutivas à lógica incluem “modelos mínimos” (Seção 9.1.3) e a abordagem da “cobertura de conjunto” (Seção 9.1.4).

### 9.1.1 Lógicas para o raciocínio não monotônico

A não monotonicidade é uma característica importante da resolução de problemas pelos seres humanos e para o raciocínio de senso comum. Por exemplo, quando planejamos nossa ida ao trabalho, fazemos uma série de suposições sobre as ruas e o tráfego. Se verificarmos que uma dessas suposições está sendo violada, talvez por alguma construção ou acidente em nossa rota normal, mudamos nossos planos e encontramos uma rota alternativa.

O raciocínio convencional usando lógica de predicados é baseado em três suposições importantes. Primeiro, as descrições dos predicados devem ser suficientes em relação ao nosso domínio de aplicação, isto é, toda a informação necessária para resolver o problema deve ser representada. Segundo, a base de informação deve ser consistente, isto é, as partes do conhecimento não podem se contradizer. Por fim, por meio do uso de regras de inferência, a informação conhecida cresce monotonicamente. Se qualquer uma dessas três suposições não for satisfeita, a abordagem baseada em lógica convencional não funcionará.

Sistemas não monotônicos tratam dessas três questões. Em primeiro lugar, sistemas de raciocínio muitas vezes enfrentam uma escassez de conhecimento sobre um domínio. Há uma questão importante aqui: suponha que não tenhamos conhecimento sobre o predicado  $p$ ; a falta de conhecimento significa que *não temos certeza de que  $p$  é verdadeiro ou significa que temos certeza de que não  $p$  é verdadeiro?* Essa questão pode ser respondida de várias formas. Prolog (ver Seção 14.3) usa a *hipótese de mundo fechado* para determinar como falso tudo o que o seu sistema de raciocínio não pode provar como verdadeiro. Como seres humanos, muitas vezes seguimos a abordagem alternativa de supor que algo seja verdadeiro até que se possa explicitamente mostrar que seja falso.

Outra abordagem ao problema da falta de conhecimento é fazer suposições explícitas sobre a verdade. No raciocínio humano, supomos a inocência de uma pessoa que não esteja diretamente conectada a um crime. Provavelmente vamos mais além e supomos a inocência de todos os que não podem se beneficiar de um crime. O resultado dessas suposições é preencher, de modo efetivo, os detalhes que faltam ao nosso conhecimento e estender nosso raciocínio para alcançar novas conclusões com base nessas suposições. Discutimos a hipótese de mundo fechado e suas alternativas na Seção 9.1.3.

As pessoas raciocinam com base em *como o mundo normalmente funciona*. A maioria dos pássaros voa. Os pais normalmente amam e apoiam seus filhos. Fazemos inferências com base na consistência de raciocinar com

nossas suposições sobre o mundo. Nesta seção, discutimos a adição de operadores modais, como *é consistente com* e *a menos que*, para realizar o raciocínio baseado em suposições.

A segunda suposição necessária aos sistemas baseados em lógica tradicionais é que o raciocínio que suporta o conhecimento deve ser consistente. Para o raciocínio humano, isso seria uma suposição muito limitante. Ao diagnosticar um problema, frequentemente consideramos múltiplas explicações possíveis para uma situação, supondo que algo é verdadeiro até que uma suposição alternativa prove ser mais frutífera. Na análise de um acidente aéreo, por exemplo, um especialista em desastres considerará uma série de causas alternativas, somente eliminando algumas dessas, conforme novas informações são descobertas. Nós humanos utilizamos o conhecimento do mundo *como ele normalmente* é para tentar direcionar o raciocínio por meio de cenários alternativos. Desejamos que os sistemas lógicos sejam capazes de considerar hipóteses alternativas.

Por fim, se desejarmos usar lógica, devemos abordar o problema de como uma base de conhecimento é atualizada. Há duas questões aqui: em primeiro lugar, como podemos adicionar o conhecimento que é baseado apenas em suposições e, em segundo lugar, o que podemos fazer quando uma de nossas suposições se mostra incorreta mais tarde. Para tratar da primeira questão, podemos permitir a adição de novo conhecimento com base nas suposições. Esse novo conhecimento é supostamente correto e, dessa forma, ele pode ser usado para inferir mais conhecimento novo. O custo dessa prática é que devemos rastrear todo o raciocínio e as provas que são baseadas em suposições: devemos estar preparados para reconsiderar qualquer conhecimento baseado nessas suposições.

O raciocínio não monotônico, por ter suas conclusões algumas vezes reconsideradas, é chamado de *revogável*; isto é, uma informação nova pode, algumas vezes, invalidar resultados previos. As representações e os procedimentos de busca que rastreiam os passos do raciocínio de um sistema lógico são chamados de *sistemas de manutenção da verdade*, ou SMV. No raciocínio revogável, o SMV preserva a consistência da base de conhecimento, rastreando as conclusões que possam ser questionadas mais tarde. Consideramos várias abordagens para a manutenção da verdade na Seção 9.1.2. Consideramos, primeiro, operadores que podem tornar revogáveis sistemas de raciocínio baseados em lógica tradicionais.

Ao implementar raciocínio não monotônico, podemos estender a nossa lógica com o operador *a\_menos\_que*, que suportaria inferências baseadas na crença de que seu argumento não seja verdadeiro. Suponha que tenhamos o seguinte conjunto de sentenças em lógica de predicados:

$$\begin{aligned} p(X) \ a\_menos\_que q(X) &\rightarrow r(X) \\ p(Z) \\ r(W) \rightarrow s(W) \end{aligned}$$

A primeira regra significa que podemos inferir  $r(X)$  se  $p(X)$  for verdadeiro e não acreditarmos que  $q(X)$  seja verdadeiro. Quando essas condições forem satisfeitas, inferimos  $r(X)$  e usando  $r(X)$  podemos, então, inferir  $s(X)$ . Subsequentemente, se modificarmos a nossa crença, ou constatarmos que  $q(X)$  é verdadeiro,  $r(X)$  e também  $s(X)$  devem ser revogados. Note que *a\_menos\_que* trata de questões de crença, em vez da verdade. Consequentemente, mudar o valor de seu argumento de “ou desconhecido ou presumido falso” para “presumido ou sabidamente verdadeiro” pode nos fazer retratar todas as inferências que dependem dessas crenças. Pela extensão de nossa lógica para raciocinar com crenças que podem ser retratadas mais tarde, introduzimos a não monotonicidade ao sistema.

O esquema de raciocínio descrito anteriormente pode ser usado, também, para codificar regras padrão (Reiter, 1980). Se substituirmos  $p(X) \ a\_menos\_que q(X) \rightarrow r(X)$  por  $p(X) \ a\_menos\_que \text{anorm } p(X) \rightarrow r(X)$ , onde *anorm*  $p(X)$  representa anormal  $p(X)$ , declaramos que, a menos que tenhamos uma ocorrência anormal de  $p$ , como por exemplo, um pássaro com uma asa quebrada, podemos fazer a inferência de que se  $X$  é um pássaro, então  $X$  pode voar.

McDermott e Doyle (1980) sugeriram um segundo operador modal para estender os sistemas lógicos. Eles expandem a lógica de predicados de primeira ordem com o operador modal *M*, que colocado antes de um predicado é lido como *é consistente com*. Por exemplo:

$$\forall X \text{ bom\_aluno}(X) \wedge M \text{ estuda\_muito}(X) \rightarrow \text{cola\_grau}(X)$$

Essa cláusula pode ser lida como: para todo  $X$ , onde  $X$  é um bom aluno, e se o fato de que  $X$  estuda muito for consistente com tudo o mais que conhecemos, então  $X$  colará grau. É claro que a parte difícil aqui é definir precisamente o que significa *é consistente com tudo mais que conhecemos*.

Notamos, primeiro, que *é consistente com tudo mais que conhecemos* pode não ser decidível. A razão para isso é que um operador modal forma um superconjunto de um sistema que já é indecidível (ver Seção 2.2.2) e que, por isso, será indecidível. Há duas maneiras de se abordar a indecidibilidade. Primeiro, podemos usar uma prova do tipo *negação como falha* para demonstrar *é consistente com*. No nosso exemplo, poderíamos tentar a prova de  $\neg(\text{estuda\_muito}(X))$  e, se não pudermos provar que  $X$  não estuda, então supomos que  $X$  estuda. Muitas vezes usamos essa abordagem em uma aproximação do tipo Prolog para a lógica de predicados. Infelizmente, a negação como falha pode restringir indevidamente nosso domínio de interpretação.

Uma segunda abordagem para o problema de *é consistente com* é realizar uma busca baseada em heurística e limitada (em tempo ou em memória) pela verdade do predicado, no nosso exemplo  $\text{estuda\_muito}(X)$ , e, então, se não houver evidência do contrário, assumi-lo como verdadeiro com a compreensão de que mais tarde poderá ser necessário retratar a conclusão *cola\_grau* e todas as demais conclusões baseadas nela.

Também podemos produzir resultados potencialmente contraditórios usando o operador *é consistente com*. Suponha que uma pessoa, Peter, seja um bom aluno, mas que ele goste muito de festas. Poderíamos, então, ter o seguinte conjunto de predicados que descrevem a situação:

$$\begin{aligned} & \forall X \text{ bom\_aluno}(X) \wedge M \text{ estuda\_muito}(X) \rightarrow \text{cola\_grau}(X) \\ & \forall Y \text{ pessoa\_festeira}(Y) \wedge M \neg(\text{estuda\_muito}(Y)) \rightarrow \neg(\text{cola\_grau}(Y)) \\ & \text{bom\_aluno(peter)} \\ & \text{pessoa\_festeira(peter)} \end{aligned}$$

Com esse conjunto de cláusulas, em que não temos informações adicionais sobre os hábitos de estudo de Peter, se ele estuda muito ou não, podemos inferir tanto que Peter colará grau quanto que ele não colará grau!

Um método de raciocínio que protege contra esses resultados contraditórios consiste em ligações das variáveis usadas com o operador modal *é consistente com*. Assim, uma vez que Peter tenha sido ligado a um dos predicados, ou a *estuda\_muito* ou a *\neg(estuda\_muito)*, o sistema não permitirá que ele seja ligado ao outro predicado. Outros sistemas lógicos não monotônicos (McDermott e Doyle, 1980) são ainda mais conservadores, não permitindo qualquer conclusão originada desses conjuntos de cláusulas potencialmente contraditórias. Podemos, ainda, criar outra anomalia:

$$\begin{aligned} & \forall Y \text{ muito\_esperto}(Y) \wedge M \neg(\text{estuda\_muito}(Y)) \rightarrow \neg(\text{estuda\_muito}(Y)) \\ & \forall X \neg(\text{muito\_esperto}(X)) \wedge M \neg(\text{estuda\_muito}(X)) \rightarrow \neg(\text{estuda\_muito}(X)) \end{aligned}$$

Dessas cláusulas, podemos inferir uma nova:

$$\forall Z M \neg(\text{estuda\_muito}(Z)) \rightarrow \neg(\text{estuda\_muito}(Z))$$

Outros desenvolvimentos da semântica do operador *é consistente com* abordam esse raciocínio anômalo. Outra extensão é a *lógica autoepistêmica* (Moore, 1985).

Outro sistema lógico não monotônico é a *lógica padrão*, criada por Reiter (1980). A lógica padrão emprega um novo conjunto de regras de inferência da forma:

$$A(Z) \wedge : B(Z) \rightarrow C(Z)$$

que é lido como: se  $A(Z)$  é demonstrável e *é consistente com o que sabemos para supormos B(Z)*, então podemos concluir  $C(Z)$ .

Até esse ponto, a lógica padrão se parece com a lógica não monotônica de McDermott e Doyle que acabamos de descrever. Uma diferença importante entre as duas é o método pelo qual o raciocínio é realizado. Na lógica padrão, essas regras de inferência especiais são usadas para inferir conjuntos de extensões plausíveis do conjunto original de axiomas/teoremas. Cada extensão é criada usando uma das regras de inferência da lógica padrão sobre o conhecimento representado pelo conjunto de axiomas/teoremas original. Assim, seria natural ter um número de extensões plausíveis de uma base de conhecimento original. Isso pode ser visto com a cláusula *cola\_grau*:

$$\begin{aligned} & \forall X \text{ bom\_aluno}(X) \wedge : \text{estuda\_muito}(X) \rightarrow \text{cola\_grau}(X) \\ & \forall Y \text{ pessoa\_festeira}(Y) \wedge : \neg(\text{estuda\_muito}(Y)) \rightarrow \neg(\text{cola\_grau}(Y)) \end{aligned}$$

Cada cláusula poderia ser usada para criar uma única extensão plausível baseada no conjunto de conhecimento original.

A lógica padrão permite, então, que qualquer teorema inferido em uma extensão plausível seja admitido como um axioma na continuação do raciocínio. Deve haver um outro guia para a tomada de decisão para determinar, enfim, qual extensão deve ser usada para continuar a resolução do problema. A lógica padrão não diz nada sobre como escolher entre extensões plausíveis possíveis de uma base de conhecimento. Reiter (1978), Reiter e Crisculo (1981) e Touretzky (1986) aprofundaram o desenvolvimento dessas questões.

Por fim, há também uma situação de raciocínio não monotônico criada pela busca por herança sobre representações, onde os objetos podem herdar de mais do que um pai. Peter, o bom aluno que gosta de festa mencionado anteriormente, poderia herdar um conjunto de propriedades por ser um bom aluno, isto é, o de que ele muito provavelmente colaria grau. Peter poderia herdar outro conjunto, nesse caso parcialmente conflitante, de propriedades por ser uma pessoa festeira, isto é, o de que ele não colaria grau.

Um problema importante enfrentado pelos sistemas de raciocínio não monotônico é a tarefa de revisar, eficientemente, um conjunto de conclusões à luz de crenças que se modificam. Se, por exemplo, usarmos o predicado  $r$  para inferir  $s$ , então a remoção de  $r$  remove também o suporte para  $s$ , bem como qualquer outra conclusão que use  $s$ . A menos que haja um conjunto independente de inferências que suportem  $s$ , ele deve ser retratado. Implementar esse processo de retratação requer, no pior caso, que recalculemos todas as conclusões cada vez que uma crença se modificar. Os sistemas de manutenção da verdade, apresentados a seguir, oferecem mecanismos para manter a consistência de bases de conhecimento.

### 9.1.2 Sistemas de manutenção da verdade

Um sistema de manutenção da verdade (SMV) pode ser empregado para proteger a integridade lógica das conclusões de um sistema de inferência. Como apontado na seção anterior, sempre que forem revisadas quaisquer crenças expressas pelas cláusulas de uma base de conhecimento, é necessário recalcular o suporte para os itens nessa base. Sistemas de manutenção da verdade tratam dessa questão armazenando justificativas para cada inferência e, então, reconsideram o suporte para as conclusões à luz de novas crenças.

Uma forma de considerar esse problema consiste em rever o algoritmo com retrocesso apresentado anteriormente na Seção 3.2.2. O algoritmo com retrocesso é um método sistemático para explorar todas as alternativas para pontos de decisão na resolução de problemas baseados em busca. Entretanto, uma deficiência importante do algoritmo com retrocesso é o modo como ele sistematicamente (e cegamente) retrocede de estados de “beco sem saída” do espaço e procura por alternativas entre suas escolhas mais recentes. Essa abordagem é, às vezes, chamada de *retrocesso cronológico*. Admitamos que o retrocesso cronológico verifique, sistematicamente, todas as alternativas do espaço; entretanto, a forma como isso se processa consome muito tempo, além de ser ineficiente e, em um espaço muito grande, inútil.

O que realmente queremos na busca baseada em lógica é a capacidade de retroceder diretamente para o ponto no espaço onde o problema ocorre e de fazer ajustes para a solução naquele estado. Essa abordagem é chamada de *retrocesso guiado por dependência*. Considere um exemplo de raciocínio não monotônico. Precisamos raciocinar sobre  $p$ , que não podemos inferir diretamente. Há, entretanto, uma suposição plausível  $q$  que, se verdadeira, dará suporte a  $p$ . Assim, supomos  $q$  e derivamos  $p$ . Nossa raciocínio continua e, com base em  $p$ , concluímos  $r$  e  $s$ . Seguimos em frente no raciocínio e concluímos, sem o suporte de  $p$ ,  $r$  ou  $s$ , os resultados  $t$  e  $u$ . Finalmente, provamos que nossa suposição anterior para  $q$  é falsa. O que devemos fazer?

O retrocesso cronológico reveria os passos de nosso raciocínio na ordem inversa em que eles foram feitos. O retrocesso guiado por dependência retornaria imediatamente à fonte da informação contraditória, a saber, a primeira suposição para  $q$ . Ele então avançaria retratando  $p$ ,  $r$  e  $s$ . Podemos, nesse ponto, conferir se  $r$  e  $s$  podem ser derivados independentemente de  $p$  e  $q$ . Só por eles terem sido produzidos originalmente a partir de uma suposição incorreta, não significa que eles não sejam suportados de outra forma. Por fim, pelo fato de  $t$  e  $u$  terem sido derivados sem  $p$ ,  $r$  e  $s$ , não precisaríamos reconsiderá-los.

Para usar retrocesso guiado por dependência em um sistema de raciocínio, precisamos:

1. Associar à produção de cada conclusão sua respectiva justificativa. Essa justificativa indica o processo de derivação para essa conclusão. A justificativa deve conter todos os fatos, regras e suposições usadas na produção da conclusão.
2. Criar um mecanismo que, dada uma contradição com suas justificativas, encontre o conjunto de suposições falsas naquela justificativa que levou à contradição.
3. Retirar a(s) suposição(ões) falsa(s).
4. Criar um mecanismo que siga a(s) suposição(ões) retratadas(s) e retrate qualquer conclusão que use nas suas justificativas a suposição falsa retratada.

É claro que todas as conclusões retratadas não são necessariamente falsas, sendo, portanto, necessário conferi-las novamente, para ver se elas podem ser justificadas independentemente das cláusulas retratadas. Apresentamos, a seguir, dois métodos para construir sistemas de retrocesso guiado por dependência.

Jon Doyle (1979) criou um dos primeiros sistemas de manutenção da verdade, chamado de *sistema de manutenção da verdade baseado em justificativa*, ou SMVJ. Doyle foi o primeiro pesquisador a separar explicitamente o sistema de manutenção da verdade, uma rede de proposições e suas justificativas, do sistema de raciocínio que opera em certos domínios. O resultado dessa separação é que o SMVJ se comunica com o resolvedor de problemas, talvez um provador automático de teoremas, recebendo informação sobre novas proposições e justificativas e, por sua vez, suprindo o resolvedor de problemas com informações sobre em quais proposições devemos acreditar com base nas justificativas atualmente existentes.

Há três operações principais que são realizadas pelo SMVJ. Primeiro, o SMVJ inspecciona a rede de justificativas. Essa inspeção pode ser disparada por consultas do resolvedor de problemas na seguinte forma: devo crer na proposição  $p$ ? Por que eu deveria crer na proposição  $p$ ? Em que suposições se baseia a proposição  $p$ ?

A segunda operação do SMVJ é modificar a rede de dependências, onde as modificações são guiadas por informações fornecidas pelo resolvedor de problemas. Entre as modificações se incluem: acrescentar novas proposições, acrescentar ou remover premissas, acrescentar contradições e justificar a crença em uma proposição. A operação final do SMVJ é atualizar a rede. Essa operação é executada sempre que uma mudança é feita na rede de dependências. A operação de atualização recalcula os rótulos de todas as proposições, de modo que haja consistência com as justificativas existentes.

Para demonstrar o SMVJ, construímos uma rede de dependências simples. Considere o operador modal M apresentado na Seção 9.1.1, que, colocado antes de um predicado, é lido como *é consistente com*. Por exemplo:

```

 $\forall X \text{ bom\_aluno}(X) \wedge M \text{ estuda\_muito}(X) \rightarrow \text{estuda\_muito}(X)$ 
 $\forall Y \text{ pessoa\_festeira}(Y) \rightarrow \neg(\text{estuda\_muito}(Y))$ 
bom\_aluno(david)

```

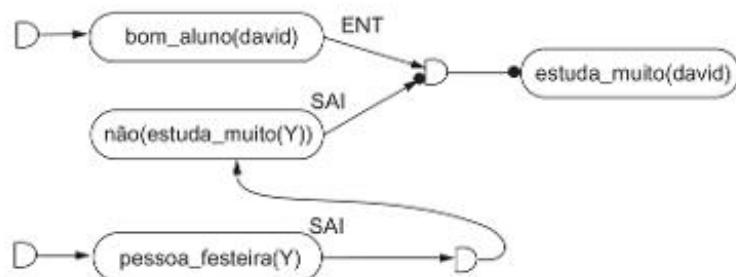
Transformamos agora esse conjunto de proposições em uma rede de justificativas.

Em um SMVJ, cada predicado que representa uma crença é associado a dois outros conjuntos de crenças. O primeiro conjunto, rotulado ENT na Figura 9.1, é o conjunto de proposições nas quais devemos acreditar para que a proposição seja válida. O segundo conjunto, rotulado SAI, são proposições nas quais não devemos acreditar para que a proposição seja válida. A Figura 9.1 representa a justificativa que suporta  $\text{estuda\_muito}(\text{david})$  derivada dos predicados listados. As notações da Figura 9.1 são adaptadas de Goodwin (1982) e explicadas na Figura 9.2. As premissas das justificativas são rotuladas como na Figura 9.2(a), e as combinações de proposições que suportam uma conclusão são rotuladas como na Figura 9.2(b).

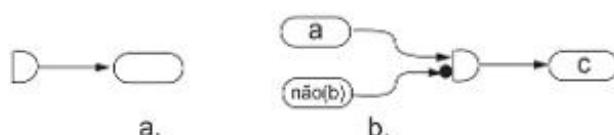
Com a informação da rede da Figura 9.1, o resolvedor de problemas pode raciocinar que existe suporte para  $\text{estuda\_muito}(\text{david})$ , porque a premissa  $\text{bom\_aluno}(\text{david})$  é considerada verdadeira e é consistente com o fato de que bons alunos estudam muito. Também não há evidência ou outra indicação, nesse exemplo, de que David não estude muito.

Suponha que acrescentemos agora  $\text{pessoa\_festeira}(\text{david})$  na premissa. Essa adição permite a derivação de  $\neg(\text{estuda\_muito}(\text{david}))$ , fazendo com que a crença  $\text{estuda\_muito}(\text{david})$  deixe de ser suportada. As justificativas para essa nova situação estão refletidas na Figura 9.3. Note a troca dos rótulos ENT e SAI.

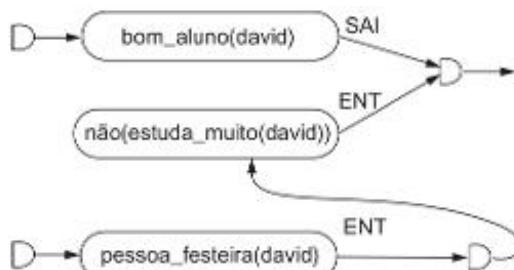
**Figura 9.1** Uma rede de justificativas para acreditar que David estuda muito.



**Figura 9.2** 9.2(a) é uma justificativa de premissa e 9.2(b), a conjunção de duas crenças, a e não b, para dar suporte a c (Goodwin, 1982).



**Figura 9.3** A nova rotulação da Figura 9.1 associada à nova premissa `pessoa_festeira(david)`.



Como as figuras 9.1 e 9.3 demonstram, o SMVJ não representa diretamente as relações entre predicados, como está expresso no conjunto original de proposições. Em vez disso, o SMVJ é uma rede simples que considera apenas as relações entre proposições atômicas e suas negações e as organiza em relações de suporte para crenças. O conjunto completo de conectivos de predicados e esquemas de inferência ( $\forall X, \wedge, \vee, \rightarrow$  etc.) é usado dentro do próprio resolvedor de problemas. Os sistemas de McAllester (1978) e Martins e Shapiro (1988) combinam o SMV e o resolvedor de problemas em uma única representação.

Um SMVJ lida apenas com as dependências entre crenças e não se preocupa com o conteúdo dessas crenças. Portanto, podemos substituir as crenças por identificadores, usualmente da forma  $n_1, n_2, \dots$ , os quais são associados com objetos da rede chamados nós. Então, a álgebra de ENTs e SAIs, que vimos implementada no exemplo do `estuda_muito`, permite que o SMVJ raciocine sobre o suporte para crenças.

Resumindo, um SMVJ trabalha com conjuntos de nós e justificativas. Os nós representam crenças e as justificativas dão suporte à crença nos nós. Os rótulos ENT e SAI estão associados com nós e indicam o estado da crença do nó associado. Podemos raciocinar sobre o suporte para qualquer nó relacionando-o a ENTs e SAIs dos outros nós que compõem a(s) sua(s) justificativa(s). As operações principais da álgebra SMVJ são realizar inspeção, modificação e atualização de operadores, discutidos anteriormente. Por fim, uma vez que a verificação de justificativa é imposta pelo retrocesso por meio dos arcos da rede de justificativas propriamente dita, temos um

exemplo de retrocesso baseado em dependências. Para obter mais informações sobre essa abordagem para o SMVJ, veja Doyle (1983) e Reinfrank (1989).

Um segundo tipo de sistema de manutenção da verdade é o *sistema de manutenção da verdade baseado em suposições*, ou SMVS. O termo *baseado em suposições* foi introduzido originalmente por deKleer (1984), embora ideias similares possam ser encontradas em Martins e Shapiro (1983). Nesses sistemas, os rótulos para os nós da rede não são mais ENT e SAI, mas sim conjuntos de premissas (suposições) que fundamentam a sua derivação. deKleer faz também uma distinção entre nós de premissas que valem universalmente e nós que podem ser suposições feitas pelo resolvidor do problema e que podem ser retratadas mais tarde.

Uma vantagem do SMVS sobre o SMVJ vem da flexibilidade adicional que o SMVS fornece ao tratar com múltiplos estados possíveis de crença. Rotulando-se crenças com os conjuntos de premissas sob as quais elas valem, não há mais um único estado de crença (em SMVJ, todos os nós rotulados como ENT), mas sim uma série de estados possíveis, o conjunto de todos os subconjuntos das premissas de suporte. A criação de diferentes conjuntos de crença, ou mundos possíveis, permite tanto uma comparação dos resultados de escolhas diferentes de premissas como, também, a existência de diferentes soluções do problema e a detecção de contradições e a sua recuperação. As desvantagens do SMVS incluem a inabilidade de representar conjuntos de premissas que são elas mesmas não monotônicas e o controle sobre o resolvidor de problemas. Entretanto, veja Dressler (1988) e Forbus e deKleer (1988) para obter mais alternativas.

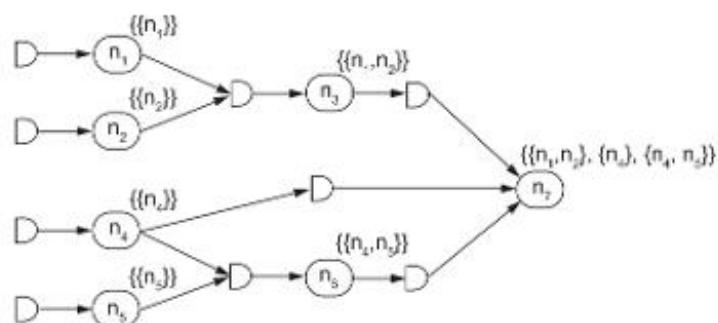
A comunicação entre o SMVS e o resolvidor de problemas é similar àquela entre o SMVJ e o resolvidor de problemas com operadores para *inspeção, modificação e atualização*. A única diferença é que com o SMVS não há mais um único estado de crença, mas sim subconjuntos de premissas de suporte potenciais. O objetivo da computação dentro do SMVS é encontrar conjuntos mínimos de premissas suficientes para o suporte de cada nó. Essa computação é feita propagando e combinando rótulos, começando com os rótulos para as premissas.

Apresentamos, a seguir, um exemplo detalhado adaptado de Martins (1991). Suponha que tenhamos a rede SMVS da Figura 9.4. Nessa rede,  $n_1, n_2, n_4$  e  $n_5$  são premissas supostas como verdadeiras. A rede de dependências também reflete as relações pelas quais, a partir de  $n_1$  e  $n_2$ , suportamos  $n_3$ , com  $n_3$  suportando  $n_7$ , com  $n_4$  suportando  $n_7$ , com  $n_4$  e  $n_5$  suportando  $n_6$  e, finalmente, com  $n_6$  suportando  $n_7$ .

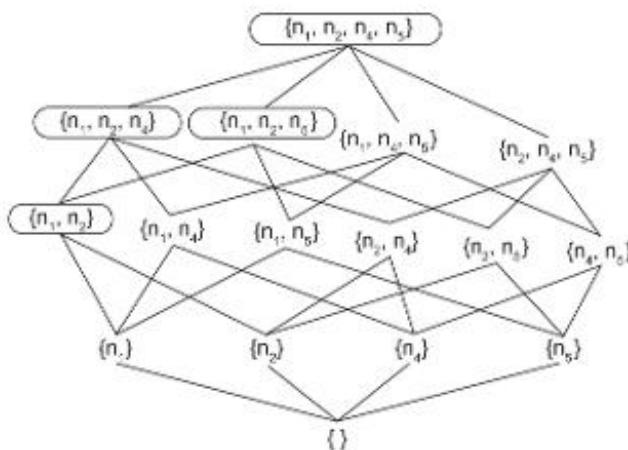
A Figura 9.5 apresenta um reticulado de subconjuntos/superconjuntos para as dependências das premissas encontradas na Figura 9.4. Esse reticulado de subconjuntos de premissas oferece uma maneira útil de visualizar o espaço de combinação de premissas. Assim, se encontrarmos uma premissa suspeita, o SMVS será capaz de determinar como essa premissa se relaciona com outros subconjuntos de premissas de suporte. Por exemplo, o nó  $n_3$  na Figura 9.4 será suportado por todos os conjuntos de premissas que estão acima de  $\{n_1, n_2\}$  no reticulado da Figura 9.5.

O raciocinador SMVS remove contradições removendo dos nós aqueles conjuntos de premissas que verificamos serem inconsistentes. Suponha, por exemplo, que revisemos o suporte para o raciocínio refletido pela Figura 9.4 para tornar  $n_3$  um nó de contradição. Como o rótulo para  $n_3$  é  $\{n_1, n_2\}$ , determina-se que esse conjunto de premissas é inconsistente. Quando essa inconsistência é descoberta, todos os conjuntos de premissas que estão na relação

**Figura 9.4** Uma rotulação de nós por SMVS em uma rede de dependências.



**Figura 9.5** O reticulado para as premissas da rede da Figura 9.4. Os conjuntos circulados indicam a hierarquia de inconsistências; adaptado de Martins (1991).



de superconjunto para  $\{n_1, n_2\}$ , na Figura 9.5, são marcados como inconsistentes (nós os circulamos) e são removidos da rede de dependências. Nessa situação, uma das rotulações possíveis que suportam  $n_7$  terá que ser removida. Uma descrição completa do algoritmo de remoção de contradições pode ser obtida em deKleer (1986).

Há uma série de outras contribuições importantes ao raciocínio SMV. O *SMV baseado em lógica* (SMVL) é originário do trabalho de McAllester (1978). No SMVL, as relações entre proposições são representadas por cláusulas que podem ser usadas para deduzir os valores-verdade de qualquer uma das proposições que elas descrevem. Outra abordagem, o *raciocinador de múltiplas crenças*, MBR (do inglês *Multiple Belief Reasoner*), é semelhante ao raciocinador SMVS, exceto que o resolvelor do problema e o sistema de manutenção da verdade são combinados em um único sistema. O MBR é baseado em uma linguagem lógica chamada SWM\*, que descreve estados de conhecimento. Cada estado de conhecimento é composto por um par de descritores, o primeiro refletindo uma base de conhecimento e o segundo, um conjunto de conjuntos de premissas sabidamente inconsistentes dentro da base de conhecimento. Em Martins (1991), podem ser encontrados algoritmos para a verificação de inconsistências durante o raciocínio. Outra discussão sobre sistemas de manutenção da verdade pode ser encontrada em *The Encyclopedia of Artificial Intelligence* (Shapiro, 1992).

### 9.1.3 Lógica baseada em modelos mínimos

Nas seções anteriores, estendemos a lógica por meio de vários operadores modais diferentes que foram concebidos especificamente para o raciocínio sobre o mundo *como normalmente ele é*, relaxando as exigências de que o nosso conhecimento do mundo seja de alguma forma completo. Esses operadores se originaram da necessidade de se criar uma visão de mundo mais flexível e capaz de ser revista. Nesta seção, apresentamos lógicas concebidas especificamente para duas situações: em primeiro lugar, para raciocinar onde um conjunto de declarações especifica apenas aquelas coisas que são verdadeiras e, em segundo, para raciocinar onde, por causa da natureza da tarefa de solução do problema, conjuntos de conjecturas são normalmente verdadeiros. Na primeira situação usamos a *hipótese de mundo fechado*, e na segunda usamos *circunscrição*. As duas abordagens para a lógica são normalmente chamadas de *raciocínio baseado em modelos mínimos*.

Vimos, na Seção 2.3, que um *modelo* é uma interpretação que satisfaz  $S$ , um conjunto de expressões de predicados, para todas as atribuições de variáveis. Existem diversas maneiras de definir o que se entende por um *modelo mínimo*. Definimos um modelo mínimo como *um modelo tal que não haja modelos menores que possam satisfazer o conjunto de expressões S para todas as atribuições de variáveis*.

O que torna os modelos mínimos importantes para o raciocínio é que existe um número (potencialmente) infinito de predicados que podem ser usados para descrever situações do mundo. Considere, por exemplo, os inúmeros predicados que podem ser usados para descrever a situação do problema do fazendeiro-lobo-cabra-repolho (Seção 3.5, Exercício 2): o barco não está afundando lentamente, as margens do rio são próximas o suficiente para que, remando, possamos atravessar o barco para a outra margem, o vento e a corrente não são fatores relevantes, e assim por diante. Quando descrevemos um problema, normalmente somos bastante parcimoniosos em nossas descrições. Criamos apenas aqueles predicados que são relevantes e necessários para resolver o problema.

A *hipótese de mundo fechado* é baseada nesse modelo mínimo de mundo. São criados exatamente aqueles predicados que são necessários para que uma solução seja criada. A hipótese de mundo fechado efetua a semântica da negação ao raciocinar. Por exemplo, se desejarmos determinar se um aluno estiver matriculado em uma turma, poderíamos acessar a base de dados de matrículas e, se o estudante não estiver listado explicitamente nessa base (o modelo mínimo), ele não estará matriculado. Da mesma forma, se desejarmos saber se duas cidades estão diretamente ligadas por alguma conexão aérea, acessaríamos a lista de todas as conexões de linhas aéreas. Se o voo direto não estiver listado lá (o modelo mínimo), tal conexão não existe.

A hipótese de mundo fechado é uma declaração de que, se nosso sistema computacional não puder concluir que  $p(X)$  é verdadeiro, então  $\neg(p(X))$  deve ser verdadeiro. Como veremos na Seção 14.4, a hipótese de mundo fechado suporta o método de inferência do Prolog. Na Seção 14.4, apresentamos as três suposições (axiomas) implícitas no uso de modelos mínimos. Esses axiomas são *o nome único*, isto é, que todos os átomos com nomes distintos são distintos; *o mundo fechado*, isto é, as únicas ocorrências de uma relação são aquelas implicadas pelas cláusulas presentes; e *o fechamento do domínio*, isto é, os átomos do domínio são exatamente aqueles do modelo. Quando esses três são satisfeitos, um modelo mínimo se torna uma especificação baseada em lógica completa. Se os axiomas não forem satisfeitos, é necessária uma forma do algoritmo de manutenção da verdade.

Enquanto o mundo fechado requer que todos os predicados que compõem um modelo sejam declarados, a *circunscrição* (McCarthy, 1980; Lifschitz, 1984, McCarthy, 1986) requer que sejam declarados *apenas* os predicados relevantes para a solução do problema. Na circunscrição, adicionam-se axiomas a um sistema que força uma interpretação mínima sobre os predicados da base de conhecimento. Esses *metapredicados* (predicados sobre os predicados de definição do problema) descrevem a maneira pela qual predicados particulares devem ser interpretados. Isto é, eles delimitam, ou circunscrevem, as interpretações possíveis para os predicados.

McCarthy (1980) introduziu a ideia de circunscrição com um experimento conceitual do problema dos misionários e dos canibais. A definição do problema pede que o resolvedor conceba uma série de movimentos nos quais seis personagens, sob um conjunto de restrições, possam usar um barco para atravessar um rio. McCarthy apresenta um grande número de situações absurdas que, legitimamente, podem ser questionadas sobre a definição do problema. Várias delas, como um barco que afunda lentamente ou um fator de vento, foram apresentadas anteriormente nesta seção. Embora os seres humanos considerem essas situações como absurdas, o raciocínio que usamos para isso não é óbvio. Os axiomas de circunscrição que McCarthy acrescentaria à especificação do problema delimitariam precisamente os predicados que descrevem o problema.

Como outro exemplo de circunscrição, considere uma expressão de predicados de uma especificação para raciocínio de senso comum orientada a objetos (Seção 9.4):

$$\forall X \text{ ave}(X) \wedge \neg(\text{anormal}(X)) \rightarrow \text{voa}(X)$$

Essa expressão poderia ocorrer no raciocínio em que uma das propriedades de *ave* é que ela *voa*. Mas o que poderia limitar a definição do predicado *anormal*? Que a *ave* não seja um pinguim, que ela não tenha uma asa quebrada, que ela não esteja morta? A especificação do predicado *anormal* é potencialmente indecidível.

A circunscrição usa um esquema de axiomas, ou um conjunto de metaregras, dentro do cálculo de predicados de primeira ordem para gerar predicados para o domínio do problema. As regras do esquema fazem com que certas fórmulas tenham as extensões menores possíveis. Por exemplo, se  $B$  é um sistema de crenças, incluindo o conhecimento do mundo  $K$  e o conhecimento de domínio  $A(p)$  sobre o predicado  $p$ , então podemos considerar que  $p$  será minimizado, se o menor número possível de  $a_i$  satisfizer  $p(a_i)$ , sendo consistente com  $A(p)$  e  $K$ . O conhecimento do mundo  $K$ , com  $A(p)$  e o esquema de circunscrição, são usados para derivar conclusões no cálculo de predicados de primeira ordem padrão. Essas conclusões são, então, acrescentadas a  $B$ , o sistema de crenças.

Como exemplo, suponha que, no mundo de blocos da Seção 8.4, tenhamos a expressão:

$$\text{é\_um\_bloco}(a) \wedge \text{é\_um\_bloco}(b) \wedge \text{é\_um\_bloco}(c)$$

que declara que a, b e c são blocos. Circunscrevendo o predicado `é_um_bloco`, temos:

$$\forall X (\text{é\_um\_bloco}(X) \leftarrow ((X = a) \vee (X = b) \vee (X = c)))$$

Essa expressão declara que os únicos blocos são a, b e c, isto é, apenas os objetos requeridos pelos predicado `é_um_bloco` são blocos nesse domínio. De forma similar, o predicado:

$$\text{é\_um\_bloco}(a) \vee \text{é\_um\_bloco}(b)$$

pode ser circunscrito a:

$$\forall X (\text{é\_um\_bloco}(X) \leftarrow ((X = a) \vee (X = b)))$$

Para obter o detalhamento completo, incluindo os axiomas de esquemas usados para derivar esses resultados, veja McCarthy (1980, Seção 4).

A circunscrição, quando usada com operadores como anormal, é muito parecida com a hipótese de mundo fechado, pois produz exatamente aquelas ligações de variáveis que anormal pode suportar. Entretanto, a álgebra de circunscrição nos permite estender esse raciocínio por meio de representações de predicados já que, como foi comentado anteriormente, se tivermos o predicado  $p(X) \vee q(X)$ , podemos circunscrever o predicado  $p$  ou o predicado  $q$ , ou ambos. Dessa forma, diferentemente da hipótese de mundo fechado, a circunscrição nos permite descrever as ocorrências que são possíveis pelos conjuntos de descrições de predicados.

Outros trabalhos sobre lógicas de circunscrição podem ser encontrados em Genesereth e Nilsson (1987). Lifschitz (1986) fez uma importante contribuição propondo uma circunscrição pontual na qual o modelo mínimo pode ser realizado para predicados particulares e suas ocorrências possíveis, em vez de o ser para todo o domínio. Outra contribuição importante é apresentada em Perlis (1988), onde o raciocínio pode se dar sobre uma falta de conhecimento de um agente em particular. Para ver mais, consulte também Shapiro (1992).

### 9.1.4 Cobertura de conjunto e abdução baseada em lógica (Stern, 1996)

Como foi citado na introdução deste capítulo, no raciocínio abdutivo, temos regras da forma  $p \rightarrow q$  com uma crença razoável em  $q$ . Queremos, então, justificar a verdade do predicado  $p$ . O raciocínio abdutivo não é consistente, mas frequentemente é denominado *raciocínio para a melhor explicação* para a presença do dado  $q$ . Nesta seção, abordamos mais de perto a geração de explicações nos domínios da inferência abdutiva.

Além dos relatos de raciocínio abdutivo já apresentados, os pesquisadores de IA têm também usado a cobertura de conjunto e a análise apoiada em lógica. A abordagem por *cobertura de conjunto* para a abdução tenta explicar o ato de adotar uma crença revogável em uma hipótese explanatória pela razão de ela explicar um conjunto de fatos que, de outra forma, seria inexplicável. A abordagem *baseada em lógica* para a abdução descreve regras de inferência para abdução com uma definição de sua(s) forma(s) legítima(s) de uso.

A abordagem por *cobertura de conjunto* define uma explicação abdutiva como uma cobertura de predicados que descrevem observações por predicados que descrevem hipóteses. Reggia et al. (1983) descreve uma cobertura baseada em uma relação binária casual  $R$ , onde  $R$  é um subconjunto de {Hipóteses X Observações}. Assim, uma explicação abdutiva de um conjunto de observações  $S_2$  é outro conjunto de hipóteses  $S_1$  suficiente para causar  $S_2$ . De acordo com a abordagem de cobertura de conjuntos, uma explicação ótima é a cobertura de conjunto mínima de  $S_2$ . O ponto fraco dessa abordagem é que ela reduz as explicações a uma simples lista de hipóteses causais (de  $S_1$ ). Em situações em que há causas inter-relacionadas ou interativas, ou em que uma compreensão da estrutura ou do sequenciamento de interações causais é necessária, o modelo de cobertura de conjuntos é inadequado.

Por outro lado, as abordagens baseadas em lógica para a abdução se baseiam em uma noção mais sofisticada de explicação. Levesque (1989) define uma explicação abdutiva de um conjunto de observações  $O$ , anteriormente inexplicável, como um conjunto mínimo de hipóteses  $H$  consistente com um conhecimento de base  $K$  de um agente. A hipótese  $H$ , com o conhecimento de base  $K$ , precisa acarretar  $O$ . Mais formalmente:

$\text{abduz}(K, O) = H$ , se, e somente se:

1.  $K$  não acarreta  $O$
2.  $H \cup K$  acarreta  $O$
3.  $H \cup K$  é consistente e
4. Nenhum subconjunto de  $H$  tem as propriedades 1, 2 e 3.

Note que, em geral, podem existir muitos conjuntos de hipóteses; isto é, pode haver muitos conjuntos abdutivos potenciais de explicação para um dado conjunto de observações  $O$ .

A definição baseada em lógica para a explicação abdutiva sugere um mecanismo correspondente para a descoberta da explicação no contexto de um sistema baseado em conhecimento. Se a hipótese explanatória precisa acarretar a observação  $O$ , então a maneira de construir uma explicação completa é raciocinando regressivamente a partir de  $O$ . Como vimos nas seções 3.3 e 8.2, podemos começar a partir dos componentes conjuntivos de  $O$  e raciocinar, regressivamente, das consequências até os antecedentes.

Essa abordagem por encadeamento regressivo também aparenta ser natural porque os condicionais que suportam o encadeamento regressivo podem ser considerados diretamente como leis causais, capturando, assim, o papel central que o conhecimento causal desempenha na construção de explicações. O modelo também é conveniente porque se enquadra perfeitamente em algo com que a comunidade de IA já tem experiência: encadeamento regressivo e modelos computacionais para dedução.

Há, também, maneiras mais engenhosas para encontrar o conjunto completo de explicações abdutivas. Os sistemas de manutenção da verdade baseados em suposições SMVS (deKleer, 1986, Seção 9.2.3) contêm um algoritmo para calcular os conjuntos de suporte mínimos, o conjunto de proposições (não axiomáticas) que logicamente acarretam uma dada proposição de uma teoria. Para encontrar todas as explicações abdutivas possíveis para um conjunto de observações, fazemos meramente o produto cartesiano sobre os conjuntos de suporte.

Apesar de ser simples, precisa e conveniente, a abordagem baseada em lógica para a abdução tem dois pontos fracos: alta complexidade computacional e semântica fraca. Selman e Levesque (1990) constataram que a complexidade das tarefas de abdução é semelhante àquela envolvida no cálculo de conjuntos de suporte para um SMVS. A prova padrão de que o problema SMVS é NP-difícil depende da existência de ocorrências do problema com um número exponencial de soluções. Selman e Levesque evitam a questão da complexidade do número de soluções potenciais questionando se encontrar um conjunto menor de soluções é também NP-difícil. Dada uma base de conhecimento constituída de cláusulas de Horn (ver Seção 14.2), Selman e Levesque produzem um algoritmo que encontra uma única explicação de ordem  $O(k^n)$ , onde  $k$  indica o número de variáveis proposicionais e  $n$  o número de ocorrência de literais. Entretanto, quando se colocam restrições sobre os tipos de explicações procuradas, o problema se torna novamente NP-difícil, mesmo para cláusulas de Horn.

Um resultado interessante da análise de Selman e Levesque (1990) é o fato de que a adição de certos tipos de objetivos, ou restrições, à tarefa de abdução na verdade torna a computação significativamente mais difícil. Da perspectiva ingênua do resolvedor de problemas humano, esse aumento de complexidade é surpreendente: as pessoas supõem que a adição de restrições à busca por explicações relevantes tornaria a tarefa mais fácil. A razão pela qual a tarefa de abdução é mais difícil no modelo baseado em lógica é que ela apenas contribui com cláusulas adicionais ao problema, e não com estrutura adicional, útil para derivar uma solução.

A descoberta de explicações no modelo baseado em lógica é caracterizada como a tarefa de encontrar um conjunto de hipóteses com certas propriedades lógicas. Essas propriedades, incluindo a consistência com o conhecimento de base e a derivabilidade do que deve ser explicado, visam capturar as condições *necessárias* das explicações: as condições mínimas que um conjunto de hipóteses explanatórias deve satisfazer para ser considerado como uma explicação abdutiva. Os proponentes dessa abordagem acreditam que, adicionando restrições, a abordagem pode ser estendida para fornecer uma caracterização de explicações boas ou razoáveis.

Uma estratégia simples para produzir explicações de qualidade é definir um conjunto de cláusulas de fatos que possam ser abduzidos, isto é, dos quais possam ser escolhidas hipóteses candidatas. Esse conjunto de cláusulas permite que a busca seja restrita previamente àqueles fatores que potencialmente possam desempenhar um papel causal no domínio escolhido. Outra estratégia é adicionar critérios de seleção para avaliar e escolher explicações. Vários critérios de seleção foram propostos, incluindo a do *conjunto mínimo*, que prefere um conjunto de

hipóteses a outro, onde ambos sejam consistentes, e impliquem no que devia ser explicado, se o primeiro estiver contido no segundo. Um critério de *simplicidade* dá preferência a conjuntos de hipóteses parcimoniosos, isto é, contendo um menor número de suposições não verificadas (Levesque, 1989).

Tanto a minimalidade quanto a simplicidade podem ser vistas como aplicação da navalha de Occam. Infelizmente, o critério do conjunto mínimo tem o seu poder limitado como uma ferramenta de poda do processo de busca; ele apenas elimina as explicações finais que são superconjuntos de explicações existentes. A simplicidade sozinha também é de validade questionável como um critério de seleção para a busca. Não é difícil construir exemplos nos quais uma explicação que requer um conjunto de hipóteses maior é preferível a um conjunto de hipóteses mais simples, mas mais superficial. De fato, mecanismos causais complexos normalmente requerem conjuntos de hipóteses maiores; entretanto, a abdução de tais mecanismos causais pode ser plenamente justificada, sobretudo quando a presença de certos elementos-chave desses mecanismos já tiver sido verificada por observação.

Dois outros mecanismos para seleção de explicações são também interessantes porque levam em consideração tanto as propriedades do conjunto de hipóteses como as propriedades do procedimento de prova. O primeiro deles, a *abdução baseada em custo*, atribui um custo tanto para hipóteses potenciais como para regras. O custo total da explicação é calculado com base no custo total das hipóteses, mais o custo das regras usadas para abduzir as hipóteses. Conjuntos de hipóteses candidatos são, então, comparados de acordo com seus custos. Esse esquema permite a associação natural de uma heurística probabilística (Charniak e Shimony, 1990). Hipóteses com custos mais altos representam eventos menos prováveis; regras com custos mais altos representam mecanismos causais menos prováveis. Métricas baseadas em custos podem ser combinadas com algoritmos de busca por custo mínimo, como a busca pela melhor escolha (Capítulo 4), reduzindo, com isso, consideravelmente a complexidade computacional da tarefa.

Um segundo mecanismo, a *seleção baseada em coerência*, é particularmente atraente quando o que deve ser explicado não for uma proposição simples, mas sim um conjunto de proposições. Ng e Mooney (1990) demonstraram que uma métrica de coerência é superior a uma métrica de simplicidade para escolher explicações na análise de textos em linguagem natural. Eles definem coerência como uma propriedade de um grafo de prova em que as explicações mais coerentes são aquelas com maior número de conexões entre um par qualquer de observações e um menor número de partições disjuntas. O critério de coerência é baseado na suposição heurística de que o que devemos explicar é um único evento ou ação com múltiplos aspectos. A justificativa para uma métrica de coerência em compreensão de linguagem natural é baseada nas condições de sucesso de Grice, que considera que o interlocutor tem a obrigação de ser coerente e pertinente (Grice, 1975). Não é difícil estender seu argumento a uma série de outras situações. Em diagnósticos, por exemplo, as observações que compreendem o conjunto inicial de fatos a serem explicados são conciliadas, porque acreditamos que elas estejam relacionadas ao mesmo mecanismo de falha subjacente.

Na Seção 9.1, consideramos extensões à lógica tradicional, que dão suporte ao raciocínio com incertezas ou dados faltantes. A seguir, descrevemos alternativas à lógica para raciocinar em situações de incerteza, incluindo a álgebra de fatores de certeza de Stanford, o raciocínio com conjuntos nebulosos e a teoria da evidência de Dempster-Shafer.

## 9.2 Abdução: alternativas à lógica

As abordagens baseadas em lógica da Seção 9.1 são complicadas e computacionalmente intratáveis para muitas aplicações, especialmente em sistemas especialistas. Alternativamente, vários projetos antigos de sistemas especialistas, como o PROSPECTOR, por exemplo, tentaram adaptar técnicas Bayesianas (Seção 9.3) para a inferência abdutiva. As suposições de independência, as atualizações contínuas de dados estatísticos e os cálculos necessários para dar suporte a algoritmos de inferência estocástica limitam essa abordagem. Um mecanismo de inferência alternativo, projetado para eliminar essas restrições, foi usado em Stanford para o desenvolvimento de sistemas especialistas, incluindo o MYCIN (Buchanan e Shortliffe, 1984).

Ao raciocinar com conhecimento heurístico, os especialistas humanos são capazes de fornecer estimativas adequadas e úteis sobre a confiança nos relacionamentos de regras. Eles ponderam conclusões com termos como *altamente provável*, *improvável*, *quase certamente ou possível*. Claramente, esses pesos não são baseados em uma análise cuidadosa de probabilidades. Em vez disso, eles mesmos são heurísticas derivadas da experiência em raciocinar sobre o domínio do problema. Na Seção 9.2, introduzimos três metodologias para a inferência abdutiva: a teoria sobre a certeza de Stanford, o raciocínio nebuloso e a teoria da evidência de Dempster-Shafer. Na Seção 9.3, apresentamos abordagens estocásticas para a incerteza.

### 9.2.1 Álgebra dos fatores de certeza de Stanford

A *teoria das certezas de Stanford* é baseada em uma série de observações. A primeira é que, na teoria das probabilidades tradicional, a soma da confiança em uma relação e da confiança contra a mesma relação deve resultar em um. Entretanto, frequentemente ocorre que um especialista humano tem, digamos, uma confiança 0,7 (em 1,0) de que uma relação seja verdadeira e não tem qualquer ideia sobre ela não ser verdadeira. Outra suposição que sustenta a teoria das certezas é que o conteúdo de conhecimento das regras é muito mais importante que a álgebra para calcular as confianças. As medidas de confiança correspondem a avaliações informais que especialistas humanos colocam nas suas conclusões, tais como “é provavelmente verdadeiro”, “é quase certamente verdadeiro” ou “é altamente improvável”.

A teoria das certezas de Stanford faz algumas suposições simples para criar medidas de confiança e tem, também, algumas regras simples para combinar essas confianças à medida que o programa avança em direção à sua conclusão. A primeira suposição é separar a “confiança em” da “confiança contra” uma relação:

Denomine  $MC(H | E)$  a medida de crença de uma hipótese  $H$  dada a evidência  $E$ .

Denomine  $MD(H | E)$  a medida de descrença de uma hipótese  $H$  dada a evidência  $E$ .

Agora, apenas uma das duas condições abaixo é verdadeira:

$1 > MC(H | E) > 0$  enquanto  $MD(H | E) = 0$  ou

$1 > MD(H | E) > 0$  enquanto  $MC(H | E) = 0$ .

Cada uma dessas duas medidas restringe a outra, já que uma porção de evidência fornecida é a favor ou contra uma hipótese particular, uma diferença importante entre a teoria das certezas e a teoria das probabilidades. Uma vez que a ligação entre essas medidas de crença e descrença tenha sido estabelecida, elas podem ser combinadas novamente por:

$$FC(H | E) = MC(H | E) - MD(H | E).$$

À medida que o fator de certeza (FC) se aproxima de 1, a evidência é maior a favor de uma hipótese; conforme FC se aproxima de -1, a confiança contra a hipótese se torna mais forte; e um FC em torno de 0 indica que existe pouca evidência tanto a favor como contra a hipótese ou então que a evidência a favor e a evidência contra estão平衡adas.

Quando especialistas estabelecem uma base de regras, eles devem estabelecer um FC para cada regra. Esse FC reflete a sua confiança na credibilidade da regra. Medidas de certeza podem ser ajustadas para otimizar o desempenho do sistema, embora pequenas variações na medida de certeza tendam a ter pouco efeito sobre a execução global do sistema. Esse papel das medidas de certeza confirma a crença que “o conhecimento é que determina o desempenho”, isto é, a integridade do conhecimento é que dá suporte à produção de diagnósticos corretos.

As premissas para cada regra são formadas por es e ou de uma série de fatos. Quando uma regra de produção é usada, os fatores de certeza associados a cada condição da premissa são combinados para produzir uma medida de certeza para a premissa global, como apresentado a seguir. Sendo P1 e P2 premissas da regra:

$$FC(P1 \text{ e } P2) = MIN(FC(P1), FC(P2)) \text{ e}$$

$$FC(P1 \text{ ou } P2) = MAX(FC(P1), FC(P2)).$$

O FC combinado das premissas, usando as regras anteriores, é, então, multiplicado pelo FC da própria regra para obter o FC para as conclusões da regra, dadas aquelas premissas. Considere, por exemplo, a seguinte regra de uma base de conhecimento:

$$(P1 \text{ e } P2) \text{ ou } P3 \rightarrow R1(0,7) \text{ e } R2(0,3)$$

onde P1, P2 e P3 são premissas e R1 e R2 são as conclusões da regra, tendo FCs de 0,7 e 0,3, respectivamente. Esses números são adicionados à regra quando ela é concebida e representam a confiança do especialista na conclusão, se todas as premissas forem conhecidas com total certeza. Se a execução do programa produzir P1, P2 e P3 com FCs de 0,6, 0,4 e 0,2, respectivamente, então R1 e R2 poderão ser acrescentados aos resultados coletados, específicos do caso, com FCs de 0,28 e 0,12, respectivamente. A seguir, mostramos os cálculos para esse exemplo:

$$FC(P1(0,6) \text{ e } P2(0,4)) = MIN(0,6, 0,4) = 0,4$$

$$FC((0,4) \text{ ou } P3(0,2)) = MAX(0,4, 0,2) = 0,4.$$

O FC para R1 é 0,7 na regra, assim R1 é acrescentado ao conjunto do conhecimento específico do caso com o FC associado de  $(0,7) \times (0,4) = 0,28$ .

O FC para R2 é 0,3 na regra, assim R2 é acrescentado ao conjunto do conhecimento específico do caso com o FC associado de  $(0,3) \times (0,4) = 0,12$ .

Precisamos ainda de outra medida: como combinar múltiplos FCs quando duas ou mais regras derem suporte ao mesmo resultado R. Essa regra reflete o procedimento análogo para a teoria das certezas, daquele da teoria das probabilidades de multiplicar as medidas de probabilidade para combinar evidências independentes. Usando essa regra repetidamente, podemos combinar os resultados de qualquer número de regras que sejam usados para determinar um resultado R. Suponha que FC(R1) seja o fator de certeza atual associado ao resultado R e que uma regra não usada previamente produza o resultado R (novamente) com FC(R2); então, o novo FC de R é calculado por:

$$FC(R1) + FC(R2) - (FC(R1) \times FC(R2)) \text{ quando } FC(R1) \text{ e } FC(R2) \text{ são positivos,}$$

$$FC(R1) + FC(R2) + (FC(R1) \times FC(R2)) \text{ quando } FC(R1) \text{ e } FC(R2) \text{ são negativos,}$$

e, caso contrário,

$$\frac{FC(R1) + FC(R2)}{1 - MIN(|FC(R1)|, |FC(R2)|)}$$

onde  $|X|$  é o valor absoluto de X.

Além de serem fáceis de calcular, essas equações para combinação dos FCs têm outras propriedades desejáveis. Em primeiro lugar, os FCs resultantes dessa regra estão sempre entre 1 e -1. Segundo, o resultado da combinação de FCs contraditórios é que eles se cancelam, como é realmente desejável. Além disso, a medida combinada de FC é uma função monotonicamente crescente (decrescente), como seria de se esperar para a combinação de evidências.

Por fim, as medidas de confiança produzidas segundo a tradição do fator de certeza de Stanford são estimativas humanas (subjetivas) de medidas de probabilidade para sintomas/causas. Como mencionado na Seção 5.4, pela tradição Bayesiana se A, B e C influenciam D, devemos isolar e combinar apropriadamente todas as probabilidades *a priori* e *a posteriori* incluindo  $P(D)$ ,  $P(D|A)$ ,  $P(D|B)$ ,  $P(D|C)$  e  $P(D|A,B)$  etc., quando desejamos raciocinar acerca de D. A tradição do fator de certeza de Stanford permite que o engenheiro do conhecimento acondicione todas essas relações em um único fator de confiança, FC, associado à regra; isto é, se A e B e C, então  $D(FC)$ . Verifica-se que essa álgebra simples reflete melhor o modo como os especialistas humanos combinam e propagam crenças.

A teoria das certezas pode ser criticada por ser excessivamente *ad hoc*. Embora ela seja definida em uma álgebra formal, o significado das medidas de certeza não é tão rigorosamente fundamentado como a teoria formal das probabilidades. Entretanto, a teoria das certezas não visa produzir uma álgebra para o raciocínio “correto”. Em vez disso, ela é a “lubrificação” que permite que o sistema especialista combine confiâncias, conforme ele avança no espaço de busca. As suas medidas são *ad hoc* do mesmo modo que a confiança de um especialista humano nos seus resultados é aproximada, heurística e informal. Quando o MYCIN é executado, os FCs são usados na busca heurística para dar uma prioridade aos objetivos a serem tentados e, também, como um ponto de corte

para que um objetivo não precise mais ser considerado. Mas, muito embora o FC seja usado para manter o programa executando e coletando informação, o poder do programa continua residindo na qualidade das regras.

### 9.2.2 Raciocinando com conjuntos nebulosos

Há duas suposições que são essenciais para o uso da teoria de conjuntos formal. A primeira diz respeito à pertinência a conjuntos: para qualquer elemento e um conjunto pertencente a um universo, o elemento é um membro do conjunto ou é um membro do complemento do conjunto. A segunda suposição, conhecida como *a lei do meio excluído*, afirma que um elemento não pode pertencer a um conjunto e, também, ao seu complemento. Essas duas suposições são violadas na *teoria dos conjuntos nebulosos* de Lotfi Zadeh. Do ponto de vista dos conjuntos nebulosos, os conjuntos e as leis de raciocínio da lógica tradicional são denominados *nítidos*.

A principal argumentação de Zadeh (Zadeh, 1983) é que, embora a teoria das probabilidades seja apropriada para medir a aleatoriedade da informação, ela é inapropriada para medir o *significado* da informação. Na verdade, grande parte da confusão acerca do uso de palavras e frases em um idioma como o português, por exemplo, está relacionada à falta de clareza (vagueza), e não a questões de aleatoriedade. Esse é um ponto crucial para analisar estruturas de linguagem e pode, também, ser importante para criar uma medida de confiança em regras de produção. Zadeh propõe a *teoria das possibilidades* como uma medida de vagueza, da mesma forma que a teoria das probabilidades mede a aleatoriedade.

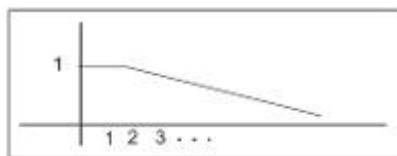
A teoria de Zadeh expressa a falta de precisão de uma maneira quantitativa, introduzindo uma função de pertinência a conjuntos, que pode assumir valores entre 0 e 1. Essa noção de um *conjunto nebuloso* pode ser descrita como: seja  $S$  um conjunto e  $s$  um membro desse conjunto. Um subconjunto nebuloso  $F$  de  $S$  é definido por uma função de pertinência  $m_F(s)$  que mede o “grau” com que  $s$  pertence a  $F$ .

A Figura 9.6 mostra um exemplo típico de um conjunto nebuloso, onde  $S$  é o conjunto dos inteiros positivos e  $F$  é o subconjunto nebuloso de  $S$ , denominado inteiros *pequenos*. Vários valores inteiros podem ter uma distribuição de “possibilidade” que define a sua “pertinência nebulosa” ao conjunto dos inteiros pequenos:  $m_F(1) = 1,0$ ,  $m_F(2) = 1,0$ ,  $m_F(3) = 0,9$ ,  $m_F(4) = 0,8$ , ...,  $m_F(50) = 0,001$  etc. Para a declaração de que o inteiro positivo  $X$  é um *inteiro pequeno*,  $m_F$  cria uma distribuição de possibilidades ao longo de todos os inteiros positivos ( $S$ ).

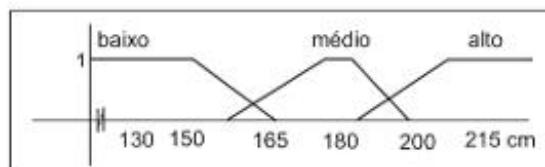
A teoria dos conjuntos nebulosos não está preocupada com como essas distribuições de possibilidades são criadas, mas sim com as regras para calcular as possibilidades combinadas sobre expressões que contêm variáveis nebulosas. Por isso, ela fornece regras para combinar medidas de possibilidades para expressões contendo variáveis nebulosas; de fato, as leis para as operações ou, e e não dessas expressões são semelhantes áquelas apresentadas há pouco para a álgebra de fatores de certeza de Stanford (ver Seção 9.2.1).

Para a representação por conjuntos nebulosos de inteiros pequenos (Figura 9.6), cada inteiro pertence a esse conjunto com uma medida de confiança associada. Na lógica tradicional dos conjuntos “nítidos”, a confiança de que um elemento está em um conjunto deve ser 1 ou 0. A Figura 9.7 apresenta funções de pertinência a conjuntos que representam os conceitos referentes a *baixo*, *médio* e *alto* para a altura de homens. Note que qualquer pessoa pode pertencer a mais do que um conjunto, por exemplo, um homem de 190 cm pertence tanto ao conjunto dos homens *médios* como ao conjunto dos homens *altos*.

**Figura 9.6** Representação por conjunto nebuloso para inteiros pequenos.



**Figura 9.7** Uma representação por conjuntos nebulosos para os conjuntos dos homens baixos, médios e altos.

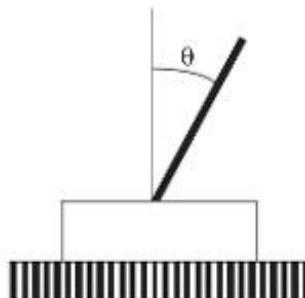


Demonstramos, a seguir, regras para combinar e propagar medidas nebulosas apresentando parte de um problema, que agora é clássico na literatura dos conjuntos nebulosos, um controlador para um pêndulo invertido. A Figura 9.8 apresenta um pêndulo invertido, o qual desejamos manter equilibrado, na posição vertical. Conseguimos manter o pêndulo equilibrado movendo a base do sistema de modo a contrabalançar a força da gravidade que age sobre ele. Existem conjuntos de equações diferenciais que, de modo determinístico, podem manter o pêndulo em equilíbrio (Ross, 1995). A vantagem da abordagem nebulosa para controlar esse sistema de pêndulo é que se pode estabelecer um algoritmo para controlar o sistema eficientemente e em tempo real. A seguir, mostramos esse controlador.

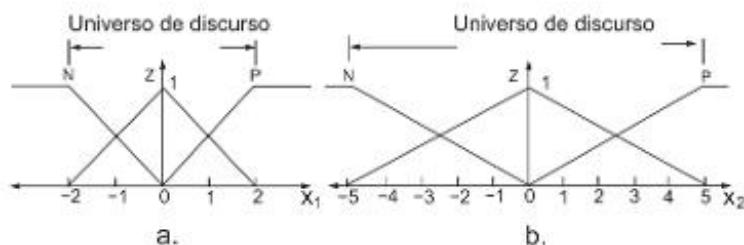
Simplificamos o problema do pêndulo representando-o em duas dimensões. Há duas medidas que usamos como valores de entrada para o controlador, como pode ser visto na Figura 9.9: o ângulo  $\theta$ , que é o desvio do pêndulo da vertical, e a velocidade  $d\theta/dt$  na qual o pêndulo se move. Essas duas medidas são positivas no quadrante à direita da vertical e são negativas à esquerda. Esses dois valores são fornecidos ao controlador nebuloso a cada iteração do sistema. A saída do controlador é um movimento e uma direção para a base do sistema, instruções que pretendem manter o pêndulo em equilíbrio.

Para esclarecer as ações do controlador nebuloso, descrevemos, a seguir, o processo de solução por conjuntos nebulosos. Os dados que descrevem o estado do pêndulo,  $\theta$  e  $d\theta/dt$ , são interpretados como medidas nebulosas (veja a Figura 9.9) e são apresentados a um conjunto de regras nebulosas. Normalmente, pode-se tornar esse passo muito eficiente pelo uso de uma estrutura chamada de *matriz associativa nebulosa*, ou MAN (Figura 9.12), onde as relações de entrada e saída são codificadas diretamente. Diferentemente da solução de problemas baseada em regras tradicional, aqui as regras não são encadeadas entre si. Em vez disso, todas as regras habilitadas por casamento disparam e seus resultados são, então, combinados. Esse resultado, normalmente representado por uma área do espaço de parâmetros de saída nebuloso (Figura 9.10) é, então, “desnevoado” para retornar à resposta de controle. Note que tanto a entrada original como a saída eventual do controlador são valores nítidos. Eles são leituras exatas de um monitor, as entradas, e instruções precisas para o controle, a saída.

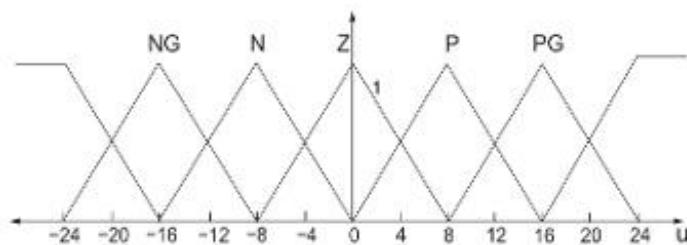
**Figura 9.8** O pêndulo invertido e os valores de entrada referentes ao ângulo  $\theta$  e a  $d\theta/dt$ .



**Figura 9.9** Regiões nebulosas para os valores de entrada  $\theta$  (a) e a  $d\theta/dt$  (b).



**Figura 9.10** Regiões nebulosas do valor de saída  $u$  indicando o movimento da base do pêndulo.



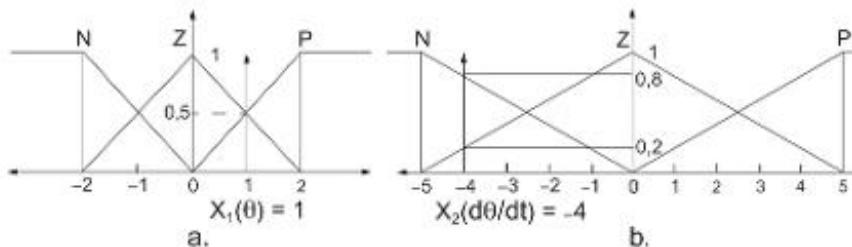
Descrevemos, a seguir, as regiões nebulosas para os valores de entrada  $\theta$  e  $d\theta/dt$ . Esse exemplo simplifica a situação, por exemplo, no número de regiões nebulosas de valores de entrada, mas mostra o ciclo completo de aplicação das regras e a resposta do controlador. O valor de entrada  $\theta$  é dividido em três regiões, Negativo, Zero e Positivo, onde  $\theta$  se encontra no intervalo entre  $-2$  e  $+2$  radianos, como pode ser visto na Figura 9.9(a). A Figura 9.9(b) representa as três regiões nas quais o segundo valor de entrada,  $d\theta/dt$ , é dividido, novamente Negativo, Zero e Positivo, ocupando o intervalo entre  $-5$  e  $+5$  graus por segundo.

A Figura 9.10 representa a divisão do espaço de saída, onde usamos as cinco regiões médias, Negativo Grande, Negativo, Zero, Positivo e Positivo Grande. A medida, entre  $-24$  e  $+24$ , representa o movimento e a direção de cada resposta.

Suponha que a simulação comece e os primeiros valores apresentados ao controlador sejam  $\theta = 1$  e  $d\theta/dt = -4$ . A Figura 9.11 reflete o enevoamento dessas medidas de entrada. Em cada situação, o valor de entrada afeta duas regiões do espaço de entrada nebuloso. Para  $\theta$ , os valores são Zero, com medida de possibilidade de 0,5, e Positivo, com medida de possibilidade de 0,5. Para  $d\theta/dt$ , esses valores são Negativo, com 0,8 de medida de possibilidade, e Zero, com 0,2 de medida de possibilidade.

A Figura 9.12 apresenta uma forma simplificada da matriz associativa nebulosa para esse problema. Os valores de entrada da tabela para  $\theta$ , ou  $x_1$ , estão na primeira coluna à esquerda e, para  $d\theta/dt$ , ou  $x_2$ , eles estão na linha superior da matriz. A tabela  $3 \times 3$  do canto inferior direito da MAN fornece os valores de saída. Por exemplo, se  $\theta$  for Positivo e  $d\theta/dt$  for Negativo, a MAN retorna o valor Zero para o movimento do sistema do pêndulo. Note que a resposta ainda precisa ser desenevoada a partir da região de saída Zero da Figura 9.10.

Nesse caso, devem-se aplicar quatro regras, já que cada valor de entrada toca duas regiões do espaço de entrada. Como pode ser visto anteriormente, as regras de combinação para sistemas nebulosos são similares às da álgebra dos fatores de certeza de Stanford. Na verdade, Zadeh (Buchanan e Shortliffe, 1984) foi o primeiro (historicamente) a propor essas regras de combinação para a álgebra do raciocínio nebuloso. Se as medidas de duas premissas forem combinadas por uma operação E, a medida da regra é calculada como o mínimo das medidas das premissas. Se fizermos um OU de duas premissas, toma-se o máximo de suas medidas.

**Figura 9.11** Enevoamento das medidas de entrada  $x_1 = 1$ ,  $x_2 = -4$ .**Figura 9.12** A matriz associativa nebulosa (MAN) para o problema do pêndulo. Os valores de entrada estão à esquerda e acima.

$x_2$	P	Z	N
$x_1$			
P	PG	P	Z
Z	P	Z	N
N	Z	N	NG

No nosso exemplo, todos os pares de premissas são combinados pela operação  $\ominus$ , de modo que o resultado da medida da regra é calculado como o mínimo das medidas de suas premissas:

$$\text{SE } x_1 = P \text{ E } x_2 = Z, \text{ ENTÃO } u = P \\ \min(0.5, 0.2) = 0.2 P$$

$$\text{SE } x_1 = P \text{ E } x_2 = N, \text{ ENTÃO } u = Z \\ \min(0.5, 0.8) = 0.5 Z$$

$$\text{SE } x_1 = Z \text{ E } x_2 = Z, \text{ ENTÃO } u = Z \\ \min(0.5, 0.2) = 0.2 Z$$

$$\text{SE } x_1 = Z \text{ E } x_2 = N, \text{ ENTÃO } u = N \\ \min(0.5, 0.8) = 0.5 N$$

A seguir, os resultados de saída são combinados. No nosso exemplo, unimos as duas áreas da Figura 9.10, indicadas pelos resultados desse conjunto de regras disparadas. Existem diversas técnicas possíveis de desnevoamento (Ross, 1995). Escolhemos uma das mais comuns, o *método do centroíde*. Nesse método, o centroíde da união das áreas dos valores de saída é o valor que o controlador aplica ao pêndulo. A união, bem como o centroíde da união, são apresentados na Figura 9.13. Depois que essa saída ou resultado é aplicada ao sistema,  $\theta$  e  $d\theta/dt$  são novamente amostrados e o ciclo de controle é repetido.

Há uma série de questões que não abordamos ao descrever sistemas de raciocínio nebuloso, incluindo padrões de oscilações no processo de convergência e taxas de amostragem ótimas. Os sistemas nebulosos, especialmente na área de controle, oferecem aos engenheiros uma ferramenta poderosa e eficiente para lidar com a imprecisão de medidas.

coletadas, esperamos que esses intervalos encolham, representando o aumento de confiança nas hipóteses. Em um domínio Bayesiano, começariamos provavelmente (sem evidências) distribuindo igualmente as probabilidades *a priori* entre as duas hipóteses, atribuindo para cada uma  $p(h_i) = 0,5$ . Dempster-Shafer deixam claro que não temos nenhuma evidência quando começamos; a abordagem Bayesiana, por outro lado, pode resultar na mesma medida de probabilidade, não importando quantos dados tenhamos. Assim, Dempster-Shafer pode ser muito útil quando é importante tomar uma decisão com base na quantidade de evidência que foi coletada.

Resumindo, Dempster e Shafer tratam do problema de medir a certeza fazendo uma distinção essencial entre falta de certeza e ignorância. Na teoria das probabilidades, somos forçados a expressar a extensão de nosso conhecimento acerca de uma hipótese  $h$  em um único número,  $p(h)$ . O problema com esse método, segundo Dempster e Shafer, é que simplesmente não podemos saber sempre os valores das probabilidades que a suportam e, portanto, qualquer escolha particular de  $p(h)$  pode não ser justificada.

As funções de crença de Dempster-Shafer satisfazem axiomas que são mais fracos que aqueles da teoria das probabilidades, isto é, elas se reduzem à teoria das probabilidades, quando todas as probabilidades são alcançáveis. As funções de crença nos permitem usar nosso conhecimento para ligar a atribuição de probabilidades a eventos na ausência de probabilidades exatas.

A teoria de Dempster-Shafer é baseada em duas ideias: primeiro, a ideia de obter graus de crença para uma questão a partir de probabilidades subjetivas para questões correlacionadas e, segundo, o uso de uma regra para combinar esses graus de crença quando eles são baseados em itens independentes de evidência. A regra de combinação foi proposta originalmente por Dempster (1968). A seguir, apresentamos um exemplo informal do raciocínio de Dempster-Shafer, em seguida apresentamos a regra de Dempster e, por fim, aplicamos essa regra a uma situação mais realística.

Suponha que eu tenha probabilidades subjetivas para a confiabilidade da minha amiga Melissa. A probabilidade de ela ser confiável é de 0,9 e a probabilidade de ela não ser confiável é de 0,1. Suponha que Melissa me diga que o meu computador foi invadido. Essa afirmação é verdadeira se Melissa for confiável, mas ela não é necessariamente falsa se ela não for confiável. Assim, o testemunho de Melissa sozinho justifica um grau de crença de 0,9 de que o meu computador foi invadido e uma crença de 0,0 de que ele não foi invadido. Uma crença de 0,0 não significa que eu tenho certeza de que meu computador não foi invadido, como seria o caso com uma probabilidade de 0,0. Ela simplesmente significa que o testemunho de Melissa não me dá motivos para acreditar que meu computador não tenha sido invadido. A medida de plausibilidade,  $pl$ , nessa situação é:

$$pl(\text{computador\_invadido}) = 1 - \text{cre}(\neg(\text{computador\_invadido})) = 1 - 0,0$$

ou 1,0 e minha medida de crença para Melissa é [0,9 1,0]. Note que ainda não há evidência que meu computador não tenha sido invadido.

A seguir, consideraremos a regra de Dempster para combinar evidências. Suponha que meu amigo Bill também me diga que meu computador foi invadido. Suponhamos que a probabilidade de Bill ser confiável seja de 0,8 e a de ele não ser confiável, de 0,2. Eu também devo supor que os testemunhos de Bill e de Melissa sobre o meu computador sejam independentes entre si, ou seja, que eles tenham suas próprias razões para me dizer que acham que meu computador foi invadido. O evento de Bill ser confiável precisa também ser independente da confiabilidade de Melissa. A probabilidade de tanto Bill como Melissa serem confiáveis é o produto de suas confiabilidades, ou 0,72; a probabilidade de eles não serem ambos confiáveis é o produto 0,02. A probabilidade de ao menos um dos dois ser confiável é de  $1 - 0,02$ , ou 0,98. Já que ambos disseram que meu computador foi invadido e como existe uma probabilidade de 0,98 de ao menos um deles ser confiável, então será atribuído ao evento de meu computador ter sido invadido um grau de crença de [0,98 1,0].

Suponha que Bill e Melissa discordem sobre o fato de meu computador ter sido invadido: Melissa afirma que ele foi e Bill afirma que não. Nesse caso, os dois não podem estar corretos nem podem ser confiáveis. Ou ambos não são confiáveis ou apenas um deles é confiável. A probabilidade *a priori* de apenas Melissa ser confiável é de  $0,9 \times (1 - 0,8) = 0,18$ , de apenas Bill ser confiável é de  $0,8 \times (1 - 0,9) = 0,08$  e de nenhum dos dois ser confiável é  $0,2 \times 0,1 = 0,02$ . Dado que ao menos um deles não é confiável,  $(0,18 + 0,08 + 0,02) = 0,28$ , podemos também calcular a probabilidade *a posteriori* de apenas Melissa ser confiável como  $0,18/0,28 = 0,643$  e de meu computador

ter sido invadido, ou a probabilidade *a posteriori* de apenas Bill estar certo,  $0,08/0,28 = 0,286$  e de meu computador não ter sido invadido.

Acabamos de usar a regra de Dempster para combinar crenças. Quando Melissa e Bill declararam a invasão do computador, somamos as três situações hipotéticas que dão suporte à invasão: Bill e Melissa são confiáveis; Bill é confiável e Melissa não; e Melissa é confiável e Bill não. A crença, 0,98, é a soma desses cenários hipotéticos possíveis de suporte. No segundo uso da regra de Dempster, as testemunhas discordaram. Somamos novamente todos os cenários possíveis. A única situação impossível era que ambos fossem confiáveis; assim, ou Melissa era confiável e Bill não era, ou Bill era confiável e Melissa não era, ou nenhum dos dois era confiável. A soma desses três fatores resulta em uma crença de invasão de 0,64. A crença de meu computador não ter sido invadido (a opinião de Bill) é de 0,286; como a plausibilidade de invasão é de  $1 - \text{cre}(\text{não(invasão)})$ , ou 0,714, a medida de crença é de [0,28 0,714].

Para usar a regra de Dempster, obtemos graus de crença para uma pergunta (meu computador foi invadido?) a partir de probabilidades de outra pergunta (as testemunhas são confiáveis?). A regra começa com a suposição de que as perguntas para as quais temos probabilidades são independentes, mas que essa independência é apenas *a priori*. Ela desaparece quando há conflito entre os diferentes itens de evidência.

O uso da abordagem de Dempster-Shafer em uma situação específica envolve resolver dois problemas relacionados. Primeiro, separamos as incertezas da situação em partes de evidência *a priori* independentes. Segundo, empregamos a regra de Dempster. Essas duas tarefas são relacionadas entre si: suponhamos, novamente, que Bill e Melissa tenham me dito, independentemente, que eles acreditam que o meu computador tenha sido invadido. Suponhamos, também, que eu tenha chamado um técnico para verificar o meu computador e que Bill e Melissa tenham testemunhado isso. Por causa desse evento comum, eu não posso mais comparar graus de crença. Entretanto, se eu considerar explicitamente a possibilidade do trabalho do técnico com o meu computador, então eu tenho três itens independentes de evidência: a confiabilidade de Melissa, a confiabilidade de Bill e a evidência da presença do técnico, que eu posso, então, combinar pela regra de Dempster.

Suponha que tenhamos um conjunto exaustivo de hipóteses mutuamente exclusivas que chamamos de Q. O nosso objetivo é atribuir uma medida de crença  $m$  aos vários subconjuntos Z de Q; m é chamada, algumas vezes, de *função densidade de probabilidade* para um subconjunto de Q. Na realidade, nem toda evidência dá suporte diretamente aos elementos individuais de Q. De fato, na maioria das vezes, as evidências dão suporte a diferentes subconjuntos Z de Q. Além disso, como supomos que os elementos de Q são mutuamente exclusivos, a evidência em favor de alguns elementos pode ter efeito sobre a nossa crença em outros elementos. Em um sistema puramente Bayesiano (Seção 9.3), tratamos essas duas situações listando todas as combinações de probabilidades condicionais. No sistema Dempster-Shafer, tratamos essas interações manipulando diretamente os conjuntos de hipóteses. A quantidade  $m_n(Z)$  mede a quantidade de crença que é atribuída ao subconjunto Z de hipóteses, e n representa o número de fontes de evidência.

Pela regra de Dempster, temos:

$$m_n(Z) = \frac{\sum_{X \cap Y = Z} m_{n-2}(X)m_{n-1}(Y)}{1 - \sum_{X \cap Y = \emptyset} m_{n-2}(X)m_{n-1}(Y)}$$

Por exemplo, a crença em uma hipótese Z, com  $n = 3$  fontes de evidência,  $m_3(Z)$ , é a soma dos produtos das situações hipotéticas,  $m_1(X)$  e  $m_2(Y)$ , cuja concorrência dá suporte a Z, isto é,  $X \cap Y = Z$ . O denominador da regra de Dempster indica, como veremos no próximo exemplo, que X e Y podem ter uma interseção vazia, e a soma das confianças deve ser normalizada por um menos a soma desses valores.

A seguir, aplicamos a regra de Dempster a uma situação de diagnóstico médico. Suponhamos que Q represente o domínio de nosso foco, contendo quatro hipóteses: que um paciente tem resfriado (R), gripe (G), enxaqueca (E) ou meningite (M). Nossa tarefa é associar medidas de crença a conjuntos de hipóteses do domínio Q. Como foi mencionado, esses são *conjuntos de hipóteses*, já que uma evidência não precisa dar suporte, exclusivamente, a hipóteses individuais. Por exemplo, ter febre pode dar suporte a {R,G,M}. Como os elementos de Q são tratados como hipóteses mutuamente exclusivas, uma evidência em favor de alguns pode afetar a crença em outros. Como já mencionado, a abordagem de Dempster-Shafer trata das interações manipulando os conjuntos de hipóteses diretamente.

Para a função densidade de probabilidade,  $m$ , e todos os subconjuntos  $Z$  do conjunto  $Q$ , a quantidade  $m(q_i)$  representa a crença que é atribuída correntemente a cada  $q_i$  de  $Q$ , sendo a soma de todos os  $m(q_i)$  igual a um. Se  $Q$  contiver  $n$  elementos, então existem  $2^n$  subconjuntos de  $Q$ . Embora lidar com  $2^n$  valores possa parecer assustador, normalmente se verifica que muitos dos subconjuntos jamais ocorrem. Assim, há uma simplificação do processo de solução, já que esses valores podem ser ignorados, pois eles não têm utilidade no domínio do problema. Finalmente, a plausibilidade de  $Q$  é  $p_l(Q) = 1 - \sum m(q_i)$ , onde os  $q_i$  são os conjuntos de hipóteses que têm alguma crença de suporte. Se não tivermos informação sobre qualquer hipótese, como é o caso frequentemente quando iniciamos um diagnóstico, então  $p_l(Q) = 1.0$ .

Suponha que a primeira evidência seja que o paciente tenha febre e que isso suporte  $\{R, G, M\}$  em 0,6. Denominamos essa primeira crença  $m_1$ . Se essa for a nossa única hipótese, então  $m_1\{R, G, M\} = 0,6$ , onde  $m_1(Q) = 0,4$ , para levar em consideração a distribuição de crença restante. É importante notar que  $m_1(Q) = 0,4$  representa o restante da nossa distribuição de crença, isto é, todas as outras crenças possíveis em  $Q$ , e não a nossa crença no complemento de  $\{R, G, M\}$ .

Suponha que adquirimos, agora, dados novos para o diagnóstico; digamos que o paciente tem muita náusea, o que sugere  $\{R, G, E\}$  com nível de suporte de 0,7. Para essa crença,  $m_2$ , temos  $m_2\{R, G, E\} = 0,7$  e  $m_2(Q) = 0,3$ . Usamos a regra de Dempster para combinar essas duas crenças,  $m_1$ , e  $m_2$ . Sendo  $X$  o conjunto de subconjuntos de  $Q$  para os quais  $m_1$  atribui um valor diferente de zero, e  $Y$  o conjunto de subconjuntos de  $Q$  para os quais  $m_2$  atribui um valor diferente de zero. Criamos, então, uma crença combinada,  $m_3$ , definida sobre subconjuntos  $Z$  de  $Q$  usando a regra de Dempster.

Ao aplicar a regra de Dempster ao diagnóstico, note que não há conjuntos  $X \cap Y$  que sejam vazios e, assim, o denominador é 1. A distribuição de crenças para  $m_3$  é vista na Tabela 9.1.

Usando a regra de Dempster, os quatro conjuntos  $Z$ , todas as maneiras possíveis de realizar a interseção de  $X$  e  $Y$  constituem a coluna mais à direita da Tabela 9.1. O nível de crença de cada conjunto é calculado multiplicando-se as crenças para os elementos correspondentes de  $X$  e  $Y$  sob  $m_1$  e  $m_2$ , respectivamente. Note também que, nesse exemplo, cada conjunto em  $Z$  é único, o que frequentemente não é o caso.

Estendemos o exemplo pela última vez para mostrar como conjuntos de crença vazios são fatorados na análise. Suponha que tenhamos um fato novo, o resultado de uma cultura em laboratório que está associado à meningoite. Temos, agora,  $m_4\{M\} = 0,8$  e  $m_4(Q) = 0,2$ . Podemos usar a fórmula de Dempster para combinar  $m_3$ , os resultados de nossa análise prévia, com  $m_4$  para obter  $m_5$ , como pode ser visto na Tabela 9.2.

Primeiro, note que  $m_5\{M\}$  é produzido pelas interseções de dois pares diferentes de conjuntos, de modo que  $m_5\{M\} = 0,240$ . Temos, também, o caso em que várias interseções produzem o conjunto vazio  $\{\}$ . Assim, o denominador para a equação de Dempster é  $1 - (0,336 + 0,224) = 1 - 0,56 = 0,44$ . A função de crença combinada final para  $m_5$  é:

$$m_5\{M\} = 0,545$$

$$m_5\{R, G, E\} = 0,127$$

$$m_5\{R, G\} = 0,191$$

$$m_5\{R, G, M\} = 0,082$$

$$m_5\{\} = 0,56$$

$$m_5\{Q\} = 0,055$$

Três comentários finais. Primeiro, uma crença grande atribuída ao conjunto vazio, como em  $m_5\{\} = 0,56$ , indica que há evidências conflitantes dentro do conjunto de crenças  $m_1$ . Na verdade, concebemos esse exemplo para mostrar várias características do raciocínio de Dempster-Shafer e, como consequência, sacrificamos a integridade médica. Além disso, quando existem grandes conjuntos de hipóteses, bem como conjuntos de evidências

**Tabela 9.1** Usando a regra de Dempster para obter uma distribuição de crenças para  $m_3$ .

$m_1$	$m_2$	$m_3$
$m_1\{R, G, M\} = 0,6$	$m_2\{R, G, E\} = 0,7$	$m_3\{R, G\} = 0,42$
$m_1(Q) = 0,4$	$m_2\{R, G, E\} = 0,7$	$m_3\{R, G, E\} = 0,28$
$m_1\{R, G, M\} = 0,6$	$m_2\{Q\} = 0,3$	$m_3\{R, G, M\} = 0,18$
$m_1(Q) = 0,4$	$m_2\{Q\} = 0,3$	$m_3\{Q\} = 0,12$

**Tabela 9.2** Usando a regra de Dempster para combinar  $m_3$  e  $m_4$  para obter  $m_5$ .

$m_3$	$m_4$	$m_5$ (sem denominador)
$m_3\{R,G\} = 0,42$	$m_4\{M\} = 0,8$	$m_5\{\} = 0,336$
$m_3\{Q\} = 0,12$	$m_4\{M\} = 0,8$	$m_5\{M\} = 0,096$
$m_3\{R,G\} = 0,42$	$m_4\{Q\} = 0,2$	$m_5\{R,G\} = 0,084$
$m_3\{Q\} = 0,12$	$m_4\{Q\} = 0,2$	$m_5\{Q\} = 0,024$
$m_3\{R,G,E\} = 0,28$	$m_4\{M\} = 0,8$	$m_5\{\} = 0,224$
$m_3\{R,G,M\} = 0,18$	$m_4\{M\} = 0,8$	$m_5\{M\} = 0,144$
$m_3\{R,G,E\} = 0,28$	$m_4\{Q\} = 0,2$	$m_5\{R,G,E\} = 0,056$
$m_3\{R,G,M\} = 0,18$	$m_4\{Q\} = 0,2$	$m_5\{R,G,M\} = 0,036$

complexos, os cálculos para os conjuntos de crenças podem se tornar complexos, mesmo assim, como mencionado anteriormente, a quantidade de cálculos é, ainda, consideravelmente menor que para o raciocínio Bayesiano. Finalmente, a abordagem de Dempster-Shafer é uma ferramenta muito útil quando as conclusões Bayesianas mais fortes não podem ser justificadas.

Dempster-Shafer é um exemplo de uma álgebra que suporta o uso de probabilidades subjetivas no raciocínio. Muitas vezes, constatamos que as probabilidades subjetivas refletem melhor o raciocínio de um especialista humano. Na próxima seção, a última do Capítulo 9, consideraremos outras técnicas de raciocínio baseadas em extensões da regra de Bayes, introduzida na Seção 5.3.

### 9.3 Abordagem estocástica para a incerteza

Usando a teoria das probabilidades, podemos determinar, frequentemente a partir de um argumento *a priori*, as chances de ocorrência de eventos. Podemos descrever, também, como combinações de eventos são capazes de influenciar um ao outro. Embora os retoques finais da teoria das probabilidades tivessem que esperar pelos matemáticos do início do século XX, incluindo Fisher, Neyman e Pearson, a tentativa de criar uma álgebra combinatória remonta aos gregos, passando pela Idade Média, incluindo Llull, Porfírio e Platão (Glymour et al., 1995a). A ideia que suporta a teoria das probabilidades é que podemos entender a frequência com que eventos ocorreram no passado e usar essa informação (como um viés indutivo) para interpretar e raciocinar a respeito de dados do presente.

No Capítulo 5, notamos que há uma série de situações em que a análise probabilística é apropriada. Por exemplo, quando o mundo é genuinamente aleatório, como no caso de um jogo com cartas bem embaralhadas, ou ao se girar uma roleta. Além disso, muitos eventos do mundo podem não ser verdadeiramente aleatórios, mas é impossível conhecer e medir, suficientemente bem, todas as causas e suas interações para que se possa prever eventos; as correlações estatísticas são um substituto útil para a interpretação do mundo. Outro papel da estatística é o de base para a indução automatizada e o aprendizado de máquina (por exemplo, o algoritmo ID3 da Seção 10.3). Por fim, trabalhos recentes tentaram ligar diretamente as noções de probabilidade e causalidade (Glymour e Cooper, 1999; Pearl, 2000).

O mecanismo de inferência principal nos domínios estocásticos é uma forma da regra de Bayes. Porém, conforme observamos na Seção 5.3, o uso completo da inferência Bayesiana em domínios complexos rapidamente se

torna intratável. Os modelos gráficos probabilísticos são projetados especificamente para resolver essa complexidade, além de serem "... um casamento entre a teoria da probabilidade e a teoria dos grafos" (Jordan, 1999).

Os modelos gráficos probabilísticos também vêm se tornando mais importantes no aprendizado de máquina (Capítulo 13) porque podem resolver os problemas de incerteza e complexidade, problemas que estão presentes (e podem ser fundamentalmente limitadores) para a IA moderna. Jordan explica que o problema de modularidade é a base dos módulos gráficos: partes simples de um modelo são combinadas usando a teoria da probabilidade, que dá suporte à consistência do sistema como um todo e, ao mesmo tempo, integra o modelo gráfico com os dados de uma aplicação (Jordan, 1999). Ao mesmo tempo, a teoria dos grafos oferece aos modelos gráficos tanto um modo intuitivo de representar conjuntos de componentes altamente interativos quanto uma estrutura de dados que dá suporte a algoritmos de inferência eficientes.

Na Seção 9.3 e novamente no Capítulo 13, consideramos dois tipos de modelos gráficos probabilísticos: as redes Bayesianas de crença, direcionadas, e as diversas formas dos modelos de Markov, e as árvores de grupos, não direcionadas, e os campos aleatórios de Markov. Os campos de Markov e outros modelos gráficos não direcionados são capazes de representar muitas dependências cíclicas que não são possíveis de representar com grafos direcionados. Por outro lado, as redes Bayesianas de crença são modelos gráficos direcionados capazes de capturar relações e dependências deduzidas implícitas de forma mais precisa e eficiente do que com os modelos gráficos não direcionados.

Nas próximas seções, apresentamos as redes Bayesianas de crença (RBCs) e diversas técnicas de inferência projetadas especificamente para resolver a complexidade computacional dos grafos não direcionados e/ou o raciocínio Bayesiano completo. Mais adiante, na Seção 9.3.5, apresentamos os processos de Markov e mencionamos diversas variações representativas importantes. Uma delas é a rede Bayesiana dinâmica, e a outra é o processo discreto de Markov.

Uma das principais limitações das RBC e dos MOMs (modelos ocultos de Markov) tradicionais é a sua inerente natureza proposicional. Para tratar dessa limitação, foram feitas várias tentativas de mesclar a lógica proposicional, o cálculo de predicados completo, baseado em variável, com os modelos gráficos probabilísticos. Resumimos vários desses esforços na Seção 9.3.7 e retornamos aos sistemas probabilísticos de primeira ordem no Capítulo 13.

### 9.3.1 Um modelo gráfico direcionado: a rede Bayesiana de crença

Embora a teoria de probabilidade Bayesiana, conforme apresentada no Capítulo 5, ofereça uma base matemática para o raciocínio sob condições de incerteza, a complexidade encontrada ao aplicá-la a domínios de problema reais pode ser proibitiva. Felizmente, podemos podar essa complexidade focando a busca em conjuntos menores de eventos e evidências mais relevantes. Uma abordagem, as *redes Bayesianas de crença* (Pearl, 1988), oferece um modelo computacional para o raciocínio, para explicar o melhor possível um conjunto de dados no contexto das relações causais esperadas de um domínio de problema.

As redes Bayesianas de crença podem reduzir bastante o número de parâmetros do modelo Bayesiano completo e mostram como os dados de um domínio (ou mesmo a ausência de dados) podem dividir e focar o raciocínio. Além disso, a estrutura modular de um domínio de problema muitas vezes permite que o projetista do programa faça muitas suposições de independência não permitidas no modelo Bayesiano completo. Na maioria das situações, não é necessário construir uma grande tabela de probabilidades conjuntas, na qual sejam listadas as probabilidades para todas as combinações possíveis de eventos e evidências. Em vez disso, especialistas humanos podem selecionar os fenômenos locais que eles sabem que interagirão e obter as medidas de probabilidades e influências que refletem apenas esses grupos de eventos. Os especialistas supõem que todos os outros eventos são condicionalmente independentes ou que as suas correlações são tão pequenas que podem ser ignoradas. As RBCs tornam essas intuições precisas.

Como um exemplo de rede Bayesiana de crença, considere novamente o exemplo do tráfego da Seção 5.4, representado pela Figura 9.14. Lembre-se de que obras na estrada é O, um acidente, A, a presença de cones laranjas, C, tráfego ruim, T, e luzes piscando, L. Para calcular a probabilidade conjunta de todos os parâmetros do exemplo,

tomando a abordagem Bayesiana completa, é preciso o conhecimento ou as medidas para os parâmetros de todos os estados particulares. Assim, a probabilidade conjunta, utilizando uma sequência de variáveis ordenada de modo topológico, é:

$$p(O,A,C,T,L) = p(O) p(A|O) p(C|O,A) p(T|O,A,C) p(L|O,A,C,T)$$

O número de parâmetros nessa probabilidade conjunta é  $2^5$ , ou 32. Essa tabela é exponencial no número de parâmetros envolvidos. Para um problema de qualquer complexidade, digamos, com trinta ou mais parâmetros, a tabela de distribuição conjunta teria mais de um bilhão de elementos!

Note, porém, que se pudermos dar suporte à suposição de que os parâmetros desse problema dependem apenas das probabilidades de seus pais, ou seja, se pudermos supor que os nós são independentes de todos os não descendentes, dado o conhecimento de seus pais, o cálculo de  $p(O,A,C,T,L)$  torna-se:

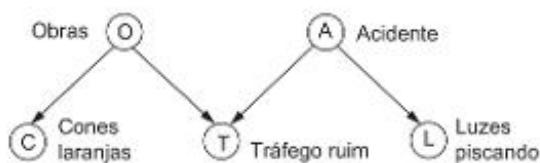
$$p(O,A,C,T,L) = p(O) * p(A) * p(C|O) * p(T|O,A) * p(L|A)$$

Para ver melhor as simplificações que fizemos, considere  $p(C|O,A)$  da equação anterior que reduzimos para  $p(C|O)$  nessa equação mais recente. Essa redução se baseia na suposição de que obras na estrada não é um efeito causal para haver um acidente. Da mesma forma, a presença de cones laranjas não é uma causa de tráfego ruim, mas obras e acidente o são, dando como resultado  $p(T|O,A)$ , em vez de  $p(T|O,A,C)$ . Finalmente, a relação probabilística  $p(L|O,A,C,T)$  foi reduzida para  $p(L|A)$ . A distribuição de probabilidades para  $p(O,A,C,T,L)$  tem, agora, apenas 20 (em vez de 32) parâmetros. Se formos para um problema mais realístico, digamos com 30 variáveis, e se cada estado tiver, no máximo, dois pais, haverá, no máximo, 240 elementos na distribuição. Se cada estado tiver três pais, o máximo será de 490 elementos na distribuição: consideravelmente menos que o bilhão necessário para a abordagem Bayesiana completa!

Precisamos justificar essa dependência de um nó em uma rede de crença em relação apenas aos seus pais. Os elos entre os nós de uma rede de crença representam as probabilidades condicionais para a influência causal. No raciocínio que usa inferência causal está implícita a suposição de que essas influências são diretas, isto é, a presença de um evento *causa* outros eventos na rede. Além disso, o raciocínio de influência causal não é circular, ou seja, um efeito não pode circular de volta para causar ele mesmo. Por essas razões, as *redes Bayesianas de crença* têm uma representação natural como *grafos direcionados acíclicos* (GDA) (Seção 3.1.1), onde padrões coerentes de raciocínio podem ser refletidos como caminhos por meio de relações entre causa e efeito. As redes Bayesianas de crença são um exemplo do que costuma ser chamado de *modelos gráficos*.

No caso do nosso exemplo de tráfego, temos uma situação ainda mais forte, onde não há ciclos não direcionados. Isso nos permite calcular, de forma muito simples, a distribuição de probabilidades em cada nó. A distribuição de nós que não têm pais é acessada diretamente. Os valores dos nós filhos são calculados usando apenas as distribuições de probabilidades dos pais de cada filho, fazendo os cálculos apropriados sobre a tabela de probabilidades condicionais do nó filho e as distribuições dos pais. Isso é possível porque não precisamos nos preocupar com as correlações entre os pais de outros nós não descendentes. Isso produz uma separação abutiva natural em que acidente não tem qualquer correlação com a presença de cones laranjas, como é visto no exemplo da Figura 9.14.

**Figura 9.14** Modelo gráfico para o problema de tráfego apresentado inicialmente na Seção 5.3.



Resumimos nossa discussão sobre RBCs com a seguinte definição:

### Definição

#### REDE BAYESIANA DE CRENÇA

Um modelo gráfico é denominado *rede Bayesiana de crença (RBC)* se seu grafo, anotado com probabilidades condicionais, for direcionado e acíclico. Além disso, RBCs consideram que os nós são independentes de todos os seus não descendentes, dado o conhecimento de seus pais.

Uma *rede Bayesiana dinâmica (RBD)* (ou *temporal*) é uma sequência de redes Bayesianas idênticas cujos nós são ligados na dimensão (direcionada) do tempo. Veremos mais sobre a RBD na Seção 9.3.4 e no Capítulo 13; veja também Friedman (1998) ou Ghahramani e Jordan (1997).

Consideramos, a seguir, uma suposição implícita muitas vezes no raciocínio especialista humano: que a presença ou ausência de dados em um domínio (implicitamente causal) pode dividir e focar a busca por explicações dentro desse domínio. Esse fato também tem implicações importantes sobre a complexidade para o espaço de busca.

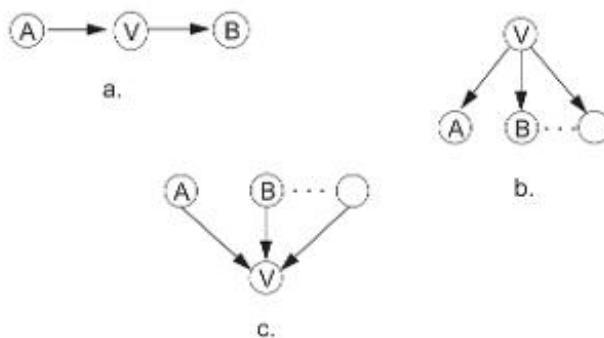
#### 9.3.2 Modelos gráficos direcionados: d-separação

Uma vantagem importante de representar domínios de aplicação como modelos gráficos é que a presença ou a ausência de informações pode levar ao particionamento do modelo e, como resultado, ao controle da complexidade da busca. A seguir, apresentamos vários exemplos disso e, depois, fornecemos a definição de *d-separação*, que dá suporte a essas intuições.

Consideremos o diagnóstico de problemas com óleo em um motor de automóvel: suponha que anéis de pistão gastos causem consumo excessivo de óleo, que, por sua vez, cause uma leitura de nível de óleo baixo. Essa situação está refletida na Figura 9.15(a), onde A é anéis de pistão gastos, V é consumo excessivo de óleo e B é nível de óleo baixo. Se não soubermos nada sobre o consumo excessivo de óleo, então temos uma relação causal entre anéis de pistão gastos e nível de óleo baixo. Entretanto, se um teste mostrar o estado de consumo excessivo de óleo, então anéis de pistão gastos e nível de óleo baixo são independentes entre si.

**Figura 9.15** A Figura 9.15(a) é uma conexão serial de nós onde a influência se propaga entre A e B, a menos que V seja conhecido.

A Figura 9.15(b) é uma conexão divergente, onde a influência se propaga entre os filhos de V, a menos que V seja conhecido. Na Figura 9.15(c), uma conexão convergente, se V não for conhecido, então seus pais são independentes; caso contrário, existe correlação entre seus pais.



Em um segundo exemplo, anéis de pistão gastos podem causar tanto escapamento azul como nível de óleo baixo. Essa situação é mostrada na Figura 9.15(b), onde V representa anéis de pistão gastos, A é escapamento azul e B é nível de óleo baixo. Se soubermos que anéis de pistão gastos é verdadeiro ou falso, então não sabemos se escapamento azul e nível de óleo baixo estão correlacionados; se não tivermos informação sobre anéis de pistão gastos, então escapamento azul e nível de óleo baixo estão correlacionados.

Finalmente, se nível de óleo baixo pode ser causado por consumo excessivo de óleo ou por um vazamento de óleo, então, dado o conhecimento sobre nível de óleo baixo, as suas duas causas possíveis estão correlacionadas. Se o estado de nível de óleo baixo for desconhecido, então as duas causas possíveis são independentes. Além disso, se nível de óleo baixo for verdadeiro, então, estabelecendo-se que vazamento de óleo é verdadeiro, a explicação de consumo excessivo de óleo será afastada. Em todos os casos, a informação sobre nível de óleo baixo é um elemento-chave no processo de raciocínio. Vemos essa situação na Figura 9.15(c), com A como consumo excessivo de óleo, B como vazamento de óleo e V como nível de óleo baixo.

Tornamos essas intuições mais precisas definindo a d-separação de nós em uma rede de crença ou outro modelo gráfico (Pearl, 1988):

### Definição

#### d-SEPARAÇÃO

Dois nós (A e B) em um grafo direcionado acíclico são *d-separados* se todo caminho entre eles estiver bloqueado. Um caminho é qualquer série contínua de conexões no grafo [ligando nós em qualquer direção; por exemplo, existe um caminho de A para B na Figura 9.15(b)]. Um caminho está bloqueado se houver um nó intermediário V no caminho com uma das seguintes propriedades:

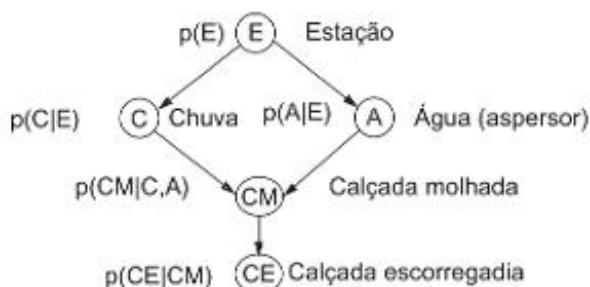
- a conexão é *serial* ou *divergente* e o estado V é conhecido ou
- a conexão é *convergente* e nem V nem outro de seus filhos possuem evidências.

A Figura 9.16 apresenta outros exemplos de relações entre nós seriais, divergentes e convergentes e, também, como a d-separação influencia caminhos de argumentação.

Aproveitando, ainda, os grafos da Figura 9.15, demonstramos como as suposições de uma rede Bayesiana de crença simplificam o cálculo das probabilidades condicionais. Usando a lei de Bayes, qualquer distribuição de probabilidade conjunta pode ser decomposta em um produto de probabilidades condicionais. Assim, na Figura 9.15(a), a probabilidade conjunta das três variáveis, A, V, B é:

$$p(A, V, B) = p(A) p(V|A) p(B|A, V).$$

**Figura 9.16** Um exemplo de uma rede probabilística Bayesiana, onde as dependências das probabilidades estão próximas a cada nó. Esse exemplo é de Pearl (1988).



Usamos a suposição de uma rede Bayesiana de crença de que a probabilidade condicional de uma variável, dado o conhecimento de todos os seus predecessores, é igual à sua probabilidade condicional dado o conhecimento apenas dos seus pais. Como resultado, na equação anterior,  $p(B|A,V)$  se torna  $P(B|V)$ , pois V é um pai de B e A não o é. As distribuições de probabilidades conjuntas para as três redes da Figura 9.15 são:

- a)  $p(A,V,B) = p(A) p(V|A) p(B|V)$ ,
- b)  $p(V,A,B) = p(V) p(A|V) p(B|V)$  e
- c)  $p(A,B,V) = p(A) p(B) p(V|A,B)$ .

Como no exemplo do tráfego mostrado na Figura 9.14, para redes Bayesianas de crença maiores, podem ser eliminadas muito mais variáveis nas probabilidades condicionais. É essa simplificação que torna as redes Bayesianas de crença e outros modelos gráficos muito mais tratáveis computacionalmente que uma análise Bayesiana completa. Em seguida, apresentamos um modelo gráfico mais complexo, contendo um ciclo não direcionado, e propomos um algoritmo de inferência eficiente, a *propagação da árvore de cliques*.

### 9.3.3 Modelos gráficos direcionados: um algoritmo de inferência

Nosso próximo exemplo, adaptado de Pearl (1988), mostra uma rede Bayesiana mais complexa. Como mostrado na Figura 9.16, a estação do ano determina a probabilidade de chuva, bem como a probabilidade de água de um sistema aspersor. A calçada molhada estará correlacionada com a chuva ou a água do aspersor. Finalmente, a calçada estará escorregadia, a depender de se ela for, ou não, uma calçada molhada. Na figura, expressamos a relação probabilística que cada um desses parâmetros tem com os seus pais. Note também que, quando comparado com o exemplo do tráfego, o exemplo da calçada escorregadia tem um ciclo se o grafo for não direcionado.

Desejamos saber agora como a probabilidade de calçada molhada,  $p(CM)$ , pode ser descrita. Isso não pode ser feito como antes, onde  $p(A) = p(A|E)p(E)$  ou  $p(C) = p(C|E)p(E)$ . As duas causas de CM não são mutuamente independentes, por exemplo, se E = verão, então  $p(A)$  e  $p(C)$  poderiam ser aumentados. Assim, devem ser calculadas as correlações completas das duas variáveis, além das suas correlações com E. Nessa situação, podemos fazê-lo, mas como vimos, esse cálculo é exponencial com o número de causas possíveis de CM. Os cálculos estão representados na Tabela 9.3. Calculamos, agora, uma posição dessa tabela, x, onde C e A são verdadeiros; por simplificação, supomos que a estação E ou é quente ou é fria.

$$\begin{aligned} x &= p(C=v, A=v) \text{ para as duas (quente, fria) condições de } E, \text{ estação} \\ &= p(E=\text{quente}) p(C=v | E=\text{quente}) p(A=v | E=\text{quente}) + \\ &\quad p(E=\text{fria}) p(C=v | E=\text{fria}) p(A=v | E=\text{fria}) \end{aligned}$$

Da mesma forma, o restante da Tabela 9.3 pode ser completado. Com isso, compomos a probabilidade conjunta para chuva e água do aspersor. Esse “macroelemento” maior representa  $p(CM) = p(CM | C, A)p(C, A)$ . Obtivemos com isso uma maneira bem razoável de realizar esses cálculos; o problema é que esses cálculos crescem exponencialmente com o número de pais do estado.

**Tabela 9.3** A distribuição de probabilidades para  $p(CM)$ , uma função de  $p(A)$  e  $p(C)$ , dado o efeito de E. Calculamos o efeito para x, onde C = v e A = v.

C	A	$p(CM)$	{
v	v	x	
v	f		
f	v		
f	f		

Chamamos esse macroelemento de *variável combinada*, ou *clique*, para o cálculo de  $p(CM)$ . Introduzimos, agora, o conceito de uma clique, de modo que possamos substituir a propagação de restrições do GDA da Figura 9.16 por uma árvore de cliques acíclica, como mostrado na Figura 9.17. Os retângulos da Figura 9.17(a) refletem as variáveis que as cliques acima e abaixo deles compartilham. A tabela que passa os parâmetros relevantes para a clique seguinte é exponencial no número desses parâmetros. Deve-se notar também que uma variável de ligação deve estar presente na clique com todos os seus pais. Dessa forma, ao se especificar uma rede de crença ou outro modelo gráfico (o processo de engenharia do conhecimento), devemos ter cuidado com o número de variáveis que são pais de um estado. As cliques também terão sobreposições, como mostrado na Figura 9.17(b), para que possam passar informações através de toda a árvore de cliques, denominada árvore de junções. A seguir, apresentamos um algoritmo desenvolvido por Lauritzen e Spiegelhalter (1988) que cria uma árvore de junções a partir de uma rede Bayesiana de crença.

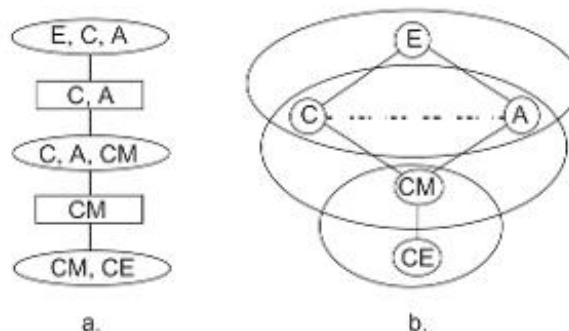
1. Para todos os nós da rede de crença, troque todos os arcos direcionados para não direcionados.
2. Para cada nó, desenhe arcos entre todos os seus pais [a linha tracejada entre C e A na Figura 9.17(b)].
3. No grafo resultante, identifique os ciclos de comprimento maior que três e acrescente novos arcos que reduzam esses ciclos para um comprimento de três. Esse processo é chamado de *triangulação* e não é necessário no exemplo da Figura 9.17(b).
4. Forme a árvore de junções com a estrutura triangulada resultante. Isso é feito encontrando-se as cliques máximas (cliques que são subgrafos completos e não subgrafos de uma clique maior). As variáveis nessas cliques são colocadas em *junções*, e a árvore de junções resultante é criada pela conexão de duas junções quaisquer que compartilham, pelo menos, uma variável, como na Figura 9.17(a).

O processo de triangulação descrito no passo 3 que acabamos de citar é crítico, pois desejamos que a árvore de junções resultante tenha custo computacional mínimo ao propagar informações. Infelizmente, essa decisão de projetar árvores de junções de custo ótimo é NP-difícil. Entretanto, muitas vezes um algoritmo guloso simples pode ser suficiente para produzir resultados úteis. Note que os tamanhos das tabelas necessárias para transmitir informação através da árvore de junções da Figura 9.17 são  $2^2 \cdot 2^2$ ,  $2^2 \cdot 2^2$  e  $2^2$ .

Finalmente, retomamos a rede do exemplo da Figura 9.16 e retornamos à questão da d-separação. Lembre-se de que a ideia da d-separação é que, com alguma informação, no cálculo das distribuições de probabilidade, partes da rede de crença possam ser ignoradas.

1. CE é d-separada de C, E e A, desde que CM seja conhecido.
2. A d-separação é simétrica, isto é, E também é d-separada de CE (e não uma explicação possível para ele), dado o conhecimento de CM.
3. C e A são dependentes por causa de E, mas conhecendo-se E, C e A são d-separados.
4. Se conhecermos CM, então C e A não são d-separados; se não conhecermos E, então C e A o são.
5. Dada a cadeia C → CM → CE, se conhecermos CM, então C e CE são d-separados.

**Figura 9.17** Uma árvore de junções (a) para a rede probabilística Bayesiana de (b). Note que começamos a construir a tabela de transições para o retângulo C, A.



Devemos ser cuidadosos quando temos informação sobre os descendentes de um estado em particular. Por exemplo, se conhecemos CE, então C e A NÃO são d-separados, já que CE está correlacionado com CM, e, conhecendo CM, C e A não são d-separados.

Um comentário final: as redes Bayesianas de crença parecem refletir como as pessoas raciocinam em domínios complexos, onde alguns fatores são conhecidos e relacionados *a priori* com outros. Conforme o raciocínio avança pela ocorrência progressiva de informações, a busca se torna mais restrita e, como resultado, mais eficiente. Essa eficiência da busca contrasta fortemente com a abordagem suportada pelo uso da distribuição conjunta completa de que mais informação produz uma necessidade exponencialmente maior de relações estatísticas, resultando em uma busca mais ampla.

Existe uma série de algoritmos disponíveis para se construir redes de crença e para propagar argumentos conforme novas evidências são adquiridas. Recomendamos, especialmente, a abordagem da *passagem de mensagem*, de Pearl (1988), e o *método da triangulação de árvores de cliques*, proposto por Lauritzen e Spiegelhalter (1988). Druzdzel e Henrion (1993) também propuseram algoritmos para propagar influências em uma rede. Dechter (1996) apresenta o algoritmo de eliminação de pacotes como uma estrutura unificadora para a inferência probabilística.

Na próxima seção, introduzimos as redes Bayesianas dinâmicas.

### 9.3.4 Modelos gráficos direcionados: redes Bayesianas dinâmicas

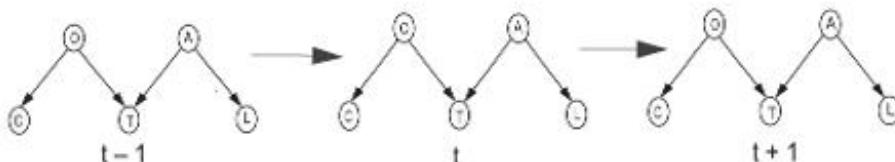
Em seguida, consideraremos uma generalização de redes Bayesianas de crença, a *rede Bayesiana dinâmica* (ou *temporal*) (RBD). A ideia que dá suporte às RBDs é a representação de um estado do domínio ou de um tempo de observação com um conjunto de variáveis aleatórias, não apenas com uma única variável aleatória. Com essa extensão, as redes Bayesianas de crença podem ser usadas para representar a independência condicional entre variáveis de diferentes pontos de vista, por exemplo, entre períodos de tempo.

A maior parte dos eventos que as pessoas encontram, e aos quais devemos reagir de modo inteligente, é temporal: eles se apresentam por períodos de tempo. Dois exemplos podem ser vistos nas figuras 9.18 e 9.19. Na Figura 9.18, tomamos nosso modelo de trânsito da Figura 9.14 e o mapeamos por períodos de tempo. Isso poderia capturar o ponto de vista de um motorista mudando com o tempo, digamos, a cada minuto, pelos parâmetros mutáveis de uma rede Bayesiana. No primeiro período de tempo, o motorista apenas está ciente da diminuição de velocidade, no segundo período de tempo ele nota a presença de cones laranjas, e no terceiro período de tempo existem ainda mais cones etc. A RBD deverá capturar a percepção crescente da obra na estrada, juntamente com uma confiança (muito) menor de um acidente.

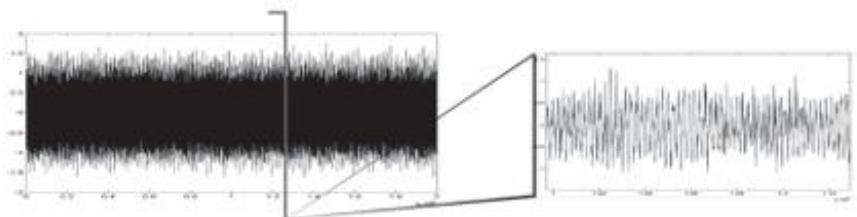
Na Figura 9.19 são apresentadas amostras de dados de sinais vindos de um ambiente complexo. (Estes, na realidade, são múltiplos sinais de calor, vibração e outros sinais vindos do sistema de rotor de um helicóptero.) O que a RBD nessa situação deverá medir é o modo como esses sinais mudam com o tempo e aconselhar o controlador sobre a “saúde contínua” atual do sistema. Consideraremos esse exemplo melhor, além de definir a RBD com mais precisão, na Seção 13.2.

As RBDs são empregadas com mais frequência para rastrear dois tipos de sistemas de variação de tempo. Primeiro, sequências dependentes no tempo, conforme descritas nas figuras 9.18 e 9.19, e segundo, fenômenos que ocorrem naturalmente com o tempo, como a amostragem de fonemas ou palavras em uma aplicação de análise da

**Figura 9.18** O problema de trânsito da Figura 9.14 representado como uma rede Bayesiana dinâmica.



**Figura 9.19** Dados típicos de série temporal a serem analisados por uma rede Bayesiana dinâmica.



linguagem natural. Observe que é possível representar correlações entre variáveis aleatórias na mesma fatia de tempo (correlações instantâneas) com as arestas direcionadas e não direcionadas dos modelos gráficos. Porém as correlações entre elementos de grafos refletindo dados de série temporal devem ser capturadas com modelos gráficos direcionados.

Para resumir: se cada aresta conectando uma série de modelos gráficos representando dados relacionados ao tempo for direcionada para refletir a dimensão de tempo, o modelo é denominado rede Bayesiana dinâmica. A RBD também pode capturar dados sequenciais não temporais, como a linguagem, onde a informação nova reflete a mudança de estado do processamento. Para ver mais detalhes sobre RBDs, consulte Friedman (1998) ou Ghahramani e Jordan (1997) e a Seção 13.2.

### 9.3.5 Modelos de Markov: o processo de Markov discreto

Na Seção 3.1.2, apresentamos *máquinas de estados finitos* como representações gráficas onde os estados são mudados dependendo do conteúdo de um fluxo de entrada. Os estados e suas transições refletem propriedades de uma linguagem formal. Depois, apresentamos um *reconhecedor de estado finito* (a máquina de Moore), uma máquina de estado que era capaz de “reconhecer” sequências com diversas propriedades. Na Seção 5.3, apresentamos a *máquina probabilística de estado finito*, uma máquina de estado em que a função do próximo estado foi representada por uma distribuição de probabilidade sobre o estado atual. O *processo de Markov discreto* é uma especialização dessa técnica, em que o sistema ignora seus valores de entrada.

A Figura 9.20 é uma *máquina de estados de Markov*, às vezes chamada *cadeia de Markov*, com quatro estados distintos. Essa classe geral de sistema pode ser descrita em qualquer período de tempo como estando em um de um conjunto de  $S$  estados distintos,  $s_1, s_2, s_3, \dots, s_n$ . O sistema passa por mudanças de estado, com a possibilidade de permanecer no mesmo estado em intervalos de tempo discretos regulares. Descrevemos o conjunto ordenado de tempos  $T$  que estão associados aos intervalos discretos como  $t_1, t_2, t_3, \dots, t_r$ . O sistema muda de estado de acordo com a distribuição de probabilidades associadas a cada estado. Indicamos o estado real da máquina no tempo  $t$  como  $\sigma_t$ .

Uma descrição probabilística completa desse sistema requer, no caso geral, a especificação do estado atual  $\sigma_t$ , em termos de todos os seus estados predecessores. Assim, a probabilidade do sistema estar em qualquer estado em particular  $\sigma_t$  é:

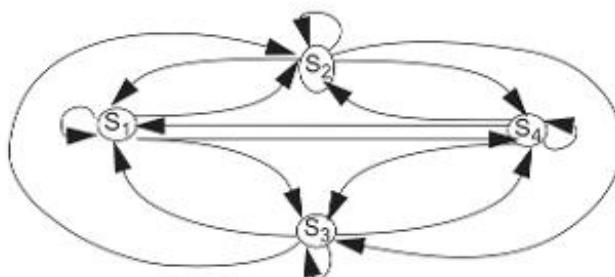
$$p(\sigma_t) = p(\sigma_t | \sigma_{t-1}, \sigma_{t-2}, \sigma_{t-3}, \dots)$$

onde  $\sigma_{t-1}$  são os estados predecessores de  $\sigma_t$ . Em uma *cadeia de Markov de primeira ordem*, a probabilidade do estado atual é uma função somente de seu estado predecessor direto:

$$p(\sigma_t) = p(\sigma_t | \sigma_{t-1})$$

onde  $\sigma_{t-1}$  precede  $\sigma_t$ . (Em geral, um estado em uma *cadeia de Markov de ordem n* depende dos seus  $n$  estados anteriores.) Supomos também que o lado direito dessa equação é invariante no tempo, ou seja, formulamos a hipótese de que, por todos os períodos de tempo do sistema, as transições entre estados específicos retêm as mesmas

**Figura 9.20** Uma máquina de estados de Markov, ou cadeia de Markov, com quatro estados,  $s_1, \dots, s_4$ .



relações probabilísticas. Com base nessas suposições, agora podemos criar um conjunto de probabilidades de transição de estado  $a_{ij}$  entre dois estados  $s_i$  e  $s_j$  quaisquer:

$$a_{ij} = p(\sigma_t = s_i | \sigma_{t-1} = s_j), 1 \leq i, j \leq N$$

Lembre-se de que  $i$  pode igualar  $j$  quando o sistema permanece no mesmo estado. As restrições tradicionais permanecem nessas distribuições de probabilidade; para cada estado  $s_i$ :

$$a_{ij} \geq 0 \text{ e } \sum_{i=1}^N a_{ij} = 1$$

O sistema que acabamos de descrever é denominado *modelo de Markov observável de primeira ordem*, pois a saída do sistema é o conjunto de estados em cada intervalo de tempo discreto, e cada estado do sistema corresponde a um evento físico (observável). Tornamos o modelo de Markov observável mais formal com a definição e depois damos um exemplo.

### Definição

#### MODELO DE MARKOV (OBSERVÁVEL)

Um modelo gráfico é chamado de *modelo de Markov (observável)* se seu grafo for direcionado e a probabilidade de chegar a qualquer estado  $s_i$  do conjunto de estados  $S$  em um tempo discreto  $t$  for uma função das distribuições de probabilidade de estar nos estados anteriores de  $S$  em tempos anteriores. Cada estado  $s_i$  de  $S$  corresponde a uma situação fisicamente observável.

Um modelo de Markov observável é de *primeira ordem* se a probabilidade de estar no estado atual  $s_t$  em qualquer instante  $t$  for uma função apenas de estar no estado anterior  $s_{t-1}$  no instante  $t-1$ , onde  $s_t$  e  $s_{t-1}$  pertencem ao conjunto de estados observáveis  $S$ .

Note que qualquer distribuição de probabilidade tem a propriedade de ser um modelo de Markov. O poder dessa técnica vem das suposições de primeira ordem. Como um exemplo de um modelo de Markov observável de primeira ordem, considere o clima ao meio-dia para determinado local. Consideraremos que esse local tenha quatro estados discretos para a variável clima:  $s_1 = \text{sol}$ ,  $s_2 = \text{nuvem}$ ,  $s_3 = \text{neblina}$ ,  $s_4 = \text{chuva}$ . Os intervalos de tempo para o modelo de Markov serão o meio-dia de cada dia consecutivo. Também consideraremos que as transições entre os estados de clima permanecem constantes com o passar do tempo (o que não é verdade para a maioria dos locais!), e que o observável, clima, pode permanecer no mesmo estado por vários dias. Essa situação é representada pela Figura 9.20, e é apoiada pela matriz de transições de estado  $a_{ij}$ :

	$s_1$	$s_2$	$s_3$	$s_4$
$a_{ij} =$	$s_1$	0,4	0,3	0,2
	$s_2$	0,2	0,3	0,2
	$s_3$	0,1	0,3	0,3
	$s_4$	0,2	0,3	0,2

Nessa matriz de transição  $a_{ij}$ , semelhante à máquina de estados finitos da Seção 3.1.2, a primeira linha representa as transições de  $s_1$  para cada um dos estados, incluindo permanecer no mesmo estado; a segunda linha são as transições de  $s_2$ , e assim por diante. As propriedades exigidas para que as transições sejam distribuições de probabilidade de cada estado são atendidas (elas somam 1,0).

Agora, fazemos perguntas do nosso modelo. Suponha que hoje,  $s_1$ , tenha sol; qual é a probabilidade dos próximos cinco dias terem sol? Ou qual é a probabilidade de os próximos cinco dias serem de sol, sol, nuvem, nuvem, chuva? Para resolver esse segundo problema, determinamos a probabilidade de o primeiro dia,  $s_1$ , ter o sol observado hoje:

$$O = s_1, s_1, s_1, s_2, s_2, s_4$$

A probabilidade desses estados observados, dado o modelo de Markov de primeira ordem, M, é:

$$\begin{aligned} p(O | M) &= p(s_1, s_1, s_1, s_2, s_2, s_4 | M) \\ &= p(s_1) \times p(s_1 | s_1) \times p(s_1 | s_1) \times p(s_2 | s_1) \times p(s_2 | s_2) \times p(s_4 | s_2) \\ &= 1 \times a_{11} \times a_{11} \times a_{12} \times a_{22} \times a_{24} \\ &= 1 \times (0,4) \times (0,4) \times (0,3) \times (0,3) \times (0,3) \\ &= 0,00432 \end{aligned}$$

Isso vem da suposição de primeira ordem de Markov, onde o clima a cada dia é uma função (somente) do clima do dia anterior. Também observamos o fato de que hoje o dia está ensolarado.

Podemos estender esse exemplo para determinar, visto que sabemos o clima de hoje, a probabilidade de o clima ser o exatamente mesmo para os próximos t dias, ou seja, que o clima permanecerá igual até o dia  $t + 1$  em que ele será diferente. Para qualquer estado de clima  $s_i$ , e modelo de Markov M, temos a observação O:

$$O = \{s_i, s_i, s_i, \dots, s_i, s_i\}, \text{ onde há exatamente } (t+1) s_i, \text{ e onde } s_i \neq s_j, \text{ então: } p(O | M) = 1 \times a_{ii}^t \times (1 - a_{ii})$$

onde  $a_{ii}$  é a probabilidade de transição de passar do estado  $s_i$  para si mesmo. Esse valor é denominado *função de densidade de probabilidade discreta para a duração de t períodos de tempo no estado  $s_i$*  do modelo M. Essa função de densidade de duração indica a duração do estado em um modelo de Markov. Com base nesse valor, podemos calcular, dentro do modelo M, o número esperado de observações de  $d_i$  ou sua duração dentro de qualquer estado  $s_i$ , dado que a primeira observação está nesse estado:

$$\begin{aligned} d_i &= \sum_{d=1}^n d(a_{ii})^{(d-1)}(1-a_{ii}) \quad \text{onde } n \text{ tende a } \infty, \text{ ou:} \\ &= \frac{1}{1-a_{ii}} \end{aligned}$$

Por exemplo, o número esperado de dias de chuva consecutivos, dado esse modelo, é  $1/(1 - 0,3)$ , ou 1,43. De modo semelhante, o número de dias de sol consecutivos que se pode esperar com esse modelo é de 1,67. Em seguida, resumimos diversas formas de modelos de Markov, muitas delas vistas com mais detalhes no Capítulo 13.

### 9.3.6 Modelos de Markov: variações

Nos modelos de Markov que vimos até aqui, cada estado corresponde a um evento físico discreto e observável, como o valor de clima em determinada hora do dia. Essa classe de modelo é realmente bastante limitada, e agora a generalizamos para uma classe maior de modelos. Descrevemos várias dessas abordagens brevemente. A Seção 13.1 oferece uma apresentação mais abrangente dos modelos ocultos de Markov com diversas variações importantes.

#### **Modelos ocultos de Markov**

Nos modelos de Markov já apresentados, cada estado corresponde a um evento físico observável e discreto. Em um *modelo oculto de Markov*, ou *MOM*, os valores observados são considerados como funções probabilísticas de um estado oculto atual.

Por exemplo, a voz observada das pessoas é um reflexo de um som que a pessoa pretende produzir. O som pronunciado está relacionado apenas probabilisticamente ao estado real ou à intenção da pessoa. Assim, o MOM é um processo estocástico duplamente embutido, onde o processo estocástico observável (o som da pessoa) tem o suporte de um processo estocástico não observável (o estado ou intenção da pessoa). O MOM é um caso especial das RBDs que vimos na Seção 9.3.2. Para obter mais detalhes sobre MOMs, consulte as seções 13.1 e 13.2 e Rabiner (1989).

#### **Modelos semi-Markovianos**

Um *modelo semi-Markoviano* é uma cadeia de Markov de dois componentes. O primeiro componente é responsável pela escolha da *transição do próximo estado*, e o segundo, responsável pelo *tempo de transição*. Quando um modelo semi-Markoviano entra em um estado  $s_i$ , ele permanece nesse estado pelo tempo  $t_i$ , que é tomado de uma distribuição de probabilidade de duração de estado. Quando o tempo  $t_i$  termina, o processo passa para o próximo estado  $s_{i+1}$  de acordo com a distribuição de probabilidade de transição de estado, e o tempo de transição de estado,  $t_{i+1}$ , é selecionado novamente. Um modelo semi-Markoviano tem suporte para quaisquer distribuições de probabilidade de duração de estado em comparação com as transições de tempo fixas especificadas pelos modelos tradicionais de Markov.

#### **Processos de decisão de Markov**

Duas outras variantes dos modelos de Markov, bastante usadas no aprendizado por reforço, são o *processo de decisão de Markov*, ou *MDP* (do inglês *Markov Decision Process*), e o *processo de decisão de Markov parcialmente observável*, ou *POMDP* (do inglês *Partially Observable Markov Decision Process*). O MDP é definido sobre dois espaços: o espaço de estados para o problema em consideração e o espaço de possíveis ações. A transição para um novo estado no espaço de estados depende do estado atual e da ação atual e é orientada pela distribuição de probabilidade condicional. A cada estado, uma recompensa é calculada dado o estado atual e uma ação. Normalmente, a tarefa do aprendizado por reforço é maximizar uma função de recompensa cumulativa sob as condições atuais.

Enquanto o MDP trabalha com um estado observado (determinístico) antes de usar uma matriz de transição baseada em probabilidade para selecionar seu próximo estado, o POMDP tem apenas conhecimento probabilístico de seu estado atual (bem como a matriz de probabilidade para determinar seu próximo estado). Assim, o POMDP pode provar ser computacionalmente intratável para ambientes complexos. Consideramos melhor os MDPs, os POMDPs e o aprendizado por reforço nas seções 10.7 e 13.3.

### 9.3.7 Alternativas de primeira ordem às RBCs para modelagem probabilística

Até este ponto na Seção 9.3, apresentamos representações baseadas no cálculo proposicional para o raciocínio com incerteza. Só é natural perguntar em que sentido os modelos baseados em *lógica de predicados probabilística* poderiam ser usados para o raciocínio. A força da representação pela lógica de predicados completa é uma se-

mântica declarativa junto a uma estrutura representativa baseada em variável. As limitações representativas das técnicas baseadas na lógica proposicional tradicional incluem uma capacidade limitada de lidar com ruído e incerteza, bem como uma visão estática das representações.

Assim, os modelos Bayesiano e Markoviano apresentados na Seção 9.3 são limitados a uma representação de nível proposicional, onde pode ser complicado representar relações gerais (baseadas em variável) com uma distribuição, por exemplo,  $\forall X \text{ homem}(X) \rightarrow \text{esperto}(X)$  (0,49 0,51). Muitos domínios de problema complexos exigem esse nível de expressividade.

Também é importante para domínios complexos, especialmente nas áreas de diagnóstico e prognóstico, poder representar estruturas repetitivas, recursivas e (potencialmente) infinitas. Portanto, se quisermos representar sistemas que mudam de estado com o tempo, precisamos ter esse poder representativo. Isso requer um regime de controle do tipo recursivo, ou repetir até terminar. Se quisermos recriar dinamicamente uma RBC tradicional, somos forçados a reconstruir a rede inteira por períodos de tempo, pois essas estruturas não possuem uma semântica declarativa e orientada a tempo.

A atividade de pesquisa recente criou uma série de extensões importantes aos modelos gráficos, incluindo diversos sistemas de primeira ordem (baseados em variável e recursivos). Há também uma série de modelos Bayesianos hierárquicos e decomponíveis. Listamos vários desses novos formalismos de modelagem que dão suporte a tal representação. No Capítulo 13, daremos exemplos de várias dessas representações quando apresentarmos os métodos probabilísticos incorporados com o aprendizado de máquina.

#### **Construção de rede Bayesiana a partir de bases de conhecimento** (Haddawy, 1994; Ngo e Haddawy, 1997; Ngo et al., 1997)

Haddawy, Ngo e seus colegas montaram um sistema lógico de primeira ordem com redes Bayesianas de crença. Como resultado, eles criaram uma lógica probabilística de primeira ordem usada como uma linguagem de representação para especificar certa classe de redes na forma de uma base de conhecimento. A semântica formal da base de conhecimento (Bayesiana) é imposta usando uma linguagem representacional estritamente definida, especificando as sentenças da base. Haddawy e Ngo forneceram um algoritmo de criação de rede Bayesiana provavelmente correto, implementando uma combinação de uma semântica formal para a linguagem lógica subjacente, bem como uma semântica de dependência para a base de conhecimento. Um recurso do algoritmo de criação é que ele evita produzir nós irrelevantes para a rede, usando evidências especificadas pelo usuário.

Haddawy e Ngo argumentam que o raciocínio se torna muito ineficaz quando uma base de conhecimento se torna muito grande. Como uma solução, eles propõem usar informação de contexto para indexar as sentenças probabilísticas de uma base de conhecimento. Quando um algoritmo de criação de modelo constrói uma rede, ele omite as sentenças da base de conhecimento cujo contexto não é relevante como uma tarefa atual. O modelo resultante é significativamente menor que se a base de conhecimento inteira fosse usada para construção do modelo. Essa técnica de Haddawy, Ngo e seus colegas é uma das primeiras tentativas de pesquisa no campo da modelagem lógica estocástica que usa explicitamente a informação contextual sobre um domínio como um modo de focar a base de conhecimento na informação relevante e reduzir o tamanho de um modelo necessário para resolver uma tarefa de raciocínio em particular.

#### **Programas de lógica Bayesiana, PLBs** (Kersting e DeRaedt, 2000)

*Programas de lógica Bayesiana* oferecem um arcabouço representacional incluindo programação lógica e modelos gráficos. Mais especificamente, PLBs combinam redes Bayesianas de crença com a lógica de cláusula de Horn (Prolog, ver Seção 14.3). Dada uma consulta específica, o sistema gera uma rede Bayesiana para responder à consulta usando um conjunto de regras de primeira ordem com parâmetros de incerteza. A representação resultante é fácil de interpretar e seus projetistas lhe atribuem uma semântica teórica de modelo semelhante à que foi vista na Seção 2.3.

#### **Modelos relacionais probabilísticos, MRPs** (Friedman et al., 1999; Getoor et al., 2001)

Friedman, Getoor e seus colegas desenvolveram outra abordagem para os sistemas probabilísticos de primeira ordem. Em vez de criar uma linguagem tipo lógica declarativa, MRPs especificam um modelo de probabilidade

sobre classes de objetos e definem restrições de probabilidade no nível de classe, de modo que essas restrições possam ser usadas com qualquer objeto dessa classe. Com MRPs, também é possível especificar dependências probabilísticas entre atributos de classes relacionadas. Diversas extensões às MRPs permitem a subclassificação que dá suporte a dependências probabilísticas em níveis de detalhe apropriados (Getoor et al., 2001). Hierarquias de classes permitem a modelagem tanto no nível individual (a rede Bayesiana de crença tradicional) como para classes. Outra extensão dos MRPs é tratar as estruturas relacionais dos próprios modelos como incertas, em vez de fixas. Por esse ponto de vista, há uma relação estreita entre MRPs e bancos de dados relacionais, pois eles podem ter estruturas similares.

#### Redes lógicas de Markov, RLMs (Richardson e Domingos, 2006)

Richardson e Domingos propõem outro sistema probabilístico baseado em lógica, chamado redes lógicas de Markov (RLMs). A maior parte das técnicas de modelagem estocástica anteriores, baseadas em lógica, utiliza subconjuntos restritos da lógica geral, com cláusulas de Horn sendo a representação mais comum. Para remover restrições na lógica, Richardson e Domingos usam a lógica geral de primeira ordem, cujas sentenças são convertidas para uma forma normal conjuntiva (FNC). Elas também usam campos aleatórios de Markov (Pearl, 1988) como um equivalente probabilístico da lógica, outra diferença importante dos sistemas descritos anteriormente (as redes Bayesianas de crença são, de longe, a representação mais utilizada). Consequentemente, a correspondência entre as sentenças lógicas na FNC e os campos aleatórios de Markov é direta. RLMs são a primeira estrutura teórica com uma correspondência completa entre o cálculo de predicados de primeira ordem com símbolos de função e distribuições de probabilidade.

#### Lógica ciclica (Pless et al., 2006; Chakrabarti et al., 2006; Sakhanenko et al., 2006)

A lógica ciclica é uma linguagem probabilística declarativa de primeira ordem e completa de Turing. Ela tem esse nome devido ao uso do algoritmo de *propagação de crença ciclica*, Pearl (1998), para a inferência. Suas representações declarativas baseadas em lógica são primeiro traduzidas para campos aleatórios de Markov. O aprendizado de parâmetro com algoritmo de maximização de expectativa (ME) integra-se bem com a propagação de crença ciclica. Na Seção 13.2, demonstramos o sistema lógico cíclico com seu uso do campo aleatório de Markov e aprendizado de parâmetro baseado em ME.

Há inúmeras outras extensões às RBCs tradicionais que as tornam de primeira ordem e completas de Turing. Outros exemplos de modelagem estocástica de primeira ordem incluem a linguagem *IBAL* de Pfeffer (2001) e o *Cálculo Estocástico Lambda* de Pless e Luger (2002). Entre as representações estocásticas orientadas a objeto se incluem Koller e Pfeffer, 1997, 1998; Pfeffer et al., 1999; Xiang et al., 2000; Pless et al., 2000. Os métodos estocásticos são importantes por todo o campo de IA; veja, por exemplo, os agentes probabilísticos (Kosoresow, 1993). Veremos os métodos estocásticos aplicados ao aprendizado de máquina no Capítulo 13 e ao processamento da linguagem natural na Seção 15.4.

## 9.4 Epílogo e referências

Desde o começo das pesquisas em IA, existe um importante subconjunto da comunidade que considera que a lógica e suas extensões oferecem uma representação suficiente para caracterizar a inteligência. Foram propostas alternativas importantes ao cálculo de predicados de primeira ordem para descrever o raciocínio sob condições de incerteza:

- 1. Lógicas multivaloradas.** Elas estendem a lógica pela adição de novos valores verdade como desconhecido aos valores-padrão de verdadeiro e falso. Com isso, pode-se, por exemplo, fornecer um veículo com a capacidade de distinguir entre asserções que se sabe serem falsas daquelas que simplesmente não se sabe se são verdadeiras.
- 2. Lógicas modais.** A lógica modal adiciona operadores que permitem que um sistema lógico trate problemas de conhecimento e crença, necessidade e possibilidade. Neste capítulo, discutimos operadores modais para *a menos que* e *é consistente com*.

3. **Lógicas temporais.** Lógicas temporais nos permitem quantificar expressões considerando-se o tempo, indicando, por exemplo, que uma expressão é *sempre verdadeira* ou *será verdadeira em algum instante de tempo no futuro*.
4. **Lógicas de ordem mais elevada.** Muitas categorias de conhecimento envolvem conceitos de ordem mais elevada, onde predicados, e não apenas variáveis, podem ser quantificados. Realmente precisamos de lógicas de ordem mais elevada para lidar com esse conhecimento, ou podemos fazer isso com lógica de primeira ordem? Se forem necessárias lógicas de ordem mais elevada, como elas seriam formuladas?
5. **Formulações lógicas de definições, protótipos e exceções.** Exceções são vistas frequentemente como uma característica necessária de um sistema baseado em definições. Entretanto, se o uso de exceções não for cuidadoso, fica prejudicada a semântica da representação. Outra questão é a diferença entre uma definição e um protótipo, ou a representação de um indivíduo *típico*. Qual é a diferença exata entre as propriedades de uma classe e as propriedades de um membro típico? Como devem ser representados os indivíduos protótipos? Quando um protótipo é mais apropriado que uma definição?

As representações baseadas em lógica continuam a ser uma importante área para pesquisa (McCarthy, 1968; Hayes, 1979; Weyhrauch, 1980; Moore, 1982; Turner, 1984).

Há várias outras contribuições importantes ao raciocínio por sistema de manutenção da verdade (SMV). O *SMV baseado em lógica* é originário do trabalho de McAllester (1978). No SMVL, as relações entre proposições são representadas por cláusulas que podem ser usadas para deduzir os valores verdade de qualquer proposição que eles descrevem. Outra abordagem, o *raciocinador por crenças múltiplas*, RCM, é como o raciocinador SMVS (de Kleer, 1984); ideias similares podem ser encontradas em Martins e Shapiro (1983). O RCM se baseia em uma linguagem lógica chamada SWM\* que descreve estados de conhecimento. Em Ginsburg (1997) e Martins (1990, 1991), encontram-se algoritmos para a verificação de inconsistências geradas pelo raciocínio com a base de conhecimento. Para obter outras informações sobre a álgebra de nós que fundamenta SMVJ, veja Doyle (1983) ou Reinfrank (1989). A lógica padrão permite que qualquer teorema inferido em uma extensão de um sistema seja admitido como um axioma para posterior raciocínio. Reiter e Criscuolo (1981) e Touretzky (1986) desenvolvem essas questões.

Existe uma literatura bastante rica sobre raciocínio não monotônico, lógicas de crenças e manutenção de verdade, além dos artigos originais na área (Doyle, 1979; Reiter, 1985; de Kleer, 1986; McCarthy, 1977, 1980). Sobre modelos estocásticos, veja em *Probabilistic Reasoning in Intelligent Systems*, de Pearl (1988); *Readings in Uncertain Reasoning*, de Shafer e Pearl (1990); *Representations of Commonsense Knowledge*, de Davis (1990), e em inúmeros artigos em anais recentes da AAAI, UAI e IJCAI. Recomendamos *The Encyclopedia of Artificial Intelligence*, de Stuart Shapiro (segunda edição, 1992), para a cobertura de muitos dos modelos de raciocínio apresentados neste capítulo. Josephson e Josephson (1994) editaram uma coleção de artigos em *Abductive Inference: Computation, Philosophy, and Technology*. Veja também *Formal Theories of the Commonsense World* (Hobbs e Moore, 1985). Em *Causality*, Pearl (2000) apresenta uma contribuição para se compreender a noção de relações de causa-efeito no mundo.

Outros trabalhos sobre lógica circunscritiva e de modelo mínimo podem ser encontrados em Genesereth e Nilsson (1987), Lifschitz (1986) e McCarthy (1986). O raciocínio de Perlis (1988) sobre a falta de conhecimento de um determinado agente é outra contribuição à inferência circunscritiva. Ginsburg (1987) editou uma importante coleção de artigos sobre sistemas não monotônicos, *Readings in Nonmonotonic Reasoning*.

Para aprofundar o estudo dos sistemas nebulosos, recomendamos o artigo original de Lotfi Zadeh (1983) e as discussões sobre integrações modernas dessa tecnologia, encontradas em *Fuzzy Sets, Neural Networks and Soft Computing*, de Yager e Zadeh (1994), e em *Fuzzy Logic with Engineering Applications*, de Timothy Ross (1995). A solução para o problema do pêndulo invertido, apresentada na Seção 9.2.2, foi adaptada do texto de Ross.

Sobre algoritmos para inferência de redes Bayesianas de crença, recomendamos a abordagem de Pearl (1988), para *passagem de mensagens*, e o *método de triangulação de cliques*, de Lauritzen e Spiegelhalter (1988) (veja a Seção 9.3). Recomendamos a discussão desses algoritmos em *Encyclopedia of AI* (Shapiro, 1992) e Huang e Darwiche, 1996. A edição da primavera de 1996 de *AISB Quarterly* contém uma introdução às *redes Bayesianas de crença* (van der Gaag, 1996); recomendamos também a discussão das *redes probabilísticas qualitativas* de Wellman (1990) e Druzdzel (1996).

As representações estocásticas e os algoritmos estocásticos continuam sendo uma área de pesquisa muito ativa (Xiang et al., 1993; Laskey e Mahoney, 1997). Limitações das representações Bayesianas motivaram a pesquisa de modelos Bayesianos hierárquicos e componíveis (Koller e Pfeffer, 1997, 1998; Pfeffer et al., 1999; Xiang et al., 2000). Outras extensões dos modelos gráficos baseados em proposição que apresentamos podem ser encontradas na literatura. Recomendamos a leitura de Koller e Pfeffer (1998), e Pless et al. (2000) para obter ideias sobre representações estocásticas orientadas a objeto. A linguagem *IBAL*, de Pfeffer (2001), e *Stochastic Lambda Calculus*, de Pless e Luger (2002), são exemplos de linguagens funcionais estocásticas de primeira ordem. Cussens (2001) e Pless e Luger (2003) criaram linguagens baseadas em lógica de primeira ordem (declarativas) para inferência estocástica. Vemos também as muitas referências dadas na Seção 9.3.7 que não foram repetidas aqui.

Veremos muitas questões da Seção 9.3 novamente, especialmente na Parte IV, Capítulo 13, a apresentação das abordagens probabilísticas do aprendizado de máquina.

## 9.5 Exercícios

1. Identifique três domínios de aplicação onde o raciocínio sob condições de incerteza é necessário. Escolha uma dessas áreas e estabeleça seis regras de inferência que refletam o raciocínio nesse domínio.
2. Dadas as regras a seguir de uma aplicação de sistema especialista de “encadeamento para trás”:

$$A \wedge \neg(B) \Rightarrow C (0,9)$$

$$C \vee D \Rightarrow E (0,75)$$

$$F \Rightarrow A (0,6)$$

$$G \Rightarrow D (0,8)$$

O sistema pode concluir os seguintes fatos (com confianças):

$$F(0,9)$$

$$B(-0,8)$$

$$G(0,7)$$

Use a álgebra de fatores de certeza de Stanford para determinar E e sua confiança.

3. Considere a regra simples, semelhante à do MYCIN: Se  $A \wedge (B \vee C) \Rightarrow D (0,9) \wedge E (0,75)$ . Discuta as questões que surgem ao se captar essas incertezas em um contexto Bayesiano. Como essa regra seria tratada em um raciocínio de Dempster-Shafer?
4. Crie um novo exemplo de raciocínio de diagnóstico e use as equações de Dempster-Shafer da Seção 9.2.3 para obter distribuições de crença como nas tabelas 9.1 e 9.2.
5. Use os axiomas de esquemas apresentados no artigo de McCarthy (1980, Seção 4) para criar os resultados de circunscrição apresentados na Seção 9.1.3.
6. Crie outra rede de raciocínio semelhante àquela da Figura 9.4 e mostre o reticulado de dependências para as suas premissas, como foi feito na Figura 9.5.
7. Raciocinar supondo-se um modelo mínimo é importante no dia a dia do ser humano. Elabore dois outros exemplos que supõem modelos mínimos.
8. Continue o exemplo do pêndulo invertido da Seção 9.2.2. Faça mais duas iterações do controlador nebuloso onde a saída de uma iteração se torna a entrada para a próxima iteração.
9. Escreva um programa que implemente o controlador nebuloso da Seção 9.2.2.
10. Procure na literatura, por exemplo, em Ross (1995), e descreva duas outras áreas onde o controle nebuloso poderia ser apropriado. Construa um conjunto de regras nebulosas para esses domínios.
11. Tom investiu algumas de suas economias em um portfólio de cinco ações,  $H = \{\text{Banco da China, Citibank, Intel, Nokia, Legend}\}$ . Essas ações podem ser classificadas em *ações de banco* (Banco da China e Citibank), *ações de alta tecnologia* (Intel, Nokia e Legend) e *ações da China* (Banco da China e Legend). Você se ofereceu para criar um sistema especialista que possa dar conselhos sobre como ele deve administrar seu portfólio de ações. Para tanto, você entrevistou muitos consultores financeiros e chegou a um

conjunto de regras para o sistema especialista. Essas regras dizem a Tom como ele deverá dividir seu dinheiro entre as diferentes ações em seu portfólio. As proporções recomendadas são diretamente proporcionais à probabilidade do aumento do preço das ações ou do tipo da ação em questão:

R1: SE a taxa de juros subir, ENTÃO compre ações de banco, ações de alta tecnologia e Nokia na proporção de 0,8, 0,15 e 0,05, respectivamente.

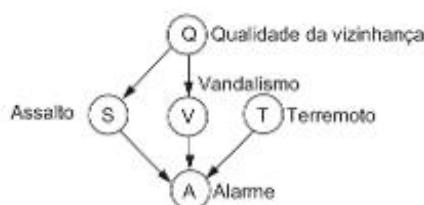
R2: SE a taxa de emprego subir, ENTÃO compre ações do Banco da China, Intel e alta tecnologia na proporção de 0,5, 0,2 e 0,3, respectivamente.

R3: SE a taxa de inflação baixar, ENTÃO compre ações da Intel, Citibank e alta tecnologia na proporção de 0,1, 0,4 e 0,5, respectivamente.

Tom notou que a taxa de juros acabou de subir. Usando a teoria das evidências de Dempster-Shafer, calcule os intervalos de crença para os três tipos de ações, ou seja, as ações de banco, as ações da China e as ações de alta tecnologia, e para a Nokia. Existe alguma diferença na crença e nos intervalos de crença entre ações da Nokia e de alta tecnologia? Por quê? (Esse problema foi proposto por James Liu Nga Kwok, da Universidade Politécnica de Hong Kong. A solução encontrada no Manual do Instrutor, na Sala Virtual para o professor, é de um aluno, Qi Xianming.)

12. Coloque outro elo na Figura 9.16, por exemplo, conectando estação diretamente a calçada molhada, e, então, crie uma árvore de cliques para representar essa situação. Compare as questões de complexidade com as da árvore de cliques da Figura 9.17.
13. Complete as avaliações simbólicas que são necessárias para terminar a Tabela 9.3.
14. Crie um algoritmo para a propagação de crenças Bayesianas e o aplique no domínio da calçada escorregadia da Seção 9.3.2. Você poderia usar a técnica de passagem de mensagens de Pearl (1988) ou o método de triangulação proposto por Lauritzen e Spiegelhalter (1988).
15. Crie um diagrama de crenças Bayesianas para outra aplicação, por exemplo, diagnóstico médico, descoberta geológica ou análise de falhas automotivas. Apresente exemplos de d-separação e crie árvores de cliques para essa rede.
16. Crie cliques e uma árvore de junções para a seguinte situação (vista na Figura 9.21). Assalto, vandalismo ou um terremoto podem disparar o alarme de uma casa. Há também uma medida dos perigos potenciais na vizinhança da casa.
17. Transforme a situação de raciocínio de diagnóstico desenvolvida nas tabelas 9.1 e 9.2 do modelo de Dempster-Shafer da Seção 9.2.3 em uma rede Bayesiana de crença. Indique semelhanças e diferenças entre essas duas abordagens de diagnóstico.
18. Você deseja projetar um modelo de Markov de segunda ordem, ou seja, onde cada estado observável seria dependente dos dois estados observáveis anteriores. Como você faria isso? Como seria a matriz de probabilidade de transição?
19. Dado o modelo de Markov observável para o clima, da Seção 9.3.4:
  - a. Determine a probabilidade de que (exatamente) os próximos cinco dias sejam de sol.
  - b. Qual é a probabilidade de exatamente três dias de sol, depois um dia de chuva e depois exatamente um dia de sol?

**Figura 9.21** Uma rede de crenças representando a possibilidade de um alarme de uma casa disparar em uma vizinhança perigosa.



## Aprendizado de máquina

*Lógica não é o fim da sabedoria, mas o começo...*

—SPOCK, *Star Trek VI*

*"Sei o que você está pensando"*, disse Tweedledum, "mas não é assim, de maneira nenhuma". "Pelo contrário", continuou Tweedledee, "se fosse, poderia ser; e se fosse assim, seria; mas, como não é, não é. Isso é lógica".

—LEWIS CAROLL, *Through the Looking Glass* (1871)

*Vamos ensinar adivinhação...*

—GEORGE POLYA

### Métodos simbólico, conexionista, genético e estocástico para o aprendizado de máquina

Quando questionadas sobre quais habilidades são essencialmente mais humanas e mais difíceis de serem computadorizadas, além da criação artística, da tomada de decisão ética e da responsabilidade social, a maioria das pessoas menciona linguagem e aprendizado. Ao longo dos anos, essas duas áreas têm funcionado como objetivo, desafio e meio de teste para o progresso da IA. Uma das razões de a linguagem e o aprendizado serem áreas de pesquisa difíceis, embora importantes, é que elas englobam muitas outras habilidades inteligentes humanas. Se, em algum momento, reivindicarmos a criação de uma inteligência artificial, deveremos abordar questões sobre linguagem natural, raciocínio automático e aprendizado de máquina. Na Parte IV, consideraremos várias técnicas de aprendizado de máquina; a Parte V apresenta tópicos sobre raciocínio automático e compreensão de linguagem natural.

No Capítulo 10, consideramos métodos de aprendizado simbólico, começando com um conjunto de símbolos que representam entidades e relações de um domínio de problema. Algoritmos de aprendizado simbólico tentam inferir generalizações novas, válidas e úteis que podem ser expressas usando esses símbolos.

As abordagens conexionistas discutidas no Capítulo 11 representam conhecimento como padrões de atividade em redes de pequenas unidades de processamento individuais. Inspiradas na arquitetura dos cérebros de animais, as redes conexionistas aprendem modificando sua estrutura e seus pesos em resposta aos dados de treinamento. Em vez de realizar uma busca entre as possíveis generalizações proporcionadas por uma linguagem de represen-

tação simbólica, os modelos conexionistas reconhecem padrões invariantes em dados e representam esses padrões dentro de sua própria estrutura.

Da mesma forma que as redes conexionistas são inspiradas pelo sistema neural biológico, os modelos emergentes do Capítulo 12 são inspirados por seus análogos genético e evolucionário. Os algoritmos genéticos começam com uma população de soluções candidatas para o problema. As soluções candidatas são avaliadas segundo sua habilidade de resolver ocorrências do problema: apenas as mais ajustadas sobrevivem e se combinam entre si para produzir a próxima geração de possíveis soluções. Assim, emergem soluções cada vez mais poderosas como ocorre no universo darwiniano. Pode-se argumentar que é pouco apropriado o fato de que essas abordagens procuram as origens da inteligência nos mesmos processos que originaram a própria vida.

Por fim, no Capítulo 13, apresentamos as abordagens estocásticas para o aprendizado de máquina. A base para o aprendizado estocástico é a ideia que dá suporte à regra de Bayes: a experiência de situações em um domínio condiciona as expectativas do conhecedor em interpretar novos dados nesse domínio (e em outros). Nossa principal abordagem ocorre por meio da apresentação e da discussão de diversos processos Markovianos. Usamos esses modelos para demonstrar a interpretação de diversos conjuntos de dados no raciocínio diagnóstico e a interpretação de máquina da linguagem. Também apresentamos o aprendizado por reforço, que reflete como a resposta de um agente na solução do problema pode ser apoiada ou desencorajada pela realimentação do domínio desse problema.

Há uma série de importantes questões filosóficas e epistemológicas subjacentes à pesquisa em aprendizado de máquina. Aí se inclui o problema da *generalização*, ou de como uma máquina pode identificar padrões invariantes em dados de uma forma suficiente para usar esses invariantes para a subsequente solução inteligente de problemas, por exemplo, um ajuste a novos dados não vistos anteriormente. Um segundo problema para o aprendizado de máquina é a natureza de um *viés indutivo* no aprendizado. Esse viés se refere a como os projetistas do sistema usaram a própria intuição e heurísticas na concepção de representações, modelos e algoritmos que dão suporte ao processo de aprendizado. A identificação de conceitos “importantes” dentro do domínio do problema, o uso de “um algoritmo específico de busca ou de recompensa”, ou a seleção de uma arquitetura específica de rede neural para o aprendizado são exemplos disso. A questão aqui é que esse viés indutivo tanto habilita o processo de aprendizado quanto limita o que pode ser aprendido pelo sistema. Uma questão filosófica final, chamada de *dilema do empírico*, analisa a situação do ângulo oposto: se não houver um viés predeterminado no sistema de aprendizado, como se pode aprender algo útil, ou, o que é pior, como podemos saber se algo foi aprendido? Essa questão surge frequentemente em modelos não supervisionados e emergentes de aprendizado de máquina. Essas três questões são discutidas novamente na Seção 16.2.

# Aprendizado de máquina: simbólico

*Como eu já disse, se a mente receber um grande número de ideias simples transmitidas pelos sentidos, da forma como elas foram encontradas nas coisas exteriores, ou por reflexão sobre suas próprias operações, e se ela notar que várias dessas ideias simples aparecem constantemente juntas... então, mais tarde, inadvertidamente seremos capazes de considerá-las como uma única ideia simples.*

— JOHN LOCKE, *Essay Concerning Human Understanding*

*A mera observação de algo não tem valor algum. Observar se transforma em notar, notar se transforma em pensar, pensar se transforma em estabelecer conexões, de maneira que podemos dizer que todo olhar atento que lançamos sobre o mundo é um ato de teorizar. Contudo, isso deve ser feito conscientemente, com autocritica, com liberdade e, para usar uma palavra audaciosa, com ironia.*

— GOETHE

## 10.0 Introdução

A capacidade de aprender deve fazer parte de qualquer sistema que reivindique possuir inteligência em um sentido geral. De fato, no nosso mundo de símbolos e interpretações, a própria noção de intelecto imutável parece ser uma contradição nos termos. Agentes inteligentes devem ser capazes de se modificarem ao longo do curso de suas interações com o mundo, bem como pela experiência de seus próprios estados e processos internos. Apresentamos quatro capítulos sobre aprendizado de máquina, refletindo quatro abordagens para esse problema: primeiro, a abordagem simbólica; a seguir, o ponto de vista conexionista; depois, as perspectivas genéticas/evolucionárias; e, por fim, a abordagem dinâmica/estocástica.

O aprendizado é importante para aplicações práticas de inteligência artificial. Feigenbaum e McCorduck (1983) identificaram o “gargalo da engenharia do conhecimento” como o maior obstáculo para o uso em larga escala de sistemas inteligentes. Esse “gargalo” é o custo e a dificuldade de se construir sistemas usando as técnicas tradicionais de aquisição de conhecimento vistas na Seção 7.1. Uma solução para esse problema seria os programas começarem com uma quantidade mínima de conhecimento e aprenderem a partir de exemplos, de aconselhamentos de alto nível ou, ainda, de suas próprias explorações do domínio.

Herbert Simon define o aprendizado como:

qualquer mudança em um sistema que melhore o seu desempenho na segunda vez que ele repetir a mesma tarefa ou outra tarefa tirada da mesma população (Simon, 1983).

Essa definição, embora resumida, sugere muitas das questões envolvidas no desenvolvimento de programas que aprendem. O aprendizado envolve a generalização a partir da experiência: o desempenho deve melhorar não apenas na “repetição da mesma tarefa”, mas também em tarefas semelhantes no domínio. Pelo fato de os domínios interessantes tenderem a ser grandes, um sistema aprendiz normalmente examina apenas uma fração de todos os exemplos possíveis; a partir dessa experiência limitada, o sistema aprendiz deve generalizar corretamente para ocorrências do domínio não vistas anteriormente. Esse é o problema da *indução* que é central para o aprendizado. Na maioria dos problemas de aprendizado, os dados disponíveis não são suficientes para garantir a generalização ideal, não importando que algoritmo seja usado. As máquinas de aprendizado devem generalizar heuristicaamente, isto é, devem selecionar aqueles aspectos da sua experiência que são os mais prováveis de se mostrarem efetivos no futuro. Esses critérios de seleção são conhecidos como *vieses indutivos*.

A definição de Simon descreve o aprendizado como algo que faz com que o sistema “melhore o seu desempenho em uma segunda vez”. Como indica o parágrafo anterior, selecionar as mudanças possíveis de um sistema, de modo que ele melhore o seu desempenho, é uma tarefa difícil. As pesquisas em aprendizado devem lidar com a possibilidade de que as mudanças possam, na realidade, piorar o desempenho do sistema. Outra questão para um algoritmo de aprendizado é prevenir e detectar tais problemas.

Aprender envolve modificações no sistema de aprendizado; isso é óbvio. Entretanto, a natureza exata dessas modificações e a melhor maneira de representá-las não são nada óbvias. Uma abordagem para isso modela o aprendizado como a aquisição de conhecimento do domínio representado de forma explícita. Com base na sua experiência, a máquina de aprendizado constrói ou modifica expressões em uma linguagem formal, como a lógica, por exemplo, e retém esse conhecimento para uso futuro. *Abordagens simbólicas*, caracterizadas pelos algoritmos do Capítulo 10, são construídas sob a suposição de que a principal influência sobre o comportamento do programa seja a sua base de conhecimento do domínio representada explicitamente.

As *redes neurais*, ou *conexionistas*, que veremos no Capítulo 11, não aprendem adquirindo sentenças em uma linguagem simbólica. Como o cérebro de um animal, que consiste em um grande número de células nervosas interconectadas, as redes conexionistas são sistemas de neurônios artificiais interconectados. O conhecimento está implícito na organização e na interação desses neurônios. As redes neurais não aprendem adicionando representações à sua base de conhecimento; em vez disso, elas aprendem modificando a sua estrutura global, de modo a se adaptar às contingências do mundo que habitam. No Capítulo 12, consideraremos o *aprendizado genético e evolucionário*. Esse modelo de aprendizado pode ser visto nos sistemas humano e animal, que evoluíram por meio do equilíbrio com o mundo. Essa abordagem de aprendizado por meio de adaptação se reflete nas pesquisas sobre algoritmos genéticos, programação genética e vida artificial.

Por fim, as técnicas de aprendizado *dinâmica* e *probabilística* são apresentadas no Capítulo 13. Muitas situações complexas no mundo, ou não entendidas totalmente, como a geração e a compreensão da linguagem humana, podem ser mais bem “entendidas” com ferramentas probabilísticas. De modo semelhante, os sistemas dinâmicos, como falhas no funcionamento de um motor mecânico, podem ser diagnosticados.

No Capítulo 10, começamos nossa exploração do aprendizado de máquina com a abordagem simbólica, onde os algoritmos variam nos seus objetivos, nos dados de treinamento disponíveis e nas estratégias de aprendizado e nas linguagens de representação do conhecimento que eles empregam. Entretanto, todos esses algoritmos aprendem por busca em um espaço de conceitos possíveis para encontrar uma generalização aceitável. Na Seção 10.1, descrevemos um arcabouço para o aprendizado de máquina simbólico, que enfatiza as suposições comuns por trás de todo esse trabalho.

Embora a Seção 10.1 descreva uma série de tarefas de aprendizado, este capítulo enfoca basicamente o *aprendizado indutivo*. A indução, que é o aprendizado de uma generalização a partir de um conjunto de exemplos, é uma das tarefas de aprendizado mais fundamentais. O *aprendizado de conceito* é um problema típico de aprendizado indutivo; dados exemplos de um determinado conceito, como “gato”, “doença da soja” ou “investimento em boas ações”, tentamos inferir uma definição que permitirá à máquina de aprendizado reconhecer corretamente futuras ocorrências daquele conceito. As seções 10.2 e 10.3 examinam dois algoritmos usados para indução de conceitos, a *busca em espaço de versões* e o *ID3*.

A Seção 10.4 considera o papel do *vies indutivo* no aprendizado. Os espaços de busca encontrados no aprendizado tendem a ser extremamente grandes, mesmo para os padrões da solução de problemas baseada em busca.

Esses problemas de complexidade são exacerbados pelo problema de escolher entre as diferentes generalizações suportadas pelos dados de treinamento. O viés indutivo se refere a qualquer método que um programa de aprendizado usa para restringir o espaço de generalizações possíveis.

Os algoritmos das seções 10.2 e 10.3 são guiados pelos dados. Eles não usam qualquer conhecimento prévio do domínio do problema, baseando-se em grandes números de exemplos para definir as propriedades essenciais de um conceito geral. Os algoritmos que generalizam com base em padrões encontrados nos dados de treinamento são denominados algoritmos *baseados em similaridade*.

Em contraste com os métodos baseados em similaridade, um algoritmo de aprendizado pode usar conhecimento prévio do domínio para orientar a generalização. Os seres humanos, por exemplo, não precisam de um grande número de exemplos para aprender eficientemente. Quase sempre, um único exemplo, uma analogia, ou uma única orientação de alto nível é suficiente para comunicar um conceito geral. O uso efetivo de tal conhecimento pode ajudar um agente a aprender mais eficientemente e com menor probabilidade de erro. A Seção 10.5 examina o *aprendizado baseado em explicações*, o aprendizado por analogia e outras técnicas que usam conhecimento prévio para aprender a partir de dados de treinamento limitados.

Os algoritmos apresentados nas seções 10.2 a 10.5, embora diferindo quanto às estratégias de busca, linguagens de representação e na quantidade de conhecimento prévio utilizado, supõem que os dados de treinamento são classificados por um professor ou por algum outro método. O algoritmo de aprendizado sempre recebe a informação se uma ocorrência é um exemplo positivo ou negativo de um conceito-alvo. Essa dependência em relação a exemplos de treinamento com classificação previamente conhecida define a tarefa de *aprendizado supervisionado*.

A Seção 10.6 continua o estudo de indução examinando o *aprendizado não supervisionado*, que trata do problema de como um agente inteligente pode adquirir conhecimento útil na ausência de dados de treinamento classificados corretamente. A *formação de categorias*, ou *agrupamento conceitual*, é um problema fundamental no aprendizado não supervisionado. Dado um conjunto de objetos que apresentam várias propriedades, como um agente pode dividir os objetos em categorias úteis? Como podemos saber se uma categoria será útil? Nesta seção, examinamos dois algoritmos para a formação de categorias, CLUSTER/2 e COBWEB.

Finalmente, na Seção 10.7, apresentamos o aprendizado por reforço. Aqui, um agente é localizado em um ambiente e recebe realimentação desse contexto. Esse aprendizado requer que o agente atue e interprete o resultado dessas ações. O aprendizado por reforço difere do aprendizado supervisionado, pois aqui não existe um “professor” que responde diretamente a cada ação; em vez disso, o próprio agente deve criar uma política para interpretar todos os resultados. O aprendizado por reforço se ajusta muito bem à epistemologia construtivista, como descrito na Seção 16.2.

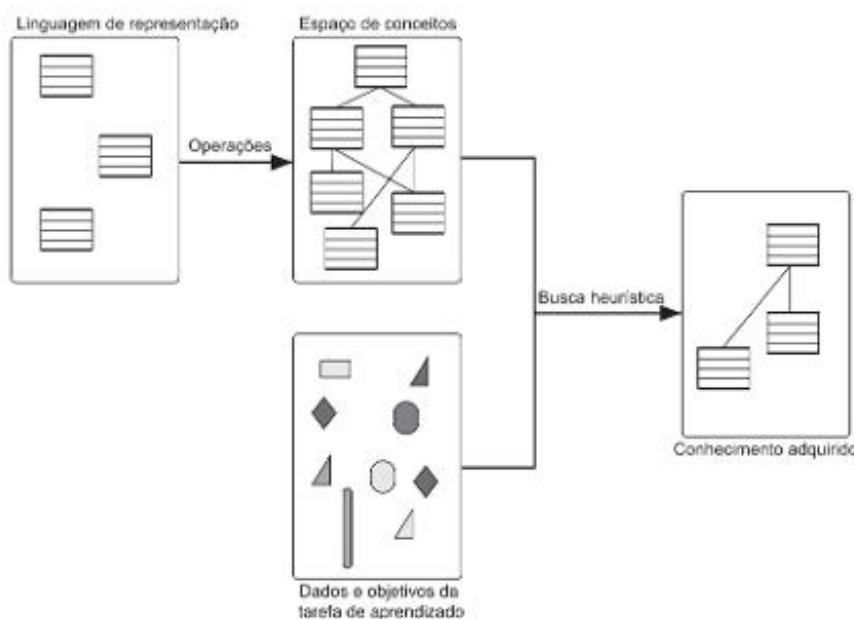
Todas as técnicas de aprendizado apresentadas neste capítulo têm uma coisa em comum: elas são vistas como uma variedade de busca em espaço de estados. Mesmo o aprendizado por reforço deriva uma função de valor sobre um espaço de estados. A seguir, apresentamos um arcabouço geral baseado em busca para o trabalho no aprendizado de máquina.

## 10.1 Um arcabouço para o aprendizado simbólico

Os algoritmos de aprendizado podem ser caracterizados segundo várias dimensões, como mostra a Figura 10.1:

- 1. Dados e objetivos da tarefa de aprendizado.** Uma das principais maneiras de se classificar problemas de aprendizado é de acordo com os objetivos do algoritmo de aprendizado e os dados que são fornecidos a ele. Os algoritmos de aprendizado de conceito das seções 10.2 e 10.3, por exemplo, começam com uma coleção de exemplos positivos (e normalmente negativos) de uma classe-alvo; o objetivo é inferir uma definição geral que permitirá que o algoritmo reconheça ocorrências futuras da classe. Em contraste com a abordagem intensiva de dados seguida por esses algoritmos, o *aprendizado baseado em explicações* (Seção 10.5) tenta inferir um conceito geral a partir de um único exemplo de treinamento e uma base prévia de conhecimento específico do domínio. Os algoritmos de agrupamento conceitual, discutidos na

**Figura 10.1** Um modelo geral do processo de aprendizado.



Seção 10.6, ilustram outra variação do problema de indução: em vez de um conjunto de ocorrências de treinamento de categorização conhecida, esses algoritmos começam com um conjunto de ocorrências não classificadas. Sua tarefa é descobrir categorizações que possam ter alguma utilidade para o aprendiz.

Exemplos não são a única fonte de dados de treinamento. Os seres humanos, por exemplo, aprendem frequentemente por orientação de alto nível. Ao ensinar programação, um professor geralmente diz aos alunos que todos os laços devem ter uma condição de parada. Essa orientação, embora correta, não é diretamente útil: ela precisa ser traduzida em regras específicas para manipular contadores de iterações ou condições lógicas em uma linguagem de programação. As analogias (Seção 10.5.4) são outro tipo de dados de treinamento que devem ser corretamente interpretados antes de poderem ser usados. Se um professor disser a um aluno que a eletricidade é como a água, o aluno deve deduzir a intenção correta da analogia: assim como a água flui através de canos, a eletricidade flui através de fios. Como acontece com a água corrente, podemos medir a quantidade de eletricidade (amperagem) e a pressão (tensão) relacionadas com o fluxo. Entretanto, diferentemente da água, a eletricidade não molha objetos nem nos ajuda a lavar as mãos. A interpretação de analogias envolve encontrar semelhanças que façam sentido e evitar deduções falsas ou sem sentido.

Podemos, também, caracterizar um algoritmo de aprendizado pelo objetivo, ou *alvo*, do aprendizado. O objetivo de muitos algoritmos de aprendizado é um *conceito*, ou uma descrição geral de uma classe de objetos. Algoritmos de aprendizado podem, também, adquirir planos, heurísticas para resolver problemas, ou outras formas de conhecimento procedimental.

As propriedades e a qualidade dos próprios dados de treinamento formam outra dimensão ao longo da qual classificamos tarefas de aprendizado. Os dados podem ser providenciados por um professor, originados do ambiente externo, ou podem ser gerados pelo próprio programa. Os dados podem ser confiáveis ou podem conter ruído. Eles podem ser apresentados de um modo bem estruturado ou podem estar desorganizados. Eles podem incluir exemplos positivos e negativos ou apenas exemplos positivos. Os dados podem estar imediatamente disponíveis, ou o programa deve construir experimentos ou realizar alguma outra forma de aquisição de dados.

2. **Representação do conhecimento aprendido.** Os programas de aprendizado de máquina utilizam todas as linguagens de representação discutidas neste livro. Por exemplo, programas que aprendem a classificar

objetos podem representar esses conceitos como expressões do cálculo de predicados ou podem usar uma representação estruturada como quadros (*frames*) ou objetos. Planos podem ser descritos como uma sequência de operações ou como uma tabela triangular. Heurísticas podem ser representadas como regras para a solução de problemas.

Uma formulação simples do problema de aprendizado de conceitos representa ocorrências de um conceito como sentenças conjuntivas contendo variáveis. Por exemplo, duas ocorrências de “bola” (insuficientes para aprender o conceito) podem ser representadas por:

$$\begin{aligned} \text{tamanho(obj1, pequeno)} \wedge \text{cor(obj1, vermelho)} \wedge \text{forma(obj1, redonda)} \\ \text{tamanho(obj2, grande)} \wedge \text{cor(obj2, vermelho)} \wedge \text{forma(obj2, redonda)} \end{aligned}$$

O conceito geral de “bola” poderia ser definido por:

$$\text{tamanho}(X, Y) \wedge \text{cor}(X, Z) \wedge \text{forma}(X, \text{redonda})$$

onde qualquer sentença que se junte a essa definição geral representa uma bola.

- 3. Um conjunto de operações.** Dado um conjunto de ocorrências de treinamento, a máquina de aprendizado deve construir uma generalização, uma regra heurística ou um plano que satisfaça seus objetivos. Isso requer a habilidade de manipular representações. Operações típicas incluem expressões simbólicas de generalização ou especialização, ajuste de pesos em uma rede neural ou outra forma de modificar as representações do programa.

No exemplo de aprendizado de conceito que acabamos de considerar, uma máquina de aprendizado poderia generalizar uma definição substituindo constantes por variáveis. Se começarmos com o conceito:

$$\text{tamanho(obj1, pequeno)} \wedge \text{cor(obj1, vermelho)} \wedge \text{forma(obj1, redonda)}$$

a substituição de uma única constante por uma variável produz as generalizações:

$$\begin{aligned} \text{tamanho(obj1, } X \text{)} \wedge \text{cor(obj1, vermelho)} \wedge \text{forma(obj1, redonda)} \\ \text{tamanho(obj1, pequeno)} \wedge \text{cor(obj1, } X \text{)} \wedge \text{forma(obj1, redonda)} \\ \text{tamanho(obj1, pequeno)} \wedge \text{cor(obj1, vermelho)} \wedge \text{forma(obj1, } X \text{)} \\ \text{tamanho(X, pequeno)} \wedge \text{cor(X, vermelho)} \wedge \text{forma(X, redonda)} \end{aligned}$$

- 4. Espaço de conceitos.** A linguagem de representação, com as operações que descrevemos, define um espaço de definições de conceitos potenciais. A máquina de aprendizado deve realizar uma busca nesse espaço para encontrar o conceito desejado. A complexidade desse espaço de conceitos é uma medida fundamental da dificuldade de um problema de aprendizado.

- 5. Busca heurística.** Programas de aprendizado devem se comprometer com uma direção e com a ordem de busca, bem como com o uso de dados de treinamento disponíveis e com heurísticas para realizar a busca de modo eficiente. No nosso exemplo de aprendizado do conceito “bola”, um algoritmo plausível pode tomar o primeiro exemplo como um *conceito candidato* e generalizá-lo para incluir exemplos subsequentes. Por exemplo, ao receber o único exemplo de treinamento

$$\text{tamanho(obj1, pequeno)} \wedge \text{cor(obj1, vermelho)} \wedge \text{forma(obj1, redonda)}$$

o algoritmo fará desse exemplo um conceito candidato; esse conceito classifica corretamente a única ocorrência positiva apresentada.

Se o algoritmo receber uma segunda ocorrência positiva

$$\text{tamanho(obj2, grande)} \wedge \text{cor(obj2, vermelho)} \wedge \text{forma(obj2, redonda)}$$

ele poderá generalizar o conceito candidato substituindo constantes por variáveis, caso seja necessário para formar um conceito que case com ambas as ocorrências. O resultado será um conceito candidato mais geral que está mais próximo do nosso conceito-alvo para “bola”.

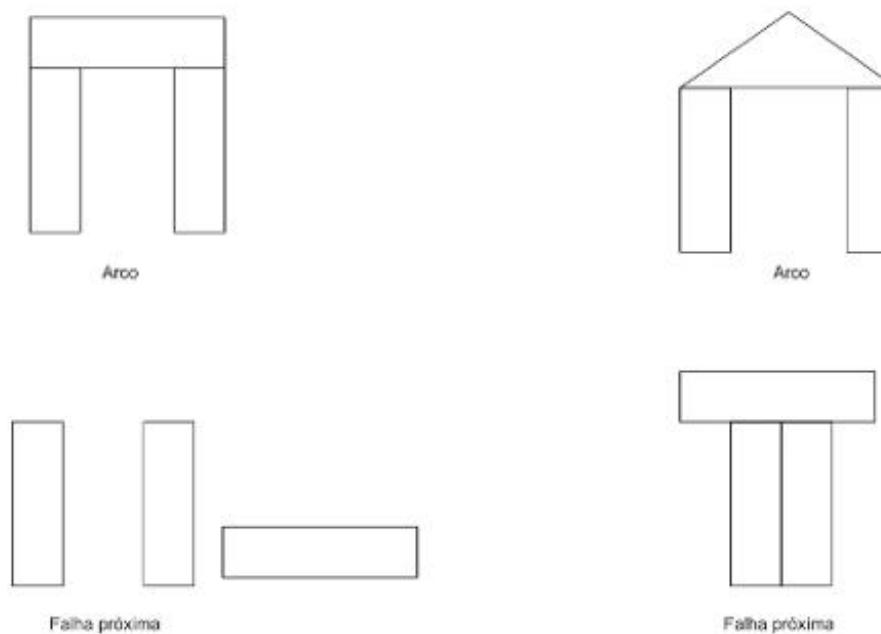
$$\text{tamanho}(X, Y) \wedge \text{cor}(X, \text{vermelho}) \wedge \text{forma}(X, \text{redonda})$$

O trabalho de Patrick Winston (1975a) sobre aprendizado de conceitos, a partir de exemplos positivos e negativos, ilustra esses componentes. O programa aprende definições gerais de conceitos estruturais, como “arco”, em um mundo de blocos. Os dados de treinamento são uma série de exemplos positivos e negativos do conceito: exemplos de estruturas do mundo de blocos que se encaixam na categoria, com *fallhas próximas*. Essas falhas são ocorrências que quase pertencem à categoria, mas que falham em uma propriedade ou relação. As falhas próximas permitem que o programa individualize características que podem ser usadas para excluir ocorrências negativas do conceito-alvo. A Figura 10.2 apresenta exemplos positivos e falhas próximas para o conceito “arco”.

O programa representa conceitos como redes semânticas, como na Figura 10.3. Ele aprende refinando uma descrição candidata do conceito-alvo, conforme ocorrências de treinamento são apresentadas. O programa de Winston refina descrições candidatas por meio de operações de generalização e de especialização. A generalização modifica o grafo de modo que ele acomode novos exemplos do conceito. A Figura 10.3(a) mostra um arco construído com três retângulos e um grafo que o descreve. O próximo exemplo de treinamento, mostrado na Figura 10.3(b), é um arco com uma pirâmide, em vez de um retângulo no seu topo. Esse exemplo não casa com a descrição candidata. O programa compara esses grafos tentando encontrar um isomorfismo parcial entre eles. O comparador de grafos usa os nomes dos nós para guiar o processo de casamento. Uma vez que o programa tenha comparado os grafos, ele pode detectar diferenças entre eles. Na Figura 10.3, os grafos se casam em todos elementos, exceto pelo fato de que o elemento do topo no primeiro grafo é retângulo, e o nó correspondente do segundo exemplo é pirâmide. Parte do conhecimento de base do programa é uma hierarquia de generalização desses conceitos [Figura 10.3(c)]. O programa generaliza o grafo substituindo esse nó pelo último supertipo comum de retângulo e pirâmide; nesse exemplo, ele é polígono. O resultado é o conceito da Figura 10.3(d).

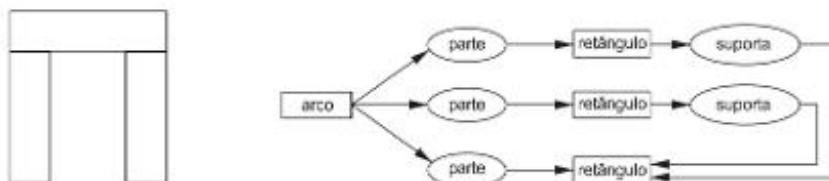
Ao receber uma falha próxima, um exemplo que difere do conceito-alvo em uma única propriedade, o programa especializa a descrição candidata de modo a excluir o exemplo. A Figura 10.4(a) é uma descrição candidata. Ela difere da falha próxima da Figura 10.4(b) pelas relações de tocar do exemplo de falha próxima. O programa especializa o grafo acrescentando elos `não_deve_tocar` para excluir a falha próxima [Figura 10.4(c)]. Note que o algoritmo depende muito da proximidade dos exemplos negativos em relação ao conceito-alvo. Diferindo do alvo em apenas uma única propriedade, uma falha próxima ajuda o algoritmo a determinar exatamente como especializar o conceito candidato.

**Figura 10.2** Exemplos e falhas próximas para o conceito “arco”.

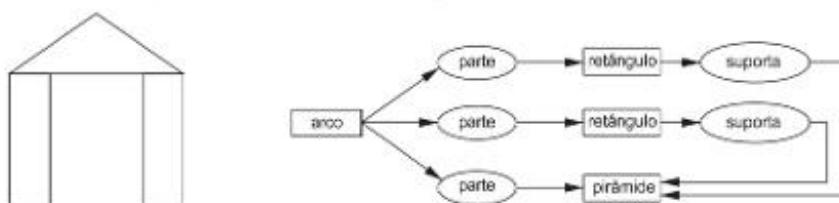


**Figura 10.3** Generalização de descrições para incluir múltiplos exemplos.

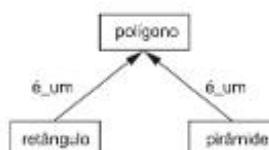
a. Exemplo de um arco e sua descrição de rede



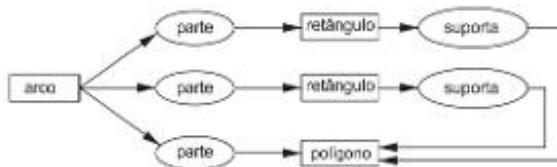
b. Exemplo de outro arco e sua descrição de rede



c. Conhecimento de base prévio que declara que retângulos e pirâmides são tipos de polígonos



d. Generalização que inclui ambos os exemplos

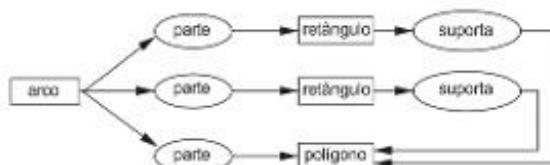


Essas operações — especializar uma rede adicionando elos e generalizá-la por substituição de nomes de nós, ou elos, por um conceito mais genérico — definem um espaço de possíveis definições de conceitos. O programa de Winston realiza uma busca por subida de encosta sobre o espaço de conceitos guiada pelos dados de treinamento. Como o programa não realiza retrocesso, o seu desempenho é muito sensível à ordem dos exemplos de treinamento; uma ordenação ruim pode fazer com que o programa fique preso em um beco sem saída do espaço de busca. Os exemplos de treinamento devem ser apresentados ao programa em uma ordem que auxilie o aprendizado do conceito desejado, da mesma forma que um professor organiza as lições de modo a ajudar seus alunos a aprenderem. A qualidade e a ordem dos exemplos de treinamento são importantes também para o algoritmo de casamento de grafos do programa; um casamento eficiente requer que os grafos não sejam muito diferentes.

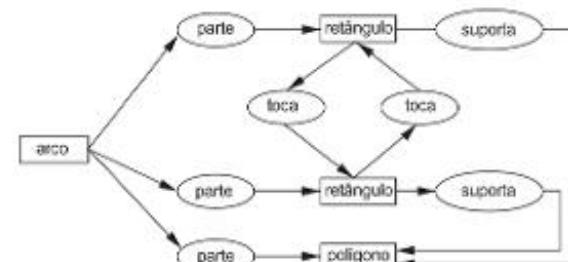
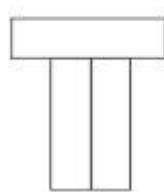
Embora seja um exemplo antigo de aprendizado indutivo, o programa de Winston ilustra as características e os problemas compartilhados pela maioria das técnicas de aprendizado de máquina deste capítulo: o uso de operações de generalização e especialização para definir um espaço de conceitos, o uso de dados para guiar a busca através desse espaço e a sensibilidade do algoritmo de aprendizado à qualidade dos dados de treinamento. As próximas seções examinam esses problemas e as técnicas que o aprendizado de máquina desenvolveu para a sua solução.

**Figura 10.4** Especialização de uma descrição para excluir uma falha próxima. Em 10.4(c), acrescentamos restrições a 10.4(a), de modo que ela não possa casar com 10.4(b).

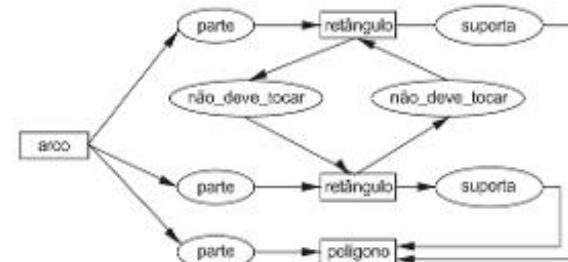
a. Descrição candidata de um arco



b. Falha próxima e sua descrição



c. Descrição de um arco especializada para excluir a falha próxima



## 10.2 Busca em espaço de versões

A *busca em espaço de versões* (Mitchell, 1978, 1979, 1982) ilustra a implementação do aprendizado indutivo como busca através de um espaço de conceitos. A busca em espaço de versões aproveita o fato de que as operações de generalização impõem uma ordenação ao espaço de conceitos, usando, então, essa ordenação para guiar a busca.

### 10.2.1 Operadores de generalização e espaço de conceitos

Generalização e especialização são os tipos mais comuns de operações para definir um espaço de conceitos. As operações de generalização fundamentais usadas no aprendizado de máquina são:

1. Substituir constantes por variáveis. Por exemplo,

`cor(bola, vermelho)`

é generalizada como

`cor(X, vermelho)`

**2.** Retirar condições de uma expressão conjuntiva.

forma(X, redondo)  $\wedge$  tamanho(X, pequeno)  $\wedge$  cor(X, vermelho)

é generalizada como

forma(X, redondo)  $\wedge$  cor(X, vermelho)

**3.** Acrescentar um termo disjuntivo a uma expressão.

forma(X, redondo)  $\wedge$  tamanho(X, pequeno)  $\wedge$  cor(X, vermelho)

é generalizada como

forma(X, redondo)  $\wedge$  tamanho(X, pequeno)  $\wedge$  (cor(X, vermelho)  $\vee$  cor(X, azul))

**4.** Substituir uma propriedade por seu pai em uma hierarquia de classes. Se soubermos que cor\_primária é uma superclasse de vermelho, então

cor(X, vermelho)

é generalizada como

cor(X, cor\_primária)

Podemos considerar a generalização em termos da teoria de conjuntos: sejam P e Q os conjuntos de sentenças que casam com as expressões do cálculo de predicados p e q, respectivamente. A expressão p é mais geral que q se, e somente se,  $P \supseteq Q$ . Nos exemplos anteriores, o conjunto de sentenças que casam com cor(X, vermelho) contém o conjunto de elementos que casam com cor(bola, vermelho). Da mesma forma, no exemplo 2, podemos considerar o conjunto de objetos redondos e vermelhos como um superconjunto do conjunto de objetos pequenos, vermelhos e redondos. Note que a relação “mais geral que” define uma ordenação parcial no espaço de sentenças lógicas. Expressamos isso usando o símbolo “ $\geq$ ”, onde  $p \geq q$  significa que p é mais geral que q. Essa ordenação é uma poderosa fonte de restrições sobre a busca realizada por um algoritmo de aprendizado.

Formalizamos essa relação por meio da noção de *cobertura*. Se o conceito p for mais geral que o conceito q, dizemos que p *cobre* q. Definimos a relação de cobertura como: sejam  $p(x)$  e  $q(x)$  descrições que classificam objetos como sendo exemplos positivos de um conceito. Em outras palavras, para um objeto x,  $p(x) \rightarrow$  positivo(x) e  $q(x) \rightarrow$  positivo(x). p cobre q se, e somente se,  $q(x) \rightarrow$  positivo(x) for uma consequência lógica de  $p(x) \rightarrow$  positivo(x).

Por exemplo, cor(X, Y) cobre cor(bola, Z), que, por sua vez, cobre cor(bola, vermelho). Como um exemplo simples, considere um domínio de objetos que têm as seguintes propriedades e valores:

Tamanhos = {grande, pequeno}

Cores = {vermelho, branco, azul}

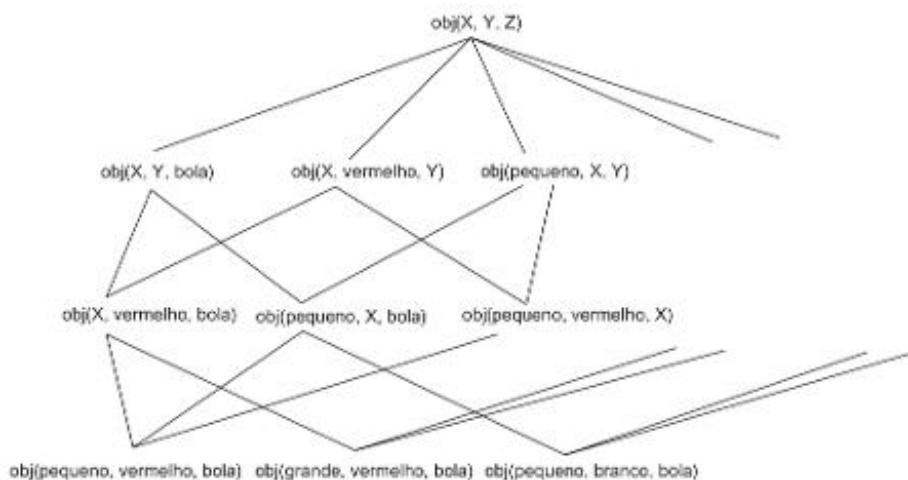
Formas = {bola, retângulo, cubo}

Esses objetos podem ser representados usando o predicado obj(Tamanhos, Cores, Formas). A operação de generalização de substituir constantes por variáveis define o espaço da Figura 10.5. Podemos ver o aprendizado indutivo como busca nesse espaço por um conceito que seja consistente com todos os exemplos de treinamento.

### 10.2.2 Algoritmo de eliminação de candidatos

Esta seção apresenta três algoritmos (Mitchell, 1982) para realizar a busca no espaço de conceitos. Esses algoritmos se baseiam na noção de um *espaço de versões*, que é o conjunto de todas as descrições de conceitos consistentes com os exemplos de treinamento. Esses algoritmos atuam reduzindo o tamanho do espaço de versões, conforme mais exemplos se tornam disponíveis. Os dois primeiros algoritmos reduzem o espaço de versões agindo na direção do específico para o genérico ou, então, do genérico para o específico, respectivamente. O terceiro algoritmo, chamado de *eliminação de candidatos*, combina essas abordagens em uma busca bidirecional. A seguir, descrevemos e avaliamos esses algoritmos.

**Figura 10.5** Um espaço de conceitos.



Esses algoritmos são guiados por dados; eles generalizam com base em regularidades encontradas nos dados de treinamento. Ao usar dados de treinamento com classificação conhecida, esses algoritmos realizam um tipo de *aprendizado supervisionado*.

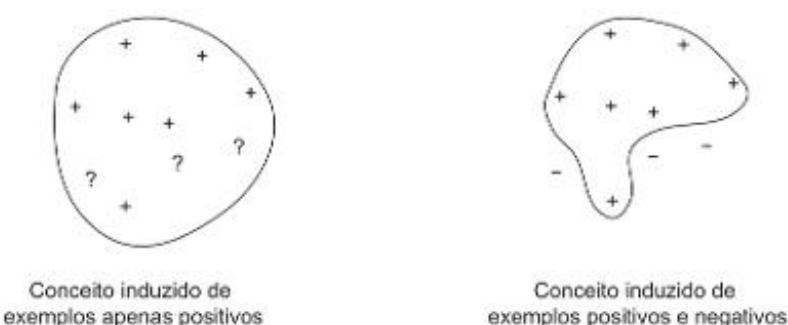
Como no programa de Winston para o aprendizado de descrições estruturais, a busca no espaço de versões usa exemplos do conceito-alvo, tanto positivos como negativos. Embora seja possível generalizar a partir apenas de exemplos positivos, os exemplos negativos são importantes para evitar que o algoritmo generalize excessivamente. O conceito aprendido deve ser não apenas suficientemente geral para cobrir todos os exemplos positivos; ele também precisa ser suficientemente específico para excluir todos os exemplos negativos. No espaço da Figura 10.5, um conceito que cobriria todos os conjuntos de exemplos exclusivamente positivos seria simplesmente  $\text{obj}(X, Y, Z)$ . Entretanto, esse conceito provavelmente é muito genérico, pois implica que todos os exemplos pertencem ao conceito-alvo. Uma forma de evitar a supergeneralização é generalizar tão pouco quanto possível para cobrir os exemplos positivos; outra forma é usar exemplos negativos para eliminar conceitos muito gerais. Como ilustra a Figura 10.6, exemplos negativos evitam a supergeneralização, forçando a máquina de aprendizado a especializar conceitos, de modo a excluir ocorrências negativas. Os algoritmos desta seção usam essas duas técnicas. Para o conjunto de hipóteses  $S$ , definimos a busca *do específico ao geral* como:

```

Início
  Inicialize S com o primeiro exemplo de treinamento positivo;
  N é o conjunto de todos os exemplos negativos vistos até o momento;

  Para cada exemplo positivo p
    Início
      Para todo s ∈ S, se s não casar com p, substitua s por sua generalização
        mais específica que case com p;
      Retire de S todas as hipóteses que são mais gerais que uma outra hipótese em S;
      Retire de S todas as hipóteses que casem com um exemplo negativo em N,
        previamente observado;
    Fim;

  Para todo exemplo negativo n
    Início
      Retire todos os membros de S que casem com n;
      Acrescente n a N para verificar a supergeneralização em futuras hipóteses;
    Fim;
Fim
  
```

**Figura 10.6** Papel dos exemplos negativos na prevenção da supergeneralização.

A busca do específico ao geral mantém um conjunto  $S$  de *hipóteses*, ou definições de conceito candidatas. Para evitar a supergeneralização, essas definições candidatas são as *generalizações maximamente específicas* dos dados de treinamento. Um conceito  $c$  é maximamente específico se ele cobrir todos os exemplos positivos, nenhum exemplo negativo e, ainda, para qualquer outro conceito  $c'$  que cubra os exemplos positivos,  $c \leq c'$ . A Figura 10.7 mostra um exemplo da aplicação desse algoritmo ao espaço de versões da Figura 10.5. O algoritmo de busca em espaço de versões do específico para o geral é construído em Prolog na Sala Virtual deste livro.

Podemos, também, realizar a busca na direção do geral para o específico. Esse algoritmo mantém um conjunto  $G$  de *conceitos maximamente gerais* que cobrem todos os exemplos positivos e nenhum exemplo negativo. Um conceito  $c$  é maximamente geral se ele não cobrir nenhum dos exemplos negativos de treinamento e, para qualquer outro conceito  $c'$  que não cubra nenhum dos exemplos negativos,  $c \geq c'$ . Nesse algoritmo, exemplos negativos causam a especialização de conceitos candidatos; o algoritmo usa exemplos positivos para eliminar conceitos superespecializados.

```

Início
  Inicialize G de modo a conter o conceito mais geral do espaço;
  P contém todos os exemplos positivos vistos até o momento;

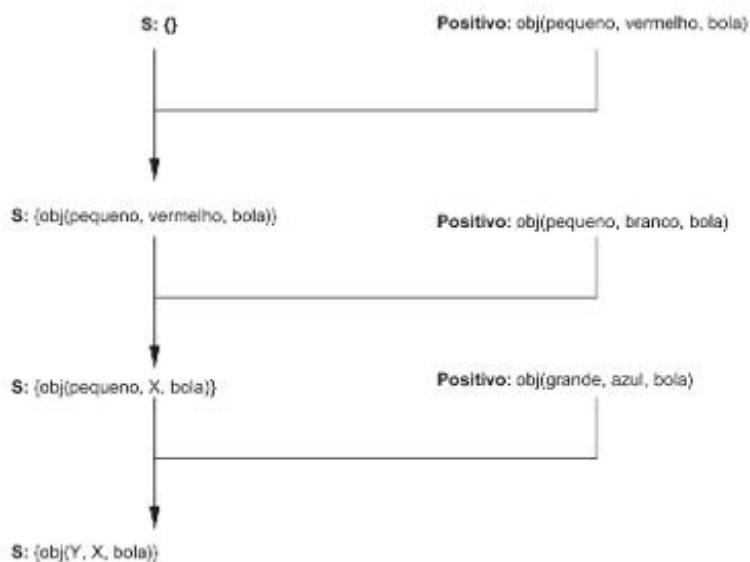
  Para cada exemplo negativo n
    Início
      Para cada  $g \in G$  que case com n, substitua g por suas especializações
        mais gerais que não casem com n;
      Retire de G todas as hipóteses mais específicas que outra hipótese em G;
      Retire de G todas as hipóteses que não casem com um exemplo positivo em P;
    Fim;

  Para cada exemplo positivo p
    Início
      Retire de G todas as hipóteses que não casem com p;
      Acrescente p a P;
    Fim;
  Fim

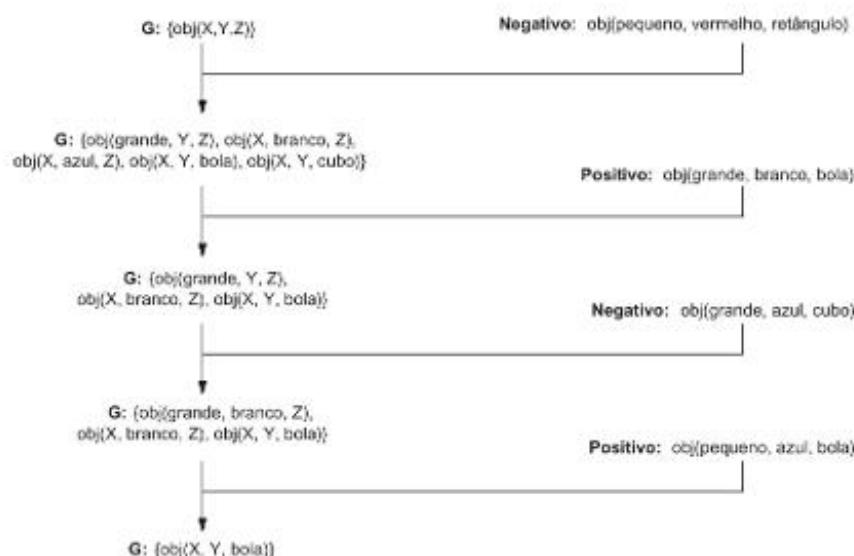
```

A Figura 10.8 mostra um exemplo de aplicação desse algoritmo ao espaço de versões da Figura 10.5. Nesse exemplo, o algoritmo usa o conhecimento prévio de que tamanho pode ter os valores {grande, pequeno}, que cor pode ter valores {vermelho, branco, azul} e que forma pode ter valores {bola, retângulo, cubo}. Esse conhecimento é essencial se quisermos que o algoritmo especialize conceitos pela substituição de constantes por variáveis.

**Figura 10.7** Busca do específico ao geral no aprendizado por espaço de versões do conceito “bola”.



**Figura 10.8** Busca do geral para o específico no aprendizado por espaço de versões do conceito bola.



O algoritmo de eliminação de candidatos combina essas abordagens em uma busca bidirecional. Essa abordagem bidirecional tem uma série de benefícios para o aprendizado. O algoritmo mantém dois conjuntos de conceitos candidatos: G é o conjunto de conceitos candidatos maximamente gerais, e S é o conjunto de candidatos maximamente específicos. O algoritmo especializa G e generaliza S até que eles converjam para o conceito-alvo. O algoritmo é definido como:

Início

  Initialize G de modo que ele seja o conceito mais geral do espaço;

  Initialize S com o primeiro exemplo de treinamento positivo;

Para cada novo exemplo positivo p

Início

Retire todos os membros de G que não casem com p;

Para todo s ∈ S, se s não casar com p, substitua s por suas generalizações mais específicas que casem com p;

Retire de S toda hipótese que seja mais geral que uma outra hipótese em S;

Retire de S toda hipótese que seja mais geral que uma hipótese em G;

Fim;

Para cada novo exemplo negativo n

Início

Retire todos os membros de S que casem com n;

Para cada g ∈ G que case com n, substitua g por suas especializações mais gerais que não casem com n;

Retire de G toda hipótese que seja mais específica que uma outra hipótese em G;

Retire de G toda hipótese que seja mais específica que uma hipótese em S;

Fim;

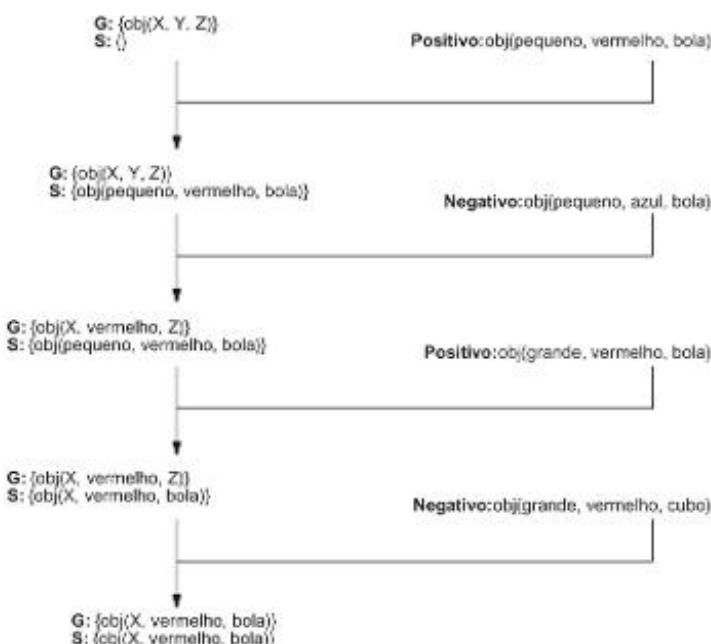
Se G = S e ambos forem conjuntos unitários, então o algoritmo encontrou um conceito único que é consistente com todos os dados e o algoritmo para;

Se G e S se tornam vazios, então não existe um conceito que cubra todos os exemplos positivos e nenhum dos exemplos negativos;

Fim

A Figura 10.9 ilustra o comportamento do algoritmo de eliminação de candidatos ao buscar o espaço de versões da Figura 10.5. Note que a figura não mostra os conceitos que foram produzidos pela generalização ou pela eliminação, mas que foram eliminados por serem supergeneralizados ou superespecíficos. Deixamos a elaboração dessa parte do algoritmo como exercício e mostramos uma implementação parcial em Prolog na Sala Virtual deste livro.

**Figura 10.9** Algoritmo de eliminação de candidatos aprendendo o conceito bola vermelha.

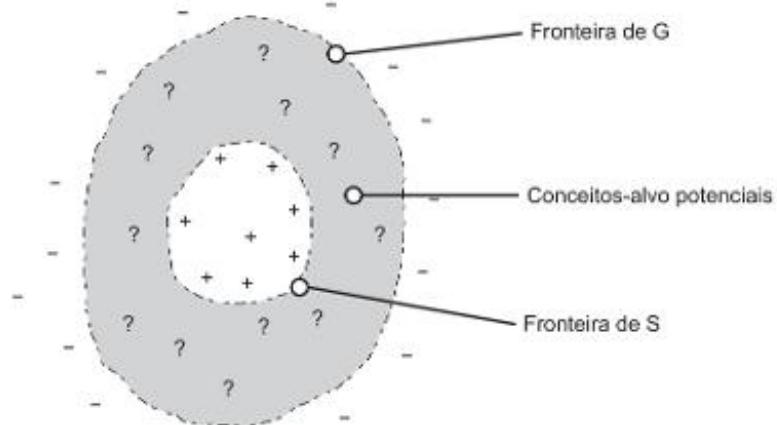


A combinação das duas direções de busca em um único algoritmo traz uma série de benefícios. Os conjuntos  $G$  e  $S$  resumem a informação contida nos exemplos negativos e positivos, respectivamente, eliminando a necessidade de salvar esses exemplos. Por exemplo, após generalizar  $S$  para cobrir um exemplo positivo, o algoritmo usa  $G$  para eliminar conceitos em  $S$  que não cobrem qualquer exemplo negativo. Como  $G$  é o conjunto de conceitos *maximamente gerais* que não casam com nenhum exemplo de treinamento negativo, qualquer membro de  $S$  que seja mais geral que qualquer membro de  $G$  deve casar com um exemplo negativo. Da mesma forma, como  $S$  é o conjunto de generalizações *maximamente específicas* que cobrem todos os exemplos positivos, qualquer novo membro de  $G$  que for mais específico que um membro de  $S$  não deve cobrir algum exemplo positivo e pode, também, ser eliminado.

A Figura 10.10 mostra uma descrição abstrata do algoritmo de eliminação de candidatos. Os sinais de “+” representam exemplos de treinamento positivos; sinais de “-” indicam exemplos negativos. O círculo mais interno contém o conjunto de exemplos conhecidamente positivos cobertos pelos conceitos em  $S$ . O círculo mais externo contém os exemplos cobertos por  $G$ ; qualquer exemplo fora desse círculo é negativo. A parte sombreada do gráfico contém o conceito-alvo com conceitos que podem ser supergenéricos ou superspecíficos (os “?”). A busca “encolhe” o conceito mais externo para que exemplos negativos possam ser excluídos; ela também “expande” o conceito interno para incluir novos exemplos positivos. Em algum momento, os dois conjuntos convergem para o conceito-alvo. Dessa forma, a eliminação de candidatos pode detectar quando ela encontrou um conceito-alvo único e consistente. Quando  $G$  e  $S$  convergem para o mesmo conceito, o algoritmo pode parar. Se  $G$  e  $S$  se tornam vazios, então não existe um conceito que cubra todos os exemplos positivos e nenhum exemplo negativo. Isso pode ocorrer se os dados de treinamento forem inconsistentes ou se o conceito-alvo não puder ser expresso na linguagem de representação (Seção 10.2.4).

Um aspecto interessante da eliminação de candidatos é a sua natureza incremental. Um algoritmo de aprendizado incremental aceita um exemplo de treinamento por vez, formando uma generalização usável, embora possivelmente incompleta, após cada exemplo. Isso contrasta com algoritmos por lote (por exemplo, o ID3, na Seção 10.3), que requerem que todos os exemplos de treinamento sejam apresentados antes do início do aprendizado. Mesmo antes de o algoritmo de eliminação de candidatos convergir para um conceito único, os conjuntos  $G$  e  $S$  fornecem restrições usáveis do conceito: se  $c$  é o conceito-alvo, então para todo  $g \in G$  e  $s \in S$ ,  $s \leq c \leq g$ . Qualquer conceito que seja mais geral que um conceito em  $G$  cobrirá exemplos negativos; qualquer conceito que seja mais específico que um conceito em  $S$  não cobrirá alguns exemplos positivos. Isso sugere que exemplos que tenham um “bom ajuste” em relação aos conceitos limitados por  $G$  e  $S$  são, ao menos, exemplos plausíveis do conceito.

**Figura 10.10** Convergência das fronteiras dos conjuntos  $G$  e  $S$  no algoritmo de eliminação de candidatos.



Na próxima seção, esclarecemos essa intuição com um exemplo de um programa que usa eliminação de candidatos para aprender a buscar por heurísticas. O sistema LEX (Mitchell et al., 1983) aprende heurísticas para resolver problemas de integração simbólica. Esse trabalho não apenas demonstra o uso de G e S para definir conceitos parciais, mas também ilustra questões adicionais, como a complexidade do aprendizado de tarefas com múltiplos passos, atribuição de crédito/culpa e a relação entre os componentes de aprendizado e de solução de problemas de um sistema complexo.

### 10.2.3 LEX: induzindo heurísticas de busca

O LEX aprende heurísticas para resolver problemas de integração simbólica. Ele integra expressões algébricas por meio de uma busca heurística, começando com a expressão a ser integrada e buscando o seu objetivo: uma expressão que não contenha sinais de integral. O componente de aprendizado do sistema usa dados do resolvedor de problema para induzir heurísticas que melhorem o desempenho desse resolvedor.

LEX busca um espaço definido por operações sobre expressões algébricas. Os seus operadores são as transformações típicas usadas na integração. Por exemplo:

- OP1:  $\int r f(x) dx \rightarrow r \int f(x) dx$
- OP2:  $\int u dv \rightarrow uv - \int v du$
- OP3:  $1^*f(x) \rightarrow f(x)$
- OP4:  $\int (f_1(x) + f_2(x)) dx \rightarrow \int f_1(x) dx + \int f_2(x) dx$

Operadores são regras cujo lado esquerdo define quando eles podem ser aplicados. Embora o lado esquerdo defina as circunstâncias sob as quais o operador pode ser usado, ele não inclui heurísticas para quando o operador *deveria* ser usado. O LEX deve aprender heurísticas usáveis por meio da sua própria experiência. Heurísticas são expressões da forma:

Se o estado atual do problema casar com P, então aplicar o operador O com ligações B.

Por exemplo, uma heurística que LEX poderia aprender é:

Se um estado do problema casar com  $\int x \text{transcendental}(x) dx$ ,

então aplicar OP2 com ligações

$$u = x$$

$$dv = \text{transcendental}(x) dx$$

Aqui, a heurística sugere aplicar integração por partes para resolver a integral de x vezes uma função transcendental, por exemplo, uma função trigonométrica em x.

A linguagem do LEX para representar conceitos consiste nos símbolos descritos na Figura 10.11. Note que os símbolos existem em uma hierarquia de generalização, com qualquer símbolo casando com qualquer um de seus descendentes na hierarquia. LEX generaliza expressões substituindo um símbolo por seu ancestral nessa hierarquia.

Por exemplo, dada a expressão:

$$\int 3x \cos(x) dx$$

LEX pode substituir cos por trig. Isso produz a expressão:

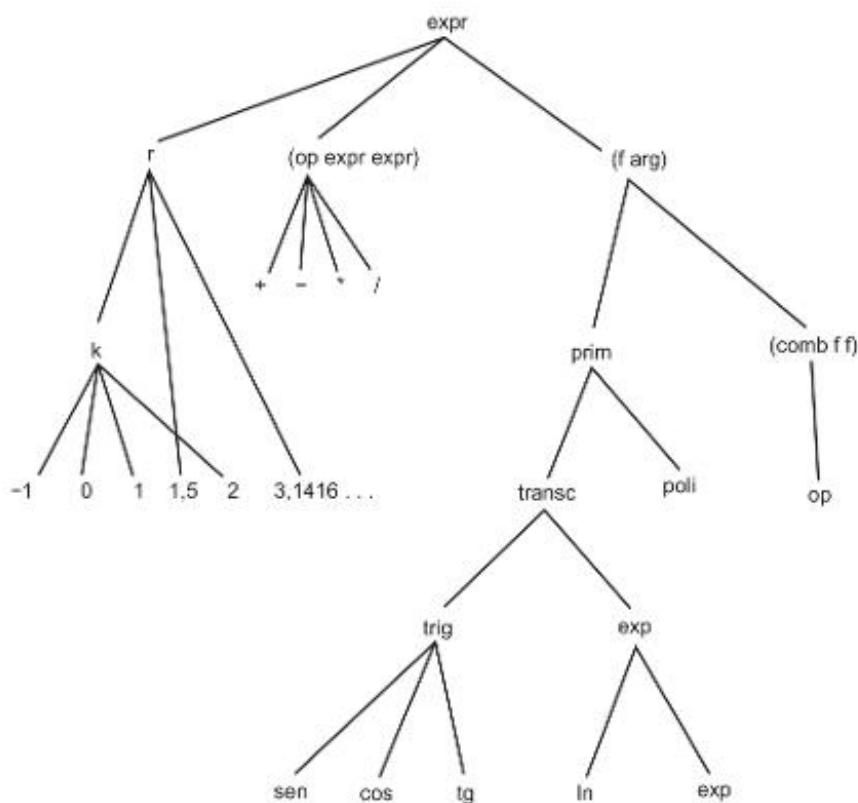
$$\int 3x \text{trig}(x) dx$$

Como alternativa, ele pode substituir 3 pelo símbolo k, que representa qualquer inteiro:

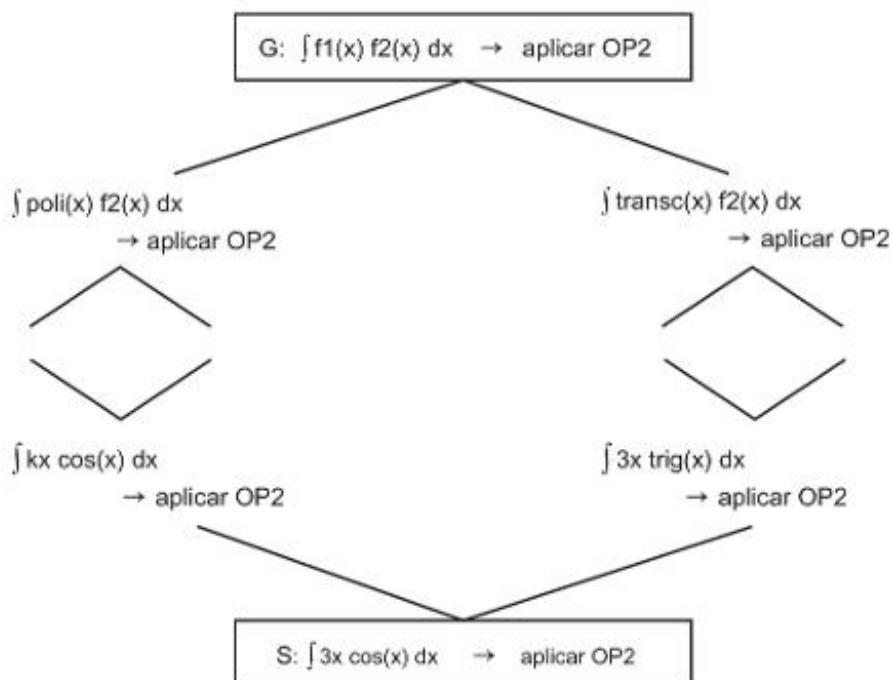
$$\int kx \cos(x) dx$$

A Figura 10.12 mostra um espaço de versões para OP2 como definido por essas generalizações.

**Figura 10.11** Uma parte da hierarquia de símbolos do LEX.



**Figura 10.12** Um espaço de versões para OP2, adaptado de Mitchell et al. (1983).



A arquitetura global do LEX consiste em quatro componentes:

1. um *generalizador*, que usa eliminação de candidatos para encontrar heurísticas;
2. um *resolvedor de problemas*, que produz rastros de soluções de problemas;
3. um *crítico*, que produz exemplos positivos e negativos a partir de um rastro de problema;
4. um *gerador de problemas*, que produz novos problemas candidatos.

O LEX mantém um conjunto de espaços de versões. Cada espaço de versões está associado a um operador e representa uma heurística parcialmente aprendida para aquele operador. O generalizador atualiza esses espaços de versões usando exemplos positivos e negativos da aplicação do operador, conforme gerado pelo crítico. Ao receber um exemplo positivo, LEX determina se um espaço de versões associado àquele operador inclui o exemplo. Um espaço de versões inclui um exemplo positivo se o exemplo for coberto por algum conceito contido em G. LEX usa, então, o exemplo positivo para atualizar aquela heurística. Se nenhuma heurística casar com o exemplo, LEX cria um novo espaço de versões usando aquele exemplo como o primeiro exemplo positivo. Isso pode levar à criação de múltiplos espaços de versões, para diferentes heurísticas, e um operador.

O resovedor de problemas do LEX constrói uma árvore do espaço buscado ao resolver um problema de integração. Isso limita o tempo de processamento que o resovedor de problemas pode usar para solucionar um problema. LEX realiza uma busca pela melhor escolha, usando as heurísticas que ele mesmo construiu. Um aspecto interessante do desempenho de LEX é o uso de G e S como definições parciais de uma heurística. Se mais de um operador puder ser aplicado em um dado estado, LEX escolhe aquele que exibe o mais alto grau de casamento parcial em relação ao estado do problema. O grau parcial de casamento é definido como a porcentagem de todos os conceitos incluídos entre G e S que casam com o estado atual. Como o custo computacional de se testar o estado em relação a todos os conceitos candidatos é proibitivo, LEX estima o grau de casamento como a porcentagem de conceitos em G e em S que casam com o estado. Note que o desempenho melhora continuamente, conforme ele melhora as suas heurísticas. Resultados empíricos confirmam essa conjectura.

LEX obtém exemplos positivos e negativos de aplicações do operador a partir de um rastro de soluções gerado pelo resovedor de problemas. Na ausência de um professor, ele deve classificar aplicações do operador como positivas ou negativas; esse é um exemplo do problema de *atribuição de crédito*. Quando o aprendizado acontece no contexto de solução de problemas de múltiplos passos, frequentemente não fica claro qual ação, em uma sequência, deveria ser responsabilizada pelo resultado. Se um resovedor de problemas chegar a uma resposta errada, como podemos saber qual dos diferentes passos realmente causou o erro? O crítico de LEX aborda esse problema supondo que o rastro da solução produzida pelo resovedor de problemas represente um caminho mais curto até um objetivo. LEX classifica as aplicações de operadores ao longo desse caminho (supostamente) mais curto como exemplos positivos; as aplicações de operadores que divergirem desse caminho são tratadas como exemplos negativos.

Entretanto, ao tratar o rastro do resovedor de problemas como uma solução de caminho mais curto, o crítico deve lidar com o fato de que não se pode garantir que as heurísticas em evolução de LEX sejam admissíveis (Seção 4.3). O caminho de solução encontrado pelo resovedor de problemas pode realmente não ser uma solução de caminho mais curto. Para garantir que ele não tenha classificado erroneamente uma aplicação de operadores como negativa, LEX primeiramente estende os caminhos iniciados por tais operadores para ter certeza de que eles não levam a uma solução melhor. Normalmente, uma solução de problema produz de 2 a 20 exemplos de treinamento. O LEX passa esses exemplos positivos e negativos para o generalizador, que os usa para atualizar os espaços de versões para os operadores associados.

O gerador de problemas é a parte menos desenvolvida do programa. Embora tenham sido usadas várias estratégias para automatizar a seleção de problema, a maioria dos exemplos é escolhida manualmente. Contudo, foi construído um gerador de problemas que explora uma série de estratégias. Uma dessas abordagens gera exemplos que são cobertos pelas heurísticas parciais para dois operadores diferentes para fazer LEX aprender a discriminá-los.

Testes empíricos mostram que LEX é eficaz em aprender heurísticas úteis. Em um teste, LEX recebeu 5 problemas de teste e 12 problemas de treinamento. Antes do treinamento, ele resolveu os 5 problemas de teste em uma média de 200 passos; essas soluções não usaram heurísticas para guiar a busca. Após ter desenvolvido heurísticas, a partir dos 12 problemas de treinamento, ele resolveu esses mesmos problemas de teste em uma média de 20 passos.

O LEX aborda diversas questões relativas ao aprendizado, incluindo problemas como a atribuição de crédito, a seleção de exemplos de treinamento e a relação entre os componentes de solução do problema e de generalização de um algoritmo de aprendizado. Ele também reforça a importância de uma representação apropriada para os conceitos. Grande parte da eficácia do LEX vem da organização hierárquica dos conceitos. Essa hierarquia é suficientemente pequena para restringir o espaço de heurísticas em potencial, permitindo uma busca eficiente, ao mesmo tempo em que é suficientemente rica para representar heurísticas efetivas.

#### 10.2.4 Avaliando a eliminação de candidatos

O algoritmo de eliminação de candidatos demonstra o modo como a representação de conhecimento e a busca no espaço de estados podem ser aplicadas ao problema de aprendizado de máquina. Entretanto, assim como nas pesquisas mais importantes, o algoritmo não deve ser avaliado apenas em termos de seus sucessos. Ele levanta problemas que continuam a formar uma porção considerável da agenda das pesquisas em aprendizado de máquina.

O aprendizado baseado em busca, assim como todos os problemas de busca, precisam lidar com a explosão combinatória dos espaços de problema. Como o algoritmo de eliminação de candidatos realiza uma busca em amplitude, ele pode ser ineficiente. Se uma aplicação for tal que  $G$  e  $S$  cresçam excessivamente, pode ser útil desenvolver heurísticas para podar estados de  $G$  e  $S$ , implementando uma busca pela melhor escolha ou, melhor ainda, uma busca em feixe (ver Seção 4.5) do espaço.

Outra abordagem para esse problema, discutida na Seção 10.4, envolve o uso de *viés indutivo* para reduzir mais o tamanho do espaço de conceitos. Esse viés restringe a linguagem usada para representar conceitos. O LEX impõe um viés por meio da escolha de conceitos na sua hierarquia de generalização. Embora não seja completa, a linguagem de conceitos do LEX é suficientemente poderosa para capturar muitas heurísticas eficientes; igualmente importante é o fato de ele reduzir o tamanho do espaço de conceitos para proporções manejáveis. Linguagens com viés são essenciais para reduzir a complexidade do espaço de conceitos, mas elas podem tornar a máquina de aprendizado incapaz de representar o conceito que ela está tentando aprender. Nesse caso, a eliminação de candidatos falharia em convergir para um conceito-alvo, deixando  $G$  e  $S$  vazios. Esse compromisso entre expressividade e eficiência é uma questão essencial no aprendizado.

A falta de convergência do algoritmo pode se dever, também, a ruído ou inconsistência nos dados de treinamento. O problema de aprender, a partir de dados ruidosos, é particularmente importante em aplicações reais, em que não se pode garantir que os dados sejam completos ou consistentes. A eliminação de candidatos não é, de forma alguma, resistente a ruído. Mesmo um único exemplo de treinamento com classificação incorreta pode fazer com que o algoritmo não convirja para um conceito consistente. Uma solução para esse problema é manter múltiplos conjuntos  $G$  e  $S$ . Além do espaço de versões derivado de todos os exemplos de treinamento, isso mantém espaços adicionais baseados em todos os exemplos de treinamento com exceção de um, com exceção de dois, e assim por diante. Se  $G$  e  $S$  não convergirem, o algoritmo poderá examinar essas alternativas para encontrar aqueles que permanecem consistentes. Infelizmente, essa abordagem causa a proliferação de conjuntos candidatos e, na prática, é ineficiente demais na maioria dos casos.

Outra questão levantada por esse estudo é o papel do conhecimento prévio no aprendizado. A hierarquia de conceitos do LEX resume uma grande parte do conhecimento sobre álgebra; esse conhecimento é essencial para o desempenho do algoritmo. Será que quantidades maiores de conhecimento do domínio podem tornar o aprendizado ainda mais efetivo? A Seção 10.5 examina esse problema usando o conhecimento prévio do domínio para orientar as generalizações.

Uma contribuição importante deste trabalho é a sua explicação da relação entre representação de conhecimento, generalização e busca no aprendizado indutivo. Embora a eliminação de candidatos seja apenas um dos muitos algoritmos de aprendizado, ela levanta questões gerais a respeito de complexidade, expressividade e uso de conhecimento e dados para guiar a generalização. Esses problemas são centrais a todos os algoritmos de aprendizado de máquina, e continuamos a considerá-los no restante do capítulo.

### 10.3 Algoritmo ID3 para indução de árvores de decisão

O algoritmo ID3 (Quinlan, 1986a), assim como a eliminação de candidatos, induz conceitos a partir de exemplos. Ele é particularmente interessante por sua representação do conhecimento aprendido, sua abordagem para o gerenciamento da complexidade, sua heurística para selecionar conceitos candidatos e seu potencial para lidar com dados ruidosos. O ID3 representa conceitos como *árvores de decisão*, uma representação que nos permite determinar a classificação de um objeto testando seus valores para certas propriedades.

Por exemplo, considere o problema de estimar o risco de crédito de um indivíduo com base em propriedades como histórico de créditos, dívida atual, garantia e renda. A Tabela 10.1 lista uma amostra de indivíduos com riscos de crédito conhecidos. A árvore de decisão da Figura 10.13 representa as classificações da Tabela 10.1, à medida que essa árvore pode classificar corretamente todos os objetos da tabela. Em uma árvore de decisão, cada nó interno representa um teste sobre uma propriedade, como histórico de crédito ou dívida; cada valor possível dessa propriedade corresponde a um ramo da árvore. Nós folhas representam classificações, tais como risco baixo ou moderado. Um indivíduo de tipo desconhecido pode ser classificado percorrendo-se essa árvore: em cada nó interno, teste o valor da propriedade correspondente do indivíduo e siga pelo ramo correspondente. Continue até alcançar um nó folha correspondente à classificação do objeto.

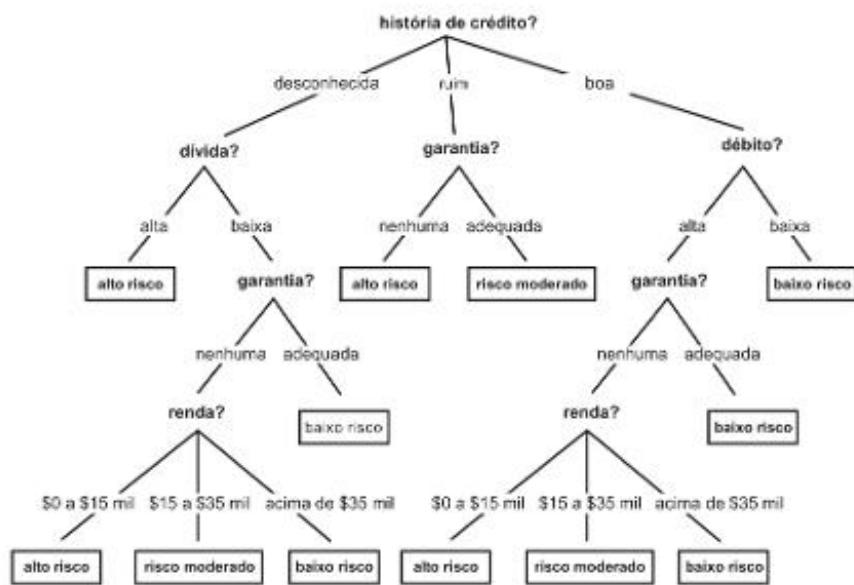
Note que, ao classificar qualquer exemplo dado, essa árvore não usa todas as propriedades presentes na Tabela 10.1. Por exemplo, se uma pessoa tem uma boa história de crédito e uma dívida baixa, podemos, de acordo com a árvore, ignorar sua garantia e sua renda e classificá-la como sendo de baixo risco. Apesar de omitir certos testes, essa árvore classifica corretamente todos os exemplos.

Em geral, o tamanho da árvore necessária para classificar determinado conjunto de exemplos varia de acordo com a ordem com que as propriedades são testadas. A Figura 10.14 mostra uma árvore que é consideravelmente mais simples do que aquela da Figura 10.13, mas que também classifica corretamente os exemplos da Tabela 10.1.

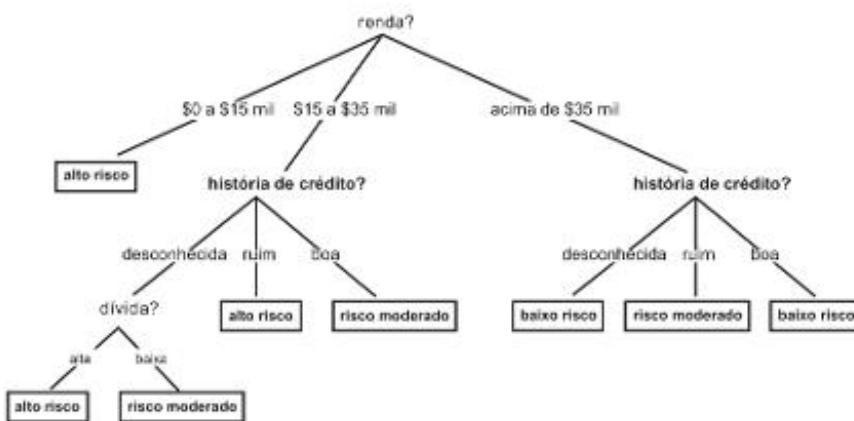
**Tabela 10.1** Dados da história de crédito de aplicações de financiamento.

Nº	Risco	História de crédito	Dívida	Garantia	Renda
1	alto	ruim	alta	nenhuma	\$0 a \$15 mil
2	alto	desconhecida	alta	nenhuma	\$15 a \$35 mil
3	moderado	desconhecida	baixa	nenhuma	\$15 a \$35 mil
4	alto	desconhecida	baixa	nenhuma	\$0 a \$15 mil
5	baixo	desconhecida	baixa	nenhuma	acima de \$35 mil
6	baixo	desconhecida	baixa	adequada	acima de \$35 mil
7	alto	ruim	baixa	nenhuma	\$0 a \$15 mil
8	moderado	ruim	baixa	adequada	acima de \$35 mil
9	baixo	boa	baixa	nenhuma	acima de \$35 mil
10	baixo	boa	alta	adequada	acima de \$35 mil
11	alto	boa	alta	nenhuma	\$0 a \$15 mil
12	moderado	boa	alta	nenhuma	\$15 a \$35 mil
13	baixo	boa	alta	nenhuma	acima de \$35 mil
14	alto	ruim	alta	nenhuma	\$15 a \$35 mil

**Figura 10.13** Uma árvore de decisão para avaliação de risco de crédito.



**Figura 10.14** Uma árvore de decisão simplificada para avaliação de risco de crédito.



Dado um conjunto de exemplos de treinamento e diversas árvores de decisão que o classificam corretamente, podemos perguntar qual árvore tem a maior probabilidade de classificar corretamente exemplos não vistos da população. O algoritmo ID3 supõe que essa é a árvore mais simples que cobre todos os exemplos de treinamento. A razão para isso é a heurística bem conhecida de se preferir a simplicidade e evitar suposições desnecessárias. Esse princípio, conhecido como *navalha de Occam*, foi articulado inicialmente pelo lógico medieval William de Occam, em 1324:

É inútil fazer com mais o que pode ser feito com menos... Não se deve multiplicar entidades além do necessário.

Uma versão mais contemporânea da navalha de Occam propõe que devemos sempre aceitar a resposta mais simples que se ajusta corretamente aos dados. Nesse caso, ela é a menor árvore de decisão que classifica corretamente todos os exemplos dados.

Embora a navalha de Occam tenha se mostrado uma heurística geral para todo tipo de atividade intelectual, seu uso aqui tem uma justificativa mais específica. Se supusermos que os exemplos dados são suficientes para construir uma generalização válida, então o nosso problema é o de distinguir as propriedades necessárias das supérfluas. A árvore de decisão mais simples que cobre todos os exemplos deve ser a que tem menor chance de incluir restrições desnecessárias. Embora essa ideia seja intuitivamente atraente, ela é uma suposição que deve ser testada empiricamente; a Seção 10.3.3 apresenta alguns desses resultados empíricos. Porém, antes de examinar esses resultados, apresentamos o algoritmo ID3 para a indução de árvores de decisão a partir de exemplos.

### 10.3.1 Indução de árvores de decisão: uma abordagem de cima para baixo

O algoritmo ID3 constrói árvores de decisão de cima para baixo. Note que, para qualquer propriedade, podemos dividir o conjunto de exemplos de treinamento em subconjuntos disjuntos, onde todos os exemplos em uma partição têm um valor comum para aquela propriedade. O ID3 seleciona uma propriedade para testar no nó atual da árvore e usa esse teste para dividir o conjunto de exemplos; o algoritmo constrói então, recursivamente, uma subárvore para cada partição. Esse processo continua até que todos os membros da partição estejam na mesma classe; essa classe se torna um nó folha da árvore. Como a ordem dos testes é crítica para a construção de uma árvore de decisão simples, o ID3 se baseia intensamente nos seus critérios para selecionar o teste na raiz de cada subárvore. Para simplificar nossa discussão, esta seção descreve o algoritmo para construir árvores de decisão, supondo uma função de seleção de teste apropriada. Na Seção 10.3.2, apresentamos a heurística de seleção do algoritmo ID3.

Por exemplo, considere o modo como o ID3 constrói a árvore da Figura 10.14 a partir da Tabela 10.1. Começando com a tabela completa de exemplos, ele seleciona renda como a propriedade raiz, usando a função de seleção descrita na Seção 10.3.2. Isso divide o conjunto de exemplos, como mostra a Figura 10.15, com os elementos de cada partição sendo listados pelo seu número na tabela.

O algoritmo de indução começa com uma amostra de membros das categorias-alvo classificados corretamente. O ID3 constrói, então, uma árvore de decisão de acordo com o algoritmo:

```

função induzir_árvore (conjunto_exemplos, Propriedades)
    inicio
        se todos elementos do conjunto_exemplo são da mesma classe
            então retorne um nó folha rotulado com esta classe
        senão, se Propriedades for vazio,
            então retorne nó folha rotulado com a disjunção de todas as classes no conjunto_exemplo
        senão início
            selecione uma propriedade P e a faça a raiz da árvore atual;
            retire P de Propriedades;
            para cada valor V de P
                inicio
                    crie um ramo da árvore rotulado com V;
                    construa partiçãoV com os elementos de conjunto_exemplos com valores V para propriedade P;
                    chame induzir_árvore(partiçãoV, Propriedades), adicione resultado ao ramo V
                fim
            fim
        fim
    fim

```

O ID3 aplica recursivamente a função `induzir_árvore` a cada partição. Por exemplo, a partição {1, 4, 7, 11} consiste inteiramente em indivíduos de alto risco; o ID3 cria um nó folha correspondente a ela. Ele seleciona a propriedade história de crédito como raiz da subárvore para a partição {2, 3, 12, 14}. Na Figura 10.16, história de crédito divide agora essa partição em {2, 3}, {14} e {12}. Continuando a selecionar testes e

**Figura 10.15** Uma árvore de decisão parcialmente construída.



**Figura 10.16** Outra árvore de decisão parcialmente construída.



construindo subárvores dessa maneira, o ID3 acaba produzindo a árvore da Figura 10.14. O leitor pode continuar essa construção; apresentamos uma implementação em Lisp na Sala Virtual deste livro.

Antes de apresentarmos a heurística de seleção de testes de ID3, vale a pena examinar a relação entre o algoritmo de construção de árvores e nossa visão de aprendizado como busca através de um espaço de conceitos. Podemos considerar o conjunto de todas as árvores de decisão possíveis como definindo um espaço de versões. Nossas operações para nos deslocarmos nesse espaço consistem da adição de testes a uma árvore. O algoritmo ID3 implementa uma forma de busca gulosa no espaço de todas as árvores possíveis: ele adiciona uma subárvore à árvore atual e continua a sua busca; ele não realiza retrocesso. Isso torna o algoritmo muito eficiente, mas o torna, também, dependente dos critérios para a seleção de propriedades para o teste.

### 10.3.2 Seleção de teste baseada na teoria da informação

Podemos considerar cada propriedade de um exemplo como contribuidora de certa quantidade de informação para a sua classificação. Por exemplo, se o objetivo for determinar a espécie de um animal, a descoberta de que ele põe ovos contribui com uma certa quantidade de informação a esse objetivo. O ID3 mede a informação obtida tornando cada propriedade a raiz da subárvore atual, então, seleciona a propriedade que fornece o maior ganho de informação.

A teoria da informação (Shannon, 1948) oferece uma base matemática para medir o conteúdo de informação de uma mensagem. Podemos considerar uma mensagem como uma ocorrência em um universo de mensagens possíveis; o ato de transmitir uma mensagem é o mesmo de selecionar uma dessas possíveis mensagens. Por esse ponto de vista, é razoável definir o conteúdo de informação de uma mensagem como dependente tanto do tamanho desse universo como da frequência com que cada mensagem possível ocorre.

A importância do número de mensagens possíveis é evidente em um exemplo de jogo: compare uma mensagem predizendo corretamente o resultado de um giro de roleta com a predição do resultado da jogada de uma moeda não viciada. Como a roleta pode ter mais resultados que o lançamento de uma moeda, uma mensagem sobre o resultado da roleta é muito mais valiosa: ganhar na roleta vale mais que ganhar no lançamento de uma moeda. Consequentemente, devemos considerar que essa mensagem transmite mais informação.

A influência da probabilidade de cada mensagem sobre a quantidade de informação é evidente em outro exemplo de aposta. Suponha que tenhamos manipulado uma moeda de modo que ocorra cara em  $\frac{3}{4}$  das vezes. Como já conhecemos o suficiente sobre a moeda para apostar corretamente em  $\frac{3}{4}$  das vezes, uma mensagem informando o resultado de um lançamento é menos valiosa para mim do que seria para uma moeda não manipulada.

Shannon formalizou isso definindo a quantidade de informação em uma mensagem como uma função da probabilidade de ocorrência  $p$  de cada mensagem possível, ou seja,  $-\log_2 p$ . Dado um universo de mensagens,  $M = \{m_1, m_2, \dots, m_n\}$  e a probabilidade  $p(m_i)$  para a ocorrência de cada mensagem, o conteúdo de informação esperado de uma mensagem  $M$  é dado por:

$$I[M] = \left( \sum_{i=1}^n -p(m_i) \log_2(p(m_i)) \right) = E[-\log_2 p(m_i)]$$

A informação em uma mensagem é medida em bits. Por exemplo, o conteúdo de informação de uma mensagem dizendo o resultado do lançamento de uma moeda não manipulada é:

$$\begin{aligned} I[\text{lançamento de moeda}] &= -p(\text{cara}) \log_2(p(\text{cara})) - p(\text{coroa}) \log_2(p(\text{coroa})) \\ &= -\frac{1}{2} \log_2\left(\frac{1}{2}\right) - \frac{1}{2} \log_2\left(\frac{1}{2}\right) \\ &= 1 \text{ bit} \end{aligned}$$

Entretanto, se a moeda foi manipulada para que dê cara em 75% das vezes, então o conteúdo de informação de uma mensagem é:

$$\begin{aligned} I[\text{lançamento de moeda}] &= -\frac{3}{4} \log_2\left(\frac{3}{4}\right) - \frac{1}{4} \log_2\left(\frac{1}{4}\right) \\ &= -\frac{3}{4} * (-0,415) - \frac{1}{4} * (-2) \\ &= 0,811 \text{ bit} \end{aligned}$$

Essa definição formaliza muitas das nossas intuições sobre o conteúdo de informação de mensagens. A teoria da informação é bastante usada em ciência da computação e em telecomunicações, incluindo aplicações como a determinação da capacidade de transmitir informação de canais de comunicação, o desenvolvimento de algoritmos de compressão de dados e o desenvolvimento de estratégias de comunicação resistentes a ruído. O ID3 usa a teoria da informação para selecionar o teste que fornece o maior ganho de informação para a classificação de exemplos de treinamento.

Podemos considerar uma árvore de decisão como transmissora de informações sobre a classificação de exemplos da tabela de decisão; o conteúdo de informação da árvore é calculado a partir das probabilidades das diferentes classificações. Por exemplo, se supusermos que todos os exemplos da Tabela 10.1 ocorrem com igual probabilidade, então:

$$p(\text{risco é alto}) = \frac{6}{14}, p(\text{risco é moderado}) = \frac{3}{14}, p(\text{risco é baixo}) = \frac{5}{14}$$

Como resultado, a distribuição descrita na Tabela 10.1,  $D_{10.1}$ , e, consequentemente, qualquer árvore que cubra esses exemplos, é:

$$\begin{aligned} I[D_{10.1}] &= -\frac{6}{14} \log_2\left(\frac{6}{14}\right) - \frac{3}{14} \log_2\left(\frac{3}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) \\ &= -\frac{6}{14} * (-1,222) - \frac{3}{14} * (-2,222) - \frac{5}{14} * (-1,485) \\ &= 1,531 \text{ bit} \end{aligned}$$

O ganho de informação fornecido pela realização de um teste na raiz da árvore atual é igual à informação total na árvore menos a quantidade de informação necessária para completar a classificação após a realização do teste. A quantidade de informação necessária para completar a árvore é definida como a média ponderada da informação em todas as suas subárvore. Calculamos a média ponderada multiplicando o conteúdo de informação de cada subárvore pela porcentagem de exemplos presentes nessa subárvore e somando esses produtos.

Considere um conjunto de exemplos de treinamento  $C$ . Se tornarmos a propriedade  $P$ , com  $n$  valores, a raiz da árvore atual, isso dividirá  $C$  em subconjuntos  $\{C_1, C_2, \dots, C_n\}$ . A informação esperada necessária para completar a árvore, após tornar  $P$  a raiz da árvore, é:

$$E[P] = \sum_{i=1}^n \frac{|C_i|}{|C|} I[C_i]$$

O ganho da propriedade  $P$  é calculado pela subtração da informação esperada para completar a árvore da informação do conteúdo total da árvore:

$$\text{ganho}(P) = I[C] - E[P]$$

No exemplo da Tabela 10.1, se tornarmos renda a propriedade testada na raiz da árvore, isso dividirá a tabela de exemplos nas partições  $C_1 = \{1, 4, 7, 11\}$ ,  $C_2 = \{2, 3, 12, 14\}$  e  $C_3 = \{5, 6, 8, 9, 10, 13\}$ . A informação esperada necessária para completar a árvore é:

$$\begin{aligned} E[\text{renda}] &= \frac{4}{14} * I[C_1] + \frac{4}{14} * I[C_2] + \frac{6}{14} * I[C_3] \\ &= \frac{4}{14} * 0,0 + \frac{4}{14} * 1,0 + \frac{6}{14} * 0,650 \\ &= 0,564 \text{ bit} \end{aligned}$$

O ganho de informação para a distribuição da Tabela 10.1 é:

$$\begin{aligned} \text{ganho}(\text{renda}) &= I[D_{10,1}] - E[\text{renda}] \\ &= 1,531 - 0,564 \\ &= 0,967 \text{ bit} \end{aligned}$$

De modo semelhante, podemos mostrar que:

$$\begin{aligned} \text{ganho}(\text{história de crédito}) &= 0,266 \\ \text{ganho}(\text{dívida}) &= 0,063 \\ \text{ganho}(\text{garantia}) &= 0,206 \end{aligned}$$

Como a renda fornece o maior ganho de informação, o ID3 a selecionará como a raiz da árvore. O algoritmo continuará a aplicar essa análise recursivamente para cada subárvore até completar a árvore.

### 10.3.3 Avaliando o ID3

Embora o algoritmo ID3 produza árvores de decisão simples, não é óbvio que tais árvores sejam eficazes em predizer a classificação de exemplos desconhecidos. O ID3 foi avaliado tanto em testes controlados como em aplicações e provou que funciona bem na prática.

Quinlan, por exemplo, avaliou o desempenho do ID3 no problema de aprender a classificar configurações de tabuleiros em jogadas finais de xadrez (Quinlan, 1983). As jogadas finais envolviam as peças brancas jogando com um rei e uma torre, e as peças pretas jogando com um rei e um cavalo. O objetivo do ID3 era aprender a reconhecer tabuleiros que levavam à derrota das peças pretas em três movimentos. Os atributos eram diferentes propriedades de alto nível de uma configuração de tabuleiro, como “incapacidade de movimentar o rei com segurança”. O teste usou 23 desses atributos.

Levando em conta as simetrias do tabuleiro, o domínio completo do problema consistia em 1,4 milhão de diferentes configurações de tabuleiro, das quais 474.000 eram derrotas para as peças pretas em três movimentos. O ID3 foi testado fornecendo a ele um conjunto de treinamento selecionado aleatoriamente e testando-o com 10.000 diferentes configurações de tabuleiros, também selecionados aleatoriamente. O teste de Quinlan forneceu os resultados mostrados na Tabela 10.2. Os erros máximos previstos foram obtidos de um modelo estatístico do comportamento do ID3 no domínio. Para obter mais detalhes, veja Quinlan (1983).

Esses resultados são apoiados por estudos empíricos e pelos resultados anedóticos de outras aplicações. Variações de ID3 foram desenvolvidas para lidar com problemas como ruído e conjuntos de treinamento excessivamente grandes. Para mais detalhes, consulte Quinlan (1986, *a, b*).

**Tabela 10.2** Avaliação do ID3.

Tamanho do conjunto de treinamento	Porcentagem do universo	Erros em 10.000 tentativas	Erros máximos previstos
200	0,01	199	728
1.000	0,07	33	146
5.000	0,36	8	29
25.000	1,79	6	7
125.000	8,93	2	1

### 10.3.4 Questões envolvendo árvores de decisão: empacotamento, reforço

Quinlan (1983) foi o primeiro a sugerir o uso da teoria da informação para produzir subárvore no aprendizado de árvores de decisão, e seu trabalho foi a base da nossa apresentação. Entretanto, nossos exemplos eram bem comportados e seu uso era direto. Existe uma série de questões que não abordamos e que ocorrem frequentemente em grandes conjuntos de dados:

1. Os dados são ruins. Isso pode acontecer quando dois (ou mais) conjuntos de atributos idênticos dão diferentes resultados. O que podemos fazer se não tivermos motivo *a priori* para eliminarmos dados?
2. Faltam dados de algum conjunto de atributos, talvez porque seja muito custoso obtê-los. Devemos extrapolar os dados? Podemos criar um novo valor “desconhecido”? Como podemos amenizar essas irregularidades?
3. Alguns dos conjuntos de atributos são contínuos. Tratamos disso dividindo o valor contínuo “renda” em subconjuntos de valores convenientes e, então, usamos esses agrupamentos. Existem métodos melhores?
4. O conjunto de dados pode ser muito grande para o algoritmo de aprendizado. Como podemos lidar com isso?

O tratamento dessas questões produziu novas gerações de algoritmos de árvores de decisão pelo aperfeiçoamento do ID3. O mais notável desses algoritmos é o C4.5 (Quinlan, 1996). Essas questões também levaram a técnicas como empacotamento e reforço. Como os dados para os sistemas de aprendizado de classificadores são vetores ou exemplos do tipo atributo-valor, procura-se manipular os dados para ver se diferentes classificadores são produzidos.

O *empacotamento* (*bagging*) produz conjuntos de treinamento replicados por amostragem com reposição a partir dos exemplos de treinamento. O *reforço* (*boosting*) usa todos os exemplos a cada replicação, mas mantém um peso para cada exemplo do conjunto de treinamento. A intenção desse peso é refletir a importância do vetor. Quando os pesos são ajustados, são produzidos diferentes classificadores, uma vez que os pesos fazem com que a máquina de aprendizado focalize exemplos diferentes. Nos dois casos, os múltiplos classificadores produzidos são combinados por votação para formar um classificador composto. No empacotamento, cada classificador componente tem o mesmo voto, enquanto o reforço atribui diferentes forças de voto para os classificadores componentes de acordo com a sua precisão.

Ao se trabalhar com grandes conjuntos de dados, é comum dividir os dados em subconjuntos, construir a árvore de decisão sobre um subconjunto e, depois, testar sua precisão sobre outros subconjuntos. A literatura sobre o aprendizado de árvores de decisão é atualmente bastante extensa, com uma série de conjuntos de dados disponibilizados on-line e uma série de resultados empíricos publicados, mostrando os resultados do uso de várias versões de algoritmos de árvores de decisão sobre esses dados.

Finalmente, pode-se converter diretamente uma árvore de decisão em um conjunto de regras correspondente. O que se faz é transformar cada caminho possível ao longo da árvore de decisão em uma única regra. O padrão para essa regra, seu lado esquerdo, consiste em decisões que levam ao nó folha. A ação, ou lado direito da regra, é o nó folha ou o resultado da árvore. Esse conjunto de regras pode ser ainda manipulado para capturar subárvores no interior da árvore de decisão. Esse conjunto de regras pode então ser utilizado para classificar novos dados.

## 10.4 Viés indutivo e capacidade de aprendizado

Até agora, nossa discussão enfatizou o uso de dados empíricos para guiar a generalização. Entretanto, a indução bem-sucedida depende também de conhecimento prévio e de suposições *a priori* sobre a natureza dos conceitos que estão sendo aprendidos. O *viés indutivo* se refere a qualquer critério que uma máquina de aprendizado use para restringir o espaço de conceitos ou para selecionar conceitos desse espaço. Na próxima seção, examinamos a necessidade desse viés e os tipos de vieses que os programas de aprendizado geralmente utilizam. A Seção 10.4.2 introduz os resultados teóricos da quantificação da eficácia dos vieses indutivos.

### 10.4.1 Viés indutivo

Os espaços de aprendizado tendem a ser grandes; sem um modo de podá-los, o aprendizado baseado em busca seria impossível na prática. Por exemplo, considere o problema de aprender uma classificação de sequências de bits (cadeias de 0s e 1s) a partir de exemplos positivos e negativos. Como uma classificação é simplesmente um subconjunto dos conjuntos de todas as sequências possíveis, o número total de classificações possíveis é igual ao conjunto potência, ou conjunto de todos os subconjuntos, da população inteira. Se existem  $m$  exemplos, há  $2^m$  classificações possíveis. Mas, para sequências de  $n$  bits, há  $2^n$  sequências diferentes. Assim, há  $2$  elevado à potência  $2^n$  classificações diferentes de sequências de bits de comprimento  $n$ . Para  $n = 50$ , esse número é maior que o número de moléculas no universo conhecido! Sem restrições heurísticas, seria impossível para uma máquina de aprendizado buscar efetivamente tais espaços, a não ser em domínios muito triviais.

Outra razão para a necessidade de viés é a natureza da própria generalização indutiva. Generalização não necessariamente preserva a verdade. Por exemplo, se encontramos um político honesto, isso justifica afirmar que todos os políticos são honestos? Quantos políticos honestos precisamos encontrar para que essa suposição seja justificada? Hume discutiu esse problema, conhecido como o problema da indução, há muitos anos:

Você pode afirmar que uma proposição é uma inferência de outra; mas você precisa reconhecer que a inferência não é intuitiva, tampouco demonstrativa. Qual é a sua natureza então? Dizer que ela é experimental é evitar a questão. Como sua fundamentação, suponha que, para todas as inferências a partir da experiência, o futuro se pareça com o passado e que forças similares estejam associadas a qualidades perceptivas similares (Hume, 1748).

Incidentalmente, no século XVIII, o trabalho de Hume era visto como uma ameaça intelectual, especialmente às tentativas da comunidade religiosa para provar matematicamente a existência e os atributos da divindade. Entre as teorias contrárias, propostas para resgatar a “certeza”, havia a do reverendo Bayes, um clérigo inglês; veja nos capítulos 5, 9 e 13. Mas voltemos à indução.

No aprendizado indutivo, os dados de treinamento são apenas um subconjunto de todos os exemplos do domínio; consequentemente, qualquer conjunto de treinamento pode dar suporte a muitas generalizações diferentes. No nosso exemplo de um classificador de uma sequência de bits, suponha que a máquina de aprendizado tenha

recebido as sequências {1100, 1010} como exemplos positivos de uma classe de sequências. Muitas generalizações são consistentes com esses exemplos: o conjunto de todas as sequências começando com “1” e terminando com “0”, o conjunto de todas as sequências começando com “1”, o conjunto de todas as sequências de paridade par ou qualquer outro subconjunto de toda a população que inclua {1100, 1010}. O que a máquina pode usar para escolher entre essas generalizações? Apenas os dados não são suficientes; todas essas escolhas são consistentes com os dados. A máquina deve fazer suposições adicionais sobre conceitos “prováveis”.

Em aprendizado, essas suposições frequentemente tomam a forma de heurísticas para escolher um ramo do espaço de busca. A função de seleção de testes baseada na teoria da informação do ID3 (Seção 10.3.2) é um exemplo de uma dessas heurísticas. O ID3 realiza uma busca do tipo subida de encosta através do espaço de possíveis árvores de decisão. A cada estágio da busca, ele examina todos os testes que poderiam ser usados para estender a árvore e escolhe o teste que ganha a maior informação. Essa é uma heurística “gulosa”: ela favorece ramos do espaço de busca que avançam uma distância maior em direção ao estado objetivo.

Essa heurística permite que o ID3 busque eficientemente o espaço de árvores de decisão e aborda, também, o problema da escolha de generalizações plausíveis a partir de dados limitados. O ID3 supõe que a menor árvore que classifica corretamente todos os dados apresentados terá maior probabilidade de classificar corretamente futuros exemplos de treinamento. O raciocínio para essa suposição é que as menores árvores têm menor probabilidade de fazer suposições não suportadas pelos dados. Se o conjunto de treinamento for suficientemente grande e realmente representar toda a população, tais árvores devem incluir todos os testes essenciais para determinar a pertinência de classe. Como discutido na Seção 10.3.3, avaliações empíricas demonstraram que essa suposição se justifica. Essa preferência para definições simples de conceitos é usada em vários algoritmos de aprendizado, tais como o CLUSTER/2 da Seção 10.6.2.

Outra forma de viés indutivo consiste em restrições sintáticas sobre a representação dos conceitos aprendidos. Tais vieses não são heurísticas para selecionar um ramo do espaço de conceitos. Em vez disso, eles limitam o próprio tamanho do espaço exigindo que os conceitos aprendidos sejam expressos em uma linguagem de representação restrita. As árvores de decisão, por exemplo, são uma linguagem muito mais restrita que o cálculo de predicados completo. A redução correspondente do tamanho do espaço de conceitos é essencial para a eficiência do ID3.

Um exemplo de um viés sintático que pode ser efetivo na classificação de sequências de bits seria limitar as descrições de conceitos a padrões de símbolos do conjunto {0, 1, #}. Um padrão define a classe de todas as sequências que casam com ele, onde esse casamento é determinado de acordo com as seguintes regras:

Se o padrão tem um “0” em certa posição, então a sequência-alvo deve ter um “0” na posição correspondente.

Se o padrão tem um “1” em certa posição, então a sequência-alvo deve ter um “1” na posição correspondente.

Um “#” em uma dada posição pode casar tanto com “1” quanto com “0”.

Por exemplo, o padrão “1##0” define o conjunto de sequências {1110, 1100, 1010, 1000}.

Considerando apenas aquelas classes que podem ser representadas como um único padrão desse tipo, reduzimos consideravelmente o tamanho do espaço de conceitos. Para sequências de comprimento  $n$ , podemos definir  $3^n$  padrões diferentes. Isso é muito menor que os  $2^n$ , na potência  $2^n$ , conceitos possíveis no espaço irrestrito. Esse viés também permite implementar diretamente a busca em espaço de versões, onde a generalização envolve substituir um 1 ou um 0 em um padrão candidato por um #. Entretanto, o custo que incorremos para esse viés é a incapacidade de representar (e, consequentemente, aprender) certos conceitos. Por exemplo, um único padrão desse tipo não pode representar a classe de todas as sequências de paridade par.

Esse compromisso entre expressividade e eficiência é típico. O LEX, por exemplo, não distingue entre inteiros ímpares ou pares na sua taxonomia de símbolos. Consequentemente, ele não pode aprender uma heurística qualquer que dependa dessa distinção. Embora tenham sido desenvolvidos trabalhos sobre programas que podem mudar o seu viés em resposta aos dados (Utgoff, 1986), a maioria dos programas de aprendizado assume um viés indutivo fixo.

O aprendizado de máquina já explorou uma série de vieses representacionais:

*Vieses conjuntivos* restringem o conhecimento aprendido a conjunções de literais. Isso é particularmente comum porque o uso de disjunção nas descrições de conceitos cria problemas para a generalização. Por exemplo, suponha que seja permitido o uso arbitrário de disjunções na representação de conceitos no algoritmo de eliminação de candidatos. Como a generalização maximamente específica de um conjunto de exemplos positivos é simplesmente a disjunção de todos os exemplos, a máquina de aprendizado não conseguirá generalizar. Ela adicionará disjunções *ad infinitum*, implementando uma forma de aprendizado por repetição (Mitchell, 1980).

*Limitações no número de disjunções.* Vieses puramente conjuntivos são muito limitados para várias aplicações. Uma abordagem que aumenta a expressividade de uma representação, ao mesmo tempo em que trata dos problemas da disjunção, é permitir um número pequeno e limitado de disjunções.

*Vetores de características* são uma representação que descreve objetos como um conjunto de características cujos valores diferem de um objeto para outro. Os objetos apresentados na Tabela 10.1 são representados como conjuntos de características.

*Árvores de decisão* são uma representação de conceitos que se mostrou eficaz no algoritmo ID3.

*Cláusulas de Horn* requerem uma restrição, na forma de implicações, que tem sido usada em raciocínio automatizado, bem como por uma série de programas para o aprendizado de regras a partir de exemplos. Apresentamos as cláusulas de Horn na Seção 14.2.

Além dos vieses sintáticos discutidos nesta seção, vários programas utilizam conhecimento específico do domínio para considerar a semântica conhecida ou suposta do domínio. Esse conhecimento pode fornecer um viés extremamente efetivo. A Seção 10.5 examina essas abordagens baseadas em conhecimento. Entretanto, antes de considerar o papel do conhecimento no aprendizado, examinaremos rapidamente os resultados teóricos que quantificam o viés indutivo. Na Seção 16.2, apresentamos também uma discussão que resume o viés indutivo em sistemas de aprendizado.

#### 10.4.2 Teoria do aprendizado

O objetivo do viés indutivo é restringir o conjunto de conceitos-alvo de tal forma que possamos tanto buscar eficientemente o conjunto como encontrar definições de conceitos de alta qualidade. Um conjunto interessante de trabalhos teóricos aborda o problema de quantificar a eficiência de um viés indutivo.

Definimos a qualidade de conceitos em termos de sua habilidade de classificar corretamente objetos que não estavam incluídos no conjunto de exemplos de treinamento. Não é difícil escrever um algoritmo de aprendizado que produza conceitos que classifiquem corretamente todos os exemplos vistos no treinamento; o aprendizado por repetição é suficiente para isso. Entretanto, devido ao grande número de exemplos existentes na maioria dos domínios, ou devido ao fato de alguns exemplos não estarem disponíveis, os algoritmos podem apenas examinar uma parte de todos os exemplos possíveis. Assim, o desempenho de um conceito aprendido sobre novas ocorrências é criticamente importante. Ao testarmos algoritmos de aprendizado, geralmente dividimos o conjunto de todos os exemplos em conjuntos não sobrepostos de exemplos de treinamento e de teste. Após treinar um programa sobre o conjunto de treinamento, o testamos sobre o conjunto de teste.

É útil considerar a eficiência e a correção como propriedades da linguagem para expressar conceitos, isto é, o viés indutivo, e não de um algoritmo de aprendizado em particular. Os algoritmos de aprendizado buscam um espaço de conceito; se esse espaço é manejável e contiver conceitos que são eficazes, então qualquer algoritmo de aprendizado razoável deverá encontrar essas definições; se o espaço for muito complexo, o sucesso de um algoritmo será limitado. Um exemplo extremo esclarecerá esse ponto.

O conceito de bola pode ser aprendido, dada uma linguagem adequada para descrever as propriedades de objetos. Após ver um número relativamente pequeno de bolas, uma pessoa será capaz de defini-las de forma concisa: bolas são redondas. Compare isso com um conceito que não é possível de ser aprendido: suponha que um grupo de pessoas percorra o planeta e selecione um conjunto de milhões de objetos inteiramente ao acaso, chamando a classe resultante de *monte\_de\_coisas*. Um conceito que fosse induzido de uma amostra de *monte\_de\_*

coisas não apenas requereria uma representação extremamente complexa, como também seria improvável que esse conceito classificasse corretamente membros do conjunto que não tivessem sido vistos.

Essas observações não fazem qualquer suposição quanto aos algoritmos de aprendizado utilizados, a não ser o fato de que eles podem encontrar no espaço um conceito consistente com os dados. O conceito bala pode ser aprendido porque podemos defini-lo em termos de algumas poucas características: o conceito pode ser expresso em uma linguagem que fornece um viés. A tentativa de descrever o conceito `monte_de_coisas` requer uma definição de conceito que é tão longa quanto a lista de todas as propriedades de todos os objetos da classe.

Assim, em vez de definirmos a possibilidade de aprendizado em termos de algoritmos específicos, fazemos em termos da linguagem usada para representar conceitos. Além disso, para mantermos a generalidade, não definimos a possibilidade de aprendizado em domínios de problemas específicos, como no aprendizado de `monte_de_coisas`. Em vez disso, nós a definimos em termos das propriedades sintáticas da linguagem de definição de conceitos.

Ao definirmos a possibilidade de aprendizado, não devemos apenas levar em consideração a eficiência; devemos, também, lidar com o fato de que temos dados limitados. Em geral, não podemos esperar encontrar exatamente o conceito correto a partir de uma amostra aleatória de exemplos. Em vez disso, assim como na estimativa estatística da média de um conjunto, procuramos encontrar um conceito que seja aproximadamente correto. Em consequência, a exatidão de um conceito é a probabilidade, sobre toda a população de exemplos, de que ele classifique corretamente um exemplo.

Além da exatidão de conceitos aprendidos, também devemos considerar a probabilidade de um algoritmo encontrar tais conceitos. Isto é, existe uma pequena chance de que as amostras que vemos serem tão atípicas que o aprendizado se torne impossível. Assim, uma distribuição particular de exemplos positivos, ou um conjunto de treinamento particular selecionado desses exemplos, pode ou não ser suficiente para selecionar um conceito de alta qualidade. Portanto, estamos preocupados com duas probabilidades: a probabilidade de que nossas amostras não sejam atípicas e a de que o algoritmo encontre um conceito de qualidade, o erro de estimativa normal. Essas duas probabilidades são limitadas por  $\delta$  e  $\epsilon$ , respectivamente, na definição de possibilidade de aprendizado PAC que apresentaremos a seguir.

Resumindo, a possibilidade de aprendizado é uma propriedade dos espaços de conceitos e é determinada pela linguagem requerida para representar conceitos. Ao avaliar esses espaços, devemos levar em consideração tanto a probabilidade de que os dados sejam por coincidência pobres quanto a probabilidade com a qual o conceito resultante classificará corretamente exemplos não vistos anteriormente. Valiant (1984) formalizou essas intuições na teoria do aprendizado provável e aproximadamente correto (PAC).

Uma classe de conceitos pode ser *aprendida*, segundo a PAC, se existir um algoritmo eficiente que tenha alta probabilidade de encontrar um conceito *aproximadamente correto*. Aproximadamente correto significa que o conceito classifica corretamente uma alta porcentagem de novos exemplos. Assim, exigimos que o algoritmo não só encontre, com alta probabilidade, um conceito que seja aproximadamente correto, mas também que ele próprio seja eficiente. Um aspecto interessante dessa definição é que ela não depende necessariamente da distribuição de exemplos positivos no espaço de exemplos. Ela depende da natureza da linguagem de conceitos, isto é, do viés e do grau desejado de correção. Por fim, ao se fazer suposições sobre as distribuições de exemplos, normalmente é possível se obter um desempenho melhor usando um número menor de amostras que aquele que a teoria requer.

Formalmente, Valiant define a possibilidade de aprendizado PAC como segue. Seja  $C$  um conjunto de conceitos  $c$  e  $X$  um conjunto de exemplos. Os conceitos podem ser algoritmos, padrões ou qualquer outro meio de dividir  $X$  em exemplos positivos e negativos.  $C$  é possível de ser aprendido, segundo a PAC, se existir um algoritmo com as seguintes propriedades:

1. Se, para um erro de conceito  $\epsilon$  e uma probabilidade de falha  $\delta$ , existir um algoritmo que, dada uma amostra aleatória de exemplos de tamanho  $n = |X|$  polinomial em  $1/\epsilon$  e em  $1/\delta$ , o algoritmo produz um conceito  $c$ , um elemento de  $C$ , tal que a probabilidade de que  $c$  tenha um erro de generalização maior que  $\epsilon$  seja menor que  $\delta$ . Isto é, para  $y$  amostrado da mesma distribuição de onde os exemplos em  $X$  foram retirados:

$$P[P[y \text{ é classificado errado por } c] \geq \epsilon] \leq \delta.$$

2. O tempo de execução para o algoritmo é polinomial em  $n$ , em  $l/\epsilon$  e em  $1/\delta$ .

Usando essa definição de aprendizado PAC, os pesquisadores mostraram que vários vieses indutivos eram tratáveis. Valiant (1984), por exemplo, provou que se pode aprender a classe de expressões k-FNC. Expressões k-FNC são sentenças na forma normal conjuntiva com um limite no número de termos disjuntivos; expressões são formadas pela conjunção de cláusulas,  $c_1 \wedge c_2 \wedge \dots \wedge c_n$ , onde cada  $c_i$  é a disjunção de não mais que  $k$  literais. Esse resultado teórico dá suporte à restrição comum de conceitos até a forma conjuntiva usada em muitos algoritmos de aprendizado. Não repetiremos a prova aqui, mas indicamos o artigo de Valiant, onde ele prova esse resultado com a possibilidade de aprendizado de outros vieses. Para resultados adicionais sobre a possibilidade de aprendizado e viés indutivo, veja Haussler (1988) e Martin (1997).

## 10.5 Conhecimento e aprendizado

O ID3 e o algoritmo de eliminação de candidatos generalizam com base nas regularidades dos dados de treinamento. Tais algoritmos são normalmente classificados como *baseados em similaridade*, pois a generalização é fundamentalmente uma função de similaridades ao longo dos exemplos de treinamento. Os vieses empregados por esses algoritmos são limitados a restrições sintáticas sobre a forma do conhecimento aprendido; eles não fazem suposições mais fortes sobre a semântica dos domínios. Nesta seção, examinamos algoritmos, como o *aprendizado baseado em explicação*, que usa conhecimento prévio do domínio para orientar a generalização.

A princípio, parece ser contraditória a ideia de que o conhecimento prévio seja necessário para o aprendizado. Entretanto, tanto os pesquisadores em aprendizado de máquina como em ciência cognitiva empregaram exatamente essa noção, argumentando que o aprendizado mais eficaz ocorre quando a máquina de aprendizado já tem um conhecimento considerável do domínio. Um argumento a favor da importância do conhecimento no aprendizado é que técnicas de aprendizado baseadas em similaridade dependem de quantidades relativamente grandes de dados de treinamento. Os seres humanos, ao contrário, podem formar generalizações confiáveis a partir de quantidades tão pequenas quanto um único exemplo de treinamento, e muitas aplicações práticas exigem que um programa de aprendizado faça o mesmo.

Outro argumento para a importância de conhecimento prévio é o reconhecimento de que qualquer conjunto de exemplos de treinamento pode dar suporte a um número ilimitado de generalizações, a maioria das quais é irrelevantes ou não faz sentido. O viés indutivo é uma forma de se fazer essa distinção. Nesta seção, examinamos algoritmos que vão além de um viés puramente sintático e consideramos o papel de um conhecimento forte do domínio no aprendizado.

### 10.5.1 Meta-DENDRAL

O Meta-DENDRAL (Buchanan e Mitchell, 1978) é um dos primeiros exemplos, e ainda um dos melhores, de uso de conhecimento em aprendizado indutivo. O Meta-DENDRAL adquire regras a serem usadas pelo programa DENDRAL para analisar dados de espectrografia de massa. O DENDRAL infere a estrutura de moléculas orgânicas a partir de suas fórmulas químicas e dados de espectrografia de massa.

Um espectrômetro de massa bombardeia moléculas com elétrons, fazendo com que algumas ligações químicas se quebrem. Os químicos medem o peso dos fragmentos resultantes e interpretam esses resultados para ganhar uma compreensão da estrutura do composto. DENDRAL emprega conhecimento na forma de regras para interpretar dados de espectrografia de massa. A premissa de uma regra DENDRAL é um grafo de uma porção de uma estrutura molecular. A conclusão da regra é esse grafo com a localização da clivagem indicada.

O Meta-DENDRAL infere essas regras a partir de resultados de espectrografia em moléculas de estrutura conhecida. Ele recebe a estrutura de um composto conhecido com a massa e a abundância relativa dos fragmentos produzidos pela espectrografia. Ele as interpreta, gerando um relatório de onde as quebras ocorreram.

Essas explicações das fragmentações em moléculas específicas são usadas como exemplos para a construção de regras gerais.

Para determinar a posição de uma clivagem em uma rodada de treinamento, o programa DENDRAL utiliza uma “teoria de ordem um meio” da química orgânica. Essa teoria, embora não seja suficientemente poderosa para possibilitar a construção direta de regras DENDRAL, possibilita a interpretação de clivagens dentro de moléculas conhecidas. A teoria de ordem um meio consiste em regras, restrições e heurísticas, tais como:

Ligações duplas e triplas não se quebram.

Apenas fragmentos maiores que dois átomos de carbono aparecem nos dados.

Usando a teoria de ordem um meio, o meta-DENDRAL constrói explicações sobre a clivagem. Essas explicações indicam os locais prováveis de clivagem com possíveis migrações de átomos por meio da quebra.

Essas explicações se tornam o conjunto de exemplos positivos para um programa de indução de regras. Esse componente induz as restrições nas premissas de regras DENDRAL, por meio de uma busca do geral para o específico. Essa busca começa com uma descrição geral de uma clivagem:  $X_1^*X_2$ . Esse padrão significa que uma clivagem, indicada pelo asterisco, pode ocorrer entre dois átomos quaisquer. Ela especializa o padrão:

adicionando átomos:  $X_1^*X_2 \rightarrow X_3 - X_1^*X_2$

onde o operador “-” indica uma ligação química, ou

instanciando átomos ou atributos de átomos:  $X_1^*X_2 \rightarrow C^*X_2$

O Meta-DENDRAL aprende apenas a partir de exemplos positivos e realiza uma busca do tipo subida de encosta no espaço de conceitos. Ele evita a generalização excessiva, limitando as regras candidatas a cobrir apenas cerca de metade dos exemplos de treinamento. Os componentes subsequentes do programa avaliam e refinam essas regras, procurando regras redundantes ou modificando regras que possam ser excessivamente genéricas ou específicas.

A força do meta-DENDRAL está no seu uso de conhecimento de domínio para transformar os dados brutos em uma forma mais utilizável. Isso confere ao programa uma resistência a ruído, devido ao uso da sua teoria para eliminar dados estranhos ou potencialmente errados e pela habilidade de aprender a partir de relativamente poucos exemplos de treinamento. A visão de que os dados de treinamento devem ser assim interpretados para serem totalmente utilizáveis é a base do aprendizado baseado em explicações.

### 10.5.2 Aprendizado baseado em explicações

O *aprendizado baseado em explicações* (ABE) utiliza uma teoria do domínio explicitamente representada para construir uma explicação para um exemplo de treinamento, normalmente uma prova de que o exemplo deve-se logicamente da teoria. Generalizando a explicação do exemplo, em vez dele próprio, o aprendizado baseado em explicações filtra ruído, seleciona aspectos relevantes da experiência e organiza os dados de treinamento em uma estrutura sistemática e coerente.

Há diversas formulações alternativas para essa ideia. Por exemplo, o programa STRIPS, para representar operadores gerais para planejamento (veja a Seção 8.4), tem exercido uma influência poderosa nessas pesquisas (Fikes et al., 1972). O Meta-DENDRAL, como acabamos de discutir, estabeleceu o poder da interpretação baseada em teoria de exemplos de treinamento. Mais recentemente, diversos autores (DeJong e Mooney, 1986; Minton, 1988) propuseram formulações alternativas para essa ideia. O algoritmo de *generalização baseada em explicação*, de Mitchell et al. (1986), também é típico do gênero. Nesta seção, examinamos uma variação do algoritmo ABE desenvolvido por DeJong e Mooney (1986).

O ABE começa com:

1. *Um conceito-alvo.* A tarefa de aprendizado é determinar uma definição efetiva desse conceito. Dependendo da aplicação específica, o conceito-alvo pode ser uma classificação, um teorema para ser provado, um plano para alcançar um objetivo ou uma heurística para um resolvedor de problemas.
2. *Um exemplo de treinamento,* um exemplo do alvo.

3. *Uma teoria do domínio*, um conjunto de regras e fatos que são usados para explicar como o exemplo de treinamento é uma ocorrência do conceito-alvo.
4. *Critérios operacionais*, uma maneira de descrever a forma que as definições podem assumir.

Para ilustrar o ABE, apresentamos um exemplo de aprendizado quando um objeto é uma caneca. Essa é uma variação de um problema explorado por Winston et al. (1983) e adaptado para o aprendizado baseado em explicações de Mitchell et al. (1986). O conceito-alvo é uma regra que pode ser usada para inferir se um objeto é uma caneca:

$$\text{premissa}(X) \rightarrow \text{caneca}(X)$$

onde premissa é uma expressão conjuntiva contendo a variável X.

Suponha uma teoria do domínio que inclua as seguintes regras sobre canecas:

$$\begin{aligned} &\text{portável}(X) \wedge \text{mantém\_líquido}(X) \rightarrow \text{caneca}(X) \\ &\text{parte\_de}(Z, W) \wedge \text{cônico}(W) \wedge \text{aponta\_para\_cima}(W) \rightarrow \text{mantém\_líquido}(Z) \\ &\text{leve}(Y) \wedge \text{parte\_de}(Y, \text{alça}) \rightarrow \text{portável}(Y) \\ &\text{pequeno}(A) \rightarrow \text{leve}(A) \\ &\text{feito\_de}(A, \text{penas}) \rightarrow \text{leve}(A) \end{aligned}$$

O exemplo de treinamento é uma ocorrência do conceito-alvo. Isto é, nos é dado:

$$\begin{aligned} &\text{caneca}(\text{obj1}) \\ &\text{pequeno}(\text{obj1}) \\ &\text{parte\_de}(\text{obj1}, \text{alça}) \\ &\text{possui}(\text{roberto}, \text{obj1}) \\ &\text{parte\_de}(\text{obj1}, \text{fundo}) \\ &\text{parte\_de}(\text{obj1}, \text{cavidade}) \\ &\text{aponta\_para\_cima}(\text{cavidade}) \\ &\text{cônico}(\text{cavidade}) \\ &\text{cor}(\text{obj1}, \text{vermelho}) \end{aligned}$$

Finalmente, suponha que os critérios operacionais exijam que os conceitos-alvo sejam definidos em termos de propriedades estruturais observáveis dos objetos, tais como parte\_de e aponta\_para\_cima. Podemos fornecer regras do domínio que possibilitem ao programa de aprendizado inferir se uma descrição é operacional ou podemos simplesmente listar predicados operacionais.

Usando essa teoria, um provador de teoremas pode construir uma explicação de por que o exemplo é, de fato, uma ocorrência do conceito de treinamento: uma prova de que o conceito-alvo decorre logicamente do exemplo, como na primeira árvore da Figura 10.17. Note que essa explicação elimina conceitos irrelevantes como cor(obj1, vermelho) dos dados de treinamento e captura daqueles aspectos do exemplo que se sabe que são relevantes para o alvo.

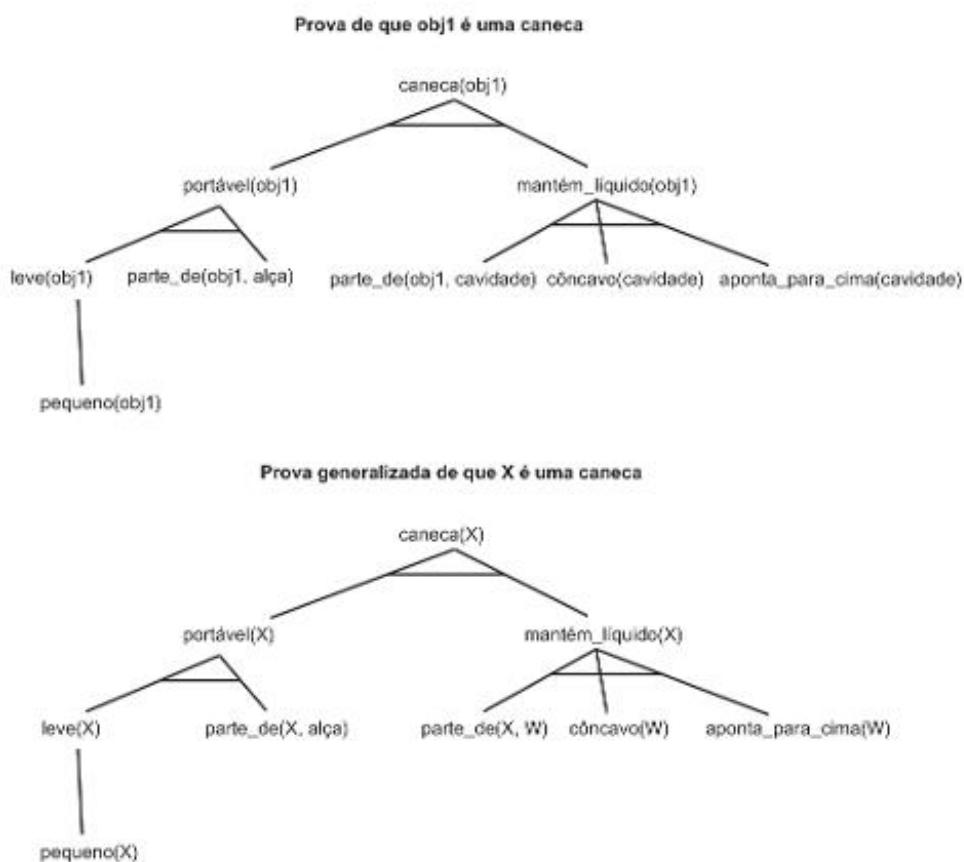
O estágio seguinte do aprendizado baseado em explicações generaliza a explicação para produzir uma definição de conceito que possa ser usada para reconhecer outras canecas. O ABE realiza isso substituindo por variáveis aquelas constantes na árvore de prova que dependam apenas do exemplo de treinamento particular, como na Figura 10.17. Com base na árvore generalizada, o ABE define uma nova regra, cuja conclusão é a raiz da árvore e cuja premissa é a conjunção das folhas:

$$\text{pequeno}(X) \wedge \text{parte\_de}(X, \text{alça}) \wedge \text{parte\_de}(X, W) \wedge \text{cônico}(W) \wedge \text{aponta\_para\_cima}(W) \rightarrow \text{caneca}(X).$$

Ao construirmos uma árvore de prova generalizada, nosso objetivo é substituir por variáveis aquelas constantes que fazem parte do exemplo de treinamento, ao mesmo tempo em que mantemos aquelas constantes e as restrições que são parte da teoria do domínio. Nesse exemplo, a constante alça é originada da teoria do domínio, e não do exemplo de treinamento. Nós a mantivemos como uma restrição essencial na regra adquirida.

Podemos construir uma árvore de prova generalizada de várias maneiras, usando um exemplo de treinamento como guia. Mitchell et al. (1986) fizeram isso construindo, inicialmente, uma árvore de prova que é específica para o exemplo de treinamento e, subsequentemente, generalizando a prova por meio de um processo chamado de

Figura 10.17 Provas específica e genérica de que um objeto (X) é uma caneca.



*regressão ao objetivo.* A regressão ao objetivo casa o objetivo generalizado (no nosso exemplo, `caneca(X)`) com a raiz da árvore de prova, substituindo constantes por variáveis conforme exigido pelo casamento. O algoritmo aplica essas substituições recursivamente ao longo da árvore até que todas as constantes apropriadas tenham sido generalizadas. Veja Mitchell et al. (1986) para uma descrição detalhada desse processo.

DeJong e Mooney (1986) propõem uma abordagem alternativa que, essencialmente, constrói em paralelo as árvores generalizada e específica. Isso é feito mantendo uma variação da árvore de prova que consiste em regras usadas para provar o objetivo distintamente das substituições das variáveis usadas na prova real. Isso é chamado de *estrutura de explicação*, como vemos na Figura 10.18, e representa a estrutura abstrata da prova. O programa de aprendizado mantém duas listas de substituições distintas para a estrutura de explicação: uma lista das substituições específicas, requeridas para explicar o exemplo de treinamento, e uma lista de substituições genéricas, requeridas para explicar o objetivo generalizado. Essas listas são construídas conforme o programa constrói a estrutura de explicação.

Construímos as listas das substituições gerais e específicas da seguinte forma: sejam  $s_s$  e  $s_g$  as listas de substituições específica e genérica, respectivamente. Para cada casamento entre duas expressões  $e_1$  e  $e_2$  na estrutura de explicação, atualizar  $s_s$  e  $s_g$  de acordo com a seguinte regra:

se  $e_1$  está na premissa de uma regra do domínio e  $e_2$  é a conclusão de uma regra do domínio  
então inicio

$$T_s = \text{o unificador mais geral de } e_1 s_s \text{ e } e_2 s_s$$

$$s_s = s_s T_s$$

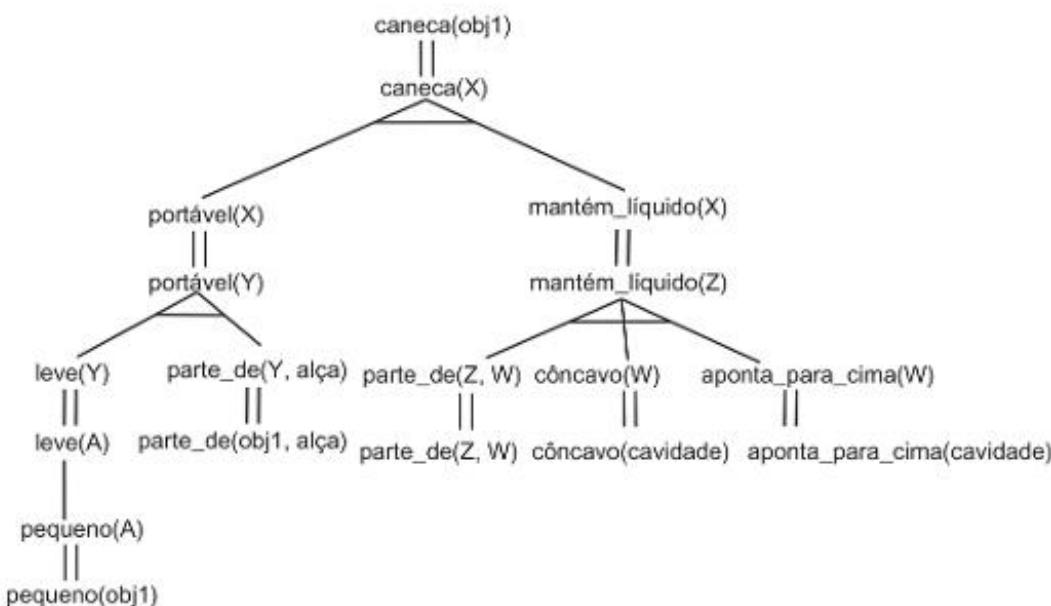
$$T_g = \text{o unificador mais geral de } e_1 s_g \text{ e } e_2 s_g$$

% unifica  $e_1$  e  $e_2$  sob  $s_s$

% atualiza  $s_s$  compondo-o com  $T_s$

% unifica  $e_1$  e  $e_2$  sob  $s_g$

**Figura 10.18** Uma estrutura de explicação do exemplo da caneca.



$$s_g = s_g T_g$$

fim

% atualiza  $s_g$  compondo-o com  $T_g$

se  $e_1$  está na premissa de uma regra do domínio e  $e_2$  é um fato do exemplo de treinamento

então início

% atualiza apenas  $s_s$

$T_s$  = o unificador mais geral de  $e_1 s_s$  e  $e_2 s_s$

% unifica  $e_1$  e  $e_2$  sob  $s_s$

$s_s = s_s T_s$

% atualiza  $s_s$ , compondo-o com  $T_s$

fim

No exemplo da Figura 10.18:

$$s_s = \{obj1/X, obj1/Y, obj1/A, obj1/Z, cavidade/W\}$$

$$s_g = \{X/Y, X/A, X/Z\}$$

Aplicando essas substituições à estrutura de explicação da Figura 10.18, temos como resultado as árvores de prova específica e genérica da Figura 10.17.

O aprendizado baseado em explicação oferece uma série de benefícios:

1. Os exemplos de treinamento frequentemente contêm informação irrelevantes, como a cor da caneca no exemplo anterior. A teoria do domínio possibilita que o programa de aprendizado selecione os aspectos relevantes do exemplo de treinamento.
2. Um dado exemplo pode possibilitar numerosas generalizações possíveis, a maioria das quais é inútil, sem sentido ou, então, errada. O ABE forma generalizações sabidamente relevantes para os objetivos específicos e que são logicamente consistentes com a teoria do domínio.
3. Pelo uso de conhecimento do domínio, o ABE permite aprender a partir de um único exemplo.
4. A construção de uma explicação possibilita que o programa de aprendizado faça hipóteses sobre relacionamentos não declarados entre os seus objetivos e a sua experiência, tais como deduzir uma definição de caneca com base nas suas propriedades estruturais.

O ABE foi aplicado a uma série de problemas de aprendizado. Por exemplo, Mitchell et al. (1983) discutem a adição de ABE ao algoritmo LEX. Suponha que o primeiro exemplo positivo de uso de OPI esteja na solução do exemplo  $\int 7x^2 dx$ . O LEX tornará esse exemplo um membro de  $S$ , o conjunto das generalizações maximamente

específicas. Entretanto, uma pessoa reconheceria imediatamente que as técnicas usadas para resolver esse exemplo não dependem dos valores específicos do coeficiente e do expoente, mas que elas funcionariam também para qualquer valor real desde que o expoente não seja igual a  $-1$ . O programa de aprendizado poderia justificadamente inferir que OP1 deveria ser aplicado a qualquer ocorrência da forma  $\int r_1 x^{(r_2-1)} dx$ , onde  $r_1$  e  $r_2$  são números reais quaisquer. LEX foi estendido para usar seu conhecimento de álgebra com aprendizado baseado em explicação para fazer esse tipo de generalização. Por fim, implementamos um algoritmo de aprendizado baseado em explicação em Prolog na Sala Virtual deste livro.

### 10.5.3 ABE e aprendizado em nível de conhecimento

Embora o ABE seja uma formulação elegante do papel do conhecimento em aprendizado, ele levanta uma série de questões importantes. Uma das mais óbvias diz respeito à questão sobre o que um programa de aprendizado baseado em explicação aprende. O ABE puro pode apenas aprender regras contidas no *fechamento dedutivo* da sua teoria. Isso significa que as regras aprendidas poderiam ter sido inferidas da base de conhecimento sem usar qualquer exemplo de treinamento. A única função do exemplo de treinamento é focar o provador de teoremas em aspectos relevantes do domínio do problema. Consequentemente, o ABE é visto frequentemente como uma forma de *aprendizado acelerado* ou reformulação da base de conhecimento; o ABE pode fazer com que o aprendizado ocorra mais rapidamente porque não precisa reconstruir a árvore de prova subjacente à nova regra. Porém, como ele poderia sempre ter reconstruído a prova, acaba não permitindo que o programa de aprendizado faça coisas novas. Essa distinção foi formalizada por Dietterich na sua discussão do *aprendizado em nível de conhecimento* (1986).

O ABE extrai informação implícita em um conjunto de regras e a torna explícita. Por exemplo, considere o jogo de xadrez: um conhecimento mínimo das regras, quando acoplado com a capacidade de realizar previsões ilimitadas sobre estados de tabuleiro, permitiria que um computador jogasse extremamente bem. Infelizmente, xadrez é muito complexo para essa abordagem. Um aprendizado baseado em explicação que pudesse dominar estratégias de xadrez aprenderia realmente algo novo, para todos os efeitos práticos.

O ABE nos permite, também, abandonar a exigência de que o programa de aprendizado tenha uma teoria completa e correta do domínio e focar em técnicas para refinar teorias incompletas dentro do seu contexto. Aqui, o programa constrói uma árvore de solução parcial. Aqueles ramos da prova que não podem ser completados indicam deficiências na teoria. Uma série de questões interessantes ainda deve ser examinada nessa área. Aí se incluem: o desenvolvimento de heurísticas para raciocinar com teorias imperfeitas, metodologias de atribuição de crédito e a escolha de uma entre várias provas malsucedidas que deve ser reparada.

Outro uso para o aprendizado baseado em explicação é a sua integração com abordagens de aprendizado baseado em similaridade. Aqui, novamente, uma série de esquemas básicos pode ser empregada, como o uso de ABE para refinar dados de treinamento em que a teoria se aplique, passando, então, esses dados parcialmente generalizados para um programa de aprendizado baseado em similaridade para continuar a generalização. Como alternativa, poderíamos usar explicações malsucedidas como um meio de identificar deficiências em uma teoria e, a partir disso, guiar a coleta de dados para um programa de aprendizado baseado em similaridade.

Outras questões nas pesquisas em ABE incluem técnicas para raciocinar com teorias inconsistentes, alternativas à prova de teoremas como um meio para construir explicações, métodos de lidar com dados de treinamento ruidosos e faltantes e métodos para determinar quais regras geradas devem ser salvas.

### 10.5.4 Raciocínio analógico

Enquanto o ABE “puro” é limitado ao aprendizado dedutivo, o uso de analogia oferece um método mais flexível de usar o conhecimento existente. O raciocínio analógico supõe que, se soubermos que duas situações são similares em alguns aspectos, é provável que elas sejam similares em outros aspectos. Por exemplo, se duas casas têm localizações, tipos de construção e condições similares, então elas provavelmente têm o mesmo valor de venda.

Diferentemente das provas usadas em ABE, a analogia não é consistente logicamente. Nesse sentido, ela se assemelha à indução. Como Russell (1989) e outros observaram, a analogia é uma espécie de indução de um único exemplo: no nosso exemplo das casas, induzimos propriedades de uma casa a partir do que se sabe sobre outra casa.

Como discutido na apresentação do raciocínio baseado em casos (Seção 7.3), a analogia é muito útil para se aplicar conhecimento existente a novas situações. Por exemplo, suponha que um estudante esteja tentando aprender sobre o comportamento da eletricidade e que o professor lhe diga que eletricidade é semelhante à água, com a voltagem correspondendo à pressão, a amperagem correspondendo à quantidade de fluxo, e a resistência correspondendo à capacidade de um cano. Usando o raciocínio analógico, o estudante pode compreender mais facilmente conceitos como a lei de Ohm.

O modelo computacional padrão de analogia define a *fonte* de uma analogia como uma solução do problema, um exemplo ou uma teoria que é relativamente bem entendida. O *alvo* não é completamente entendido. A analogia constrói um *mapeamento* entre elementos correspondentes do alvo e da fonte. As inferências analógicas estendem esse mapeamento a novos elementos do domínio-alvo. Continuando com a analogia “eletricidade é como água”, se sabemos que essa analogia mapeia chaves para válvulas, amperagem para a quantidade de fluxo, e voltagem para a pressão de água, podemos inferir logicamente que deve haver alguma analogia para a capacidade de um cano de água (isto é, a sua área transversal); isso pode levar a um entendimento da resistência elétrica.

Vários autores propuseram um arcabouço unificador para modelos computacionais de raciocínio analógico (Hall, 1989; Kedar-Cabelli, 1988; Wolstencroft, 1989). Um arcabouço típico consiste nos seguintes estágios:

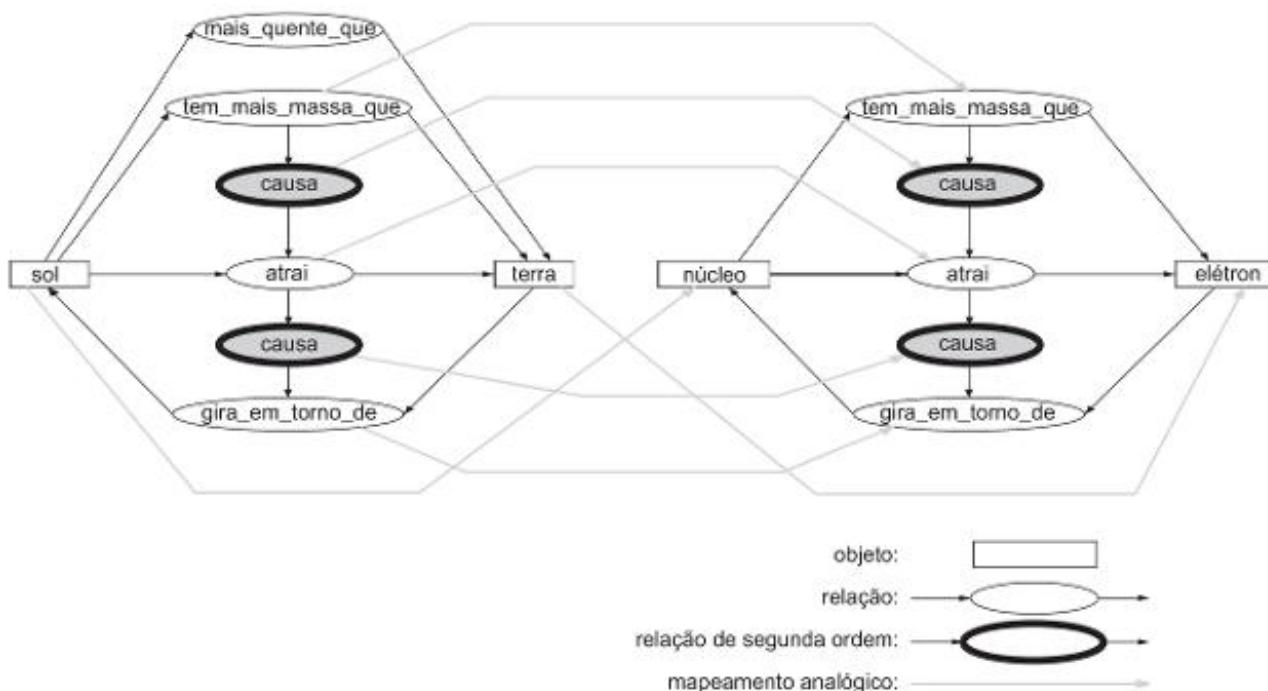
1. *Recuperação*. Dado um problema-alvo, é necessário selecionar uma fonte de analogia em potencial. Problemas em recuperação analógica incluem selecionar aquelas características do alvo e da fonte que aumentam a probabilidade de recuperar uma fonte analógica útil e indexar o conhecimento de acordo com essas características. Geralmente, a recuperação estabelece os elementos iniciais de um mapeamento analógico.
2. *Elaboração*. Uma vez que a fonte tenha sido recuperada, muitas vezes é necessário derivar características e relações adicionais da fonte. Pode ser necessário, por exemplo, desenvolver um rastreamento (ou explicação) de uma solução específica de um problema no domínio de origem como base para a analogia com o alvo.
3. *Mapeamento e inferência*. Esse estágio envolve desenvolver o mapeamento dos atributos da fonte para o domínio-alvo. Isso envolve tanto as similaridades conhecidas quanto as inferências analógicas.
4. *Justificativa*. Aqui, determinamos que o mapeamento é realmente válido. Esse estágio pode requerer modificação do mapeamento.
5. *Aprendizado*. Nesse estágio, o conhecimento adquirido é armazenado em uma forma que será útil no futuro.

Esses estágios foram desenvolvidos numa série de modelos computacionais de raciocínio analógico. Por exemplo, a *teoria do mapeamento estrutural* (Falkenhainer, 1990; Falkenhainer et al., 1989; Gentner, 1983) não apenas trata do problema de construir analogias úteis, mas também fornece um modelo plausível de como os seres humanos entendem analogias. Uma questão central no uso de analogia é como podemos distinguir analogias expressivas e profundas de comparações mais superficiais. Gentner argumenta que as analogias verdadeiras devem enfatizar características estruturais, sistemáticas de um domínio, em vez de similaridades mais superficiais. Por exemplo, a analogia “o átomo é como o sistema solar” é mais profunda que “o girassol é como o sol”, porque a primeira captura um sistema completo de relações causais entre corpos orbitais, enquanto a última descreve similaridades superficiais, como o fato de que os girassóis e o sol são redondos e amarelos. Essa propriedade do mapeamento analógico é chamada de *sistematicidade*.

O mapeamento estrutural formaliza essa intuição. Considere o exemplo da analogia átomo/sistema solar, como mostrado na Figura 10.19, conforme explicado por Gentner (1983). O domínio-fonte inclui os predicados:

```
amarelo(sol)
azul(terra)
mais_quente_que(sol, terra)
causa(tem_mais_massa_que(sol, terra), atrai(sol, terra))
causa(atraí(sol, terra), gira_em_torno_de(terra, sol))
```

Figura 10.19 Um mapeamento analógico.



O domínio-alvo que a analogia procura explicar inclui:

`tem_mais_massa_que(núcleo, elétron)`  
`gira_em_torno_de(elétron, núcleo)`

O mapeamento estrutural procura transferir a estrutura causal da fonte para o alvo. O mapeamento é restrito pelas seguintes regras:

1. Retiram-se propriedades da fonte. Como a analogia favorece sistemas de relações, o primeiro estágio é eliminar aqueles predicados que descrevem propriedades superficiais da fonte. O mapeamento estrutural formaliza isso eliminando predicados de um argumento único (predicados unários) da fonte. A razão para isso é que predicados de alta aridade, em virtude de descrever um relacionamento entre duas ou mais entidades, têm maior probabilidade de capturar as relações sistemáticas desejadas pela analogia. No nosso exemplo, isso elimina asserções como `amarelo(sol)` e `azul(terra)`. Note que a fonte pode conter ainda asserções, como `mais_quente_que(sol, terra)`, que não são relevantes para a analogia.
2. As relações são mapeadas sem modificações da fonte para o alvo; os argumentos das relações podem ser diferentes. No nosso exemplo, relações como `gira_em_torno_de` e `tem_mais_massa_que` são as mesmas tanto na fonte como no alvo. Essa restrição é usada por muitas teorias e reduz muito o número de mapeamentos possíveis. Ela também é consistente com a heurística de dar preferência a relações no mapeamento.
3. Ao construir o mapeamento, as relações de ordem mais alta são preferidas como foco do mapeamento. No nosso exemplo, `causa` é uma relação de ordem mais alta, porque ela envolve outras relações como seus argumentos. Isso é chamado de *princípio da sistematicidade*.

Essas restrições produzem o mapeamento:

`sol → núcleo`  
`terra → elétron`

A extensão do mapeamento resulta na inferência:

```
causa(tem_mais_massa_que(núcleo, elétron), atrai(núcleo, elétron))  
causa(atraí(núcleo, elétron), gira_em_torno_de(elétron, núcleo))
```

A teoria do mapeamento estrutural foi implementada e testada em vários domínios. Embora ela esteja longe de ser uma teoria completa de analogia, não tratando de problemas como a evocação de analogia de fonte, ela se mostrou tanto computacionalmente prática como capaz de explicar muitos aspectos do raciocínio analógico humano. Finalmente, como observamos na nossa apresentação sobre *raciocínio baseado em casos* (Seção 7.3), a analogia tem um papel essencial ao criar e aplicar uma base de casos útil.

## 10.6 Aprendizado não supervisionado

Os algoritmos de aprendizado discutidos até agora implementam formas de *aprendizado supervisionado*. Eles assumem a existência de um professor, uma medida de adequação ou outro método externo de classificação de exemplos de treinamento. O *aprendizado não supervisionado* elimina o professor e requer que o próprio algoritmo de aprendizado avalie os conceitos. A ciência talvez seja o melhor exemplo de aprendizado não supervisionado em seres humanos. Os cientistas não têm o benefício de um professor. Em vez disso, eles propõem hipóteses para explicar observações; avaliam as suas hipóteses usando critérios como simplicidade, generalidade e elegância; e testam hipóteses por meio de experimentos que eles mesmos concebem.

### 10.6.1 Descoberta e aprendizado não supervisionado

O programa AM (Davis e Lenat, 1982; Lenat e Brown, 1984) é um dos primeiros e mais bem-sucedidos programas de descoberta, que derivou uma série de conceitos matemáticos interessantes, mesmo que não originais. Ele inicia com os conceitos da teoria dos conjuntos, operações para criar conhecimento novo por modificação e combinação de conceitos existentes, além de um conjunto de heurísticas para detectar conceitos “interessantes”. Buscando esse espaço de conceitos matemáticos, o AM descobriu os números naturais com vários conceitos importantes da teoria dos números, como a existência dos números primos.

Por exemplo, o AM descobriu os números naturais modificando a sua noção de “pacotes”. Um pacote é uma generalização de um conjunto que permite múltiplas ocorrências de um mesmo elemento. Por exemplo, {a, a, b, c, c} é um pacote. Especializando a definição de pacote para permitir apenas elementos de um tipo, o AM descobriu uma analogia dos números naturais. Por exemplo, o pacote {1, 1, 1, 1} corresponde ao número 4. A união de pacotes levou à noção de adição: {1, 1}  $\cup$  {1, 1} = {1, 1, 1, 1}, ou  $2 + 2 = 4$ . Continuando a explorar esses conceitos, o AM descobriu que a multiplicação é uma série de adições. Usando uma heurística que define novos operadores pela inversão de operadores existentes, ele descobriu a divisão inteira. Além disso, descobriu o conceito de números primos notando que certos números tinham exatamente dois divisores (eles mesmos e 1).

Ao criar um conceito novo, o AM o avalia de acordo com uma série de heurísticas, mantendo aqueles conceitos que se mostram “interessantes”. O AM determinou que os números primos eram interessantes por causa da frequência com que eles ocorriam. Ao avaliar conceitos usando essa heurística, o AM gera exemplos do conceito, testando cada um para verificar se o conceito é válido. Se um conceito é verdadeiro para todos os exemplos, ele é uma tautologia, e o AM lhe dá uma avaliação baixa. Da mesma forma, ele rejeita conceitos que não são verdadeiros para nenhum exemplo. Se um conceito é verdadeiro para uma porção significativa dos exemplos (como é o caso com os números primos), ele o avalia como interessante e o seleciona para continuar as modificações.

Embora o AM tenha descoberto os números primos e vários outros conceitos interessantes, ele não conseguiu progredir muito além da teoria elementar dos números. Em uma análise posterior desse trabalho, Lenat e Brown (1984) examinaram as razões para o sucesso do programa e suas limitações. Embora originalmente Lenat tenha acreditado que as heurísticas de AM fossem a fonte primordial do seu poder, essa avaliação posterior atribuiu

muito do sucesso do programa à linguagem usada para representar os conceitos matemáticos. O AM representava conceitos como estruturas recursivas em uma variação da linguagem de programação Lisp. Por estar fundamentada em uma linguagem de programação bem concebida, essa representação definia um espaço que continha uma alta densidade de conceitos interessantes. Isso era particularmente verdadeiro nos estágios iniciais da busca. Conforme a exploração continuava, o espaço crescia de forma combinatória e a porcentagem de conceitos interessantes “minguava”. Essa observação reforça ainda mais a relação entre representação e busca.

Outra razão para o AM não conseguir manter o passo impressionante de suas primeiras descobertas é a sua incapacidade de “aprender a aprender”. Ele não adquiria novas heurísticas conforme ganhava conhecimento matemático; consequentemente, a qualidade da sua busca degradava conforme a sua matemática se tornava mais complexa. Nesse sentido, AM nunca desenvolvia um entendimento profundo da matemática. Lenat tratou desse problema em um trabalho posterior com um programa chamado EURISKO, que procurava aprender novas heurísticas (Lenat, 1983).

Vários programas continuaram a explorar os problemas da descoberta automática. IL (Sims, 1987) aplica uma variedade de técnicas de aprendizado à descoberta matemática, incluindo métodos como a prova de teoremas e o aprendizado baseado em explicações (Seção 10.5). Veja também a invenção automática de sequências de inteiros em Cotton et al. (2000).

BACON (Langley et al., 1986, 1987) desenvolveu modelos computacionais da formação de leis científicas quantitativas. Por exemplo, usando dados que relacionavam as distâncias dos planetas ao sol e o período das órbitas dos planetas, BACON “redescobriu” as leis de Kepler do movimento planetário. Por representar um modelo computacional plausível de como os seres humanos podem ter realizado descobertas em uma série de domínios, BACON forneceu uma ferramenta e uma metodologia úteis para examinar o processo de descoberta científica humana. SCAVENGER (Stubblefield, 1995; Stubblefield e Luger, 1996) usou uma variação do algoritmo ID3 para melhorar a sua capacidade de formar analogias úteis. Shrager e Langley (1990) descrevem uma série de outros sistemas de descoberta.

Embora a descoberta científica seja uma área de pesquisa importante, o progresso atual tem sido modesto. Um problema mais básico e talvez mais proveitoso em aprendizado não supervisionado diz respeito à descoberta de categorias. Lakoff (1987) sugere que a categorização é fundamental à cognição humana: o conhecimento teórico de alto nível depende da capacidade de organizar as particularidades da nossa experiência em taxonomias coerentes. A maior parte de nosso conhecimento útil é sobre categorias de objetos, como vacas, em vez de ser sobre vacas individuais específicas, como a Mimosa ou a Ferdinand. Nordhausen e Langley enfatizaram a formação de categorias como a base para uma teoria unificada da descoberta científica (Nordhausen e Langley, 1990). Ao desenvolver explicações de por que substâncias químicas reagem da forma que fazem, a química se apoiou em trabalhos prévios sobre classificação de compostos em categorias como “ácidos” e “alcalinos”.

Na próxima seção, examinamos o *agrupamento conceitual*, que é o problema de descobrir categorias úteis em dados não classificados.

### 10.6.2 Agrupamento conceitual

O problema de agrupamento começa com uma coleção de objetos não classificados e um meio de medir a similaridade de objetos. O objetivo é organizar os objetos em classes que satisfazem algum padrão de qualidade, como a maximização da similaridade de objetos da mesma classe.

A *taxonomia numérica* é uma das mais antigas abordagens para o problema de agrupamento. Os métodos numéricos se baseiam na representação de objetos como uma coleção de características, cada uma podendo ter um valor numérico. Uma métrica de similaridade aceitável trata cada objeto (um vetor com  $n$  valores de características) como um ponto no espaço  $n$ -dimensional. A semelhança de dois objetos é a distância euclidiana entre eles nesse espaço.

Usando essa métrica de similaridade, um algoritmo de agrupamento comum constrói agrupamentos de uma forma de baixo para cima. Essa abordagem, normalmente chamada de estratégia de *agrupamento aglomerante*, forma categorias da seguinte maneira:

1. Examinar todos os pares de objetos, selecionando o par com o maior grau de similaridade, tornando-o um agrupamento.
2. Definir as características do agrupamento como uma função, tal como a média, das características dos membros componentes e, então, substituir os objetos componentes por essa definição do agrupamento.
3. Repetir esse processo sobre a coleção de objetos até que todos eles tenham sido reduzidos a um único agrupamento.
4. Muitos algoritmos de aprendizado não supervisionado podem ser vistos como realizando *estimativas de densidade da máxima verossimilhança*, o que significa encontrar uma distribuição da qual os dados pudessem ser retirados mais provavelmente. Um exemplo é a interpretação de um conjunto de fonemas em uma aplicação de linguagem natural (veja o Capítulo 15).

O resultado desse algoritmo é uma árvore binária cujos nós folhas são ocorrências e cujos nós internos são agrupamentos de tamanho crescente.

Podemos estender esse algoritmo para objetos representados como conjuntos de características simbólicas, em vez de numéricas. O único problema é medir a similaridade de objetos definidos usando valores simbólicos, em vez de numéricos. Uma abordagem razoável define a similaridade de dois objetos como a proporção de características que eles têm em comum. Dados os objetos:

```
objeto1 = {pequeno, vermelho, borracha, bola}  
objeto2 = {pequeno, azul, borracha, bola}  
objeto3 = {grande, preto, madeira, bola}
```

essa métrica calcularia os seguintes valores de similaridade:

$$\begin{aligned}\text{similaridade}(\text{objeto1}, \text{objeto2}) &= 3/4 \\ \text{similaridade}(\text{objeto1}, \text{objeto3}) &= \text{similaridade}(\text{objeto2}, \text{objeto3}) = 1/4\end{aligned}$$

Entretanto, os algoritmos de agrupamento baseados em similaridade não captam adequadamente o papel subjacente do conhecimento semântico na formação do agrupamento. Por exemplo, constelações são descritas tanto com base na sua proximidade no céu quanto por conceitos humanos como a “Ursa Maior”.

Ao definir categorias, não podemos dar pesos iguais a todas as características. Em qualquer contexto dado, algumas das características de um objeto são mais importantes que outras; métricas de similaridade simples tratam todas as características igualmente. As categorias humanas dependem muito mais dos objetivos da categorização e do conhecimento prévio do domínio do que da similaridade superficial. Considere, por exemplo, a classificação de baleias como mamíferos, em vez de peixes. Similaridades superficiais não são capazes de determinar essa classificação, que depende de objetivos mais amplos da classificação biológica e de extensas evidências fisiológicas e evolucionárias.

Algoritmos de agrupamento tradicionais não apenas não levam em consideração objetivos e conhecimento prévio, como também não conseguem produzir explicações semânticas significativas das categorias resultantes. Esses algoritmos representam agrupamentos *extensionalmente*, ou seja, por enumeração de todos os seus membros. Os algoritmos não produzem definições *intencionais*, ou seja, regras gerais que definem a semântica da categoria e que podem ser usadas para classificar tanto os membros conhecidos como futuros membros da categoria. Por exemplo, uma definição extensional do conjunto de pessoas que serviram como secretário-geral das Nações Unidas listaria simplesmente esses indivíduos. Uma definição intencional, como:

$\{X \mid X \text{ tenha sido eleito secretário-geral das Nações Unidas}\}$

teria os benefícios adicionais de definir a classe semanticamente e de nos permitir reconhecer futuros membros da categoria.

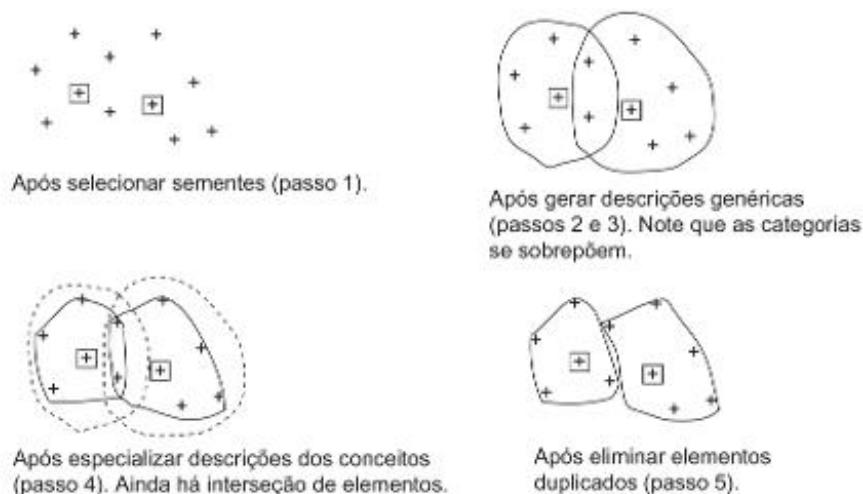
O *agrupamento conceitual* aborda esses problemas usando técnicas de aprendizado de máquina para produzir definições gerais de conceitos e aplicando conhecimento prévio à formação de categorias. O CLUSTER/2 (Michalski e Stepp, 1983) é um bom exemplo dessa abordagem. Ele usa conhecimento prévio na forma de vieses na linguagem usada para representar categorias.

O CLUSTER/2 forma  $k$  categorias construindo indivíduos em torno de  $k$  objetos *sementes*.  $k$  é um parâmetro que pode ser ajustado pelo usuário. O CLUSTER/2 avalia os agrupamentos resultantes, selecionando novas sementes e repetindo o processo até que os seus critérios de qualidade sejam satisfeitos. O algoritmo é definido:

1. Selecionar  $k$  sementes do conjunto de objetos observados. Isso pode ser feito aleatoriamente ou de acordo com uma função de seleção.
2. Para cada semente, usando a semente como um exemplo positivo e todas as outras sementes como exemplos negativos, produzir uma definição maximamente genérica que cubra todos os exemplos positivos e nenhum exemplo negativo. Note que isso pode levar a múltiplas classificações de outros objetos ainda não vistos.
3. Classificar todos os objetos da amostra de acordo com essas descrições. Substituir cada descrição maximamente genérica por uma descrição maximamente específica que cubra todos os objetos da categoria. Isso diminui a probabilidade de que as classes se sobreponham para objetos não vistos.
4. As classes podem ainda se sobrepor em objetos dados. O CLUSTER/2 inclui um algoritmo para ajustar definições que se sobreponhem.
5. Usando uma métrica de distância, selecionar um elemento o mais perto possível do centro de cada classe. A métrica de distância poderia ser semelhante à métrica de similaridade discutida anteriormente.
6. Usando esses elementos centrais como novas sementes, repetir os passos 1 a 5. Parar quando os agrupamentos forem satisfatórios. Uma métrica de qualidade típica é a complexidade das descrições genéricas de classes. Por exemplo, uma variação da navalha de Occam daria preferência a agrupamentos que produzem definições sintaticamente mais simples, tais como aquelas com um pequeno número de conjuntivos.
7. Se os agrupamentos não forem satisfatórios e não houver melhoria ao longo de várias iterações, selecionar novas sementes que estejam mais próximas da fronteira do agrupamento, em vez de estarem próximas ao centro.

A Figura 10.20 mostra os estágios de uma execução de CLUSTER/2.

**Figura 10.20** Os passos de uma rodada de CLUSTER/2.



### 10.6.3 COBWEB e a estrutura de conhecimento taxonômico

Muitos algoritmos de agrupamento, bem como muitos algoritmos de aprendizado supervisionado como o ID3, definem as categorias em termos das condições necessárias e suficientes para a pertinência à categoria. Essas condições formam um conjunto de propriedades que todos os membros de uma categoria, e apenas dessa categoria, pos-

suem. Embora muitas categorias, tais como o conjunto de todos os delegados das Nações Unidas, possam ser definidas dessa forma, as categorias humanas nem sempre se enquadram nesse modelo. De fato, a categorização humana é caracterizada por uma maior flexibilidade e por uma estrutura mais rica do que a que foi examinada até agora.

Por exemplo, se as categorias humanas fossem realmente definidas por condições necessárias e suficientes para a sua pertinência, não poderíamos distinguir graus de pertinência à categoria. Entretanto, psicólogos notaram um forte senso de prototipagem na categorização humana (Rosch, 1978). Por exemplo, geralmente consideramos um pardal como um exemplo mais apropriado de ave do que uma galinha; um carvalho é um exemplo mais típico de árvore do que uma palmeira (principalmente no hemisfério norte).

A *teoria da semelhança familiar* (Wittgenstein, 1953) dá suporte a essas noções de prototipagem argumentando que as categorias são definidas por sistemas complexos de semelhanças entre membros, em vez de o ser por condições necessárias e suficientes para pertinência. Tais categorias podem não ter nenhuma propriedade compartilhada por todos os seus membros. Wittgenstein cita o exemplo de jogos: nem todos requerem dois ou mais jogadores, como paciência; nem todos são agradáveis para os jogadores, como pedra, papel e tesoura; nem todos têm regras bem articuladas, como os jogos de “faz de conta” das crianças; e nem todos os jogos envolvem competição, como pular corda. Entretanto, consideramos a categoria como bem definida e não ambígua.

As categorias humanas também diferem das hierarquias de herança mais formais, uma vez que nem todos os níveis das taxonomias humanas são igualmente importantes. Psicólogos (Rosch, 1978) demonstraram a existência de categorias de *nível básico*. A categoria de nível básico é a classificação mais comumente usada para descrever objetos, a primeira terminologia que as crianças aprendem e o nível que, de alguma forma, captura a classificação mais fundamental de um objeto. Por exemplo, a categoria “cadeira” é mais básica que as suas generalizações, como “móvel”, ou que suas especializações, como “cadeira de escritório”. “Carro” é mais básico que “sedã” ou “veículo”.

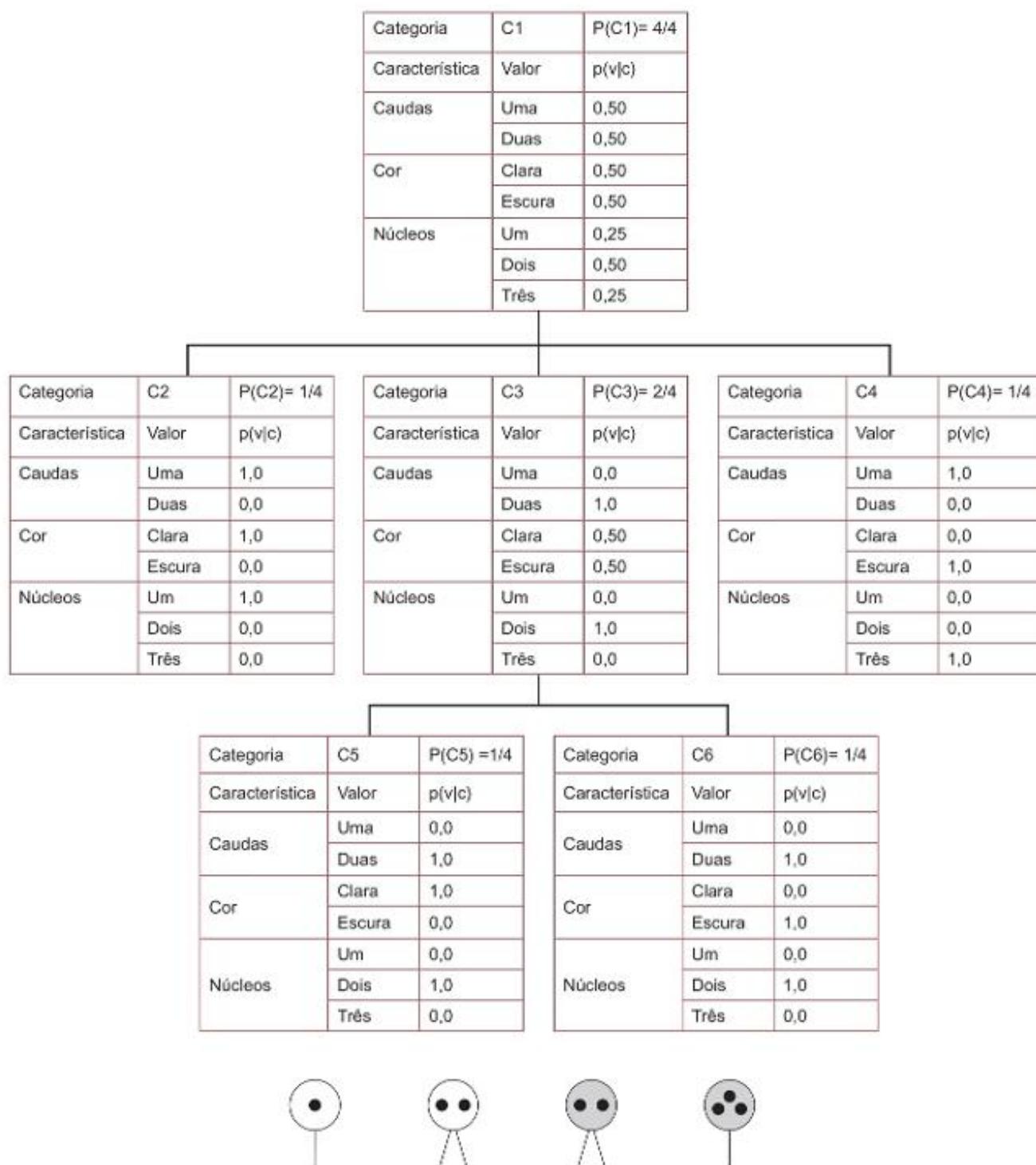
Os métodos comuns para representar pertinência a classes e hierarquias, como a lógica, sistemas de herança, vetores de características ou árvores de decisão, não consideram esses efeitos. Entretanto, isso não é apenas importante para os cientistas cognitivos, cujo objetivo é a compreensão da inteligência humana; isso também é valioso para a engenharia de aplicações úteis de IA. Os usuários avaliam um programa em termos de sua flexibilidade, sua robustez e sua habilidade em se comportar, de modo que pareça racional para os padrões humanos. Embora não exijamos que os algoritmos de IA se assemelhem à arquitetura da mente humana, qualquer algoritmo que se propõe a descobrir categorias deve satisfazer expectativas do usuário relativas à estrutura e ao comportamento dessas categorias.

O COBWEB (Fisher, 1987) trata dessas questões. Embora ele não tenha a intenção de ser um modelo de cognição humana, ele trata a categorização de nível básico e os graus de pertinência à categoria. Além disso, o COBWEB aprende de modo incremental: ele não requer que todos os exemplos estejam disponíveis antes de começar o aprendizado. Em muitas aplicações, o aprendiz adquire dados ao longo do tempo. Nessas situações, ele precisa construir descrições de conceitos aplicáveis, a partir de uma coleção inicial de dados, e atualizar essas descrições conforme mais dados se tornem disponíveis. O COBWEB lida, também, com o problema de determinar o número correto de agrupamentos. O CLUSTER/2 produzia um número predeterminado de categorias. Embora o usuário pudesse variar esse número ou o algoritmo pudesse tentar diferentes valores em um esforço para melhorar a categorização, tais abordagens não são particularmente flexíveis. O COBWEB usa métricas de qualidade globais para determinar o número de agrupamentos, a profundidade da hierarquia e a pertinência à categoria de novas ocorrências.

Diferentemente do que foi visto até aqui, o COBWEB representa categorias de forma probabilística. Em vez de definir a pertinência a uma categoria como um conjunto de valores que deve estar presente para cada característica de um objeto, o COBWEB representa a probabilidade com a qual cada valor de característica está presente.  $p(f_i = v_{ij} | c_k)$  é a probabilidade condicional (Seção 5.2) com a qual a característica  $f_i$  tem o valor  $v_{ij}$ , dado que um objeto pertença à categoria  $c_k$ .

A Figura 10.21 ilustra uma taxonomia realizada por COBWEB, conforme Gennari et al. (1989). Nesse exemplo, o algoritmo formou uma categorização dos quatro animais unicelulares mostrados na figura. Cada animal é definido por seu valor para as características: cor, número de caudas e número de núcleos. A categoria C3, por exemplo, tem uma probabilidade de 1,0 de ter 2 caudas, uma probabilidade de 0,5 de ter cor clara e uma probabilidade de 1,0 de ter 2 núcleos.

Como é ilustrado na figura, cada categoria na hierarquia inclui probabilidades de ocorrência de todos os valores de todas as características. Isso é essencial tanto para caracterizar novas ocorrências como para modificar a

**Figura 10.21** Um agrupamento COBWEB para quatro organismos unicelulares, adaptado de Gennari et al. (1989).

estrutura da categoria para melhor se ajustar a novas ocorrências. De fato, sendo um algoritmo incremental, o COBWEB não separa essas ações. Quando o COBWEB recebe uma nova ocorrência, ele considera a qualidade global de colocá-la em uma categoria existente ou modificar a hierarquia para acomodar a ocorrência. O critério que o COBWEB usa para avaliar a qualidade de uma classificação é chamado de *utilidade de categorias* (Gluck e

Corter, 1985). A utilidade de categorias foi desenvolvida no âmbito de pesquisas sobre a categorização humana. Ela leva em conta efeitos de nível básico e outros aspectos da estrutura de categorização humana.

A utilidade de categorias procura maximizar tanto a probabilidade de dois objetos da mesma categoria terem valores em comum como a probabilidade de objetos em diferentes categorias terem valores de propriedades diferentes. A utilidade de categorias é definida como:

$$\sum_k \sum_i \sum_j p(f_i = v_{ij}) p(f_i = v_{ij} | c_k) p(c_k | f_i = v_{ij})$$

Essa soma é feita sobre todas as categorias,  $c_k$ , todas as características,  $f_i$ , e todos os valores de características,  $v_{ij}$ .  $p(f_i = v_{ij} | c_k)$ , chamada de *previsibilidade*, é a probabilidade de um objeto ter o valor  $v_{ij}$  para a característica  $f_i$ , dado que o objeto pertença à categoria  $c_k$ . Quanto maior essa probabilidade, mais provável será que dois objetos compartilhem os mesmos valores de característica em uma categoria.  $p(c_k | f_i = v_{ij})$ , chamada de *previdência*, é a probabilidade com a qual um objeto pertence à categoria  $c_k$ , dado que ele tenha o valor  $v_{ij}$  para a característica  $f_i$ . Quanto maior essa probabilidade, menos provável será que objetos não pertencentes à categoria tenham esses valores de característica.  $p(f_i = v_{ij})$  serve como peso, assegurando que valores de características que ocorram frequentemente exerçam uma influência mais forte na avaliação. Combinando esses valores, uma alta utilidade de categorias indica uma alta probabilidade de que objetos na mesma categoria compartilhem propriedades, ao mesmo tempo em que diminui a probabilidade de objetos em categorias diferentes terem propriedades comuns.

O algoritmo COBWEB é definido por:

```

cobweb(Nó, Ocorrência)
  inicio
    se Nó é uma folha
      então inicio
        criar dois filhos de Nó, L1 e L2;
        fixar as probabilidades de L1 com o valor das probabilidades de Nó;
        inicializar as probabilidades de L2 com o valor das probabilidades de Ocorrência;
        adicionar Ocorrência a Nó, atualizando as probabilidades de Nó;
    fim
    senão inicio
      adicionar Ocorrência a Nó, atualizando as probabilidades de Nó;
      para cada filho C de Nó, calcular a utilidade de categorias do agrupamento
        quando Ocorrência for colocada em C;
        faça S1 ser o escore para a melhor categorização, C1;
        faça S2 ser o escore para a segunda melhor categorização, C2;
        faça S3 ser o escore para colocar a ocorrência em uma nova categoria;
        faça S4 ser o escore para aglutinar C1 e C2 em uma única categoria;
        faça S5 ser o escore para dividir C1 (substituindo-o por suas categorias filhas)
    fim
    Se S1 é o melhor escore
      então cobweb(C1, Ocorrência) % colocar a ocorrência em C1
    senão, se S3 é o melhor escore
      então inicializar as probabilidades da nova categoria com as de Ocorrência
    senão, se S4 é o melhor escore
      então inicio
        faça Cm ser o resultado da aglutinação de C1 e C2;
        cobweb(Cm, Ocorrência)
      fim
    senão, se S5 é o melhor escore
      então inicio

```

```

    dividir C1;
    cobweb(Nó, Ocorrência)
fim;
fim

```

O COBWEB realiza uma busca por subida de encosta no espaço de taxonomias possíveis usando a utilidade de categorias para avaliar e selecionar categorizações possíveis. Ele inicializa a taxonomia em uma única categoria, cujas características são as da primeira ocorrência. Para cada ocorrência subsequente, o algoritmo começa com a categoria raiz e se move ao longo da árvore. Em cada nível, ele usa a utilidade de categorias para avaliar a taxonomia que resulta de:

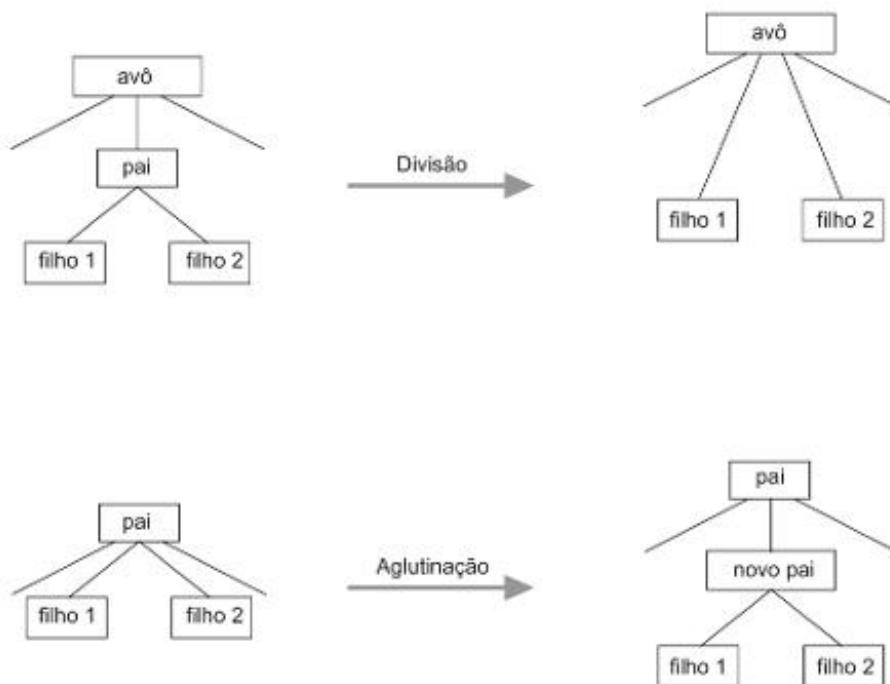
1. Colocar a ocorrência na melhor categoria existente.
2. Adicionar uma nova categoria contendo apenas a ocorrência.
3. Aglutinar duas categorias existentes em uma nova categoria e adicionar a ocorrência nessa categoria.
4. Dividir uma categoria existente em duas e colocar a ocorrência na melhor nova categoria resultante.

A Figura 10.22 ilustra os processos de aglutinação e de divisão de nós. Para aglutinar dois nós, o algoritmo cria um novo e torna os nós existentes filhos desse novo nó. Ele calcula as probabilidades para o novo nó combinando as probabilidades para os filhos. A divisão substitui um nó pelos seus filhos.

Esse algoritmo é eficiente e produz taxonomias com um número razoável de classes. Como ele permite pertinências probabilísticas, as suas categorias são flexíveis e robustas. Além disso, ele apresenta efeitos de categorias de nível básico e, por sua noção de casamento parcial de categorias, suporta noções de prototipagem e graus de pertinência. Em vez de se basear em lógica bivalorada, o COBWEB, assim como a lógica nebulosa, interpreta a “vagueza” da pertinência nas categorias como um componente necessário para aprender e raciocinar de uma forma inteligente e flexível.

Apresentamos, a seguir, o aprendizado por reforço, que, como os sistemas de classificação da Seção 11.2, interpreta a realimentação vinda do ambiente, de modo a aprender conjuntos ótimos de relações condição/resposta para solucionar problemas nesse ambiente.

**Figura 10.22** Aglutinando e dividindo nós.



## 10.7 Aprendizado por reforço

Nós, humanos, aprendemos (normalmente) a partir da interação com o nosso ambiente. Um momento de reflexão, entretanto, nos faz lembrar que a realimentação das nossas ações no mundo não é sempre imediata e direta. Nos relacionamentos humanos, por exemplo, normalmente demora um bom tempo para apreciarmos os resultados de nossas ações. A interação com o mundo nos mostra causas e efeitos (Pearl, 2000), as consequências de nossas ações e como alcançar objetivos complexos. Como agentes inteligentes, fazemos políticas para trabalhar no nosso mundo. “O mundo” é um professor, mas as suas lições são muitas vezes sutis e, algumas vezes, difíceis de aprender!

### 10.7.1 Componentes do aprendizado por reforço

No aprendizado por reforço, concebemos algoritmos computacionais para transformar situações do mundo em ações, de forma a maximizar uma medida de recompensa. O nosso agente não sabe diretamente o que fazer ou qual ação tomar; em vez disso, ele descobre por meio de exploração quais ações oferecem a maior recompensa. As ações do agente afetam não apenas a recompensa imediata, mas também têm impacto sobre ações e eventuais recompensas subsequentes. Essas duas características, busca por tentativa e erro e reforço atrasado, são as mais importantes do aprendizado por reforço. Consequentemente, o aprendizado por reforço é uma metodologia mais geral que o aprendizado visto anteriormente neste capítulo.

O aprendizado por reforço não é definido por métodos de aprendizado particulares, mas sim por ações realizadas no ambiente e por respostas recebidas de um ambiente. Qualquer método de aprendizado que possa tratar essa interação é um método aceitável para aprendizado por reforço. Ele também não é um aprendizado supervisionado, como abordam os capítulos de aprendizado de máquina deste livro. No aprendizado supervisionado, um “professor” usa exemplos para instruir diretamente ou treinar o aprendiz. No aprendizado por reforço, o próprio agente de aprendizado, por tentativa, erro e realimentação, aprende uma política ótima para alcançar objetivos no seu ambiente. (Veja o aprendizado por classificação na Seção 11.2.)

O aprendiz por reforço deve ainda considerar o compromisso entre usar apenas o que ele sabe no momento ou continuar explorando o mundo. Para otimizar as suas possibilidades de recompensa, o agente não deve apenas fazer o que ele já sabe, mas também explorar aquelas partes de seu mundo que ele ainda desconhece. A exploração permite que o agente (possivelmente) faça melhores seleções no futuro; assim, obviamente, o agente que sempre, ou então nunca, explora normalmente não terá sucesso. O agente deve explorar uma série de opções e, ao mesmo tempo, favorecer aquelas que pareçam ser as melhores. Em tarefas com parâmetros estocásticos, as ações exploratórias devem ser realizadas diversas vezes para que se ganhe estimativas confiáveis de recompensas. (Veja os detalhes sobre os aspectos estocásticos do aprendizado por reforço no Capítulo 13.)

Muitos dos algoritmos para solucionar problemas apresentados anteriormente neste livro, incluindo planejadores, tomadores de decisão e algoritmos de busca, podem ser vistos no contexto do aprendizado por reforço. Podemos, por exemplo, criar um plano com um controlador teleorreativo (Seção 7.4) e, então, avaliar o seu sucesso com um algoritmo de aprendizado por reforço. De fato, o algoritmo de reforço DYNA-Q (Sutton, 1990, 1991) integra aprendizado por modelo com planejamento e atuação. Assim, o aprendizado por reforço oferece um método para avaliar planos e modelos e a sua utilidade para realizar tarefas em ambientes complexos.

Introduzimos, agora, uma terminologia para o aprendizado por reforço:

$t$  é um passo de tempo discreto no processo de solução de problemas

$s_t$  é o estado do problema em  $t$ , dependente de  $s_{t-1}$  e  $a_{t-1}$

$a_t$  é a ação em  $t$ , dependente de  $s_t$

$r_t$  é a recompensa em  $t$ , dependente de  $s_{t-1}$  e  $a_{t-1}$

$\pi$  é uma política para realizar uma ação em um estado. Assim,  $\pi$  é um mapeamento de estados em ações

$\pi^*$  é a política ótima

$V$  mapeia um estado para o seu valor. Assim,  $V^\pi(s)$  é o valor do estado  $s$  sob a política  $\pi$

Na Seção 10.7.2, será visto o *aprendizado por diferença temporal*, que aprende um  $V$  para cada  $s$  com  $\pi$  estático.

Há quatro componentes de aprendizado por reforço, uma *política*  $\pi$ , uma *função de recompensa*  $r_t$ , um *mapeamento de valor*  $V$  e, frequentemente, um *modelo* do ambiente. A *política* define as escolhas do agente de aprendizado e o método de agir em qualquer instante de tempo. Assim, a política poderia ser representada por um conjunto de regras de produção ou uma simples tabela de consulta. A política para uma situação particular, como já citado, pode ser também o resultado de uma busca extensiva, da consulta a um modelo ou então de um processo de planejamento. Ela poderia ser também estocástica. A política é o componente crítico do agente de aprendizado, já que ela sozinha é suficiente para produzir comportamento em um dado tempo.

A *função de recompensa*  $r_t$  define as relações entre estado e objetivo do problema no tempo  $t$ . Ela mapeia cada ação ou, mais precisamente, cada par estado-resposta, em uma medida de recompensa, que indica o quanto aquela ação é desejável para atingir o objetivo. O agente no aprendizado por reforço tem a tarefa de maximizar a recompensa total que ele recebe ao realizar a sua tarefa.

A *função de valor*  $V$  é uma propriedade de cada estado do ambiente que indica a recompensa que o sistema pode esperar para ações a partir daquele estado. Enquanto a função de recompensa mede o quanto os pares estado-resposta são desejáveis imediatamente, a função de valor indica o quanto um estado do ambiente é desejável a longo prazo. Um estado obtém seu valor a partir da sua própria qualidade intrínseca, bem como da qualidade dos estados que provavelmente o sigam, isto é, a recompensa por estar naqueles estados. Por exemplo, um estado-ação pode ter baixa recompensa imediata, mas ter um alto valor porque ele é normalmente seguido por outros estados que produzem uma alta recompensa. Um valor baixo pode também indicar estados que não estão associados com caminhos de solução bem-sucedidos.

Sem uma função de recompensa, não há valores, e o único propósito de se estimar valores é para ganhar mais recompensas. Na tomada de decisão, entretanto, o que mais nos interessa são os valores, pois eles indicam estados e combinações de estados que nos dão as mais altas recompensas. Entretanto, é muito mais difícil determinar valores para estados do que determinar recompensas. As recompensas são dadas diretamente pelo ambiente, enquanto os valores são estimados e então reestimados a partir dos sucessos e dos fracassos ao longo do tempo. Na verdade, o aspecto mais crítico e mais difícil do aprendizado por reforço é criar um método para determinar valores de forma eficiente. Demonstramos um método, uma regra para aprendizado por diferença temporal, na Seção 10.7.2.

Um elemento final — e opcional — para o aprendizado por reforço é o *modelo* do ambiente. Um modelo é um mecanismo para capturar aspectos do comportamento do ambiente. Como vimos na Seção 7.3, os modelos podem ser usados não apenas para determinar falhas, como no raciocínio de diagnóstico, mas também como parte da determinação de um plano de ação. Os modelos nos permitem avaliar possíveis ações futuras sem realmente as experimentar. O planejamento baseado em modelo é uma adição recente ao paradigma do aprendizado por reforço, já que os sistemas antigos procuravam criar parâmetros de recompensas e de valor baseados apenas em ações puras de tentativa e erro de um agente.

## 10.7.2 Um exemplo: o jogo da velha modificado

Demonstramos, a seguir, um algoritmo de aprendizado por reforço para o jogo da velha, um problema que já havíamos considerado (Capítulo 4) e que é tratado na literatura de aprendizado por reforço (Sutton e Barto, 1998). É importante comparar e contrastar a abordagem de aprendizado por reforço com outros métodos de solução, por exemplo, o mín-máx.

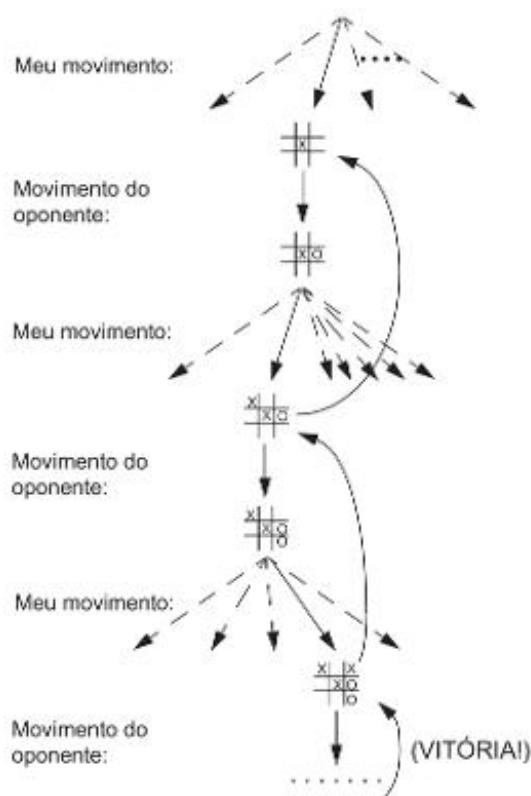
Para relembrá-lo, o jogo da velha é um jogo para duas pessoas sobre uma grade  $3 \times 3$ , como vemos na Figura II.5. Os jogadores, X e O, alternam-se colocando as suas marcas sobre a grade, até que o primeiro jogador que consiga três marcas em uma linha horizontal, vertical ou diagonal, vença. Como se sabe, quando se joga esse jogo usando informações completas e valores armazenados (veja Seção 4.3), ele sempre acaba em empate. Com aprendizado por reforço, seremos capazes de fazer algo bem mais interessante. Mostraremos como podemos capturar o desempenho de um oponente imperfeito e criar uma política que nos permita maximizar nossa vantagem sobre ele. A nossa política pode, também, evoluir, conforme o nosso oponente melhore no jogo, e, com o uso de um modelo, seremos capazes de gerar bifurcações e outros movimentos de ataques!

Primeiro, devemos construir uma tabela de números, um para cada estado possível do jogo. Esses números, os *valores* dos estados, refletirão a estimativa atual da probabilidade de vencer a partir daquele estado. Isso dará suporte a uma política para uma estratégia estritamente vencedora, isto é, ou uma vitória do nosso oponente, ou um empate, será considerada uma perda para nós. Essa abordagem de empate como perda nos permite construir uma política focada na vitória e é diferente do modelo de informação perfeita, vitória-derrota-empate, da Seção 4.3. Essa é uma diferença importante, realmente; capturaremos a habilidade de um oponente real, e não a informação perfeita de um oponente idealizado. Assim, inicializamos a tabela de valores com 1 para cada posição de vitória para nós, 0 para cada configuração de perda, ou empate, e 0,5 para as outras configurações, refletindo a suposição inicial de que temos 50% de chance de vencer a partir desses estados.

Realizamos, agora, vários jogos contra esse oponente. Para simplificar, suponha que sejamos o X e o oponente, o O. A Figura 10.23 reflete uma sequência de movimentos possíveis, tanto os considerados como os escolhidos, dentro de uma situação de jogo. Para gerar um movimento, primeiro consideramos cada estado que é um movimento válido do nosso estado atual, isto é, qualquer estado aberto que podemos reivindicar para X. Consultamos a medida de valor atual para esse estado armazenado na tabela. Na maior parte do tempo, podemos realizar movimentos gulosos, isto é, escolhendo o estado que tem a melhor função de valor. Ocasionalmente, podemos fazer um movimento exploratório e selecionar outro estado aleatoriamente. Esses movimentos exploratórios são pensados para considerar alternativas que talvez não tenhamos visto durante a situação de jogo, expandindo possíveis otimizações de valor.

Enquanto jogamos, mudamos as *funções de valor* dos estados que selecionamos. Procuramos fazer com que os seus últimos valores reflitam melhor a probabilidade de eles estarem em um caminho vencedor. Antes, chamávamos isso de *função de recompensa* de um estado. Para fazer isso, retornamos o valor de um estado que selecio-

**Figura 10.23** Uma sequência de movimentos do jogo da velha. Setas tracejadas indicam escolhas de movimentos possíveis, setas em linhas sólidas para baixo indicam movimentos selecionados, setas sólidas para cima indicam recompensa, quando a função de recompensa muda o valor do estado.



namos como uma função do valor do próximo estado que escolhemos. Como pode ser visto pelas “setas para cima” da Figura 10.23, essa ação de retrocesso pula a escolha de nosso oponente, refletindo, assim, o conjunto de valores que ele nos dirigiu para a nossa próxima escolha de um estado. Assim, o valor atual de um estado que escolhemos anteriormente é ajustado para melhor refletir o valor do estado posterior (e por fim, é claro, do valor de vitória ou derrota). Normalmente, fazemos isso avançando o estado anterior em uma fração da diferença de valores entre ele mesmo e o novo estado que selecionamos. Essa fração, chamada de *parâmetro de tamanho do passo*, é refletida pelo multiplicador  $c$  na equação:

$$V(s_n) = V(s_n) + c(V(s_{n+1}) - V(s_n))$$

Nessa equação,  $s_n$  representa o estado escolhido no tempo  $n$ , e  $s_{n+1}$  o estado escolhido no tempo  $n+1$ . Essa equação de atualização é um exemplo de *regra de aprendizado por diferença temporal*, uma vez que as suas modificações são uma função da diferença,  $V(s_{n+1}) - V(s_n)$ , entre as estimativas de valor em dois tempos diferentes,  $n$  e  $n+1$ . Discutimos essas regras de aprendizado mais adiante na próxima seção.

A regra de diferença temporal tem um bom desempenho no jogo da velha. Podemos reduzir o parâmetro de tamanho do passo  $c$  ao longo do tempo, de modo que, conforme o sistema aprenda, sucessivamente ajustes menores são feitos aos valores de estado. Isso garante que as funções de valor para cada estado converjam para a probabilidade de vitória, dado o nosso oponente. Além disso, exceto para movimentos exploratórios periódicos, as escolhas feitas serão, de fato, os movimentos ideais, isto é, a política ideal, contra o nosso oponente. O que é ainda mais interessante, entretanto, é o fato de que, se o tamanho do passo nunca chegar realmente a zero, essa política se modificará continuamente para refletir qualquer mudança ou melhoria na jogada do oponente.

O nosso exemplo do jogo da velha ilustra muitas das características importantes do aprendizado por reforço. Primeiro, ocorre aprendizado enquanto há interação com o ambiente, nesse caso, o nosso oponente. Em segundo lugar, há um objetivo explícito (refletido em uma série de estados metas), e um comportamento ideal requer planejamento e antecipação que permitem compensar efeitos atrasados de movimentos particulares. Por exemplo, o algoritmo de aprendizado por reforço de fato monta armadilhas em múltiplos movimentos para o oponente ingênuo. Uma característica importante do aprendizado por reforço é que os efeitos de antecipação e planejamento podem ser de fato alcançados sem um modelo explícito do oponente ou por busca estendida.

No nosso exemplo do jogo da velha, o aprendizado começa sem conhecimento prévio além das regras do jogo. (Simplesmente iniciamos todos os estados não terminais em 0,5.) O aprendizado por reforço certamente não requer essa visão “tabula rasa”. Qualquer informação prévia disponível pode ser inserida nos valores de estado iniciais. É também possível lidar com estados em que não exista informação disponível. Por fim, se um modelo de uma situação estiver disponível, a informação resultante do modelo pode ser usada para os valores de estado. Mas é importante lembrar que o aprendizado por reforço pode ser aplicado nas duas situações: quando não for necessário um modelo, mas ele puder ser usado se estiver disponível, ou, então, quando um modelo puder ser aprendido.

No exemplo do jogo da velha, a recompensa foi amortizada sobre cada decisão de estado-ação. O nosso agente era miope, preocupado em maximizar apenas as recompensas imediatas. Na verdade, se usarmos uma antecipação mais profunda com o aprendizado por reforço, necessitaremos do *retorno descontado* de uma eventual recompensa. Se a taxa de desconto  $\gamma$  representar o valor atual de uma futura recompensa, uma recompensa recebida  $k$  passos de tempo no futuro valerá apenas  $\gamma^{k-1}$  vezes o que ela valeria se fosse recebida imediatamente. Esse desconto da medida de recompensa é importante ao se utilizar a abordagem da programação dinâmica ao aprendizado por reforço, que é apresentada na próxima seção.

O jogo da velha é um exemplo de um jogo para dois jogadores. O aprendizado por reforço pode ser usado também em situações em que não existam oponentes, mas apenas o retorno realimentado pelo ambiente. O exemplo do jogo da velha tem também um espaço de estados finito (e bastante pequeno). O aprendizado por reforço pode ser usado também quando o espaço de estados é muito grande ou mesmo infinito. No último caso, os valores de estado são gerados apenas quando um estado é encontrado e usado em uma solução. Tesauro (1995), por exemplo, usou a regra de diferença temporal descrita anteriormente, incorporada em uma rede neural, para aprender a jogar gamão. Muito embora o tamanho estimado do espaço de estados do jogo de gamão seja de  $10^{20}$  estados, o programa de Tesauro joga no nível dos melhores jogadores humanos.

### 10.7.3 Algoritmos de inferência e aplicações do aprendizado por reforço

De acordo com Sutton e Barto (1998), há três famílias diferentes de algoritmos de inferência por aprendizado por reforço: aprendizado por *diferença temporal*, *programação dinâmica* e métodos *Monte Carlo*. Essas três famílias formam a base para virtualmente todas as abordagens atuais para aprendizado por reforço. Os métodos por diferença temporal aprendem a partir de trajetórias amostradas e valores retornados de um estado para outro. Vimos um exemplo de aprendizado por diferença temporal com o jogo da velha na seção anterior.

Os métodos de programação dinâmica calculam funções de valor retornando valores de estados sucessores para estados predecessores. Os métodos de programação dinâmica atualizam sistematicamente um estado após o outro, baseando-se em um modelo de distribuição do próximo estado. A abordagem por programação dinâmica se baseia no fato de que, para uma política  $\pi$  e um estado  $s$ , vale a seguinte equação recursiva de consistência:

$$V^\pi(s) = \sum_a \pi(a | s) * \sum_s \pi(s \rightarrow s' | a) * (R^a(s \rightarrow s') + \gamma(V^\pi(s')))$$

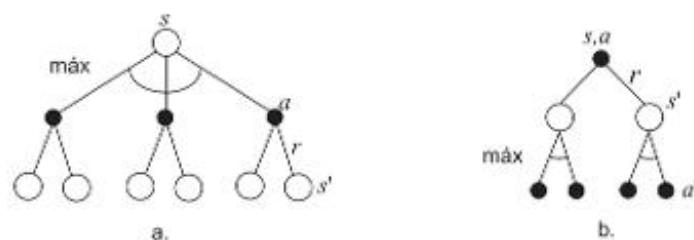
$\pi(a | s)$  é a probabilidade da ação  $a$  dado o estado  $s$  sob a política estocástica  $\pi$ ,  $\pi(s \rightarrow s' | a)$  é a probabilidade de  $s$  ir para  $s'$  sob a ação  $a$ . Essa é a equação de Bellman (1957) para  $V^\pi$ . Ela expressa uma relação entre o valor de um estado e os valores recursivamente calculados de seus estados sucessores. (Ver programação dinâmica na Seção 4.1.) Na Figura 10.24(a), apresentamos o primeiro passo do cálculo, onde, a partir do estado  $s$ , consideramos três estados sucessores possíveis. Com a política  $\pi$ , a ação  $a$  tem probabilidade  $\pi(a | s)$  de ocorrer. A partir de cada um desses três estados, o ambiente poderia responder com um entre vários estados, digamos  $s'$  com recompensa  $r$ . A equação de Bellman calcula a média sobre todas essas possibilidades, ponderando cada uma pela sua probabilidade de ocorrer. Ela determina que o valor do estado inicial  $s$  deve ser igual ao valor descontado por  $\gamma$  dos próximos estados esperados mais a recompensa gerada ao longo do caminho.

Os modelos clássicos de programação dinâmica têm utilidade limitada aqui devido à suposição de um modelo perfeito. Se  $n$  e  $m$  denotarem o número de estados e ações, um método de programação dinâmica garante encontrar uma política ótima em tempo polinomial, muito embora o número total de políticas determinísticas seja  $n^m$ . Nesse sentido, a programação dinâmica é exponencialmente mais rápida que qualquer busca direta no espaço de políticas, pois a busca direta teria que avaliar cada política exaustivamente para fornecer a mesma garantia.

Os métodos de Monte Carlo não requerem um modelo completo. Eles amostram trajetórias de estados completas para atualizar a função de valor com base nos resultados finais dos episódios. Os métodos de Monte Carlo requerem experiência, isto é, amostras de sequências de estados, ações e recompensas a partir de interações reais ou simuladas com o ambiente. A experiência real é interessante porque não requer conhecimento prévio do ambiente e, mesmo assim, pode ser ótima. O aprendizado, a partir de experiência simulada, também é poderoso. É necessário dispor de um modelo, mas ele não precisa ser um modelo analítico, mas sim de geração, isto é, que seja capaz de gerar trajetórias, mas não de calcular probabilidades explícitas. Assim, o modelo não precisa produzir as distribuições de probabilidades completas de todas as transições possíveis, que são necessárias para a programação dinâmica.

Assim, os métodos de Monte Carlo resolvem o problema do aprendizado por reforço calculando médias de amostras de retornos. Para assegurar retornos bem definidos, os métodos de Monte Carlo são definidos apenas

Figura 10.24 Diagramas para regressão de (a)  $V^*$  e (b)  $Q^*$ , adaptados e Sutton e Barto (1998).



para episódios completos, isto é, todos os episódios devem ir até o fim. Além disso, apenas na conclusão de um episódio é que as estimativas de valores e políticas são modificadas. Dessa forma, os métodos de Monte Carlo são incrementais em um sentido de episódio para episódio, e não passo a passo. O termo “Monte Carlo” é frequentemente usado de maneira mais ampla para qualquer método de estimação cuja operação envolva um componente aleatório significativo. Aqui, ele é usado especialmente para métodos baseados em cálculo de média de retornos completos. Consideraremos os métodos por reforço de Monte Carlo novamente no Capítulo 13.

Há outros métodos que são usados para o aprendizado por reforço, sendo o mais importante o aprendizado Q (Watkins, 1989), uma variante da abordagem de diferença temporal. No aprendizado Q, Q é uma função de pares de estado-ação para valores aprendidos. Para todos os estados e ações:

$$Q: (\text{estado} \times \text{ação}) \rightarrow \text{valor}$$

Para um passo do aprendizado Q:

$$Q(s_t, a_t) \leftarrow (1 - c) * Q(s_t, a_t) + c * [r_{t+1} + \gamma * \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

onde  $c$  e  $\gamma$  são  $\leq 1$  e  $r_{t+1}$  é a recompensa em  $s_{t+1}$ . Podemos visualizar a abordagem Q na Figura 10.24(b) e contrastá-la com a Figura 10.24(a), onde o nó inicial é uma situação de estado-ação. Essa regra de regressão atualiza pares de estado-ação, de modo que o estado no topo da Figura 10.24(b), a raiz da regressão, é um nó de ação acoplado com o estado que o produziu.

No aprendizado Q, a regressão se dá a partir de nós de ação, maximizando sobre todas as ações possíveis dos próximos estados com as suas recompensas. No aprendizado Q totalmente recursivo, os nós inferiores da árvore de regressão são nós terminais alcançáveis por uma sequência de ações iniciando no nó raiz com as recompensas dessas ações sucessoras. O aprendizado Q em tempo de execução (on-line), expandindo para a frente a partir de ações possíveis, não requer a construção de um modelo de mundo completo. O aprendizado Q pode também ser executado fora do tempo de execução (off-line). Como se pode notar, o aprendizado Q é uma espécie de abordagem de diferença temporal. Outros detalhes podem ser encontrados em Watkins (1989) e Sutton e Barto (1998). O aprendizado Q será discutido novamente no Capítulo 13.

Atualmente, existe uma série de problemas significativos resolvidos por meio do aprendizado por reforço, incluindo o jogo de gamão (Tesauro, 1994, 1995). Sutton e Barto (1998) também analisam o programa de damas de Samuel (ver Seção 4.3) do ponto de vista do aprendizado por reforço. Eles também discutem a abordagem do aprendizado por reforço para o *acrobata*, o *controle de elevador*, a *alocação dinâmica de canais*, o *escalonamento de tarefas* e outros problemas clássicos (Sutton e Barto, 1998).

## 10.8 Epílogo e referências

O aprendizado de máquina é um dos campos mais excitantes na prática de inteligência artificial, aborda um problema que é central para o comportamento inteligente e levanta uma série de questões importantes sobre representação de conhecimento, busca e mesmo sobre as suposições básicas da própria IA. Alguns textos sobre aprendizado de máquina são: *Elements of Machine Learning*, de Pat Langley (1995); *Machine Learning*, de Tom Mitchell (1997); e *Computational Learning Theory: An Introduction*, de Anthony Martin (1997). Veja também *An Introduction to Machine Learning*, de Nils Nilsson (<<http://robotics.stanford.edu/people/nilsson/mlbook.html>>) e a Seção 16.2.

Uma importante revisão mais antiga sobre aprendizado é *Machine Learning: An Artificial Intelligence Approach* (Kodratoff e Michalski, 1990; Michalski et al., 1983, 1986). *Readings in Machine Learning* (Shavlik e Dietterich, 1990) é uma importante coleção de artigos da área, remontando até 1958. Ao colocar todas essas pesquisas em um único volume, os editores prestaram um serviço valioso tanto para os pesquisadores como para aqueles que procuram uma introdução à área. O aprendizado indutivo é apresentado por Vere (1975, 1978) e Dietterich e Michalski (1981, 1986). *Production System Models of Learning and Development* (Klahr et al., 1987) apresenta uma coleção de artigos em aprendizado de máquina, incluindo trabalhos que refletem uma abordagem mais cognitiva.

*Computer Systems That Learn* (Weiss e Kulikowski, 1991) é uma revisão introdutória de toda a área, incluindo abordagens de redes neurais, métodos estatísticos e técnicas de aprendizado de máquina. Os leitores interessados em uma discussão mais aprofundada do raciocínio por analogia devem examinar Carbonell (1983, 1986), Holyoak (1985), Kedar-Cabelli (1988) e Thagard (1988). Para aqueles interessados em descoberta e formação de teorias, veja *Scientific Discovery: Computational Explorations of the Creative Processes* (Langley et al., 1987) e *Computational Models of Scientific Discovery and Theory Formation* (Shrager e Langley, 1990). *Concept Formation: Knowledge and Experience in Unsupervised Learning* (Fisher et al., 1991) apresenta vários artigos sobre agrupamentos, formação de conceitos e outras formas de aprendizado não supervisionado.

O ID3 tem uma longa história entre a comunidade de aprendizado de máquina. O EPAM (do inglês *Elementary Perceiver And Memorizer*), em *Perceptor e Memorizador Elementar* (Feigenbaum e Feldman, 1963), usava um tipo de árvore de decisão, chamada de *rede de discriminação*, para organizar sequências de sílabas sem sentido. Quinlan foi o primeiro a usar a teoria da informação para gerar filhos na árvore de decisão. Quinlan (1993) e outros estenderam o ID3 para o C4.5 e abordaram questões como ruído e atributos contínuos em dados (Quinlan, 1996; Auer et al., 1995). Stubblefield e Luger (1996) aplicaram o ID3 ao problema de melhorar a recuperação de fontes em um sistema raciocinador analógico.

Michie (1961) e Samuel (1959) oferecem exemplos antigos de aprendizado por reforço. O livro *Reinforcement Learning*, de Sutton e Barto (1998), foi fonte de muito da nossa apresentação sobre esse tópico. Recomendamos a tese de Watkins (1989) para uma apresentação mais detalhada do aprendizado Q, o artigo original de Sutton (1988) para uma análise do aprendizado por diferença temporal e o livro *Neuro-Dynamic Programming*, de Bertsekas e Tsitsiklis (1996), para uma apresentação mais formal de todos os algoritmos de aprendizado por reforço. Discutiremos o aprendizado por reforço novamente no contexto do aprendizado probabilístico no Capítulo 13.

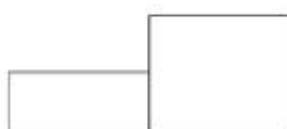
*Machine Learning* é o principal periódico da área. Outras fontes de pesquisa atual incluem os anais da *International Conference on Machine Learning* e da *European Conference on Machine Learning*, bem como os anais da *American Association of Artificial Intelligence Conference* (agora *Association for the Advancement of Artificial Intelligence*) e da *International Joint Conference on Artificial Intelligence* para a atualização mais recente sobre tópicos em aprendizado de máquina.

Apresentamos o aprendizado conexionista no Capítulo 11, os métodos social e emergente no Capítulo 12 e o aprendizado dinâmico e probabilístico no Capítulo 13. Discutimos o viés indutivo, a generalização e outras limitações no aprendizado na Seção 16.2.

## 10.9 Exercícios

1. Considere o comportamento do programa de aprendizado de conceitos de Winston quando ele aprende o conceito “degrau”, onde um degrau consiste de um retângulo baixo e de um retângulo alto, colocados em contato, como apresentado na Figura 10.25. Crie representações por redes semânticas de três ou quatro exemplos e falhas próximas e mostre o desenvolvimento do conceito.

Figura 10.25 Um degrau.

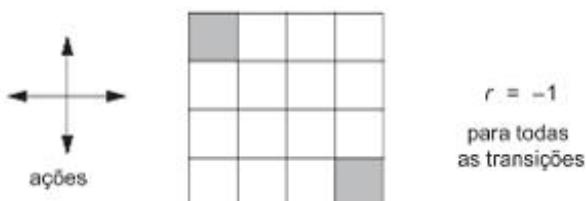


2. A execução do algoritmo de eliminação de candidatos da Figura 10.9 não mostra os conceitos candidatos que foram produzidos e eliminados, ou porque eles eram muito genéricos, ou muito específicos, ou, ainda, subordinados a algum outro conceito. Refaça a sequência de execução, mostrando esses conceitos e as razões pelas quais eles foram eliminados.
3. Construa os algoritmos de busca em espaço de versões em Prolog, ou em uma linguagem de sua escolha. Se você usar Prolog ou Java, veja as dicas para a busca no espaço de versões na Sala Virtual deste livro.
4. Usando a função de seleção baseada na teoria da informação da Seção 10.4.3, mostre em detalhes como o ID3 constrói a árvore da Figura 10.14 a partir dos exemplos da Tabela 10.1. Mostre os cálculos usados para calcular o ganho de informação para cada teste e as seleções resultantes desses testes.
5. Usando a fórmula de Shannon, determine se uma mensagem sobre o resultado do giro de uma roleta tem mais informação que uma mensagem sobre o resultado do lançamento de uma moeda. Qual é o valor da informação se a mensagem sobre o giro da roleta for “não 00”?
6. Desenvolva uma tabela de exemplos simples em algum domínio, como a classificação de animais por espécies, e mostre a construção de uma árvore de decisão pelo algoritmo ID3.
7. Implemente o ID3 em uma linguagem de programação de sua escolha e o utilize no exemplo do histórico de crédito apresentado neste capítulo. Se você usar o LISP, considere como sugestão os algoritmos e as estruturas de dados desenvolvidos na Seção 15.13.
8. Discuta os problemas que podem surgir ao se utilizar atributos contínuos nos dados, como um custo monetário, reais e centavos, ou a altura, um número real, de uma entidade. Sugira um método de tratar o problema de dados contínuos.
9. Outros problemas do ID3 são dados ruins ou faltantes. Os dados são ruins se um conjunto de atributos tiver dois resultados diferentes. Dados faltantes são aqueles em que parte do atributo não está presente, talvez porque seria muito custoso de se obter. Como essas questões poderiam ser tratadas no desenvolvimento de algoritmos ID3?
10. A partir de Quinlan (1993), obtenha o algoritmo de árvore de decisão C4.5 e teste-o em um conjunto de dados. Há programas e conjuntos de dados completos para o C4.5 disponíveis nessa referência.
11. Desenvolva uma teoria de domínio para o aprendizado baseado em explicação em alguma área escolhida. Mostre o comportamento de um sistema aprendiz baseado em explicação aplicando essa teoria a vários exemplos de treinamento.
12. Desenvolva um algoritmo de aprendizado baseado em explicação em uma linguagem qualquer. Se você usar Prolog, considere os algoritmos desenvolvidos na Sala Virtual deste livro.
13. Considere o exemplo do jogo da velha da Seção 10.7.2. Implemente o algoritmo de aprendizado por diferença temporal em uma linguagem escolhida. Se você projetar o algoritmo para levar em consideração simetrias do problema, o que você espera que aconteça? Como isso poderia limitar a sua solução?
14. O que acontece se o algoritmo de diferença temporal do Problema 13 jogar o jogo da velha consigo mesmo?
15. Analise o programa para jogar damas de Samuel do ponto de vista do aprendizado por reforço. Sutton e Barto (1998, Seção 11.2) oferecem sugestões para essa análise.
16. Você pode analisar o problema mecânico, Figura 8.8, apresentado na Seção 8.2.2, do ponto de vista do aprendizado por reforço? Construa algumas medidas simples de recompensa e use o algoritmo de diferença temporal na sua análise.
17. Outro tipo de problema excelente para o aprendizado por reforço é o chamado mundo de grade. Apresentamos um mundo de grade simples  $4 \times 4$  na Figura 10.26. Os dois cantos escuros são os estados terminais desejados para o agente. A partir de todos os outros estados, o movimento do agente pode ser para cima, para baixo, para a esquerda ou para a direita. O agente não pode se mover para fora da grade: tentar movimentos não altera o estado. A recompensa para todas as transições, exceto para os estados terminais é de -1. Desenvolva uma sequência de grades que produza uma solução com base no algorit-

mo de diferença temporal apresentado na Seção 10.7.2. Veja, no Capítulo 13, uma discussão sobre o problema do mundo de grade.

**Figura 10.26** Um exemplo de mundo de grade  $4 \times 4$ , adaptado de Sutton e Barto (1998).

---



$$r = -1$$

para todas  
as transições

---

# Aprendizado de máquina: conexionista

*Um gato que já se sentou sobre um fogão quente não se sentará novamente sobre esse fogão quente nem sobre um fogão frio...*

— MARK TWAIN

*Tudo é vago até certo grau de que você não se dá conta até que tente fazer com que seja preciso...*

— BERTRAND RUSSELL

*... como se uma lanterna mágica projetasse os nervos em padrões sobre uma tela...*

— T. S. ELIOT, *The Love Song of J. Alfred Prufrock*

## 11.0 Introdução

No Capítulo 10, enfatizamos uma abordagem simbólica do aprendizado. Um aspecto central dessa hipótese é o uso de símbolos para referenciar objetos e relações do domínio. Neste capítulo, introduzimos abordagens *neurais*, ou inspiradas na *biologia*, para aprendizado.

Modelos neurais, também conhecidos como *processamento paralelo distribuído* (PDP, do inglês *Parallel Distributed Processing*), ou sistemas *conexionistas*, tiram a ênfase do uso explícito de símbolos para solucionar problemas. Em vez disso, eles afirmam que a inteligência surge em sistemas de componentes simples, interativos (os neurônios biológicos ou artificiais), por meio de um processo de aprendizado, ou de adaptação, pelo qual as conexões entre os componentes são ajustadas. O processamento nesses sistemas é distribuído por meio de conjuntos ou camadas de neurônios. A solução de um problema é paralela no sentido de que todos os neurônios dentro do conjunto ou da camada processam suas entradas simultânea e independentemente. Esses sistemas tendem, também, a se degradar de uma forma suave, porque a informação e o processamento são distribuídos por meio de nós e camadas da rede.

Entretanto, nos modelos conexionistas, há um forte caráter representacional tanto na criação dos parâmetros de entrada como na interpretação dos valores de saída. Para construir uma rede neural, por exemplo, o projetista deve criar um esquema para codificar padrões do mundo em quantidades numéricas da rede. A escolha de um esquema de codificação desempenha um papel crucial no sucesso ou no fracasso de aprendizado da rede.

Em sistemas conexionistas, o processamento é paralelo e distribuído sem manipulação de símbolos como símbolos. Os padrões de um domínio são codificados como vetores numéricos. As conexões *entre* componentes são também representadas por valores numéricos. Finalmente, a transformação de padrões é o resultado de operações numéricas, normalmente multiplicações de matrizes. Essas “escolhas do projetista” para uma arquitetura conexionista constituem o *viés induutivo* do sistema.

Os algoritmos e as arquiteturas que implementam essas técnicas são normalmente treinados ou condicionados, em vez de serem explicitamente programados. Na verdade, esse é o grande poder da abordagem: uma arquitetura de rede e um algoritmo de aprendizado, projetados adequadamente, podem muitas vezes capturar invariâncias do mundo, até mesmo na forma de atratores estranhos, sem serem explicitamente programados para reconhecê-las. O modo como isso acontece forma o material deste capítulo.

Entre as tarefas para as quais a abordagem neural/conexionista é bem adequada estão:

*classificação*, que decide a categoria ou grupo ao qual pertence um valor de entrada;

*reconhecimento de padrões*, que identifica a estrutura ou os padrões nos dados;

*evocação de memória*, que inclui o problema da memória endereçável por conteúdo;

*predição*, que identifica, por exemplo, doenças a partir de sintomas, causas a partir de efeitos;

*otimização*, que encontra a “melhor” organização de restrições; e

*filtragem de ruído*, que separa o sinal do ruído de fundo, retirando os componentes irrelevantes de um sinal.

Os métodos deste capítulo funcionam melhor no caso de tarefas que podem ser difíceis de formular para os modelos simbólicos. Isso inclui geralmente tarefas nas quais o domínio do problema requer capacidades baseadas em percepção, ou em que falte uma sintaxe definida claramente.

Na Seção 11.1, introduzimos modelos inspirados em neurônios de um ponto de vista histórico. Apresentamos os componentes básicos do aprendizado de redes neurais, incluindo o neurônio “mecânico”, e descrevemos alguns trabalhos pioneiros, historicamente importantes, incluindo o neurônio de McCulloch-Pitts (1943). A evolução dos paradigmas de treinamento de redes neurais nos últimos 60 anos fornece um entendimento importante sobre o estado atual da disciplina.

Na Seção 11.2, continuamos a apresentação histórica com a introdução do aprendizado do *perceptron* e a regra *delta*. Apresentamos um exemplo do perceptron usado como um classificador. Na Seção 11.3, introduzimos as redes neurais com camadas ocultas e a regra de aprendizado por *retropropagação* (*backpropagation*). Essas inovações foram introduzidas na evolução das redes neurais artificiais para superar problemas que os antigos sistemas tinham em generalizar a partir de pontos de dados que não eram linearmente separáveis. O algoritmo de retropropagação distribui, proporcionalmente, a “culpa” por respostas incorretas aos nós de um sistema multicamadas com limiares contínuos.

Na Seção 11.4, apresentamos os modelos para o *aprendizado competitivo* desenvolvidos por Kohonen (1984) e Hecht-Nielsen (1987). Nesses modelos, os vetores de peso da rede são usados para representar padrões, em vez de forças de conexões. O algoritmo de aprendizado “*o-vencedor-leva-tudo*” seleciona o nó cujo padrão de pesos é o mais parecido com o vetor de entrada e o ajusta para torná-lo ainda mais parecido com o vetor de entrada. Ele é não supervisionado na medida em que *vencer* significa, simplesmente, identificar o nó cujo vetor de pesos atual se assemelha mais com o vetor de entrada. A combinação de camadas de Kohonen e de Grossberg (1982) em uma única rede oferece um modelo interessante para o aprendizado de estímulo-resposta chamado de aprendizado de *contrapropagação*.

Na Seção 11.5, apresentamos o modelo de Hebb (1949) de aprendizado por reforço. Hebb conjecturou que cada vez que um neurônio contribui para o disparo de outro neurônio, a força da conexão entre esses neurônios é aumentada. O aprendizado hebbiano é modelado por um algoritmo simples para ajustar pesos de conexões. Apresentamos as versões não supervisionada e supervisionada do aprendizado hebbiano. Introduzimos, também, o associador linear, um modelo baseado no aprendizado hebbiano para a recuperação de padrões da memória.

A Seção 11.6 introduz uma família muito importante de redes chamada *redes de atratores*, que empregam conexões realimentadoras para circular, repetidamente, um sinal pela rede. A saída da rede é considerada seu estado ao atingir o equilíbrio. Os pesos da rede são construídos de modo que um conjunto de *atratores* é criado. Padrões de entrada dentro da *bacia* de atração de um atrator alcançam o equilíbrio nesse atrator. Dessa forma, os atratores po-

dem ser usados para armazenar padrões em uma memória. Dado um padrão de entrada, recuperamos o padrão armazenado na rede mais próximo dele, ou, então, um padrão associado ao padrão mais próximo. O primeiro tipo de memória é chamado de memória *autoassociativa*; o segundo, de memória *heteroassociativa*. John Hopfield (1982), um físico teórico, definiu uma classe de redes de atratores cuja convergência pode ser representada por minimização de energia. As redes de Hopfield podem ser usadas para resolver problemas de satisfação de restrições, como o problema do caixeiro-viajante, mapeando a função de otimização em uma função de energia (Seção 11.6.4).

No Capítulo 12, apresentamos modelos evolutivos de aprendizado, como os algoritmos genéticos e a vida artificial. O Capítulo 13 apresenta os modelos de aprendizado dinâmico e estocástico. Na Seção 16.3, discutimos questões representacionais e o viés no aprendizado, bem como as vantagens de cada paradigma de aprendizado.

## 11.1 Fundamentos das redes conexionistas

### 11.1.1 História inicial

As arquiteturas conexionistas são frequentemente vistas como um desenvolvimento recente; entretanto, podemos traçar a sua origem a trabalhos iniciais em ciência da computação, psicologia e filosofia. John von Neumann, por exemplo, era fascinado tanto pelos autômatos celulares quanto pelas abordagens para a computação, inspiradas em neurônios. O trabalho inicial em aprendizado neural foi influenciado pelas teorias psicológicas de aprendizado em animais, especialmente a de Hebb (1949). Nesta seção, definimos os componentes básicos do aprendizado de redes neurais e apresentamos o trabalho inicial historicamente importante para a área.

A base das redes neurais é o neurônio artificial, como vemos na Figura 11.1. Um neurônio artificial consiste em:

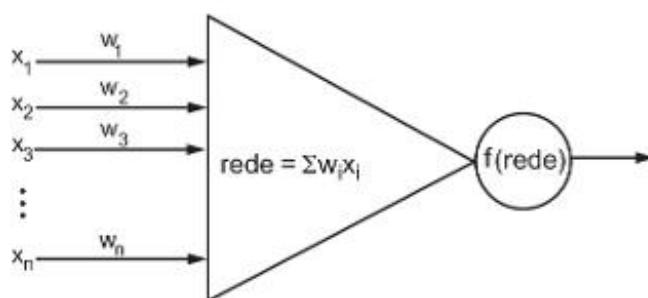
*Sinais de entrada*,  $x_i$ . Esses dados podem vir do ambiente ou da ativação de outros neurônios. Diferentes modelos se distinguem quanto ao intervalo permitido dos valores de entrada; geralmente as entradas são discretas, do conjunto  $\{0, 1\}$ ,  $\{-1, 1\}$ , ou números reais.

*Um conjunto de pesos com valor real*,  $w_i$ . Os pesos descrevem as forças de conexão.

*Um nível de ativação*  $\sum w_i x_i$ . O nível de ativação do neurônio é determinado pela força cumulativa de seus sinais de entrada, onde cada sinal de entrada é escalado pelo peso da conexão  $w_i$  ao longo da linha de entrada. Assim, o nível de ativação é calculado pela soma ponderada das entradas, isto é,  $\sum w_i x_i$ .

*Uma função de limiar*,  $f$ . Essa função calcula o estado final, ou de saída, do neurônio, determinando o quanto o nível de ativação do neurônio está abaixo ou acima de um valor de limiar. O objetivo da função de limiar é produzir o estado ligado/desligado dos neurônios reais.

**Figura 11.1** Um neurônio artificial, um vetor de entrada  $x_i$ , os pesos em cada linha de entrada e uma função de limiar  $f$  que determina o valor de saída do neurônio. Compare esta figura com o neurônio real da Figura 1.2.



Além dessas propriedades dos neurônios individuais, uma rede neural é também caracterizada por propriedades globais, tais como:

*A topologia da rede.* É o padrão de conexões entre os neurônios individuais. Essa topologia é uma fonte primária do viés indutivo da rede.

*O algoritmo de aprendizado utilizado.* Vários algoritmos para aprendizado são apresentados neste capítulo.

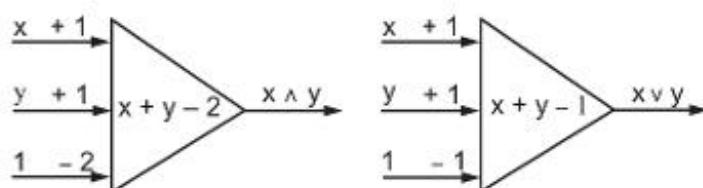
*O esquema de codificação.* Corresponde à interpretação que se dá aos dados fornecidos para a rede e ao resultado do seu processamento.

O primeiro exemplo de computação neural é o neurônio de McCulloch-Pitts (McCulloch e Pitts, 1943). As entradas de um neurônio de McCulloch-Pitts são excitatórias (+1) ou inibitórias (-1). A função de ativação multiplica cada entrada por seu peso correspondente e soma os resultados; se a soma for maior ou igual a zero, o neurônio retorna 1, caso contrário, ele retorna -1. McCulloch e Pitts mostraram como esses neurônios poderiam ser construídos de forma a calcular qualquer função lógica, demonstrando que os sistemas desses neurônios fornecem um modelo computacional completo.

A Figura 11.2 mostra os neurônios de McCulloch-Pitts para calcular funções lógicas. O neurônio E tem três entradas:  $x$  e  $y$  são os valores a serem agregados pela conjunção; o terceiro, algumas vezes chamado de viés, tem um valor constante de +1. Os dados de entrada e o viés têm pesos de +1, -1 e -2, respectivamente. Assim, para quaisquer valores de  $x$  e  $y$ , o neurônio calcula o valor de  $x + y - 2$ ; se esse valor for menor que 0, ele retorna -1; caso contrário, ele retorna 1. A Tabela 11.1 ilustra um neurônio calculando  $x \wedge y$ . De maneira semelhante, a soma ponderada dos dados de entrada para o neurônio OU (veja a Figura 11.2), é maior ou igual a 0, a menos que  $x$  e  $y$  sejam iguais a -1.

Embora McCulloch e Pitts tenham demonstrado o poder da computação neural, o interesse nessa abordagem apenas começou a florescer com o desenvolvimento de algoritmos de aprendizado práticos. Os primeiros modelos de aprendizado se baseavam bastante no trabalho do psicólogo D. O. Hebb (1949), que especulava que o aprendizado ocorria no cérebro por meio da modificação de sinapses. Hebb teorizou que o disparo repetido a partir de uma sinapse aumentava a sua sensibilidade e a probabilidade futura de seu disparo. Se um estímulo particular repetidamente causasse atividade em um grupo de células, essas células se tornariam fortemente associadas. No futuro, estímulos similares tenderiam a excitar os mesmos caminhos neurais, resultando no reconhecimento dos estímulos (ver descrição real de Hebb na Seção 11.5.1). O modelo de aprendizado de Hebb funcionava apenas no

**Figura 11.2** Neurônios de McCulloch-Pitts para calcular as funções lógicas E e OU.



**Tabela 11.1** O modelo de McCulloch-Pitts para o E lógico.

x	y	$x + y - 2$	Saída
1	1	0	1
1	0	-1	-1
0	1	-1	-1
0	0	-2	-1

reforço dos caminhos usados e ignorava a inibição, a punição por erro, ou o desgaste. Os psicólogos modernos procuraram implementar o modelo de Hebb, mas não conseguiram produzir resultados genéricos sem acrescentar um mecanismo inibitório (Rochester et al., 1988; Quinlan, 1991). Consideramos o modelo de aprendizado de Hebb na Seção 11.5.

Na próxima seção, estendemos o modelo neural de McCulloch-Pitts acrescentando camadas de mecanismos neurais conectados e algoritmos para as suas interações. A primeira versão desse tipo é chamada de *perceptron*.

## 11.2 Aprendizado do perceptron

### 11.2.1 Algoritmo de treinamento do perceptron

Frank Rosenblatt (1958, 1962) concebeu um algoritmo de aprendizado para um tipo de rede de camada única chamada *perceptron*. Na sua propagação de sinais, o perceptron é similar ao neurônio de McCulloch-Pitts, o qual veremos na Seção 11.2.2. Os valores de entrada e os níveis de ativação do perceptron são  $-1$  ou  $1$ ; os pesos têm valores reais. O nível de ativação do perceptron é dado pela soma dos valores ponderados das entradas,  $\sum x_i w_i$ . Os perceptrons usam uma função de limiar abrupto simples em que uma ativação acima de um limiar resulta em um valor de saída de  $1$ , ou, no caso contrário, de  $-1$ . Dados os valores de entrada  $x_i$ , os pesos  $w_i$  e um limiar  $t$ , o perceptron calcula o seu valor de saída como:

$$\begin{cases} 1 & \text{se } \sum x_i w_i \geq t \\ -1 & \text{se } \sum x_i w_i < t \end{cases}$$

O perceptron usa uma forma simples de aprendizado supervisionado. Após tentar resolver uma ocorrência do problema, um professor fornece a ele o resultado correto. O perceptron modifica, então, seus pesos de modo a reduzir o erro. A regra a seguir é usada. Seja  $c$  uma constante cujo tamanho determina a taxa de aprendizado, e  $d$ , o valor de saída desejado. O ajuste para o peso no  $i$ -ésimo componente do vetor de entrada,  $\Delta w_i$ , é dado por:

$$\Delta w_i = c(d - \text{sinal}(\sum x_i w_i)) x_i$$

O  $\text{sinal}(\sum x_i w_i)$  é o valor de saída do perceptron. Ele é  $-1$  ou  $1$ . A diferença entre os valores de saída desejado e real será  $0$ ,  $2$  ou  $-2$ . Assim, para cada componente do vetor de entrada:

Se os valores de saída, desejado e real, forem iguais, não faça nada.

Se o valor de saída real for  $-1$  e o desejado for  $1$ , incremente os pesos na  $i$ -ésima linha em  $2x_i$ .

Se o valor de saída real for  $1$  e o desejado for  $-1$ , reduza os pesos na  $i$ -ésima linha em  $2x_i$ .

Esse procedimento tem o efeito de produzir um conjunto de pesos que pretende minimizar o erro médio sobre o todo o conjunto de treinamento. Se existir um conjunto de pesos que forneça a saída correta para todos os membros do conjunto de treinamento, o procedimento de aprendizado do perceptron irá aprendê-lo (Minsky e Papert, 1969).

Os perceptrons foram inicialmente saudados com entusiasmo. Entretanto, Nils Nilsson (1965) e outros analisaram as limitações do seu modelo. Eles demonstraram que os perceptrons não poderiam resolver certa classe difícil de problemas, na qual os pontos de dados não são linearmente separáveis. Embora várias melhorias do modelo, incluindo os perceptrons multicamadas, fossem concebidas naquele tempo, Marvin Minsky e Seymour Papert, no seu livro *Perceptrons* (1969), argumentavam que o problema da separabilidade linear não poderia ser superado por nenhuma forma de rede de perceptrons.

Um exemplo de uma classificação não linearmente separável é o *OU-exclusivo*, que é representado pela tabela verdade:

**Tabela 11.2** Tabela verdade para o OU-exclusivo.

$x_1$	$x_2$	Saída
1	1	0
1	0	1
0	1	1
0	0	0

Considere um perceptron com duas entradas,  $x_1, x_2$ , dois pesos,  $w_1, w_2$ , e o limiar  $t$ . Para aprender essa função, uma rede precisa encontrar uma atribuição de pesos que satisfaça as seguintes desigualdades, vistas graficamente na Figura 11.3:

$$w_1 \cdot 1 + w_2 \cdot 1 < t, \text{ da linha 1 da tabela verdade.}$$

$$w_1 \cdot 1 + 0 > t, \text{ da linha 2 da tabela verdade.}$$

$$0 + w_2 \cdot 1 > t, \text{ da linha 3 da tabela verdade.}$$

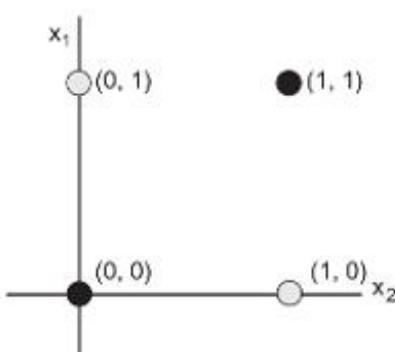
$$0 + 0 < t, \text{ ou } t \text{ deve ser positivo, da última linha da tabela.}$$

Essa série de equações em  $w_1, w_2$  e  $t$  não tem solução, provando que é impossível para um perceptron resolver o *OU-exclusivo*. Embora as redes multicamadas pudessem ser construídas de modo a resolver o problema do OU-exclusivo (ver Seção 11.3.3), o algoritmo de aprendizado do perceptron funcionava apenas em redes de uma camada.

O que torna o ou-exclusivo impossível para o perceptron é que as duas classes que devem ser distinguidas não são *linearmente separáveis*. Isso pode ser visto na Figura 11.3. É impossível desenhar uma linha reta em duas dimensões que separe os pontos de dados  $\{(0,0), (1,1)\}$  dos pontos  $\{(0,1), (1,0)\}$ .

Podemos considerar o conjunto de valores dos dados de entrada para uma rede como definidor de um espaço. Cada parâmetro dos dados de entrada corresponde a uma dimensão, com cada valor de entrada definindo um ponto no espaço. No exemplo do OU-exclusivo, os quatro valores de entrada, indexados pelas coordenadas  $x_1, x_2$ , formam os pontos de dados da Figura 11.3. O problema de aprender uma classificação binária dos exemplos de treinamento se reduz àquele de separar esses pontos em dois grupos. Para um espaço de  $n$  dimensões, uma classificação é linearmente separável se as suas classes puderem ser separadas por um hiperplano de dimensão  $n - 1$ . (Em duas dimensões, um hiperplano  $n$ -dimensional é uma linha; em três dimensões, ele é um plano etc.)

Como resultado da limitação da separabilidade linear, as pesquisas se voltaram para as arquiteturas simbólicas, refreando o progresso na metodologia conexionista. Contudo, trabalhos subsequentes nas décadas de 1980 e 1990 mostraram que esses problemas eram solucionáveis (veja Ackley et al., 1985; Hinton e Sejnowski, 1986, 1987).

**Figura 11.3** Problema do OU-exclusivo. Nenhuma linha reta de duas dimensões pode separar os pontos de dados  $(0, 1)$  e  $(1, 0)$  dos pontos  $(0, 0)$  e  $(1, 1)$ .

Na Seção 11.3, discutimos a *retropropagação*, uma extensão do aprendizado do perceptron que funciona para redes multicamadas. Antes de examinarmos a retropropagação, apresentamos um exemplo de perceptron que realiza classificações. Encerramos a Seção 11.2 definindo a *regra delta generalizada*, uma generalização do algoritmo de aprendizado do perceptron que é usado em muitas arquiteturas de redes neurais, incluindo a rede de *retropropagação*.

### 11.2.2 Um exemplo: usando um perceptron para classificação

A Figura 11.4 mostra um panorama do problema de classificação. Os dados brutos de um espaço de pontos possíveis são selecionados e transformados em um novo espaço de dados/padrões. Nesse novo espaço de padrões, são identificadas características e, finalmente, a entidade que essas características representam é classificada. Um exemplo seriam ondas sonoras gravadas em um gravador digital. De lá, os sinais acústicos são transformados em um conjunto de parâmetros de amplitudes e frequências. Finalmente, um sistema de classificação poderia reconhecer esses padrões de características como o discurso feito por uma pessoa em particular. Outro exemplo é a captação de informação por um equipamento de testes médicos, por exemplo, desfibriladores cardíacos. As características encontradas nesse espaço de padrões seriam usadas para classificar conjuntos de sintomas em diferentes categorias de doenças.

No nosso exemplo de classificação, o transdutor e o extrator de características da Figura 11.4 traduzem a informação sobre o problema em parâmetros de um espaço cartesiano bidimensional. A Figura 11.5 apresenta a análise da informação do perceptron de duas características da Tabela 11.3. As primeiras duas colunas da tabela apresentam os pontos de dados sobre os quais a rede foi treinada. A terceira coluna representa a classificação, +1 ou -1, usada como realimentação no treinamento da rede. A Figura 11.5 é um grafo dos dados de treinamento do problema, mostrando a separação linear das classes de dados, criada quando a rede treinada foi aplicada a cada ponto de dado.

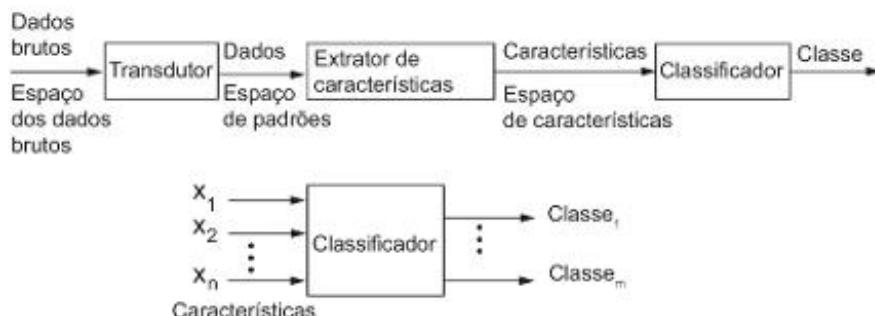
Discutimos, inicialmente, a teoria geral da classificação. Cada agrupamento de dados que um classificador identifica é representado por uma região no espaço multidimensional. Cada classe  $R_i$  tem uma função discriminante  $g_i$ , que mede a pertinência nessa região. Dentro da região  $R_i$ , a  $i$ -ésima função discriminante tem o maior valor:

$$g_i(x) > g_j(x) \text{ para todo } j, 1 < j < n.$$

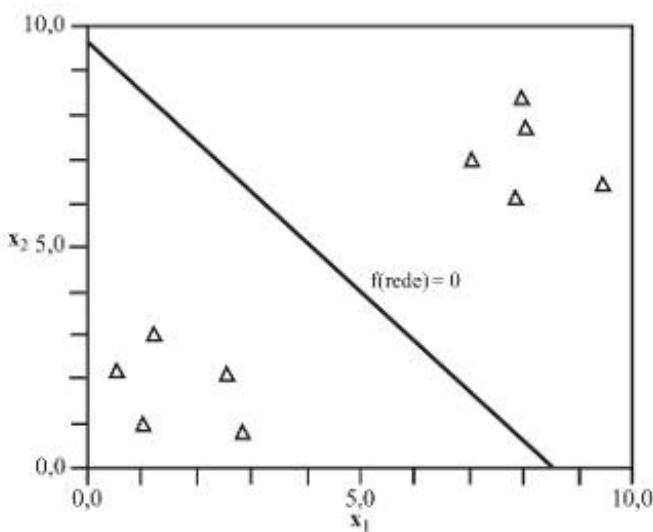
No exemplo simples da Tabela 11.3, os dois parâmetros de entrada produzem duas regiões, ou classes óbvias, no espaço, uma representada por 1, a outra representada por -1.

Um caso especial importante de funções discriminantes é aquele que avalia a pertinência de classe com base na distância a um ponto central da região. A classificação baseada nessa função discriminante é chamada de *classificação por distância mínima*. Um argumento simples mostra que, se as classes são linearmente separáveis, existe uma classificação por distância mínima.

**Figura 11.4** Um sistema de classificação completo.



**Figura 11.5** Um gráfico bidimensional dos pontos de dados da Tabela 11.3. O perceptron da Seção 11.2.1 fornece uma separação linear dos conjuntos de dados.



**Tabela 11.3** Um conjunto de dados para classificação por um perceptron.

$x_1$	$x_2$	Saída
1,0	1,0	1
9,4	6,4	-1
2,5	2,1	1
8,0	7,7	-1
0,5	2,2	1
7,9	8,4	-1
7,0	7,0	-1
2,8	0,8	1
1,2	3,0	1
7,8	6,1	-1

Se as regiões  $R_i$  e  $R_j$  são adjacentes, como as duas regiões na Figura 11.5, existe uma região de fronteira onde as funções discriminantes são iguais:

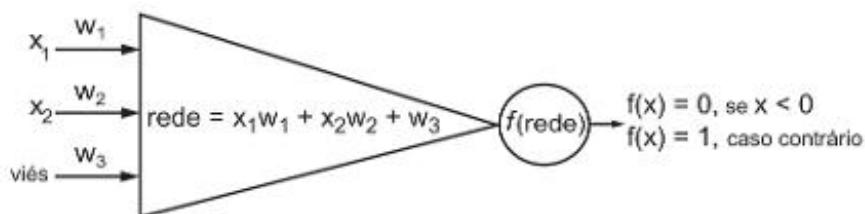
$$g_i(x) = g_j(x) \text{ ou } g_i(x) - g_j(x) = 0.$$

Se as classes são linearmente separáveis, como na Figura 11.5, a função discriminante separando as regiões é uma linha reta, ou  $g_i(x) - g_j(x)$  é linear. Como uma reta é formada por pontos equidistantes de dois pontos fixos, as funções discriminantes,  $g_i(x)$  e  $g_j(x)$ , são funções de distância mínima, medidas a partir do centro cartesiano de cada uma das regiões.

O perceptron da Figura 11.6 calcula essa função linear. Precisamos de dois parâmetros de entrada e teremos um viés com um valor constante de 1. O perceptron calcula:

$$f(\text{rede}) = f(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot 1), \text{ onde } f(x) \text{ é o sinal de } x.$$

**Figura 11.6** Perceptron para os dados do exemplo da Tabela 11.3. A função de limiar é linear e bipolar [ver Figura 11.7(a)].



Quando  $f(x)$  é  $+1$ ,  $x$  é interpretado como se estivesse em uma das classes e, quando ele é  $-1$ ,  $x$  está na outra classe. Esse limiar para  $+1$  ou  $-1$  é chamado de limiar linear bipolar [ver Figura 11.7(a)]. O viés serve para deslocar a função de limiar sobre o eixo horizontal. A extensão desse deslocamento é aprendida ajustando o peso  $w_3$  durante o treinamento.

Usamos, agora, os pontos de dados da Tabela 11.3 para treinar o perceptron da Figura 11.6. Supomos uma inicialização aleatória dos pesos com os valores  $[0,75, 0,5, -0,6]$  e usamos o algoritmo para treinamento do perceptron da Seção 11.2.1. Os índices superiores — por exemplo, o 1 em  $f(\text{rede})^1$  — representam o número da iteração atual do algoritmo. Iniciamos tomando o primeiro ponto de dado da tabela:

$$f(\text{rede})^1 = f(0,75 \cdot 1 + 0,5 \cdot 1 - 0,6 \cdot 1) = f(0,65) = 1$$

Como  $f(\text{rede})^1 = 1$ , o valor correto de saída, não ajustamos os pesos. Assim,  $W^2 = W^1$ . Para o segundo ponto de dado:

$$f(\text{rede})^2 = f(0,75 \cdot 9,4 + 0,5 \cdot 6,4 - 0,6 \cdot 1) = f(9,65) = 1$$

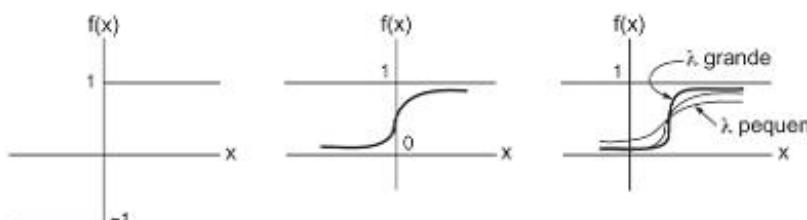
Dessa vez, o nosso resultado deveria ser  $-1$ , de modo que devemos aplicar a regra de aprendizado descrita na Seção 11.1.1:

$$W^t = W^{t-1} + c(d^{t-1} - \text{sinal}(W^{t-1} \cdot X^{t-1})) X^{t-1}$$

onde  $c$  é a constante de aprendizado,  $X$  e  $W$  são os vetores de entrada e de peso, e  $t$  é a iteração da rede.  $d^{t-1}$  é o resultado desejado no tempo  $t-1$ , ou na nossa situação, em  $t=2$ . A saída da rede em  $t=2$  é 1. Com isso, a diferença entre a saída desejada e a obtida pela rede,  $d^2 - \text{sinal}(W^2 \cdot X^2)$ , é  $-2$ . Na verdade, em um perceptron bipolar limitado abruptamente, o incremento do aprendizado será sempre  $+2c$  ou, então,  $-2c$  vezes o vetor de treinamento. Fazemos a constante de aprendizado um número real positivo pequeno, 0,2. Atualizamos o vetor de pesos:

$$W^3 = W^2 + 0,2(-1 - 1)X^2 = \begin{bmatrix} 0,75 \\ 0,50 \\ -0,60 \end{bmatrix} - 0,4 \begin{bmatrix} 9,4 \\ 6,4 \\ 1,0 \end{bmatrix} = \begin{bmatrix} -3,01 \\ -2,06 \\ -1,00 \end{bmatrix}$$

**Figura 11.7** Funções de limiar.



a. Um limiar bipolar linear.

b. Um limiar sigmoide unipolar.

c. Um limiar sigmoide deslocado por um viés e achatado. Conforme  $\lambda$  aumenta, a sigmoide se aproxima de um limiar linear.

Consideramos, agora, o terceiro ponto de dado com os pesos recém-ajustados:

$$f(\text{rede})^3 = f(-3,01 \cdot 2,5 - 2,06 \cdot 2,1 - 1,0 \cdot 1) = f(-12,84) = -1$$

Novamente, o resultado da rede não é a saída desejada. Mostramos o ajuste de  $W^4$ :

$$W^4 = W^3 + 0,2(-1-1)X^3 = \begin{bmatrix} -3,01 \\ -2,06 \\ -1,00 \end{bmatrix} + 0,4 \begin{bmatrix} 2,5 \\ 2,1 \\ 1,0 \end{bmatrix} = \begin{bmatrix} -2,01 \\ -1,22 \\ -0,60 \end{bmatrix}$$

Após 10 iterações do perceptron, é produzida a separação linear da Figura 11.5. Após o treinamento repetido sobre o conjunto de dados, cerca de 500 iterações no total, o vetor de pesos converge para  $[-1,3, -1,1, 10,9]$ . Estamos interessados na reta que separa as duas classes. Em termos das funções discriminantes  $g_i$  e  $g_j$ , a reta é definida como o local dos pontos para os quais  $g_i(x) = g_j(x)$  ou  $g_i(x) - g_j(x) = 0$ , isto é, onde a saída da rede é 0. A equação para a saída da rede é dada em termos dos pesos:

$$\text{saída} = w_1x_1 + w_2x_2 + w_3.$$

Consequentemente, a reta separando as duas classes é definida pela equação linear:

$$-1,3x_1 + -1,1x_2 + 10,9 = 0.$$

### 11.2.3 Regra delta generalizada

Uma forma direta de generalizar o perceptron é substituir a sua função de limiar abrupto por outros tipos de função de ativação. Funções de ativação contínuas, por exemplo, oferecem a possibilidade de algoritmos de aprendizado mais sofisticados por permitirem uma granularidade mais fina na medida do erro.

A Figura 11.7 mostra o gráfico de algumas funções de limiar: uma função de limiar linear bipolar [Figura 11.7(a)], semelhante àquela usada pelo perceptron, e algumas funções *sigmoídes*. Funções sigmoídes são assim chamadas porque seu gráfico tem uma curva em forma de "S", como vemos na Figura 11.7(b). Uma função de ativação sigmoide comum, chamada de função *logística*, é dada pela equação:

$$f(\text{rede}) = 1/(1 + e^{-\lambda \cdot \text{rede}}), \text{ onde } \text{rede} = \sum_i w_i x_i.$$

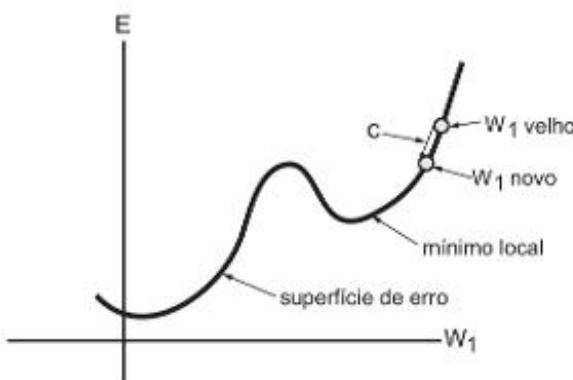
Como no caso das funções previamente definidas,  $x_i$  é a entrada na linha  $i$ ,  $w_i$  é o peso na linha  $i$  e  $\lambda$  é um "parâmetro de achatamento" usado para ajustar a curva sigmoide. Conforme  $\lambda$  aumenta, a sigmoide se aproxima de uma função de limiar linear sobre  $\{0, 1\}$ ; quando  $\lambda$  se aproxima de 1, a função se assemelha a uma reta.

Esses gráficos de limiar traçam os valores de entrada, o nível de ativação do neurônio, contra a ativação escalada ou saída do neurônio. A função de ativação sigmoide é contínua, o que permite uma medida mais precisa de erro. Assim como a função de limitação abrupta, a função de ativação sigmoide mapeia a maioria dos valores de seu domínio para regiões próximas a 0 ou 1. Entretanto, há uma região de transição rápida, mas contínua, entre 0 e 1. De certo modo, ela aproxima um comportamento de limiar, ao mesmo tempo em que produz uma função de saída contínua. O uso de  $\lambda$  no expoente ajusta a inclinação da forma sigmoide na região de transição. Um viés ponderado desloca o limiar ao longo do eixo  $x$ .

A emergência histórica de redes com funções de ativação contínuas sugeriu novas abordagens para o aprendizado por redução de erro. A regra de aprendizado de Widrow-Hoff (1960) é independente da função de ativação, minimizando o erro quadrático entre o valor de saída desejado e a ativação da rede,  $\text{rede}_i = WX_i$ . Talvez a regra de aprendizado mais importante para funções de ativação contínuas seja a *regra delta* (Rumelhart et al., 1986a).

Intuitivamente, a regra delta é baseada na ideia de uma superfície de erro, como ilustra a Figura 11.8. Essa superfície de erro representa o erro cumulativo sobre um conjunto de dados como uma função de pesos da rede. Cada configuração possível de pesos da rede é representada por um ponto sobre essa superfície de erro n-dimensional. Dada uma configuração de pesos, queremos que o algoritmo de aprendizado encontre a direção sobre essa superfície que reduza o mais rapidamente possível o erro. Essa abordagem é chamada de *aprendizado por gradiente descendente* porque o gradiente é uma medida da inclinação, como uma função de direção, a partir de um ponto sobre uma superfície.

**Figura 11.8** Uma superfície de erro em duas dimensões (de forma mais realista, deveria ser visualizado em n dimensões). A constante C controla o tamanho do passo de aprendizado.



Para usar a regra delta, a rede deve usar uma função de ativação que é contínua e, portanto, diferenciável. A fórmula logística já apresentada tem essa propriedade. A fórmula do aprendizado pela regra delta para ajustar o peso na j-ésima entrada do nó i é:

$$c(d_i - O_i) f'(rede_i) x_j,$$

onde c é a constante que controla a taxa de aprendizado,  $d_i$  e  $O_i$  são os valores de saída desejado e real do i-ésimo nó, respectivamente. A derivada da função de ativação para o i-ésimo nó é  $f'$ , e  $x_j$  é a j-ésima entrada do nó i. Mostramos, agora, a derivação dessa fórmula.

O erro médio quadrático de rede é calculado pela soma do erro quadrático de cada nó:

$$\text{Erro} = (1/2) \sum_i (d_i - O_i)^2$$

onde  $d_i$  é o valor desejado para cada nó de saída e  $O_i$  é o valor de saída real do nó. Elevamos ao quadrado cada erro, de modo que os erros individuais, alguns possivelmente com valores negativos e outros com valores positivos, não cancelam um ao outro na soma.

Consideramos, aqui, o caso em que o nó está na camada de saída; descreveremos o caso geral quando apresentarmos as redes com camadas ocultas na Seção 11.3. Queremos primeiro medir a taxa de variação do erro da rede em relação à saída de cada nó. Para fazer isso, usamos a noção de *derivada parcial*, que nos dá a taxa de variação de uma função multivariada em relação a uma variável particular. A derivada parcial do erro total em relação a cada unidade de saída i é:

$$\frac{\partial \text{Erro}}{\partial O_i} = \frac{\partial (1/2) * \sum (d_i - O_i)^2}{\partial O_i} = \frac{\partial (1/2) * (d_i - O_i)^2}{\partial O_i}$$

A segunda simplificação é possível porque consideramos um nó na camada de saída, onde seu erro não afeta qualquer outro nó. Realizando a derivada dessa quantidade, obtemos:

$$\frac{\partial (1/2) * (d_i - O_i)^2}{\partial O_i} = -(d_i - O_i)$$

O que queremos é a taxa de variação do erro da rede como uma função da variação nos pesos do nó i. Para uma variação em um peso em particular,  $w_k$ , usamos a derivada parcial, dessa vez realizando a derivada parcial do erro em cada nó em relação ao peso,  $w_k$ , daquele nó. A expansão do lado direito do sinal de igualdade é dada pela regra da cadeia das derivadas parciais:

$$\frac{\partial \text{Erro}}{\partial w_k} = \frac{\partial \text{Erro}}{\partial O_i} * \frac{\partial O_i}{\partial w_k}$$

Isso nos fornece o que precisamos para resolver a equação. Usando o nosso resultado anterior, obtemos:

$$\frac{\partial \text{Erro}}{\partial w_k} = -(d_i - O_i) * \frac{\partial O_i}{\partial w_k}$$

Continuamos a derivação considerando o fator mais à direita, a derivada parcial da saída real do  $i$ -ésimo nó em relação a cada peso desse nó. A fórmula para a saída do nó  $i$  como uma função de seus pesos é:

$$O_i = f(W_i X_i), \text{ onde } W_i X_i = \text{rede}_i.$$

Como  $f$  é uma função contínua, ao fazermos a derivada obtemos:

$$\frac{\partial O_i}{\partial w_k} = x_k * f'(W_i X_i) = f'(\text{rede}_i) * x_k$$

Substituindo na equação anterior:

$$\frac{\partial \text{Erro}}{\partial w_k} = -(d_i - O_i) f'(\text{rede}_i) * x_k$$

A minimização do erro requer que as variações dos pesos sejam feitas na direção da componente negativa do gradiente. Portanto:

$$\Delta w_k = -c \frac{\partial \text{Erro}}{\partial w_k} = -c [-(d_i - O_i) f'(\text{rede}_i) * x_k] = c(d_i - O_i) f'(\text{rede}_i) * x_k$$

Observamos que a regra delta é análoga a uma *subida de encosta* (ver Seção 4.1), pois a cada passo tenta minimizar a medida de erro local usando a derivada para encontrar a inclinação do espaço de erro em torno de um ponto particular. Isso torna o aprendizado delta vulnerável ao problema de distinguir mínimos locais de mínimos globais no espaço de erro.

A constante de aprendizado,  $c$ , exerce uma influência importante sobre o desempenho da regra delta, como a análise da Figura 11.8 ilustra. O valor de  $c$  determina quanto os valores dos pesos se movem em um único episódio de aprendizado. Quanto maior for o valor de  $c$ , mais rapidamente os pesos se movem em direção a um valor ótimo. Entretanto, se  $c$  for muito grande, o algoritmo pode passar do mínimo ou oscilar em torno dos pesos ótimos. Valores menores de  $c$  são menos suscetíveis a esse problema, mas não permitem que o sistema aprenda tão rapidamente. O valor ótimo da taxa de aprendizado, algumas vezes acrescentada de um fator de momento (Zurada, 1992), é um parâmetro ajustado para uma aplicação particular por meio de experimentação.

Embora a regra delta por si só não supere as limitações das redes de camada única, a sua forma generalizada é o ponto central para o funcionamento da retropropagação, um algoritmo para aprendizado de uma rede multcamadas. Esse algoritmo é apresentado na próxima seção.

## 11.3 Aprendizado por retropropagação

### 11.3.1 Derivação do algoritmo de retropropagação

Como vimos, as redes de perceptron de uma única camada são limitadas pelas classificações que elas podem realizar. Nas seções 11.3 e 11.4, mostramos que a adição de múltiplas camadas pode superar muitas dessas limitações. Na Seção 16.3, observaremos que redes multcamadas são computacionalmente completas, isto é, equivalentes à classe das máquinas de Turing. No entanto, os primeiros pesquisadores não foram capazes de conceber um

algoritmo de aprendizado para ser usado nessas redes. Nesta seção, apresentamos a regra delta generalizada que oferece uma solução para esse problema.

Os neurônios em uma rede multicamadas (ver Figura 11.9) estão conectados em camadas, com unidades de uma camada  $n$  passando suas ativações apenas para neurônios da camada  $n + 1$ . O processamento de sinais em múltiplas camadas significa que os erros no interior da rede podem se espalhar e evoluir em formas complexas e imprevistas por meio das camadas sucessivas. Assim, a análise da fonte de erro na camada de saída é complexa. A retropropagação fornece um algoritmo para atribuir aos neurônios a sua parcela de culpa pelo erro da rede e ajustar os pesos de forma correspondente.

A abordagem adotada pelo algoritmo consiste em iniciar na camada de saída e *propagar* o erro retroativamente através das camadas ocultas. Quando analisamos o aprendizado com a regra delta, vimos que toda a informação necessária para atualizar os pesos de um neurônio se encontrava localmente nesse neurônio, exceto pela parcela de erro. Para nós de saída, essa parcela é facilmente calculada pela diferença entre os valores de saída desejado e real. Para nós em camadas ocultas, é bem mais difícil determinar o erro para o qual um nó é responsável. A função de ativação para a retropropagação é normalmente a função logística:

$$f(\text{rede}) = \frac{1}{1 + e^{-\lambda \cdot \text{rede}}}, \text{ onde } \text{rede} = \sum_i w_i x_i.$$

Essa função é usada por quatro razões. Primeiro, ela tem a forma sigmoide. Segundo, sendo uma função contínua, ela tem uma derivada definida em todo o domínio. Terceiro, como o valor da derivada é maior onde a função sigmoide varia mais rapidamente, a atribuição da maior parte do erro é feita àqueles nós cuja ativação é menos certa. Quarto, a derivada é facilmente calculada por uma subtração e uma multiplicação:

$$f(\text{rede}) = \left( \frac{1}{1 + e^{-\lambda \cdot \text{rede}}} \right) = \lambda \cdot (f(\text{rede}) \cdot (1 - f(\text{rede}))).$$

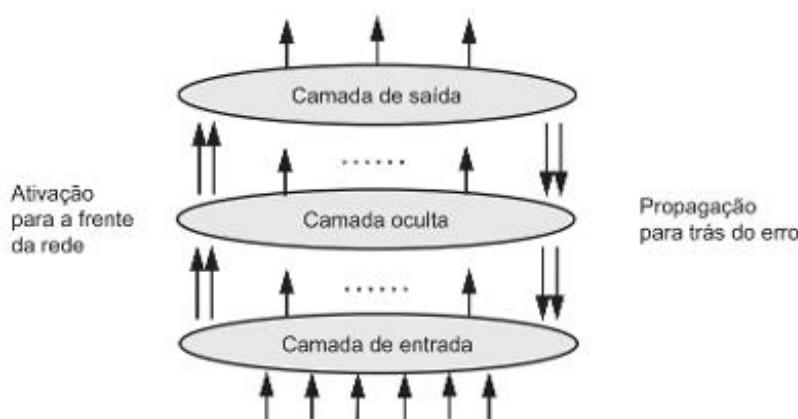
O treinamento por retropropagação usa a regra delta generalizada. Ele adota a mesma abordagem de gradiente descendente apresentada na Seção 11.2. Para nós na camada oculta, procuramos a sua contribuição ao erro da camada de saída. As fórmulas para calcular o ajuste do peso  $w_{ki}$  no caminho do nó  $k$  para o nó  $i$  no treinamento por retropropagação são:

1.  $\Delta w_{ki} = -c(d_i - O_i) * O_i(1 - O_i)x_k$ , para nós na camada de saída, e
2.  $\Delta w_{ki} = -c * O_i(1 - O_i) \sum_j (-\delta_{kj} * w_{ij})x_k$ , para nós em camadas ocultas.

Em (2),  $j$  é o índice dos nós na próxima camada para os quais os sinais de  $i$  se propagam e:

$$\delta_{kj} = \frac{\partial \text{Erro}}{\partial \text{rede}_j} = (d_j - O_j) * O_j(1 - O_j).$$

**Figura 11.9** Retropropagação em uma rede conexionista com uma camada oculta.



Mostramos, agora, a derivação dessas fórmulas. Primeiro, derivamos (1), a fórmula para ajuste de pesos em nós na camada de saída. Como antes, o que desejamos é a taxa de variação do erro da rede como uma função da variação no peso  $w_k$ ,  $w_k$ , do nó  $i$ . Tratamos essa situação na derivação da regra delta, na Seção 11.2.3, e mostramos que:

$$\frac{\partial \text{Erro}}{\partial w_k} = -((d_i - O_i) * f'(\text{rede}_i) * x_k)$$

Como  $f$ , que poderia ser qualquer função, é agora a função de ativação logística, temos:

$$f(\text{rede}) = f(1/(1 + e^{-\lambda * \text{rede}})) = f(\text{rede}) * (1 - f(\text{rede})).$$

Lembre-se de que  $f(\text{rede}_i)$  é simplesmente  $O_i$ . Substituindo na equação anterior, obtemos:

$$\frac{\partial \text{Erro}}{\partial w_k} = -((d_i - O_i) * O_i * (1 - O_i) * x_k)$$

Como a minimização do erro requer que as variações nos pesos aconteçam na direção do componente negativo do gradiente, multiplicamos por  $-c$  para obtermos o ajuste de pesos para o nó  $i$  da camada de saída:

$$\Delta w_k = c(d_i - O_i) * O_i * (1 - O_i) * x_k.$$

Depois, derivamos o ajuste de pesos para os nós ocultos. Por motivo de clareza, supomos inicialmente a existência de uma única camada oculta. Tomamos um único nó  $i$  na camada oculta e analisamos a sua contribuição para o erro total da rede. Fazemos isso considerando, inicialmente, a contribuição do nó  $i$  para o erro em um nó  $j$  da camada de saída. Somamos, então, essas contribuições sobre todos os nós da camada de saída. Finalmente, descrevemos a contribuição do  $k$ -ésimo peso de entrada do nó  $i$  para o erro da rede. A Figura 11.10 ilustra essa situação.

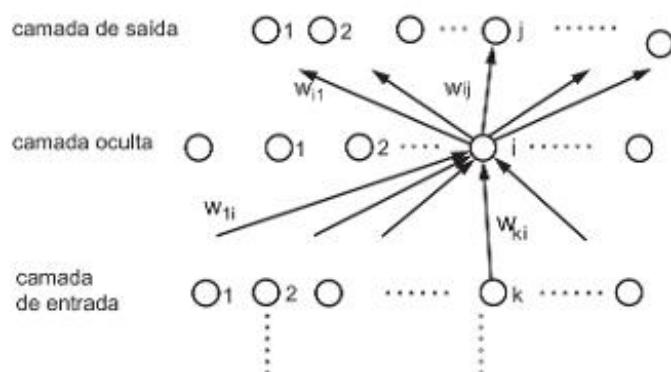
Primeiro, consideramos a derivada parcial do erro da rede em relação à saída do nó  $i$  da camada oculta. Obtemos essa derivada aplicando a regra da cadeia:

$$\frac{\partial \text{Erro}}{\partial O_i} = \frac{\partial \text{Erro}}{\partial \text{rede}_j} * \frac{\partial \text{rede}_j}{\partial O_i}$$

O negativo do primeiro termo no lado direito,  $(\partial \text{erro})/(\partial \text{rede}_j)$ , é chamado de *delta<sub>j</sub>*. Portanto, podemos reescrever a equação como:

$$\frac{\partial \text{Erro}}{\partial O_i} = -\text{delta}_j * \frac{\partial \text{rede}_j}{\partial O_i}$$

**Figura 11.10**  $\sum_j -\text{delta}_j * w_{ij}$  é a contribuição total do nó  $i$  para o erro na saída. A nossa derivação dá o ajuste para  $w_{ki}$ .



Devemos nos lembrar de que a ativação do nó  $j$ ,  $\text{rede}_j$ , na camada de saída é dada pela soma do produto de seus pesos pelos valores de saída dos nós da camada oculta:

$$\text{rede}_j = \sum_i w_{ij} O_i$$

Como consideramos a derivada parcial em relação a apenas um componente da soma, a conexão entre o nó  $i$  e o nó  $j$ , obtemos:

$$\frac{\partial \text{rede}_j}{\partial O_i} = w_{ij},$$

onde  $w_{ij}$  é o peso na conexão entre o nó  $i$  na camada oculta e o nó  $j$  na camada de saída. Substituindo esse resultado:

$$\frac{\partial \text{Erro}}{\partial O_i} = -\delta_j * w_{ij}$$

Somamos, agora, todos os componentes do nó  $i$  da camada de saída:

$$\frac{\partial \text{Erro}}{\partial O_i} = \sum_j -\delta_j * w_{ij}$$

Isso nos dá a sensibilidade do erro da rede à saída do nó  $i$  na camada oculta. A seguir, determinamos o valor de  $\delta_i$ , a sensibilidade do erro da rede à ativação líquida ( $\text{rede}$ ) no nó oculto  $i$ . Isso dá a sensibilidade do erro da rede aos pesos incidentes do nó  $i$ . Usando novamente a regra da cadeia:

$$-\delta_i = \frac{\partial \text{Erro}}{\partial \text{rede}_i} = \frac{\partial \text{Erro}}{\partial O_i} * \frac{\partial O_i}{\partial \text{rede}_i}$$

Como estamos usando a função de ativação logística,

$$\frac{\partial O_i}{\partial \text{rede}_i} = O_i * (1 - O_i)$$

Podemos substituir agora esse valor na equação para  $\delta_i$ , obtendo:

$$-\delta_i = O_i * (1 - O_i) * \sum_j -\delta_j * w_{ij}$$

Finalmente, podemos avaliar a sensibilidade do erro da rede na camada de saída aos pesos incidentes no nó oculto  $i$ . Examinamos o peso  $k$  do nó  $i$ ,  $w_k$ . Pela regra da cadeia:

$$\frac{\partial \text{Erro}}{\partial w_{ki}} = \frac{\partial \text{Erro}}{\partial \text{rede}_i} * \frac{\partial \text{rede}_i}{\partial w_{ki}} = -\delta_i * \frac{\partial \text{rede}_i}{\partial w_{ki}} = -\delta_i * x_k$$

onde  $x_k$  é a entrada  $k$  do nó  $i$ .

Agora, substituímos na equação o valor de  $-\delta_i$ :

$$\frac{\partial \text{Erro}}{\partial w_{ki}} = O_i * (1 - O_i) * \sum_j (-\delta_j * w_{ij}) * x_k$$

Como a minimização do erro requer que as variações dos pesos ocorram na direção do componente negativo do gradiente, obtemos o ajuste para o peso  $k$  do nó  $i$  multiplicando pelo negativo da constante de aprendizado:

$$\Delta w_{ki} = -c \frac{\partial \text{Erro}}{\partial w_{ki}} = c * O_i * (1 - O_i) * \sum_j (\delta_j * w_{ij}) * x_k.$$

Para redes com mais de uma camada oculta, o mesmo procedimento é aplicado recursivamente para propagar o erro da camada oculta  $n$  para a camada oculta  $n - 1$ .

Embora isso forneça uma solução para o problema do aprendizado em redes multicamadas, o algoritmo de retropropagação tem suas próprias dificuldades. Como no caso da subida de encosta, ele pode convergir para mínimos locais, como na Figura 11.8. Por fim, o custo computacional da retropropagação pode ser elevado, em especial quando a rede converge lentamente.

### 11.3.2 Retropropagação, exemplo 1: NETtalk

A NETtalk é um exemplo interessante de uma solução de rede neural para um problema de aprendizado difícil (Sejnowski e Rosenberg, 1987). Ela aprendeu a pronunciar textos em inglês. Essa pode ser uma tarefa difícil para uma abordagem explícita simbólica, por exemplo, um sistema baseado em regras, já que a pronúncia da língua inglesa é bastante irregular.

A NETtalk aprendeu a ler uma sequência de texto e a retornar um fonema e uma entonação associada a cada letra na sequência. Um fonema é a unidade básica sonora em uma linguagem; a entonação é a intensidade relativa daquele som. Como a pronúncia de uma única letra depende do seu contexto e das letras vizinhas, a NETtalk recebe uma janela de sete caracteres. Conforme o texto passa por essa janela, a NETtalk retorna um par fonema/entonação para cada letra.

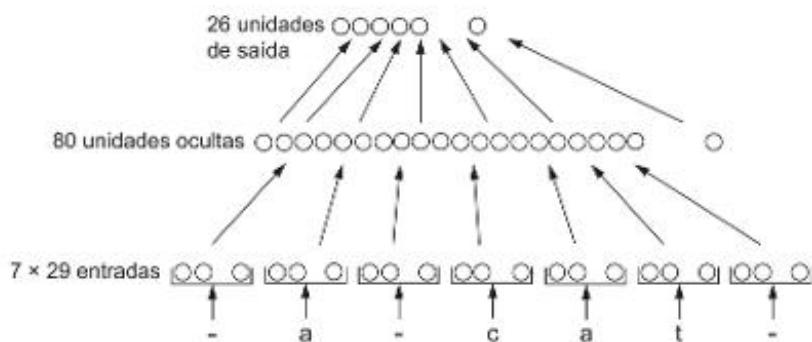
A Figura 11.11 mostra a arquitetura da NETtalk. A rede consiste em três camadas de unidades. As unidades de entrada correspondem à janela de sete caracteres no texto. Cada posição na janela é representada por 29 unidades, uma para cada letra do alfabeto, e 3 para pontuação e espaços. A letra em cada posição ativa a unidade correspondente. As unidades de saída codificam fonemas usando 21 características diferentes da articulação humana. As cinco unidades restantes codificam a entonação e as fronteiras silábicas. A NETtalk tem 80 unidades ocultas, 26 valores de saída e 18.629 conexões.

A NETtalk é treinada pela apresentação de uma janela de sete caracteres em que ela tenta pronunciar o caractere central. Comparando a sua pronúncia com a pronúncia correta, ela ajusta seus pesos usando retropropagação.

Esse exemplo ilustra uma série de propriedades interessantes das redes neurais, muitas das quais refletem a natureza do aprendizado humano. Por exemplo, o aprendizado, quando medido como uma porcentagem de respostas corretas, acontece rapidamente no início, mas se torna lento quando essa porcentagem aumenta. Como acontece com as pessoas, à medida que a rede aprende a pronunciar mais palavras, ela também pronuncia melhor novas palavras. Em experimentos em que foram alterados aleatoriamente alguns dos pesos de uma rede totalmente treinada, a rede se mostrou resistente a falhas, apresentando uma degradação suave conforme os pesos eram alterados. Os pesquisadores também descobriram que o reaprendizado em uma rede danificada é muito eficiente.

Outro aspecto interessante das redes multicamadas é o papel das camadas ocultas. Qualquer algoritmo de aprendizado deve aprender generalizações que se aplicam a ocorrências ainda não observadas no domínio do

**Figura 11.11** Topologia da rede de NETtalk.



problema. As camadas ocultas exercem um papel importante na generalização de uma rede neural. A NETtalk, assim como muitas outras redes de retropropagação, tem menos neurônios na camada oculta do que na camada de entrada. Já que um menor número de nós na camada oculta é usado para codificar a informação contida nos padrões de treinamento, deve ocorrer alguma forma de abstração. A codificação mais compacta implica que diferentes padrões na camada de entrada podem ser mapeados para padrões idênticos na camada oculta. Essa redução é uma forma de generalização.

A NETtalk aprende eficientemente, embora exija um grande número de exemplos de treinamento, bem como várias passagens repetidas pelos dados de treinamento. Em uma série de testes empíricos comparando a retropropagação com o ID3 nesse problema, Shavlik et al. (1991) constataram que os algoritmos tinham resultados equivalentes, embora o seu treinamento e o emprego dos dados fossem bastante diferentes. Eles avaliaram os algoritmos dividindo o conjunto total de exemplos em conjuntos separados de treinamento e de teste. Tanto o ID3 (Seção 9.3) como a NETtalk foram capazes de pronunciar corretamente cerca de 60 por cento dos dados de teste, após treinamento com 500 exemplos. Mas, enquanto o ID3 necessitou de apenas uma passagem pelos dados de treinamento, a NETtalk precisou de várias repetições do conjunto de treinamento. Nessa investigação, a NETtalk realizou 100 passagens pelos dados de treinamento.

Como demonstra o nosso exemplo, a relação entre os aprendizados conexionista e simbólico é mais complicada do que poderia parecer a princípio. No nosso próximo exemplo, mostraremos os detalhes de uma solução por retropropagação para o problema do OU-exclusivo.

### 11.3.3 Retropropagação, exemplo 2: OU-exclusivo

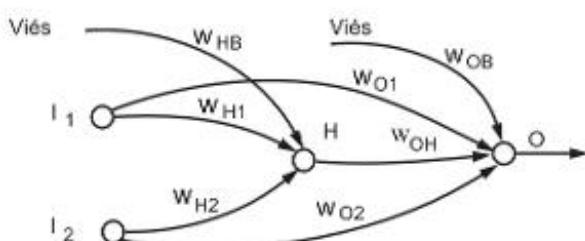
Terminamos essa seção apresentando uma solução simples com uma camada oculta para o problema do OU-exclusivo. A Figura 11.12 mostra uma rede com dois nós de entrada, um nó oculto e um nó de saída. A rede tem também dois nós de viés, o primeiro para o nó oculto e o segundo para o nó de saída. Os valores líquidos para os nós ocultos e de saída são calculados da maneira usual, como o produto entre os vetores de valores de entrada e os seus pesos treinados. O viés é adicionado a essa soma. Os pesos são treinados por retropropagação e a função de ativação é sigmoide.

Devemos notar que os nós de entrada estão também diretamente ligados, por pesos treinados, com o nó de saída. Essa ligação adicional pode fornecer uma rede com um número menor de nós na camada oculta e com convergência mais rápida. Na verdade, não há nada de único na rede da Figura 11.12; várias outras redes diferentes poderiam ser usadas para calcular o OU-exclusivo.

Treinamos a nossa rede inicializada aleatoriamente com múltiplos exemplos dos quatro padrões que representam os valores verdade do OU-exclusivo:

$$(0, 0) \rightarrow 0; (1, 0) \rightarrow 1; (0, 1) \rightarrow 1; (1, 1) \rightarrow 0$$

**Figura 11.12** Uma rede retropropagação para resolver o problema do OU-exclusivo. Os  $w_{ij}$  são os pesos e H é o nó oculto.



Um total de 1.400 ciclos de treinamento usando esses quatro exemplos produziu os seguintes valores, arredondados para o décimo mais próximo, para os parâmetros de pesos da Figura 11.12:

$$\begin{array}{llll} W_{H1} = -7,0 & W_{HB} = 2,6 & W_{O1} = -5,0 & W_{OH} = -11,0 \\ W_{H2} = -7,0 & W_{OB} = 7,0 & W_{O2} = -4,0 & \end{array}$$

Com os valores de entrada  $(0, 0)$ , a saída do nó oculto é:

$$f(0*(-7,0) + 0*(-7,0) + 1*2,6) = f(2,6) \rightarrow 1$$

O valor de saída do nó de saída para  $(0, 0)$  é:

$$f(0*(-5,0) + 0*(-4,0) + 1*(-11,0) + 1*(7,0)) = f(-4,0) \rightarrow 0$$

Com os valores de entrada  $(1, 0)$ , a saída do nó oculto é:

$$f(1*(-7,0) + 0*(-7,0) + 1*2,6) = f(-4,4) \rightarrow 0$$

O valor de saída do nó de saída para  $(1, 0)$  é:

$$f(1*(-5,0) + 0*(-4,0) + 0*(-11,0) + 1*(7,0)) = f(2,0) \rightarrow 1$$

O valor de entrada de  $(0, 1)$  é semelhante. Finalmente, verificamos a rede OU-exclusivo para os valores de entrada  $(1, 1)$ . A saída do nó oculto é:

$$f(1*(-7,0) + 1*(-7,0) + 1*2,6) = f(-11,4) \rightarrow 0$$

O valor de saída do nó de saída para  $(1, 1)$  é:

$$f(1*(-5,0) + 1*(-4,0) + 0*(-11,0) + 1*(7,0)) = f(-2,0) \rightarrow 0$$

O leitor pode ver que essa rede alimentada adiante com aprendizado por retropropagação construiu uma separação não linear desses pontos de dados. A função de limiar  $f$  é a sigmoide da Figura 11.7(b), os vieses aprendidos deslocaram essa função um pouco na direção positiva no eixo  $x$ .

Consideramos, a seguir, modelos de aprendizado competitivo.

## 11.4 Aprendizado competitivo

### 11.4.1 Aprendizado “o vencedor-leva-tudo” para classificação

O algoritmo “o vencedor-leva-tudo” (Kohonen, 1984; Hecht-Nielsen, 1987) funciona com um único nó, de uma camada de nós, que responde mais fortemente ao padrão de entrada. Esse algoritmo pode ser visto como uma competição entre um conjunto de nós da rede, como na Figura 11.13. Nessa figura, temos um vetor de valores de entrada,  $X = (x_1, x_2, \dots, x_m)$ , que passa por uma camada de nós da rede, A, B, ..., N. O diagrama mostra o nó B vencedor da competição, com um sinal de saída de 1.

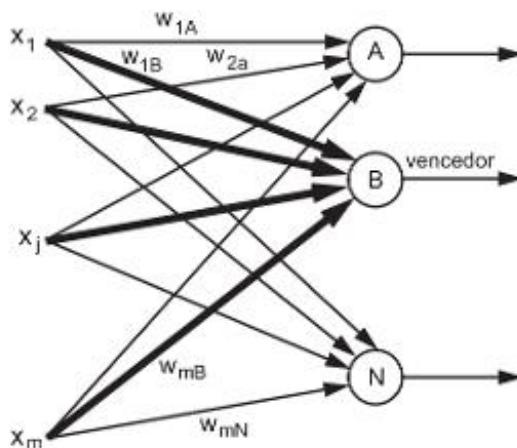
O aprendizado para o vencedor-leva-tudo é não supervisionado, pois o vencedor é determinado por um teste de “máxima ativação”. O vetor de pesos do vencedor é recompensado, trazendo os seus componentes mais perto do vetor de entrada. Para os pesos,  $W$ , do nó vencedor e os componentes  $X$  do vetor de entrada, o incremento é:

$$\Delta W^t = c(X^{t-1} - W^{t-1}),$$

onde  $c$  é uma constante de aprendizado pequena, que usualmente decresce com o prosseguimento do aprendizado. O vetor de pesos vencedor é, então, ajustado adicionando-se  $\Delta W^t$ .

Essa recompensa incrementa ou diminui cada componente do vetor de pesos do vencedor por uma fração da diferença  $x_i - w_i$ . O efeito, claro, é tornar o protótipo do nó vencedor mais parecido com o vetor de entrada. O algoritmo “o vencedor-leva-tudo” não precisa calcular diretamente os níveis de ativação para encontrar o nó com a

**Figura 11.13** Uma camada de nós para aplicação de um algoritmo “o vencedor-leva-tudo”. Os vetores de entrada antigos dão suporte ao nó vencedor.



resposta mais forte. O nível de ativação de um nó está diretamente relacionado com a proximidade do seu vetor de pesos ao vetor de entrada. Para um nó  $i$  com um vetor de pesos normalizado  $W_i$ , o nível de ativação,  $W_i X$ , é uma função da distância euclidiana entre  $W_i$  e o padrão de entrada  $X$ . Isso pode ser visto calculando-se a distância euclidiana, com  $W_i$  normalizado:

$$\|X - W_i\| = \sqrt{(X - W_i)^2} = \sqrt{X^2 - 2XW_i + W_i^2}$$

Dessa equação, pode-se ver que, para um conjunto normalizado de vetores de peso, o vetor de peso com a menor distância euclidiana,  $\|X - W\|$ , será o vetor com o valor máximo de ativação,  $WX$ . Em muitos casos, é mais eficiente determinar o vencedor calculando-se distâncias euclidianas do que comparando níveis de ativação sobre vetores de pesos normalizados.

Apresentamos a regra de aprendizado “o vencedor-leva-tudo” de Kohonen por várias razões. Primeiro, nós a consideramos um método de classificação e a comparamos com a classificação do perceptron. Segundo, ela pode ser combinada com outras arquiteturas de rede para oferecer modelos mais sofisticados de aprendizado. Examinamos a combinação do aprendizado de protótipo de Kohonen com o aprendizado supervisionado *outstar* para rede. Esse sistema híbrido, proposto originalmente por Robert Hecht-Nielsen (1987, 1990), é chamado de rede *contrapropagação*. Vemos, na Seção 11.4.3, como podemos descrever o aprendizado condicionado usando a rede contrapropagação.

Antes de encerrarmos essa introdução, há uma série de questões importantes relacionadas com algoritmos “o vencedor-leva-tudo”. Algumas vezes, um parâmetro de “consciência” é ajustado a cada iteração para evitar que nós individuais sejam vencedores com muita frequência. Isso faz com que todos os nós da rede participem pelo menos uma vez representando o espaço de padrões. Em alguns algoritmos, em vez de identificar um vencedor que leva tudo, um *conjunto* de nós próximos é selecionado e os pesos de cada nó são incrementados de modo diferente. Outra abordagem é recompensar de maneira diferente os nós vizinhos do vencedor. Os pesos são tipicamente inicializados com valores aleatórios e são, então, normalizados durante esse método de aprendizado (Zurada, 1992). Hecht-Nielsen (1990) mostra como os algoritmos “o vencedor-leva-tudo” podem ser vistos como equivalentes à análise k-médias de um conjunto de dados. Na próxima seção, apresentamos o método não supervisionado “o vencedor-leva-tudo” de Kohonen para o aprendizado de agrupamentos.

### 11.4.2 Rede de Kohonen para aprendizado de protótipos

A classificação de dados e o papel de protótipos no aprendizado são preocupações constantes de psicólogos, linguistas, cientistas da computação e cientistas cognitivos (Wittgenstein, 1953; Rosch, 1978; Lakoff, 1987). O papel de protótipos e da classificação na inteligência é, também, um tema constante deste livro. Demonstramos a classificação simbólica e os algoritmos de agrupamento probabilísticos com COBWEB e CLUSTER/2 na Seção 9.5. Em modelos conexionistas, demonstramos a classificação baseada no perceptron na Seção 11.2 e, agora, mostramos um algoritmo de agrupamento do tipo “o vencedor-leva-tudo” de Kohonen (1984).

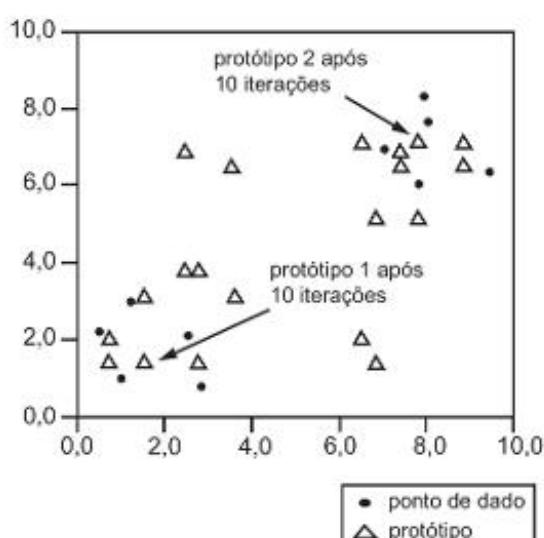
A Figura 11.14 apresenta novamente os pontos de dados da Tabela 11.3. Superposta a esses pontos se encontra uma série de protótipos criados durante o treinamento da rede. O algoritmo de treinamento do perceptron convergiu após certo número de iterações, resultando em uma configuração de pesos da rede que define uma separação linear entre as duas classes. Como vimos, a reta definida por esses pesos foi obtida calculando-se implicitamente o “centro” euclidiano da cada agrupamento. Esse centro de um agrupamento serve na classificação pelo perceptron como um protótipo da classe.

O aprendizado de Kohonen, por outro lado, é não supervisionado e tem um conjunto de protótipos criados aleatoriamente que é, então, refinado até representar explicitamente os agrupamentos de dados. Conforme o algoritmo avança, a constante de aprendizado é reduzida progressivamente, de modo que cada novo vetor de entrada cause menos perturbação nos protótipos.

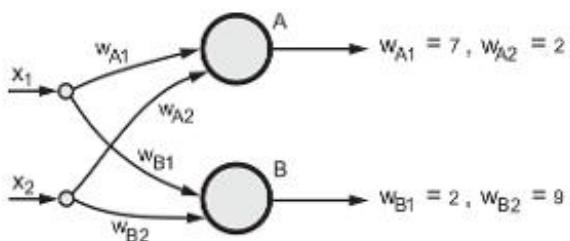
O aprendizado de Kohonen, assim como CLUSTER/2, tem um forte viés indutivo, pois o número de protótipos desejados é identificado explicitamente no início do algoritmo e continuamente refinado. Isso permite que o projetista do algoritmo da rede identifique um número específico de protótipos para representar os agrupamentos de dados. A rede contrapropagação (Seção 11.4.3) permite a manipulação adicional desse número selecionado de protótipos.

A Figura 11.15 é uma rede de aprendizado de Kohonen para a classificação dos dados da Tabela 11.3. Os dados são representados no espaço cartesiano bidimensional, de modo que os protótipos que representam os agrupamentos de dados também serão pares ordenados. Selecionamos dois protótipos, um para representar cada agrupamento de dados. Inicializamos aleatoriamente o nó A com (7, 2) e o nó B com (2, 9). A inicialização aleatória funciona apenas em problemas simples como o apresentado; uma alternativa é fazer os vetores de peso serem iguais a representantes de cada um dos agrupamentos.

**Figura 11.14** O uso de uma camada de Kohonen, não supervisionada, para gerar uma sequência de protótipos para representar as classes da Tabela 11.3.



**Figura 11.15** A arquitetura da rede com base no aprendizado de Kohonen para os dados da Tabela 11.3 e a classificação da Figura 11.14.



O nó vencedor terá um vetor de pesos que é mais próximo do vetor de entrada. Esse vetor de pesos para o nó vencedor será recompensado sendo movido para mais perto do dado de entrada, enquanto os pesos dos demais nós não são modificados. Como estamos calculando explicitamente a distância euclidiana do vetor de entrada para cada protótipo, não precisamos normalizar os vetores, como descrito na Seção 11.4.1.

O aprendizado de Kohonen é não supervisionado, pois uma medida simples da distância entre cada protótipo e o ponto de dado permite a seleção do vencedor. A classificação será “descoberta” no contexto dessa rede *auto-organizável*. Embora o aprendizado de Kohonen selecione pontos de dados para análise em uma ordem aleatória, tomamos os pontos da Tabela 11.3 na ordem de cima para baixo. Para o ponto (1, 1), medimos a distância a partir de cada protótipo:

$$\begin{aligned}\|(1, 1) - (7, 2)\| &= (1 - 7)^2 + (1 - 2)^2 = 37 \text{ e} \\ \|(1, 1) - (2, 9)\| &= (1 - 2)^2 + (1 - 9)^2 = 65.\end{aligned}$$

O nó A (7, 2) é o vencedor, pois ele é o mais próximo de (1, 1).  $\|(1, 1) - (7, 2)\|$  que representa a distância entre esses dois pontos; não precisamos aplicar a função raiz quadrada na medida da distância euclidiana porque a relação de magnitudes é invariante. Recompensamos, agora, o nó vencedor, usando a constante de aprendizado  $c$  fixada em 0,5. Para a segunda iteração:

$$\begin{aligned}W^2 &= W^1 + c(X^1 - W^1) \\ &= (7, 2) + 0,5((1, 1) - (7, 2)) = (7, 2) + 0,5((1 - 7), (1 - 2)) \\ &= (7, 2) + (-3, -0,5) = (4, 1,5)\end{aligned}$$

Na segunda iteração do algoritmo de aprendizado, temos, para o ponto de dado (9,4, 6,4):

$$\begin{aligned}\|(9,4, 6,4) - (4, 1,5)\| &= (9,4 - 4)^2 + (6,4 - 1,5)^2 = 53,17 \text{ e} \\ \|(9,4, 6,4) - (2, 9)\| &= (9,4 - 2)^2 + (6,4 - 9)^2 = 60,15.\end{aligned}$$

Novamente o nó A é o vencedor. O peso para a terceira iteração é:

$$\begin{aligned}W^3 &= W^2 + c(X^2 - W^2) \\ &= (4, 1,5) + 0,5((9,4, 6,4) - (4, 1,5)) \\ &= (4, 1,5) + (2,7, 2,5) = (6,7, 4)\end{aligned}$$

Na terceira iteração, temos, para o ponto de dado (2,5, 2,1):

$$\begin{aligned}\|(2,5, 2,1) - (6,7, 4)\| &= (2,5 - 6,7)^2 + (2,1 - 4)^2 = 21,25 \text{ e} \\ \|(2,5, 2,1) - (2, 9)\| &= (2,5 - 2)^2 + (2,1 - 9)^2 = 47,86.\end{aligned}$$

O nó A vence novamente, e prosseguimos calculando o seu novo vetor de pesos. A Figura 11.14 mostra a evolução do protótipo após 10 iterações. O algoritmo usado para gerar os dados da Figura 11.14 selecionou os dados aleatoriamente da Tabela 11.3, de modo que os protótipos mostrados diferirão daqueles criados agora. A melhoria progressiva dos protótipos pode ser vista pelo seu movimento em direção ao centro dos agrupamentos dos dados. Como já dissemos, esse é um algoritmo de reforço, não supervisionado e do tipo “o vencedor-leva-tudo”. Ele cons-

trói um conjunto de protótipos explícitos e evolutivos para representar os agrupamentos de dados. Vários pesquisadores, incluindo Zurada (1992) e Hecht-Nielsen (1990), observaram que a classificação não supervisionada de dados de Kohonen é basicamente a mesma que a análise k-médias.

A seguir, consideramos uma extensão por Grossberg, ou *outstar*, da análise de Kohonen de “o vencedor-leva-tudo”, um algoritmo que estende o poder de seleção de protótipos.

### 11.4.3 Redes *outstar* e contrapropagação

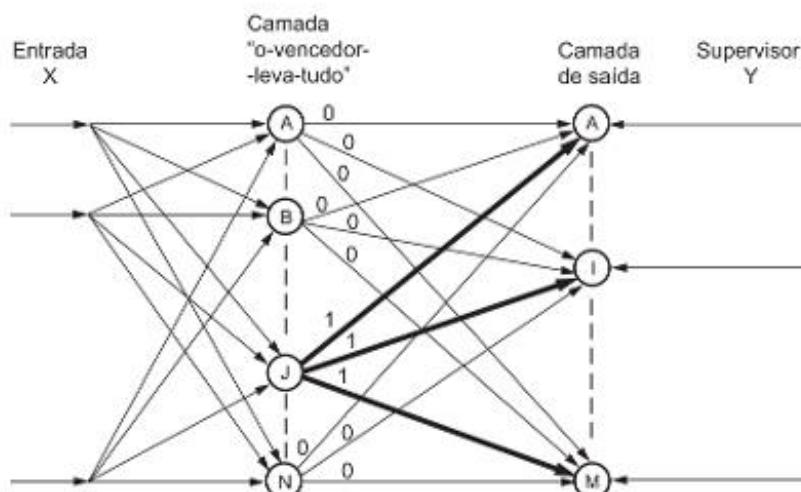
Até esse ponto, consideramos o agrupamento não supervisionado de dados de entrada. O aprendizado aqui requer pouco conhecimento prévio de um domínio de problema. A detecção gradual de características dos dados, bem como o histórico do treinamento, levam à identificação de classes e à descoberta de fronteiras entre elas. Uma vez que os pontos de dados estejam agrupados de acordo com similaridades nas suas representações vetoriais, um professor pode ajudar a calibrar ou dar nome às classes de dados. Isso é feito por uma forma de treinamento supervisionado, onde os nós de saída de uma rede “o vencedor-leva-tudo” são usados como entrada para uma segunda camada da rede. Nessa camada de saída, as decisões serão explicitamente reforçadas.

Esse treinamento supervisionado e as saídas reforçadas nos permitem, por exemplo, mapear os resultados de uma rede de Kohonen para um padrão de saída ou classe. Uma camada de Grossberg (1982, 1988), implementando um algoritmo chamado de *outstar*, nos permite fazer isso. A rede combinada, uma camada de Kohonen com uma camada de Grossberg, é chamada de *contrapropagação*, e foi inicialmente proposta por Robert Hecht-Nielsen (1987, 1990).

Na Seção 11.4.2, consideramos, com algum detalhe, a camada de Kohonen; aqui, consideramos a camada de Grossberg. A Figura 11.16 mostra uma camada de nós, A, B, ..., N, onde um nó, J, é selecionado como o vencedor. O aprendizado de Grossberg é supervisionado, uma vez que desejamos, com a ajuda de um professor, representado pelo vetor Y, reforçar o peso da conexão do nó J para o nó I na camada de saída que deve disparar. Com o aprendizado do *outstar*, identificamos e incrementamos o peso  $w_{JI}$ , na ligação de J para I.

Para treinar a rede contrapropagação, treinamos, inicialmente, a camada de Kohonen. Quando encontrado um vencedor, os valores em todos os elos que saem dele serão 1, enquanto todos os valores de saída de seus competidores permanecerão em 0. Esse nó junto a com todos os nós da camada de saída aos quais ele está conectado

**Figura 11.16** O *outstar* do nó J, o “vencedor” em uma rede “o-vencedor-leva-tudo”. O vetor Y supervisiona a resposta na camada de saída no treinamento de Grossberg. O *outstar* está destacado com todos os pesos em 1; todos os outros pesos são 0.



formam o que é chamado de um *outstar* (veja a Figura 11.16). O treinamento da camada de Grossberg é baseado em componentes *outstar*.

Se cada agrupamento de vetores de entrada representar uma única classe e se desejarmos que todos os membros de uma classe sejam mapeados para o mesmo valor na camada de saída, não precisamos de um treinamento iterativo. Precisamos apenas determinar qual nó na camada “o-vencedor-leva-tudo” está ligado a que classe é, então, atribuímos pesos desses nós para os nós de saída baseados na associação entre classes e valores de saída desejados. Por exemplo, se a J-ésima unidade “o vencedor-leva-tudo” vencer para todos os elementos do agrupamento para os quais  $I = 1$  é a saída desejada da rede, atribuímos  $w_{Jl} = 1$  e  $w_{JK} = 0$  para todos os outros pesos do *outstar* de J.

Se a saída desejada para elementos de um agrupamento variar, então é necessário um procedimento iterativo, usando o vetor de supervisão Y, para ajustar os pesos do *outstar*. O resultado desse procedimento de treinamento é que encontramos a *média* dos valores desejados de saída para os elementos de um agrupamento em particular. Treinamos os pesos das conexões do *outstar*, do nó vencedor para os nós de saída, de acordo com a equação:

$$W^{t+1} = W^t + c(Y - W^t)$$

onde c é uma constante de aprendizado, positiva e pequena,  $W^t$  é o vetor de pesos do componente *outstar* e Y é o vetor de saída desejado. Note que esse algoritmo de aprendizado tem o efeito de aumentar a conexão entre o nó J na camada de Kohonen e o nó I na camada de saída exatamente quando I for um nó vencedor com uma saída de 1 e a saída desejada de J for também 1. Isso o torna um exemplar do aprendizado hebbiano, uma forma de aprendizado na qual um caminho neural é reforçado toda vez que um nó contribui para o disparo de outro nó. Discutimos o aprendizado hebbiano com mais detalhes na Seção 11.5.

Aplicamos, a seguir, a regra de treinamento de uma rede contrapropagação para reconhecer os agrupamentos de dados da Tabela 11.3. Mostramos, também, com esse exemplo, como redes contrapropagação implementam o aprendizado de condicionamento. Suponha que o parâmetro  $x_1$  na Tabela 11.3 represente a velocidade de um motor em um sistema de propulsão e  $x_2$  represente a temperatura do motor. A velocidade e a temperatura do sistema são calibradas para produzir pontos de dados no intervalo [0, 10]. Nossa rede de monitoramento amostra pontos de dados a intervalos de tempo regulares. Sempre que a velocidade e a temperatura forem excessivamente altas, desejamos transmitir um alarme. Inicialmente, rotularemos os valores de saída da Tabela 11.3 como “seguro” para +1 e “perigoso” para -1. A nossa rede contrapropagação terá a forma da Figura 11.17.

Como sabemos exatamente quais valores cada nó vencedor da rede de Kohonen deve mapear na camada de saída da rede de Grossberg, poderíamos fixar diretamente esses valores. Entretanto, para demonstrar o aprendizado do *outstar*, treinaremos a rede utilizando a fórmula apresentada. Se tomarmos a decisão (arbitrária) em que o nó S na camada de saída deve sinalizar situações seguras e o nó D, situações perigosas, então os pesos do *outstar* para o nó A na camada de saída da rede de Kohonen deveriam ser [1, 0] e os pesos do *outstar* para B deveriam ser [0, 1]. Por causa da simetria da situação, mostraremos o treinamento apenas do *outstar* do nó A.

A rede de Kohonen deve ter estabilizado antes da rede de Grossberg ser treinada. Demonstramos a convergência dessa mesma rede de Kohonen na Seção 11.4.2. Os vetores de entrada para treinar o nó do *outstar* A são da forma  $[x_1, x_2, 1, 0]$ .  $x_1$  e  $x_2$  são os valores da Tabela 11.3 que foram agrupados pelo nó de saída A da rede de Kohonen, e as duas últimas componentes indicam que, quando A é o vencedor de Kohonen, seguro é “verdadeiro” e perigoso é “falso”, como na Figura 11.15. Inicializamos os pesos do *outstar* A em [0, 0] e usamos 0,2 como a constante de aprendizado:

$$W^1 = [0, 0] + 0,2[[1, 0] - [0, 0]] = [0, 0] + [0,2, 0] = [0,2, 0]$$

$$W^2 = [0,2, 0] + 0,2[[1, 0] - [0,2, 0]] = [0,2, 0] + [0,16, 0] = [0,36, 0]$$

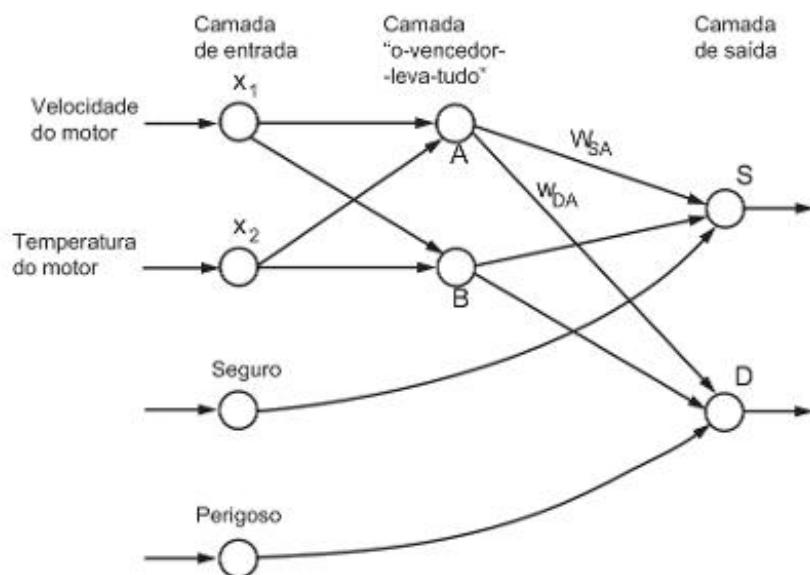
$$W^3 = [0,36, 0] + 0,2[[1, 0] - [0,36, 0]] = [0,36, 0] + [0,13, 0] = [0,49, 0]$$

$$W^4 = [0,49, 0] + 0,2[[1, 0] - [0,49, 0]] = [0,49, 0] + [0,10, 0] = [0,59, 0]$$

$$W^5 = [0,59, 0] + 0,2[[1, 0] - [0,59, 0]] = [0,59, 0] + [0,08, 0] = [0,67, 0]$$

Como podemos ver, com o treinamento esses pesos são movidos em direção a [1, 0]. Como nesse caso os elementos do agrupamento associado com A sempre mapeiam para [1, 0], é claro que poderíamos ter utilizado para o treinamento o algoritmo de atribuição simples em vez do algoritmo que calcula a média.

**Figura 11.17** Uma rede contrapropagação para reconhecer as classes na Tabela 11.3. Treinamos os pesos do *outstar* do nó A,  $W_{SA}$  e  $W_{DA}$ .



Mostramos, agora, que essa atribuição fornece a resposta apropriada para a rede contrapropagação. Quando o primeiro vetor de entrada da Tabela 11.3 é aplicado à rede da Figura 11.17, obtemos uma ativação de [1, 1] para os pesos do *outstar* do nó A e [0, 0] para o *outstar* de B. O produto escalar da ativação e dos pesos para o nó S da camada de saída é [1, 0] \* [1, 0]; isso fornece a ativação de 1 para o nó de saída S. Com os pesos de *outstar* de B treinados para [0, 1], a ativação para o nó D é [1, 0] \* [0, 1]; esses são os valores que esperávamos. Testando a segunda linha de pontos de dados da Tabela 11.3, obtemos uma ativação de [0, 0] para o nó A e [1, 1] para o nó B no nível de "o vencedor-leva-tudo". O produto escalar desses valores pelos pesos treinados fornece 0 para o nó S e 1 para D, novamente o que é esperado. O leitor pode continuar testando outros dados da Tabela 11.3.

De um ponto de vista cognitivo, podemos dar uma interpretação associacionista à rede contrapropagação. Considere novamente a Figura 11.17. O aprendizado na camada de Kohonen pode ser visto como aquele que adquire um estímulo condicionado, já que a rede está aprendendo padrões em eventos. Por outro lado, o aprendizado no nível de Grossberg é uma associação de nós (estímulos incondicionados) com uma resposta. Na nossa situação, o sistema aprende a transmitir um sinal de perigo quando os dados casam com certo padrão. Uma vez que a resposta apropriada seja aprendida, então, mesmo sem a ajuda contínua de um professor, o sistema responde adequadamente a novos dados.

Uma segunda interpretação cognitiva da contrapropagação é como reforço de ligações de memória para padrões de fenômenos. Isso é semelhante a construir uma tabela de consulta para respostas a padrões de dados.

A rede contrapropagação tem, em certos casos, uma vantagem considerável sobre a rede retropropagação. Como a retropropagação, ela é capaz de aprender classificações não linearmente separáveis. Entretanto, ela faz isso em virtude do pré-processamento que se passa na camada de Kohonen, onde o conjunto de dados é dividido em agrupamentos de dados homogêneos. Essa divisão pode resultar em uma vantagem significativa sobre a retropropagação em relação à taxa de aprendizado, já que a divisão explícita dos dados em agrupamentos separados substitui a busca normalmente extensa requerida pelas camadas ocultas da rede retropropagação.

#### 11.4.4 Máquinas de vetor de suporte (Harrison e Luger, 2002)

As máquinas de vetor de suporte (SVM, do inglês *Support Vector Machines*) oferecem outro exemplo de aprendizado competitivo. Na abordagem por vetor de suporte, medidas estatísticas são usadas para determinar um conjunto mínimo de pontos de dados (os vetores de suporte) que separam de forma máxima as ocorrências positivas e negativas de um conceito aprendido. Esses vetores de suporte, representando pontos de dados selecionados das ocorrências positivas e negativas do conceito, definem implicitamente um hiperplano separando esses dois conjuntos de dados. Por exemplo, a execução do algoritmo SVM identifica os pontos (2,5, 2,1) e (1,2, 3,0) como vetores de suporte para as ocorrências positivas e (7,0, 7,0) e (7,8, 6,1) para as ocorrências negativas dos dados da Tabela 11.3 e da Figura 11.5. Quando os vetores de suporte forem aprendidos, outros pontos de dados não precisarão mais ser retidos, pois apenas os vetores de suporte serão suficientes para determinar o hiperplano que separa os dois conjuntos.

A máquina de vetor de suporte é um classificador linear em que o aprendizado dos vetores de suporte é supervisionado. Considera-se que os dados para o aprendizado SVM são produzidos de forma independente e idêntica por uma distribuição fixa de dados, embora desconhecida. O hiperplano, definido implicitamente pelos próprios vetores de suporte, divide as instâncias de dados positivas das negativas. Os pontos de dados mais próximos do hiperplano estão na *margem de decisão* (Burges, 1998). Qualquer adição ou remoção de um vetor de suporte muda o limite do hiperplano. Conforme observado anteriormente, após o término do treinamento, é possível reconstruir o hiperplano e classificar novos conjuntos de dados a partir apenas dos vetores de suporte.

O algoritmo SVM classifica os elementos de dados pelo cálculo da distância de um ponto de dados ao hiperplano que separa os conjuntos, como um problema de otimização. O controle bem-sucedido da flexibilidade aumentada dos espaços de características, dos parâmetros (quase sempre transformados) das ocorrências a serem aprendidas, requer uma sofisticada teoria de generalização. Essa teoria precisa ser capaz de descrever com precisão as características que precisam ser controladas para formar uma boa generalização. Dentro da estatística, essa questão é conhecida como o *estudo das taxas de convergência uniforme*. Já vimos um exemplo disso na Seção 10.4.2, onde foi apresentado o modelo de aprendizado provável aproximadamente correto, ou modelo PAC. Os resultados do aprendizado PAC podem ser vistos como estabelecendo limites sobre o número de exemplos necessários para garantir um limite de erro específico. Para essa tarefa de generalização, empregam-se técnicas de compactação de dados Bayesianas ou outras. No aprendizado SVM, geralmente é utilizada a teoria de Vapnik e Chervonenkis.

A dimensão de Vapnik Chervonenkis (VC) é definida como o número máximo de pontos de treinamento que podem ser divididos em duas categorias por um conjunto de funções (Burges, 1998). Assim, a teoria VC oferece um limite independente de distribuição sobre a generalização da hipótese consistente (Cristianini e Shawe-Taylor, 2000). O algoritmo SVM usa essa teoria VC para calcular o hiperplano e controla a margem de erro para a precisão das generalizações, às vezes chamada de *capacidade* da função.

As SVMs usam uma medida de similaridade de produto escalar para mapear dados de um espaço de características. Os resultados de produto escalar representando vetores mapeados são combinados linearmente por pesos determinados pela solução de um programa quadrático (Scholkopf et al., 1998). Uma função de núcleo, como um polinômio, um spline ou uma gaussiana, é usada para criar o mapeamento do vetor de características, onde a escolha do núcleo é determinada pela distribuição do problema. As SVMs calculam distâncias para determinar a classificação do elemento de dados. Essas regras de decisão criadas pela SVM representam regularidades estatísticas nos dados. Quando a SVM é treinada, a classificação de novos pontos de dados é simplesmente uma questão de comparação com os vetores de suporte. Nos vetores de suporte, as características críticas caracterizando o conceito aprendido são agrupadas em um lado do hiperplano, aquelas descrevendo sua negação no outro, e as características que não discriminam não são usadas.

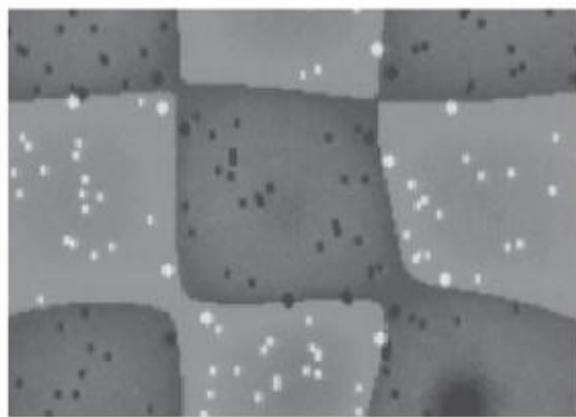
Para o algoritmo do perceptron (Seção 11.2), a separabilidade linear dos dados é importante: se os dados não forem separáveis, o algoritmo não convergirá. A SVM, como alternativa, tenta maximizar a margem de decisão e é mais robusta em sua capacidade de lidar com a separação fraca causada pela sobreposição de pontos de dados. Ela é capaz de usar variáveis de *folga* para relaxar as restrições lineares e achar uma margem flexível, com valores que indicam o nível de confiança do limite de classificação (Cristianini e Shawe-Taylor, 2000). Como resultado, alguns pontos de dados atípicos podem ser classificados erroneamente para produzir o hiperplano e, assim, a margem de decisão será estreitada quando os dados forem ruidosos.

As SVMs podem ser generalizadas a partir de problemas de classificação de duas categorias para a discriminação de múltiplas classes, executando repetidamente a SVM em cada categoria de interesse contra todas as outras categorias. SVMs são mais adequadas para problemas com dados numéricos do que de dados categóricos; como resultado, sua aplicabilidade é limitada para muitos problemas de categorização clássicos com limites qualitativos. Sua força está em suas bases matemáticas: a minimização de uma função quadrática convexa sob restrições de desigualdade linear.

As SVMs se aplicam a muitas situações de aprendizado, incluindo a classificação de páginas web. Na categorização de texto, a presença de palavras de busca e outras relacionadas são ponderadas. Cada documento, então, transforma-se em dados de entrada para serem categorizados pela SVM com base na informação de frequência de palavra (Harrison e Luger, 2002). Elas também são usadas para reconhecimento de imagem, focalizando a detecção de bordas e descrição de forma usando informações de escala de cinzas ou intensidade de cor (Cristianini e Shawe-Taylor, 2000). Na Figura 11.18, adaptada de Cristianini e Shawe-Taylor (2000), a máquina de vetor de suporte discrimina limites em um tabuleiro de xadrez. Outros detalhes em SVMs e o algoritmo de aprendizado SVM completo podem ser encontrados em Burges (1998).

**Figura 11.18** Uma SVM aprendendo os limites de um tabuleiro de xadrez a partir de pontos gerados de acordo com a distribuição uniforme usando núcleos gaussianos. Os pontos são os dados, com os pontos maiores compreendendo o conjunto de vetores de suporte; as áreas mais escuras indicam a confiança na classificação. Adaptado de Cristianini e Shawe-Taylor (2000).

---



---

## 11.5 Aprendizado hebbiano por coincidência

---

### 11.5.1 Introdução

A teoria de Hebb para o aprendizado é baseada na observação de que, em sistemas biológicos, quando um neurônio contribui para o disparo de outro neurônio, a conexão entre os dois neurônios é reforçada. Hebb (1949) afirmou que:

Quando um axônio de uma célula A está próximo o suficiente para excitar uma célula B e repetidamente e persistentemente participa do seu disparo, então ocorre um processo de crescimento ou modificação metabólica em uma ou nas duas células, de modo que a eficiência de A, como uma das células que causam o disparo de B, é aumentada.

O aprendizado hebbiano é atrativo porque estabelece conceitos de recompensa baseados em comportamento no nível neural. As pesquisas neurofisiológicas confirmam que a ideia de Hebb de que a proximidade temporal do disparo de neurônios conectados pode modificar a força sináptica está, pelo menos, muito próxima de ser correta, muito embora isso ocorra de uma forma muito mais complexa do que no simples “aumento de eficiência”. A lei de

aprendizado apresentada nesta seção é agora conhecida como aprendizado hebbiano, embora as suas ideias tenham sido mais abstratas. Esse aprendizado pertence à categoria das leis de aprendizado por *coincidência* que causam modificações em pesos, em resposta a eventos localizados em processos neurais. Descrevemos as leis de aprendizado dessa categoria por suas propriedades locais temporais e espaciais.

O aprendizado hebbiano tem sido usado em uma série de arquiteturas de rede. Ele é usado tanto em aprendizado supervisionado quanto no modo não supervisionado. O efeito do reforço da conexão entre dois neurônios, quando um contribui para o disparo do outro, pode ser simulado matematicamente ajustando-se o peso nessa conexão por uma constante multiplicada pelo sinal do produto de seus valores de saída.

Vejamos como isso funciona. Suponha que os neurônios  $i$  e  $j$  estejam conectados de modo que a saída de  $i$  seja a entrada de  $j$ . Podemos definir o ajuste do peso na conexão entre eles,  $\Delta W$ , como o sinal de  $c^*$  ( $O_i * O_j$ ), onde  $c$  é uma constante que controla a taxa de aprendizado. Na Tabela 11.4,  $O_i$  é o sinal do valor de saída de  $i$  e  $O_j$  é o valor da saída de  $j$ . Da primeira linha da tabela, vemos que quando ambos,  $O_i$  e  $O_j$ , são positivos, o ajuste de peso,  $\Delta W$ , é positivo. Isso tem o efeito de reforçar a conexão entre  $i$  e  $j$  quando  $i$  contribuiu para o “disparo” de  $j$ .

Na segunda e terceira linhas da Tabela 11.4,  $i$  e  $j$  têm sinais opostos. Como os seus sinais diferem, queremos inibir a contribuição de  $i$  para o valor de saída de  $j$ . Portanto, ajustamos o peso da conexão por um incremento negativo. Finalmente, na quarta linha,  $i$  e  $j$  têm novamente o mesmo sinal. Isso significa que incrementamos a força da sua conexão. Esse mecanismo de ajuste de pesos tem o efeito de reforçar o caminho entre neurônios quando eles têm sinais similares e inibi-los, caso contrário.

Nas próximas seções, consideraremos dois tipos de aprendizado hebbiano: o não supervisionado e o supervisionado. Começamos examinando uma forma não supervisionada.

**Tabela 11.4** Sinais e produtos dos sinais dos valores de saída dos nós.

$O_i$	$O_j$	$O_i * O_j$
+	+	+
+	-	-
-	+	-
-	-	+

### 11.5.2 Um exemplo de aprendizado hebbiano não supervisionado

Devemos recordar que, no aprendizado não supervisionado, não dispomos de um crítico que forneça o valor de saída “correto”; assim, os pesos são modificados apenas como uma função dos valores de entrada e de saída do neurônio. O treinamento dessa rede tem o efeito de reforçar as respostas da rede a padrões que ela tenha visto. No exemplo a seguir, mostramos como as técnicas hebbianas podem ser usadas para modelar o aprendizado por resposta condicionada, em que um estímulo arbitrariamente selecionado pode ser usado como uma condição para uma resposta desejada.

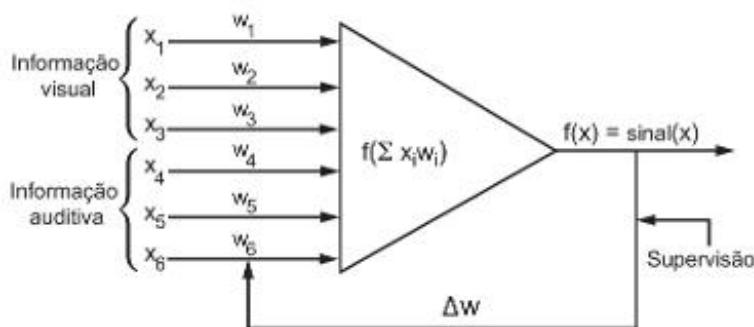
Os pesos podem ser ajustados,  $\Delta W$ , para um nó  $i$  no aprendizado hebbiano não supervisionado por:

$$\Delta W = c * f(X, W) * X$$

onde  $c$  é a constante de aprendizado, um número pequeno positivo,  $f(X, W)$  é a saída de  $i$  e  $X$  é o vetor de entrada de  $i$ .

Mostramos, agora, como uma rede pode usar o aprendizado hebbiano para transferir a sua resposta de um estímulo primário ou incondicionado para um estímulo condicionado. Isso nos permite modelar o tipo de aprendizado estudado em experimentos de Pavlov, em que o toque simultâneo de uma campainha cada vez que comida era apresentada causava a transferência da resposta de salivação de um cachorro à comida para a campainha. A rede da Figura 11.19 tem duas camadas, uma de entrada com seis nós e uma de saída com um

**Figura 11.19** Um exemplo de neurônio para a aplicação de um nó hebbiano híbrido em que o aprendizado é supervisionado.



nó. A camada de saída retorna +1, significando que a saída do neurônio disparou, ou então -1, significando que ele está em repouso.

Escolhemos como constante de aprendizado o número real positivo pequeno 0,2. Nesse exemplo, treinamos a rede com o padrão [1, -1, 1, -1, 1, -1], que é a concatenação dos dois padrões, [1, -1, 1] e [-1, 1, -1]. O padrão [1, -1, 1] representa o estímulo incondicionado e [-1, 1, -1] representa o novo estímulo.

Supomos que a rede já responda positivamente ao estímulo incondicionado, mas seja neutra em relação ao novo estímulo. Simulamos a resposta positiva da rede ao estímulo incondicionado com o vetor de pesos [1, -1, 1], casando exatamente com o padrão de entrada, enquanto a resposta neutra da rede ao estímulo novo é simulada pelo vetor de pesos [0, 0, 0]. A concatenação desses dois vetores de pesos nos dá o vetor de pesos inicial para a rede, [1, -1, 1, 0, 0, 0].

Treinamos, agora, a rede com o padrão de entrada visando induzir uma configuração de pesos que produza uma resposta positiva da rede ao novo estímulo. A primeira iteração da rede fornece:

$$\begin{aligned} W \cdot X &= (1 * 1) + (-1 * -1) + (1 * 1) + (0 * -1) + (0 * 1) + (0 * -1) \\ &= (1) + (1) + (1) = 3 \\ f(3) &= \text{sinal}(3) = 1. \end{aligned}$$

Criamos, agora, o novo peso W<sup>2</sup>:

$$\begin{aligned} W^2 &= [1, -1, 1, 0, 0, 0] + 0,2 * (1) * [1, -1, 1, -1, 1, -1] \\ &= [1, -1, 1, 0, 0, 0] + [0,2, -0,2, 0,2, -0,2, 0,2, -0,2] \\ &= [1,2, -1,2, 1,2, -0,2, 0,2, -0,2] \end{aligned}$$

A seguir, expomos a rede ajustada ao padrão de entrada original:

$$\begin{aligned} W \cdot X &= (1,2 * 1) + (-1,2 * -1) + (1,2 * 1) + (-0,2 * -1) + (0,2 * 1) + (-0,2 * -1) \\ &= (1,2) + (1,2) + (1,2) + (0,2) + (0,2) + (0,2) = 4,2 \text{ e} \\ \text{sinal}(4,2) &= 1. \end{aligned}$$

Criamos, agora, o novo peso W<sup>3</sup>:

$$\begin{aligned} W^3 &= [1,2, -1,2, 1,2, -0,2, 0,2, -0,2] + 0,2 * (1) * [1, -1, 1, -1, 1, -1] \\ &= [1,2, -1,2, 1,2, -0,2, 0,2, -0,2] + [0,2, -0,2, 0,2, -0,2, 0,2, -0,2] \\ &= [1,4, -1,4, 1,4, -0,4, 0,4, -0,4] \end{aligned}$$

Pode-se ver, agora, que o produto escalar dos vetores, W \* X, continuará a crescer na direção positiva, com o valor absoluto de cada elemento do vetor de pesos aumentando de 0,2 a cada novo ciclo de treinamento. Após mais 10 iterações do treinamento hebbiano, o vetor de pesos será:

$$W^{13} = [3,4, -3,4, 3,4, -2,4, 2,4, -2,4]$$

Usaremos, agora, esse vetor de pesos treinado para testar a resposta da rede a dois padrões parciais. Desejamos ver se a rede continua a responder positivamente ao estímulo incondicionado e, mais importante que isso, se a rede agora adquiriu uma resposta positiva ao novo estímulo condicionado. Testamos a rede primeiro com o estímulo incondicionado  $[1, -1, 1]$ . Preenchemos aleatoriamente os três últimos argumentos do vetor de entrada com valores 1 e -1. Por exemplo, testamos a rede com o vetor  $[1, -1, 1, 1, 1, -1]$ :

$$\begin{aligned}\text{sinal}(W^T X) &= \text{sinal}((3,4 \cdot 1) + (-3,4 \cdot -1) + (3,4 \cdot 1) \\ &\quad + (-2,4 \cdot 1) + (2,4 \cdot 1) + (-2,4 \cdot -1)) \\ &= \text{sinal}(3,4 + 3,4 + 3,4 - 2,4 + 2,4 + 2,4) \\ &= \text{sinal}(12,6) = +1.\end{aligned}$$

Assim, a rede ainda responde positivamente ao estímulo incondicionado original. Realizamos, agora, um segundo teste usando o estímulo incondicionado original e um vetor aleatório diferente nas últimas três posições  $[1, -1, 1, 1, -1, -1]$ :

$$\begin{aligned}\text{sinal}(W^T X) &= \text{sinal}((3,4 \cdot 1) + (-3,4 \cdot -1) + (3,4 \cdot 1) \\ &\quad + (-2,4 \cdot 1) + (2,4 \cdot -1) + (-2,4 \cdot -1)) \\ &= \text{sinal}(3,4 + 3,4 + 3,4 - 2,4 - 2,4 + 2,4) \\ &= \text{sinal}(7,8) = +1.\end{aligned}$$

O segundo vetor também produz uma resposta positiva da rede. Na verdade, notamos, nesses dois exemplos, que a sensibilidade da rede ao estímulo original, como medido pela sua ativação bruta, foi reforçada devido à exposição repetida desse estímulo.

Testamos, agora, a resposta da rede ao novo padrão de estímulo,  $[-1, 1, -1]$ , codificado nas últimas três posições do vetor de entrada. Preenchemos as três primeiras posições do vetor com atribuições aleatórias do conjunto  $\{1, -1\}$  e testamos a rede com o vetor  $[1, 1, 1, -1, 1, -1]$ :

$$\begin{aligned}\text{sinal}(W^T X) &= \text{sinal}((3,4 \cdot 1) + (-3,4 \cdot -1) + (3,4 \cdot 1) \\ &\quad + (-2,4 \cdot 1) + (2,4 \cdot 1) + (-2,4 \cdot -1)) \\ &= \text{sinal}(3,4 - 3,4 + 3,4 + 2,4 + 2,4 + 2,4) \\ &= \text{sinal}(10,6) = +1.\end{aligned}$$

O padrão do estímulo secundário também é reconhecido!

Fazemos um último experimento, com os vetores dos padrões ligeiramente degradados. Isso poderia representar a situação de estímulo, em que os sinais de entrada estariam ligeiramente alterados, talvez por terem sido utilizados uma nova comida e um som diferente da campainha. Testamos a rede com o vetor de entrada  $[1, -1, -1, 1, 1, -1]$ , em que os primeiros três parâmetros correspondem ao estímulo incondicionado original, excetuando-se um, e os últimos três parâmetros formam o estímulo condicionado, alterando-se um:

$$\begin{aligned}\text{sinal}(W^T X) &= \text{sinal}((3,4 \cdot 1) + (-3,4 \cdot -1) + (3,4 \cdot 1) \\ &\quad + (-2,4 \cdot 1) + (2,4 \cdot 1) + (-2,4 \cdot -1)) \\ &= \text{sinal}(3,4 + 3,4 - 3,4 - 2,4 + 2,4 + 2,4) \\ &= \text{sinal}(5,8) = +1.\end{aligned}$$

Até mesmo o estímulo parcialmente degradado é reconhecido!

O que o modelo de aprendizado hebbiano produziu? Criamos uma associação entre um estímulo novo e uma resposta antiga pela apresentação repetida dos estímulos novo e antigo juntos. A rede aprende a transferir a sua resposta ao novo estímulo sem qualquer supervisão. Essa sensibilidade reforçada também faz com que a rede responda da mesma maneira a uma versão um pouco degradada dos estímulos. Isso deve-se à utilização do aprendizado hebbiano por coincidência para aumentar a força da resposta da rede ao padrão total, um aumento que tem o efeito de aumentar a intensidade da resposta da rede a cada componente individual do padrão.

### 11.5.3 Aprendizado hebbiano supervisionado

A regra do aprendizado hebbiano é baseada no princípio de que a força da conexão entre neurônios é aumentada sempre que um neurônio contribuir com o disparo de outro neurônio. Esse princípio pode ser adaptado em uma situação de aprendizado supervisionado, baseando o ajuste de pesos para essa conexão na saída desejada do neurônio, em vez da saída real. Por exemplo, se a entrada do neurônio A para o neurônio B for positiva e a resposta *desejada* do neurônio B for uma saída positiva, então o peso na conexão de A para B é aumentado.

Examinaremos, agora, uma aplicação do aprendizado hebbiano supervisionado mostrando como uma rede pode ser treinada para reconhecer um conjunto de associações entre padrões. As associações são dadas por um conjunto de pares ordenados,  $\{<X_1, Y_1>, <X_2, Y_2>, \dots, <X_l, Y_l>\}$ , onde  $X_i$  e  $Y_i$  são os padrões de vetores a serem associados. Suponha que o comprimento de  $X_i$  seja  $n$  e o de  $Y_i$  seja  $m$ . Projetamos a rede para que ela se ajuste explicitamente a essa situação. Portanto, ela tem duas camadas, uma de entrada de tamanho  $n$  e uma de saída de tamanho  $m$ , como vemos na Figura 11.20.

A fórmula de aprendizado para essa rede pode ser derivada partindo-se da fórmula de aprendizado hebbiano da seção anterior:

$$\Delta W = c * f(X, W) * X$$

onde  $f(X, W)$  é a saída real do nó da rede. No aprendizado supervisionado, substituimos essa saída real de um nó pelo vetor de saída desejado  $D$ , produzindo a fórmula:

$$\Delta W = c * D * X$$

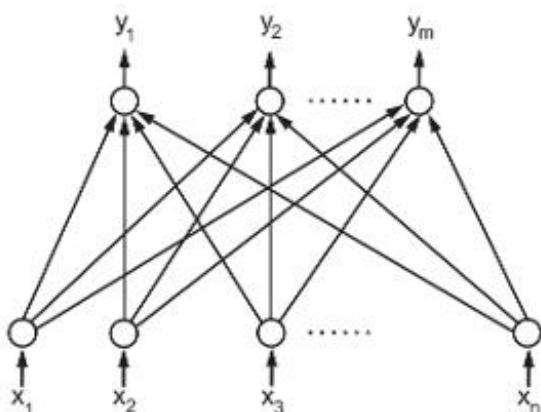
Dado um par de vetores,  $<X, Y>$  do conjunto de pares associados, aplicamos essa regra de aprendizado ao  $k$ -ésimo nó na camada de saída:

$$\Delta W_{ik} = c * d_k * x_i$$

onde  $\Delta W_{ik}$  é o ajuste do peso da  $i$ -ésima entrada para o  $k$ -ésimo nó na camada de saída,  $d_k$  é a saída desejada do  $k$ -ésimo nó e  $x_i$  é o  $i$ -ésimo elemento de  $X$ . Aplicamos essa fórmula para ajustar todos os pesos de todos os nós da camada de saída. O vetor  $<x_1, x_2, \dots, x_n>$  é apenas o vetor  $X$ , e o vetor  $<d_1, d_2, \dots, d_m>$  é o vetor de saída  $Y$ . Aplicando a fórmula aos ajustes individuais de peso ao longo de toda a camada de saída e agrupando termos, podemos escrever a fórmula para o ajuste de pesos na camada de saída como:

$$\Delta W = c * Y * X,$$

**Figura 11.20** Uma rede hebbiana supervisionada para aprender associações de padrões.



onde o produto  $Y \cdot X$  é o *produto externo dos vetores*. O produto externo  $YX$  é definido genericamente como a matriz:

$$YX = \begin{bmatrix} y_1 \cdot x_1 & y_1 \cdot x_2 & \dots & y_1 \cdot x_m \\ y_2 \cdot x_1 & y_2 \cdot x_2 & \dots & y_2 \cdot x_m \\ \dots & \dots & \dots & \dots \\ y_n \cdot x_1 & y_n \cdot x_2 & \dots & y_n \cdot x_m \end{bmatrix}$$

Para treinar a rede com todo o conjunto de pares associados, realizamos ciclos por meio desses pares, ajustando o peso para cada par  $\langle X_i, Y_i \rangle$  de acordo com a fórmula:

$$W^{t+1} = W^t + c * Y_i * X_i$$

Para o conjunto de treinamento inteiro, obtemos:

$$W^1 = W^0 + c (Y_1 * X_1 + Y_2 * X_2 + \dots + Y_t * X_t),$$

onde  $W^0$  é a configuração de pesos inicial. Se inicializarmos  $W^0$  com o vetor 0,  $\langle 0, 0, \dots, 0 \rangle$  e fixarmos a constante de aprendizado  $c$  em 1, obteremos a seguinte fórmula para atribuição dos pesos da rede:

$$W = Y_1 * X_1 + Y_2 * X_2 + \dots + Y_t * X_t.$$

Uma rede que mapeia vetores de entrada em vetores de saída usando essa fórmula para atribuição de pesos é chamada de *associador linear*. Mostramos que redes do tipo associador linear são baseadas na regra de aprendizado hebbiano. Na prática, essa fórmula pode ser aplicada diretamente na inicialização dos pesos da rede sem treinamento explícito.

Analisamos, a seguir, as propriedades do associador linear. Esse modelo, como acabamos de ver, armazena múltiplas associações em uma matriz de vetores de pesos. Isso abre a possibilidade de interações entre padrões armazenados. Analisaremos os problemas criados por essas interações nas próximas seções.

#### 11.5.4 A memória associativa e o associador linear

A rede do *associador linear* foi proposta originalmente por Tuevo Kohonen (1972) e James Anderson et al. (1977). Nesta seção, apresentamos a rede do associador linear como um método para armazenar e recuperar padrões de memória. Examinamos diferentes formas de recuperação de memória, incluindo os modelos heteroassociativo, autoassociativo e interpolativo. Analisamos a rede do associador linear como uma implementação de memória interpolativa baseada no aprendizado hebbiano. Encerramos essa seção considerando problemas com interferências ou “linhas cruzadas”. Esses problemas surgem quando se codifica múltiplos padrões na memória.

Começamos o estudo das memórias com algumas definições. Padrões e valores de memória são representados como vetores. Há sempre um viés indutivo ao se reduzir a representação de um problema a um conjunto de vetores de características. As associações que devem ser armazenadas na memória são representadas como pares de vetores,  $\{\langle X_1, Y_1 \rangle, \langle X_2, Y_2 \rangle, \dots, \langle X_t, Y_t \rangle\}$ . Para cada par de vetores  $\langle X_i, Y_i \rangle$ , o padrão  $X_i$  é uma chave para a recuperação do padrão  $Y_i$ . Existem três tipos de memórias associativas:

- 1. Heteroassociativa:** esse é um mapeamento de  $X$  para  $Y$ , tal que, se um vetor arbitrário  $X$  estiver mais próximo do vetor  $X_i$  do que qualquer outro exemplo, então o vetor associado  $Y_i$  será retornado.
- 2. Autoassociativa:** esse mapeamento é o mesmo que o heteroassociativo, exceto que  $X_i = Y_i$  para todos os pares de exemplos. Como cada padrão  $X_i$  está relacionado consigo mesmo, essa forma de memória é usada basicamente quando um padrão parcial, ou degradado, serve para evocar o padrão completo.
- 3. Interpolativa:** esse é um mapeamento  $\Phi$  de  $X$  para  $Y$ , tal que, quando  $X$  difere de um exemplo, isto é,  $X = X_i + \Delta_i$ , então a saída de  $\Phi(X) = \Phi(X_i + \Delta_i) = Y_i + E$ , onde  $E = \Phi(\Delta_i)$ . Se o vetor de entrada for um dos exemplos  $X_i$ , o  $Y_i$  associado é recuperado. Se ele diferir de um dos exemplos por um vetor  $\Delta$ , então o vetor de saída também diferirá por um vetor diferença  $E$ , onde  $E = \Phi(\Delta)$ .

As memórias autoassociativas e heteroassociativas são usadas para recuperar um dos exemplares originais. Elas constituem uma memória no sentido verdadeiro, uma vez que o padrão que é recuperado é uma cópia literal do padrão armazenado. Podemos, também, desejar construir um padrão de saída que difira dos padrões armazenados na memória de uma forma sistemática. Essa é a função de uma memória interpolativa.

A rede do associador linear da Figura 11.21 implementa uma forma de memória interpolativa. Como mostra a Seção 11.5.3, ela é baseada no modelo de aprendizado hebbiano. A inicialização dos pesos é descrita pela equação derivada na Seção 11.5.3:

$$W = Y_1 * X_1 + Y_2 * X_2 + \dots + Y_t * X_t$$

Dada essa atribuição de pesos, a rede recuperará com um casamento exato um dos exemplos; caso contrário, ela produzirá um mapeamento interpolativo.

A seguir, introduzimos alguns conceitos e a notação que nos ajudarão a analisar o comportamento dessa rede. Primeiro, introduzimos uma métrica que nos permite definir exatamente a distância entre dois vetores. Todos os nossos vetores de padrões nos exemplos são vetores de *Hamming*, isto é, vetores compostos apenas de valores +1 e -1. Usamos a *distância de Hamming* para descrever a distância entre dois vetores de Hamming. Formalmente, definimos um espaço de Hamming:

$$H^n = \{X = (x_1, x_2, \dots, x_n)\}, \text{ onde cada } x_i \text{ pertence ao conjunto }\{+1, -1\}.$$

A distância de Hamming é definida para quaisquer dois vetores de um espaço de Hamming como:

$\|X, Y\|$  = o número de componentes pelos quais X e Y diferem.

Por exemplo, a distância de Hamming, no espaço de dimensão quatro, entre:

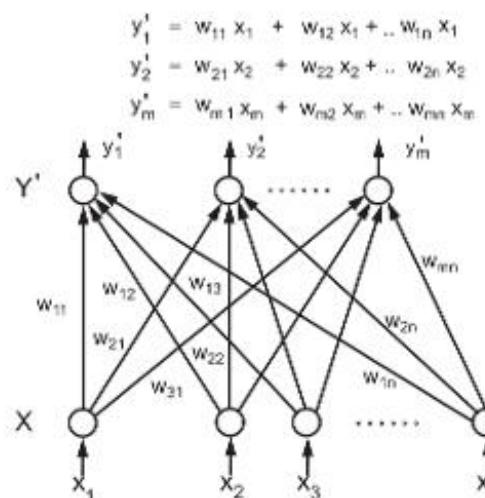
(1, -1, -1, 1) e (1, 1, -1, 1) é 1

(-1, -1, -1, 1) e (1, 1, 1, -1) é 4

(1, -1, 1, -1) e (1, -1, 1, -1) é 0.

Precisamos, ainda, de duas outras definições. Primeiro, o complemento de um vetor de Hamming é aquele em que cada elemento é trocado: +1 para -1 e -1 para +1. Por exemplo, o complemento de (1, -1, -1, -1) é (-1, 1, 1, 1).

**Figura 11.21** Rede de associação linear. O vetor  $X_i$  é apresentado como entrada, e o vetor associado  $Y'$  é produzido como saída.  $y'_i$  é uma combinação linear da entrada  $X$ . Durante o treinamento, cada  $y'_i$  é fornecido com os seus sinais de saída corretos.



Segundo, definimos a *ortonormalidade* de vetores. Vetores que são ortonormais são ortogonais, ou perpendiculares, e de comprimento unitário. Dois vetores ortonormais, quando multiplicados entre si pelo *produto escalar*, têm a soma de todos os seus termos produtos cruzados zerada. Assim, em um conjunto de vetores ortonormais, quando dois vetores quaisquer,  $X_i$  e  $X_j$ , são multiplicados, o produto é 0, a menos que eles sejam o mesmo vetor.

$$X_i X_j = \delta_{ij}, \text{ onde } \delta_{ij} = 1 \text{ quando } i = j, \text{ e } 0 \text{ caso contrário.}$$

Demonstramos, a seguir, que a rede do associador linear definida anteriormente tem duas propriedades, sendo que  $\Phi(X)$  representa a função mapeamento da rede. Primeiro, para um padrão de entrada  $X_i$  que casa exatamente com um dos exemplares, a saída da rede,  $\Phi(X_i)$ , é  $Y_i$ , o exemplar associado. Segundo, para um padrão de entrada  $X_k$ , que não casa exatamente com um dos exemplares, a saída da rede  $\Phi(X_k)$  é  $Y_k$ , que é a interpolação linear de  $X_k$ . Mais precisamente, se  $X_k = X_i + \Delta_i$ , onde  $X_i$  é um exemplar, a rede retorna:

$$Y_k = Y_i + E, \text{ onde } E = \Phi(\Delta_i).$$

Mostramos inicialmente que, quando a entrada da rede  $X_i$  é um dos exemplares, a rede retorna o exemplar associado.

$$\Phi(X_i) = W X_i, \text{ pela definição da função de ativação da rede.}$$

Como  $W = Y_1 X_1 + Y_2 X_2 + \dots + Y_n X_n$ , obtemos:

$$\begin{aligned}\Phi(X_i) &= (Y_1 X_1 + Y_2 X_2 + \dots + Y_n X_n) X_i \\ &= Y_1 X_1 X_i + Y_2 X_2 X_i + \dots + Y_n X_n X_i, \text{ pela distributividade.}\end{aligned}$$

Mas, como definido acima,  $X_i X_j = \delta_{ij}$ :

$$\Phi(X_i) = Y_1 \delta_{1i} + Y_2 \delta_{2i} + \dots + Y_n \delta_{ni}.$$

Pela condição de ortonormalidade,  $\delta_{ij} = 1$ , quando  $i = j$ , e 0 caso contrário. Assim, obtemos:

$$\Phi(X_i) = Y_1 * 0 + Y_2 * 0 + \dots + Y_i * 1 + \dots + Y_n * 0 = Y_i.$$

Pode-se também mostrar que, quando  $X_k$  não for igual a um dos exemplares, a rede realiza um mapeamento interpolativo. Isto é, para  $X_k = X_i + \Delta_i$ , onde  $X_i$  é um exemplar,

$$\begin{aligned}\Phi(X_k) &= \Phi(X_i + \Delta_i) \\ &= Y_i + E,\end{aligned}$$

onde  $Y_i$  é o vetor associado a  $X_i$  e

$$E = \Phi(\Delta_i) = (Y_1 X_1 + Y_2 X_2 + \dots + Y_n X_n) \Delta_i.$$

Omitimos os detalhes da prova.

Daremos, agora, um exemplo de processamento do associador linear. A Figura 11.22 apresenta uma rede simples do associador linear que mapeia um vetor de quatro elementos  $X$  para um vetor de três elementos  $Y$ . Como estamos trabalhando em um espaço de Hamming, a função de ativação da rede  $f$  é a função *sinal* usada anteriormente.

Se desejarmos armazenar as seguintes associações de dois vetores  $\langle X_1, Y_1 \rangle, \langle X_2, Y_2 \rangle$  e:

$$X_1 = [1, -1, -1, -1] \leftrightarrow Y_1 = [-1, 1, 1],$$

$$X_2 = [-1, -1, -1, 1] \leftrightarrow Y_2 = [1, -1, 1].$$

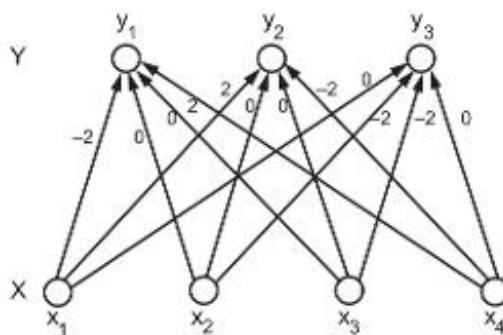
Usando a fórmula de inicialização dos pesos para associadores lineares, sendo o produto vetorial externo como definido na seção anterior:

$$W = Y_1 X_1 + Y_2 X_2 + Y_3 X_3 + \dots + Y_n X_n.$$

Podemos calcular agora  $Y_1 X_1 + Y_2 X_2$ , a matriz de pesos para a rede:

$$W = \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 0 & 0 & 2 \\ 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 \end{bmatrix}$$

**Figura 11.22** Rede de associação linear para o exemplo da Seção 11.5.4. A matriz de pesos é calculada usando a fórmula apresentada na seção anterior.



Aplicamos o associador linear a um dos exemplares. Começamos com  $X = [1, -1, -1, -1]$  do primeiro par de exemplares para retornar o  $Y$  associado:

$$\begin{aligned}y_1 &= (-2 \cdot 1) + (0 \cdot -1) + (0 \cdot -1) + (2 \cdot -1) = -4 \text{ e } \text{sinal}(-4) = -1, \\y_2 &= (2 \cdot 1) + (0 \cdot -1) + (0 \cdot -1) + (-2 \cdot -1) = 4 \text{ e } \text{sinal}(4) = 1 \text{ e} \\y_3 &= (0 \cdot 1) + (-2 \cdot -1) + (-2 \cdot -1) + (0 \cdot -1) = 4 \text{ e } \text{sinal}(4) = 1.\end{aligned}$$

Assim,  $Y_1 = [-1, 1, 1]$ , a outra metade do par exemplo é retornada.

Mostramos, agora, um exemplo de interpolação linear de um exemplar. Considere o vetor  $X [1, -1, 1, -1]$ :

$$\begin{aligned}y_1 &= (-2 \cdot 1) + (0 \cdot -1) + (0 \cdot 1) + (2 \cdot -1) = -4, \text{ e } \text{sinal}(-4) = -1, \\y_2 &= (2 \cdot 1) + (0 \cdot -1) + (0 \cdot 1) + (-2 \cdot -1) = 4, \text{ e } \text{sinal}(4) = 1 \text{ e} \\y_3 &= (0 \cdot 1) + (-2 \cdot -1) + (-2 \cdot 1) + (0 \cdot -1) = 0, \text{ e } \text{sinal}(0) = 1.\end{aligned}$$

Note que  $Y = [-1, 1, 1]$  não é um dos exemplares  $Y$  originais. Observe que o mapeamento preserva os valores que os dois exemplares  $Y$  têm em comum. De fato,  $[1, -1, 1, -1]$ , o vetor  $X$ , tem uma distância de Hamming de 1 de cada um dos dois exemplares  $X$ ; o vetor de saída  $[-1, 1, 1]$  tem, também, uma distância de 1 de cada um dos exemplares  $Y$ .

Resumimos esses resultados com algumas observações a respeito de associadores lineares. As propriedades desejáveis do associador linear dependem da necessidade de os padrões exemplares formarem um conjunto ortonormal. Isso restringe a sua praticidade de duas maneiras. Primeiro, pode não haver um mapeamento óbvio de situações do mundo real para padrões de vetores ortonormais. Segundo, o número de padrões que podem ser armazenados é limitado pela dimensionalidade do espaço vetorial. Quando o requisito de ortonormalidade é violado, ocorre interferência entre padrões armazenados, causando um fenômeno chamado de *interferência cruzada (crosstalk)*.

Observe, também, que o associador linear recupera um exemplar associado  $Y$  apenas quando o vetor de entrada casa exatamente com um exemplar  $X$ . Quando não ocorre um casamento exato com o padrão de entrada, o resultado é um mapeamento interpolativo. Pode-se argumentar que a interpolação não é uma memória no sentido verdadeiro. Frequentemente, desejamos implementar uma função de evocação de memória verdadeira, onde uma aproximação de um exemplar causa a recuperação do padrão exato que está associado a ele. O que precisamos, então, é de uma *bacia de atração* para capturar vetores na região de vizinhança.

Na próxima seção, demonstramos uma versão *atradora* da rede do associador linear.

## 11.6 Redes de atratores ou “memórias”

### 11.6.1 Introdução

As redes discutidas até este ponto são *alimentadas para a frente* (*feedforward*). Nessas redes, a informação é apresentada a um conjunto de nós de entrada e o sinal se propaga para a frente por meio dos nós e camadas de nós até que algum resultado seja obtido. Outra classe importante de redes conexionistas são as redes *recorrentes* (*feedback*).

A arquitetura dessas redes é diferente, e o sinal de saída de um nó pode ser retornado, diretamente ou indiretamente, como entrada para esse mesmo nó.

As redes recorrentes diferem das redes alimentadas adiante de várias maneiras:

1. a presença de conexões de realimentação entre nós;
2. um atraso temporal, isto é, a propagação não instantânea de sinais,
3. a saída da rede é o estado da rede em convergência,
4. a utilidade da rede depende das propriedades de convergência.

Quando uma rede recorrente alcança um instante de tempo no qual ela não muda mais, diz-se que ela alcançou um estado de equilíbrio. O estado que a rede alcança no equilíbrio é considerado a saída da rede.

Nas redes recorrentes da Seção 11.6.2, o estado da rede é inicializado com um padrão de entrada. A rede processa esse padrão, passando por uma série de estados até alcançar o equilíbrio. O estado da rede no equilíbrio é o padrão recuperado da memória. Na Seção 11.6.3, consideraremos redes que implementam uma memória heteroassociativa e, na Seção 11.6.4, uma memória autoassociativa é considerada.

Os aspectos cognitivos dessas memórias são interessantes e importantes, pois nos oferecem um modelo de memória endereçável por conteúdo. Esse tipo de associador pode descrever a recordação de um número de telefone, ou do sentimento de tristeza de uma antiga lembrança, ou mesmo o reconhecimento de uma pessoa a partir de uma visão parcial de seu rosto. Os pesquisadores têm tentado capturar muitos aspectos associativos desse tipo de memória com estruturas de dados simbólicas, incluindo redes semânticas, quadros e sistemas baseados em objetos, como visto no Capítulo 6.

Um *atrator* é definido como um estado para o qual estados contidos em uma região vizinha evoluem ao longo do tempo. Cada atrator de rede terá uma região onde qualquer estado da rede dentro dessa região evolui para esse atrator. Essa região é denominada *bacia*. Um atrator pode consistir em um único estado da rede ou em uma série de estados por meio dos quais a rede circula.

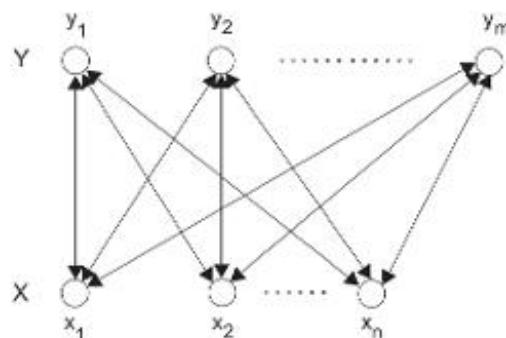
As tentativas para compreender matematicamente os atratores e suas bacias deram origem à noção de uma função de energia da rede (Hopfield, 1984). Pode-se demonstrar que redes recorrentes com uma função de energia que tenha a propriedade de reduzir a energia total da rede a cada transição sempre convergem. Descrevemos essas redes na Seção 11.6.3.

Redes de atratores podem ser usadas para implementar memórias endereçáveis por conteúdo instalando os padrões desejáveis como atratores na memória. Elas também podem ser usadas para resolver problemas de otimização, como no problema do caixeiro-viajante, criando um mapeamento entre a função de custo do problema de otimização e a energia da rede. A solução do problema se dá, então, por meio da redução da energia total da rede. Esse tipo de solução de problema é realizado pela chamada rede de Hopfield.

### 11.6.2 BAM, a memória associativa bidirecional

A rede BAM (do inglês *Bi-directional Associative Memory*), descrita pela primeira vez por Bart Kosko (1988), consiste em duas camadas totalmente interconectadas de elementos processadores. Pode haver, também, uma ligação realimentadora conectando cada nó a si mesmo. Na Figura 11.23, é apresentado o mapeamento realizado por uma BAM de um vetor de entrada de  $n$  dimensões  $X_n$  para o vetor de saída de  $m$  dimensões  $Y_m$ . Como cada arco de  $X$  para  $Y$  é bidirecional, existirão pesos associados a cada fluxo de informação que se propaga em cada direção.

**Figura 11.23** Uma rede BAM para os exemplos da Seção 11.6.2. Cada nó pode ser também conectado a si mesmo.



Como no caso dos pesos do associador linear, os pesos na rede BAM podem ser calculados previamente. Na verdade, usamos o mesmo método para calcular os pesos da rede que é usado para o associador linear. Os vetores para a arquitetura da BAM pertencem ao conjunto de vetores de Hamming.

Com os N pares de vetores que constituem o conjunto de exemplares que desejamos armazenar, construímos a matriz de pesos como fizemos na Seção 11.5.4:

$$W = Y_1 * X_1 + Y_2 * X_2 + \dots + Y_t * X_t$$

Essa equação fornece os pesos nas conexões da camada X para a camada Y, como pode ser visto na Figura 11.23. Por exemplo,  $w_{32}$  é o peso na conexão da segunda unidade na camada X para a terceira unidade na camada Y. Supomos que quaisquer dois nós tenham apenas um caminho entre eles. Portanto, os pesos que conectam os nós nas camadas X e Y são idênticos em ambas as direções. Assim, a matriz de pesos de Y para X é a transposta da matriz de pesos W.

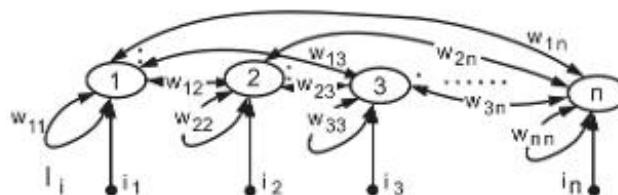
A rede BAM pode ser transformada em uma rede autoassociativa usando a mesma fórmula de inicialização de pesos para o conjunto de associações  $\langle X_1, X_1 \rangle, \langle X_2, X_2 \rangle, \dots$ . Como as camadas X e Y que resultam desse procedimento são idênticas, podemos eliminar a camada Y, o que resulta em uma rede como a mostrada na Figura 11.24. Na Seção 11.6.4, examinaremos um exemplo de uma rede autoassociativa.

A rede BAM é usada para recuperar padrões da memória inicializando a camada X com um padrão de entrada. Se o padrão de entrada for uma versão ruidosa ou incompleta de um dos exemplares, a BAM pode completar o padrão e recuperar o padrão associado.

Para recuperar dados da BAM, fazemos o seguinte:

1. Aplique um par de vetores inicial ( $X, Y$ ) aos elementos processadores. X é o padrão para o qual desejamos recuperar um exemplar. Y é inicializado aleatoriamente.
2. Propague a informação da camada X para a camada Y e atualize os valores na camada Y.

**Figura 11.24** Uma rede autoassociativa com um vetor de entrada  $i_i$ . Supomos ligações únicas entre nós com índices únicos, assim  $w_{ij} = w_{ji}$ , e a matriz de pesos é simétrica.



3. Envie a informação atualizada de Y de volta para a camada X, atualizando as unidades X.
4. Continue a executar os dois passos precedentes até os vetores se estabilizarem, isto é, até não haver mais modificações nos valores dos vetores X e Y.

O algoritmo apresentado fornece o fluxo recorrente da BAM, o seu movimento bidirecional até o equilíbrio. Esse conjunto de instruções poderia ter começado com um padrão no nível Y levando, após a convergência, à seleção de um vetor exemplar X. A rede é totalmente bidirecional: podemos escolher um vetor X como entrada e obter uma associação Y após a convergência, ou podemos começar com um vetor Y como entrada e buscar uma associação X. Analisaremos essas questões a partir de um exemplo na próxima seção.

Após a convergência, o estado de equilíbrio final retorna um dos exemplares usados para construir a matriz de pesos original. Se tudo funcionar como esperado, tomamos um vetor de propriedades conhecidas, idêntico a um dos pares de vetores exemplares, ou um pouco diferente deste. Usamos esse vetor para recuperar o outro exemplo do par exemplar. A distância é a distância de Hamming medida por comparação entre cada componente dos vetores, somando um para cada diferença entre elementos. Por causa das restrições de ortonormalidade, quando a BAM converge para um vetor, ela converge também para o seu complemento. Assim, notamos que o complemento do vetor também se torna um atrator. Na próxima seção, examinamos um exemplo disso.

Há vários aspectos que podem interferir na convergência da BAM. Se um número muito grande de exemplares for mapeado na matriz de pesos, os exemplares podem ficar muito próximos entre si e produzir pseudoestabilidades na rede. Esse fenômeno é chamado de *interferência cruzada* e ocorre como mínimos locais no espaço de energia da rede.

A seguir, consideramos brevemente o processamento da BAM. A multiplicação de um vetor de entrada pela matriz de pesos calcula a soma dos produtos vetoriais de pares de vetores para cada elemento do vetor de saída. Uma simples função de limiar traduz, então, o vetor resultante em um vetor do espaço de Hamming. Assim:

$$\text{rede}(Y) = WX, \text{ ou para cada componente } Y_i, \text{ rede}(Y_i) = \sum w_{ij} x_j,$$

com relações similares para a camada X. A função de limiar  $f$  para  $\text{rede}(Y)$  no tempo  $t + 1$  é também definida diretamente como:

$$f(\text{rede}^{t+1}) = \begin{cases} +1 & \text{se } \text{rede} > 0 \\ f(\text{rede}^t) & \text{se } \text{rede} = 0 \\ -1 & \text{se } \text{rede} < 0 \end{cases}$$

Na próxima seção, ilustramos o processamento dessa *memória associativa bidirecional* com vários exemplos.

### 11.6.3 Exemplos do processamento da BAM

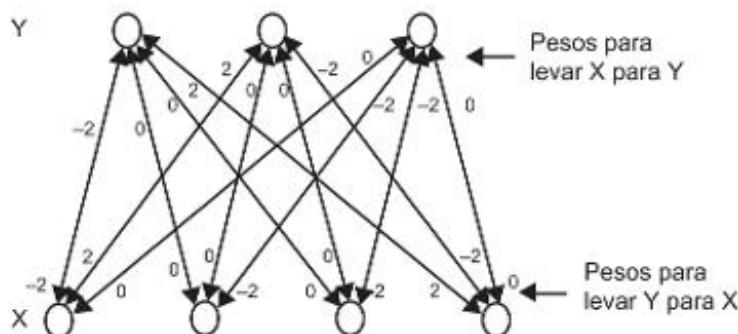
A Figura 11.25 apresenta uma pequena rede BAM, uma variação simples do associador linear apresentado na Seção 11.5.4. Essa rede mapeia um vetor X de quatro elementos para um vetor Y de três elementos e vice-versa. Suponha que desejemos criar os dois pares de exemplares:

$$\begin{aligned} x_1 &= [1, -1, -1, -1] \leftrightarrow y_1 = [1, 1, 1] \text{ e} \\ x_2 &= [-1, -1, -1, 1] \leftrightarrow y_2 = [1, -1, 1]. \end{aligned}$$

Criamos, agora, a matriz de pesos de acordo com a fórmula apresentada na seção anterior:

$$\begin{aligned} W &= Y_1 X_1^T + Y_2 X_2^T + Y_3 X_3^T + \dots + Y_N X_N^T \\ W &= \begin{bmatrix} 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -2 & -2 & 0 \\ 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 \end{bmatrix} \end{aligned}$$

**Figura 11.25** Uma rede BAM para os exemplos da Seção 11.6.3.



O vetor de pesos para o mapeamento de Y para X é a transposta de W, ou:

$$\begin{bmatrix} 0 & 2 & 0 \\ -2 & 0 & -2 \\ -2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix}$$

Selecionamos, agora, vários vetores e testamos o associador BAM. Vamos começar com um par de exemplares, escolhendo o componente X e verificando se obtemos Y. Seja  $X = [1, -1, -1, -1]$ :

$$Y_1 = (1^*0) + (-1^*2) + (-1^*2) + (0^*-1) = 4, \text{ e } f(4) = 1,$$

$$Y_2 = (1^*2) + (-1^*0) + (-1^*0) + (-1^*-2) = 4, \text{ e } f(4) = 1 \text{ e}$$

$$Y_3 = (1^*0) + (-1^*2) + (-1^*2) + (-1^*0) = 4, \text{ e } f(4) = 1.$$

Assim, a outra metade do par exemplar é recuperada. O leitor pode usar esse vetor Y como o vetor de entrada e verificar que o vetor original X,  $[1, -1, -1, -1]$ , é retornado.

Para o próximo exemplo, considere o vetor  $X = [1, 1, 1, -1]$ , com Y inicializado aleatoriamente. Mapeamos X com a nossa rede BAM:

$$Y_1 = (1^*0) + (1^*2) + (1^*2) + (-1^*0) = -4, \text{ e } f(-4) = -1,$$

$$Y_2 = (1^*2) + (1^*0) + (1^*0) + (-1^*-2) = 4, \text{ e } f(4) = 1,$$

$$Y_3 = (1^*0) + (1^*2) + (1^*2) + (-1^*0) = -4, \text{ e } f(-4) = -1.$$

Esse resultado, com a função de limiar f aplicada a  $[-4, 4, -4]$ , é  $[-1, 1, -1]$ . Mapeando de volta para X, o resultado é:

$$X_1 = (-1^*0) + (1^*2) + (-1^*0) = 2,$$

$$X_2 = (-1^*-2) + (1^*0) + (-1^*2) = 4,$$

$$X_3 = (-1^*-2) + (1^*0) + (-1^*-2) = 4,$$

$$X_4 = (-1^*0) + (1^*-2) + (-1^*0) = -2.$$

A função de limiar aplicada, novamente como já mostrado, fornece o vetor original  $[1, 1, 1, -1]$ . Como o vetor inicial produziu um resultado estável na sua primeira propagação, poderíamos pensar que descobrimos outro exemplar protótipo. Na verdade, o exemplo que selecionamos é o complemento do vetor exemplar original  $\langle X_2, Y_2 \rangle$ ! Verificamos que em uma rede BAM, quando um par de vetores é estabelecido como um protótipo exemplar, o seu complemento também o é. Portanto, a nossa rede BAM inclui mais dois protótipos:

$$X_3 = [-1, 1, 1, 1] \leftrightarrow Y_3 = [-1, -1, -1] \text{ e}$$

$$X_4 = [1, 1, 1, -1] \leftrightarrow Y_4 = [-1, 1, -1].$$

Selecionemos, agora, um vetor próximo a um exemplar X,  $[1, -1, 1, -1]$ . Note que a distância de Hamming para o mais próximo dos quatro X exemplares é 1. A seguir, inicializamos, aleatoriamente, o vetor Y em  $[-1, -1, -1]$ :

$$\begin{aligned}Y_1^{1+1} &= (1^*0) + (-1^*-2) + (1^*-2) + (-1^*0) = 0, \\Y_2^{1+1} &= (1^*2) + (-1^*0) + (1^*0) + (-1^*-2) = 4, \\Y_3^{1+1} &= (1^*0) + (-1^*-2) + (1^*-2) + (-1^*0) = 0.\end{aligned}$$

A avaliação da função rede  $f(Y_i^{1+1}) = f(Y_i^1)$  quando  $Y_i^{1+1} = 0$  vem da equação da função de limiar definida no final da Seção 11.6.2. Assim,  $Y$  é  $[-1, 1, -1]$  devido à inicialização aleatória do primeiro e terceiro parâmetros de  $Y^T$  em  $-1$ . Fazemos, agora,  $Y$  retornar para  $X$ :

$$\begin{aligned}X_1 &= (-1^*0) + (1^*2) + (-1^*0) = 2, \\X_2 &= (-1^*-2) + (1^*0) + (-1^*-2) = 4, \\X_3 &= (-1^*-2) + (1^*0) + (-1^*-2) = 4, \\X_4 &= (-1^*0) + (1^*-2) + (-1^*0) = -2.\end{aligned}$$

A função de limiar mapeia esse resultado para o vetor  $X = [1, 1, 1, -1]$ . Repetimos o processo propagando esse vetor de volta para  $Y$ :

$$\begin{aligned}Y_1 &= (1^*0) + (1^*-2) + (1^*-2) + (-1^*0) = -4, \\Y_2 &= (1^*2) + (1^*0) + (1^*0) + (-1^*-2) = 4, \\Y_3 &= (1^*0) + (1^*-2) + (1^*-2) + (-1^*0) = -4.\end{aligned}$$

A função de limiar aplicada a  $[-4, 4, -4]$  resulta novamente em  $Y = [-1, 1, -1]$ . Esse vetor é idêntico à versão mais recente de  $Y$ , de modo que a rede é estável. Isso demonstra que, após duas passagens pela rede BAM, um padrão que era próximo a  $X_4$  convergiu para o exemplar armazenado. Isso seria semelhante a reconhecer uma face ou outra imagem armazenada com parte da informação faltando ou obscurecida. A distância de Hamming entre o vetor  $X$  original  $[1, -1, 1, -1]$  e o protótipo  $X_4$   $[1, 1, 1, -1]$  era de 1. O vetor se acomodou no par de exemplares  $\langle X_4, Y_4 \rangle$ .

Nos nossos exemplos de BAM, começamos o processamento com o elemento  $X$  do par exemplar. Claro que poderíamos desenvolver os exemplos a partir do vetor  $Y$ , inicializando  $X$  quando fosse necessário.

Hecht-Nielsen (1990, p. 82) apresenta uma análise interessante da rede BAM. Ele demonstra que a propriedade ortonormal para o suporte da rede do associador linear à BAM é restritiva demais. Ele argumenta mostrando que o requisito para construir a rede é que os vetores sejam linearmente independentes, isto é, que nenhum vetor possa ser criado de uma combinação linear de outros vetores no espaço de exemplares.

#### 11.6.4 Memória autoassociativa e rede de Hopfield

A pesquisa de John Hopfield, um físico do California Institute of Technology, foi uma das principais razões para que as arquiteturas conexionistas ganhassem a sua atual credibilidade. Ele estudou as propriedades de convergência de redes usando o conceito de minimização de energia. Ele concebeu, também, uma família de redes baseada nesses princípios. Como físico, Hopfield via a estabilidade de fenômenos físicos como pontos de minimização de energia do sistema físico. Um exemplo dessa abordagem é a análise da têmpera simulada (*simulated annealing*) de metais sendo resfriados.

Reveremos, inicialmente, as características básicas das redes associativas recorrentes. Essas redes começam com um estado inicial que consiste no vetor de entrada. A rede processa esse sinal por meio de caminhos de re-alimentação até atingir um estado estável. Para utilizar essa arquitetura como uma memória associativa, desejamos que a rede tenha duas propriedades. Primeiro, partindo do estado inicial, desejamos uma garantia de que a rede convergirá para um estado estável. Em segundo lugar, gostaríamos que esse estado estável fosse aquele mais próximo do estado de entrada, segundo certa métrica de distância.

Analisamos primeiro a rede autoassociativa, criada segundo os mesmos princípios da rede BAM. Na seção anterior, observamos que as redes BAM podem ser transformadas em redes autoassociativas usando vetores idênticos nas posições  $X$  e  $Y$ . O resultado dessa transformação, como veremos a seguir, é uma matriz de pesos quadrada simétrica. A Figura 11.23 da Seção 11.6.2 mostra um exemplo disso.

A matriz de pesos para a rede autoassociativa, que armazena um conjunto de vetores exemplares  $\{X_1, X_2, \dots, X_n\}$ , é criada por:

$$W = \sum X_i X_i^\dagger \quad \text{para } i = 1, 2, \dots, n.$$

Quando criamos a memória autoassociativa, a partir da memória heteroassociativa, o peso do nó  $x_i$  para  $x_j$  será idêntico àquele de  $x_j$  para  $x_i$  e, assim, a matriz de pesos será simétrica. Essa suposição requer apenas que os dois elementos processadores sejam conectados por um caminho com um único peso. Podemos, também, ter o caso especial, plausível do ponto de vista neural, que nenhum nó da rede esteja diretamente conectado a si mesmo, isto é, não há conexões de  $x_i$  para  $x_i$ . Nessa situação, a diagonal principal da matriz de pesos,  $w_{ii}$ , onde  $i = j$ , é toda formada por zeros.

Como no caso da BAM, determinaremos a matriz de pesos a partir dos padrões a serem armazenados na memória. Esclarecemos isso com um exemplo simples. Considere o conjunto de exemplares de três vetores:

$$X_1 = [1, -1, 1, -1, 1],$$

$$X_2 = [-1, 1, 1, -1, -1],$$

$$X_3 = [1, 1, -1, 1, 1].$$

Calculamos, a seguir, a matriz de pesos usando  $W = \sum X_i X_i^\dagger$  para  $i = 1, 2, 3$ :

$$W = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

$$W = \begin{bmatrix} 3 & -1 & -1 & 1 & 3 \\ -1 & 3 & -1 & 1 & -1 \\ -1 & -1 & 3 & -3 & -1 \\ 1 & 1 & -3 & 3 & 1 \\ 3 & -1 & -1 & 1 & 3 \end{bmatrix}$$

Usamos a função de limiar:

$$f(\text{rede}^{t+1}) = \begin{cases} +1 & \text{se rede} > 0 \\ f(\text{rede}^t) & \text{se rede} = 0 \\ -1 & \text{se rede} < 0 \end{cases}$$

Primeiro, testamos a rede com um exemplar,  $X_3 = [1, 1, -1, 1, 1]$  e obtemos:

$$X_3 * W = [7, 3, -9, 9, 7],$$

e com a função de limiar,  $[1, 1, -1, 1, 1]$ . Observamos que esse vetor se estabiliza imediatamente. Isso ilustra que os próprios exemplares são estados estáveis ou atratores.

A seguir, testamos um vetor que está a uma distância de Hamming de 1 do exemplar  $X_3$ . A rede deveria retornar esse exemplar. Isso é equivalente a recuperar um padrão de memória a partir de dados parcialmente degradados. Selecionamos  $X = [1, 1, 1, 1, 1]$ :

$$X * W = [5, 1, -3, 3, 5].$$

Usando a função de limiar, obtemos o vetor  $X_3 = [1, -1, -1, 1, 1]$ .

A seguir, como terceiro exemplo, tomamos um vetor que está a uma distância de Hamming de 2 de seu protótipo mais próximo,  $X = [1, -1, -1, 1, -1]$ . Pode-se verificar que esse vetor está a uma distância 2 de  $X_3$ , 3 de  $X_1$  e 4 de  $X_2$ . Começamos calculando:

$$X * W = [3, -1, -5, 5, 3], \text{ que com o limiar produz } [1, -1, -1, 1, 1].$$

Isso não se parece com nada, nem é um ponto de estabilidade, pois:

$$[1, -1, -1, 1, 1] * W = [9, -3, -7, 7, 9], \text{ que resulta em } [1, -1, -1, 1, 1].$$

A rede está agora estável, mas não em uma das memórias originais armazenadas! Será que encontramos outro mínimo de energia? Uma inspeção mais cuidadosa nos mostra que esse novo vetor é o complemento do exemplar original  $X_2 = [-1, 1, 1, -1, -1]$ . De novo, como no caso da rede BAM heteroassociativa, nossa rede autoassociativa cria atratores para os exemplares originais, bem como para os seus complementos; nesse caso, teremos seis atratores no total.

Até esse ponto, examinamos as redes autoassociativas baseadas em um modelo de associador linear de memória. Um dos objetivos de John Hopfield era fornecer uma teoria mais geral das redes autoassociativas que fosse aplicável a qualquer rede recorrente de uma única camada, que satisfizesse um determinado conjunto de restrições simples. Para essa classe de redes recorrentes de camada única, Hopfield provou que sempre existiria uma função de energia da rede que garantisse a sua convergência.

Outro objetivo de Hopfield era substituir o modelo de atualização de tempo discreto usado anteriormente por um que se parecesse mais com o processamento de tempo contínuo dos neurônios reais. Uma maneira comum de simular a atualização assíncrona de tempo contínuo em redes de Hopfield é atualizar os nós individualmente, em vez de fazer a atualização de toda a camada de nós simultaneamente. Isso é feito usando um procedimento de seleção aleatória para escolher o próximo nó a ser atualizado e, ao mesmo tempo, aplicando algum método que assegure que, em média, todos os nós da rede sejam atualizados com a mesma frequência.

A estrutura de uma rede de Hopfield é idêntica à da rede autoassociativa apresentada: uma única camada de nós completamente conectados (ver Figura 11.23). A ativação e a aplicação do limiar também funciona como antes. Para o nó  $i$ ,

$$x_i^{\text{novo}} = \begin{cases} +1 & \text{se } \sum_j w_{ij} x_j^{\text{velho}} > T_i, \\ x_i^{\text{velho}} & \text{se } \sum_j w_{ij} x_j^{\text{velho}} = T_i, \\ -1 & \text{se } \sum_j w_{ij} x_j^{\text{velho}} < T_i, \end{cases}$$

Dada essa arquitetura, apenas uma restrição a mais é necessária para caracterizar uma rede de Hopfield. Se  $w_{ij}$  é o peso na conexão do nó  $i$  para o nó  $j$ , definimos a seguinte restrição para os pesos de uma rede de Hopfield:

$$\begin{aligned} w_{ii} &= 0 && \text{para todo } i, \\ w_{ij} &= w_{ji} && \text{para todo } i, j. \end{aligned}$$

A rede de Hopfield geralmente não tem um método de aprendizado associado a ela. Como a BAM, os seus pesos são em geral calculados previamente.

O comportamento das redes de Hopfield é atualmente mais bem compreendido que qualquer outra classe de rede, com exceção dos perceptrons. Isso se deve ao seu comportamento poder ser caracterizado em termos de uma função de energia concisa descoberta por Hopfield:

$$H(X) = -\sum_i \sum_j w_{ij} x_i x_j + 2 \sum_i T_i x_i$$

Mostraremos, agora, que essa função de energia tem a propriedade de que qualquer transição da rede reduz a energia total da rede. Dado o fato de  $H$  ter um mínimo predeterminado e que, a cada instante que  $H$  decresce, ela decresce pelo menos por uma quantidade fixa mínima, podemos inferir que, a partir de qualquer estado, a rede sempre convergirá.

Mostraremos primeiro que, para um elemento de processamento arbitrário  $k$ , que é aquele que foi atualizado mais recentemente,  $k$  muda de estado se, e somente se,  $H$  decrescer. A variação na energia,  $\Delta H$ , é:

$$\Delta H = H(X^{\text{novo}}) - H(X^{\text{velho}}).$$

Expandindo essa equação usando a definição de  $H$ , obtemos:

$$\Delta H = -\sum_i \sum_j w_{ij} x_i^{\text{novo}} x_j^{\text{novo}} - 2 \sum_i T_i x_i^{\text{novo}} + \sum_i \sum_j w_{ij} x_i^{\text{velho}} x_j^{\text{velho}} + 2 \sum_i T_i x_i^{\text{velho}}$$

Como apenas  $x_k$  variou,  $x_i^{novo} = x_i^{velho}$  para  $i$  diferente de  $k$ . Isso significa que os termos da soma que não contêm  $x_k$  se cancelam mutuamente. Rearranjando e agrupando termos, obtemos:

$$\Delta H = -2x_k^{novo} \sum_j w_{kj} x_j^{novo} + 2T_k x_k^{novo} + 2x_k^{velho} \sum_j w_{kj} x_j^{velho} - 2T_k x_k^{velho}.$$

Usando o fato de que  $w_{ii} = 0$  e  $w_{ij} = w_{ji}$ , podemos finalmente reescrever essa equação como:

$$\Delta H = 2(x_k^{velho} - x_k^{novo}) \left[ \sum_j w_{kj} x_j^{velho} - T_k \right].$$

Para mostrar que  $\Delta H$  é negativa, consideramos dois casos. Primeiro, suponhamos que  $x_k$  tenha mudado de  $-1$  para  $+1$ . Então, o termo entre colchetes deve ter sido positivo para que  $x_k^{novo}$  seja  $+1$ . Como  $x_k^{velho} - x_k^{novo}$  é igual a  $-2$ ,  $\Delta H$  deve ser negativo. Suponha, agora, que  $x_k$  tenha mudado de  $1$  para  $-1$ . Pelo mesmo raciocínio,  $\Delta H$  deve novamente ser negativo. Se  $x_k$  não mudar de estado,  $x_k^{velho} - x_k^{novo} = 0$  e  $\Delta H = 0$ .

Com base nesse resultado, partindo de qualquer estado inicial, a rede deve convergir. Além disso, o estado da rede na convergência deve ser um mínimo local de energia. Se ele não o fosse, existiria uma transição que reduziria ainda mais a energia total da rede, e o algoritmo de seleção para a atualização escolheria, em algum momento, aquele nó para ser atualizado.

Acabamos de mostrar que as redes de Hopfield têm uma das duas propriedades que desejamos em uma rede que implementa uma memória associativa. Contudo, pode-se mostrar que ela, em geral, não tem a segunda propriedade desejável: elas nem sempre convergem para o estado estável mais próximo do estado inicial. Não existe um método conhecido para tratar desse problema.

As redes de Hopfield podem ser aplicadas, também, para solucionar problemas de otimização, como no problema do caixeiro-viajante. Para isso, o projetista deve encontrar uma maneira de mapear a função de custo do problema para a função de energia de Hopfield. Ao se mover para um mínimo de energia, a rede também minimizará o custo em relação ao estado do problema. Embora tal mapeamento tenha sido encontrado para alguns problemas interessantes, incluindo o problema do caixeiro-viajante, em geral, esse mapeamento de estados do problema para estados de energia é muito difícil de ser descoberto.

Nesta seção, introduzimos as redes recorrentes heteroassociativas e autoassociativas. Analisamos as propriedades dinâmicas dessas redes e apresentamos exemplos bastante simples mostrando a evolução desses sistemas em direção a seus atratores. Mostramos como a rede do associador linear poderia ser transformada em uma rede de atratores chamada de BAM. Na nossa discussão das redes de Hopfield em tempo contínuo, vimos como o comportamento da rede pode ser descrito em termos de uma função de energia. A classe das redes de Hopfield tem a garantia de convergência, porque pode-se mostrar que, a cada transição da rede, a energia total da rede é reduzida.

Ainda restam alguns problemas com a abordagem baseada em energia das redes conexionistas. Primeiro, o estado de energia alcançado não é, necessariamente, um mínimo global do sistema. Segundo, as redes de Hopfield não convergem, necessariamente, para o atrator mais próximo do vetor de entrada. Isso as torna inadequadas para implementar memórias endereçáveis por conteúdo. Terceiro, ao usar redes de Hopfield para otimização, não existe um método geral para criar um mapeamento de restrições para a função de energia de Hopfield. Finalmente, existe um limite no número total de mínimos de energia que podem ser armazenados e recuperados de uma rede e, o que é ainda mais importante, esse número não pode ser estabelecido de forma precisa. Testes empíricos dessas redes mostram que o número de atratores é uma pequena fração do número de nós da rede. Esses e outros tópicos são questões abertas para investigação (Hecht-Nielsen, 1990; Zurada, 1992; Freeman e Skapura, 1991).

As abordagens baseadas na biologia, como os algoritmos genéticos e os autômatos celulares, procuram imitar o aprendizado implícito das formas vivas. O processamento nesses modelos é também paralelo e distribuído. No modelo do algoritmo genético, por exemplo, uma população de padrões representa as soluções candidatas de um problema. Conforme o algoritmo realiza os seus ciclos, essa população de padrões “evolui” por meio de operações que imitam a reprodução, a mutação e a seleção natural. Analisaremos essas abordagens no Capítulo 12.

## 11.7 Epílogo e referências

Neste capítulo, introduzimos o aprendizado com redes conexionistas. Essas redes foram introduzidas em uma perspectiva histórica na Seção 11.1. Para outras leituras, recomendamos McCulloch e Pitts (1943), Oliver Selfridge (1959), Claude Shannon (1948) e Frank Rosenblatt (1958). Modelos psicológicos antigos são também importantes, especialmente os de Donald Hebb (1949). A ciência cognitiva continua a explorar a relação entre cognição e arquitetura do cérebro. Entre as fontes contemporâneas estão *An Introduction to Natural Computation* (Ballard, 1997), *Artificial Minds* (Franklin, 1995), *The Cognitive Neuroscience of Action* (Jeannerod, 1997) e *Rethinking Innateness: A Connectionist Perspective on Development* (Elman et al., 1996).

Não tratamos muitos aspectos matemáticos e computacionais importantes das arquiteturas conexionistas. Para uma visão geral, recomendamos Robert Hecht-Nielsen (1990), James Freeman e David Skapura (1991), Jacek Zurada (1992) e Nello Cristianini e John Shawe-Taylor (2000). Um excelente tutorial sobre máquinas de vetor de suporte é apresentado por Christopher Burges (1988). A programação neurodinâmica é descrita por Bertsekas e Tsitsiklis (1996). Para aplicações conexionistas, investigue *Pattern Recognition in Industry*, de P. M. Bhagat (2005).

Há muitas questões, representacionais e computacionais, que os pesquisadores do aprendizado devem considerar. Entre elas se incluem a seleção da arquitetura e da conectividade da rede, bem como a determinação de quais parâmetros cognitivos do ambiente devem ser processados e o que “significam” os resultados. Há também a questão dos sistemas híbridos neurossimbólicos e como eles poderiam refletir aspectos diferentes da inteligência.

A rede retropropagação é, provavelmente, a arquitetura conexionista mais utilizada e, por isso, demora espaço considerável para as suas origens, uso e desenvolvimento. Os dois volumes de *Parallel Distributed Processing* (Rumelhart et al., 1986b) apresentam uma introdução às redes neurais tanto como ferramenta computacional como cognitiva. *Neural Networks and Natural Intelligence* (Grossberg, 1988) é outro tratamento completo desse assunto.

Há, também, outras questões para a utilização de redes retropropagação, incluindo o número de camadas e nós ocultos, a seleção do conjunto de treinamento, o ajuste fino da constante de aprendizado, o uso de nós de viés, e assim por diante. Muitas dessas questões podem ser vistas como *viés induutivo*: o papel do conhecimento, as expectativas e as ferramentas que o resolvedor de problemas utiliza para resolver o problema. No Capítulo 16, consideraremos muitas dessas questões.

Muitos dos projetistas originais de arquiteturas conexionistas descreveram seus trabalhos. Entre eles estão John Anderson et al. (1977), Stephan Grossberg (1976, 1988), Geoffrey Hinton e Terrance Sejnowski (1986), Robert Hecht-Nielsen (1989, 1990), John Hopfield (1982, 1984), Tuevo Kohonen (1972, 1984), Bart Kosko (1988) e Carver Mead (1989). Abordagens mais recentes, incluindo modelos gráficos, são apresentadas por Michael Jordan (1999) e Brendan Frey (1998). Um bom texto moderno foi escrito por Christopher Bishop (1995).

## 11.8 Exercícios

1. Projete um neurônio de McCulloch-Pitts que possa calcular a função lógica implicação,  $\Rightarrow$ .
2. Construa uma rede de perceptron em LISP e a execute sobre o exemplo da Seção 11.2.2.
  - a. Gere outro conjunto de dados semelhante ao da Tabela 11.3 e execute o classificador sobre ele.
  - b. A partir dos resultados obtidos com o classificador, use os pesos para determinar a especificação da reta que separa os conjuntos.
3. Construa uma rede retropropagação em LISP ou em C++ e use-a para resolver o problema do OU-exclusivo da Seção 11.3.3. Resolva o problema do OU-exclusivo com uma arquitetura diferente de rede retropropagação, tendo talvez dois nós ocultos e nenhum nó de viés. Compare as velocidades de convergência usando as diferentes arquiteturas.

4. Escreva uma rede de Kohonen em LISP ou em C++ e use-a para classificar os dados da Tabela 11.3. Compare seus resultados com os das seções 11.2.2 e 11.4.2.
5. Construa uma rede contrapropagação para resolver o problema do OU-exclusivo. Compare seus resultados com os da rede retropropagação da Seção 11.3.3. Use a sua rede contrapropagação para discriminar entre as classes da Tabela 11.3.
6. Use uma rede retropropagação para reconhecer os dez dígitos (manuscritos). Uma abordagem seria construir um arranjo de  $4 \times 6$  pontos. Quando se desenha um dígito nessa grade, ele cobre alguns elementos, que recebem o valor 1, deixando outros descobertos, com valor 0. Esse vetor de 24 elementos seria o valor de entrada para a sua rede. Você deve construir os seus próprios vetores de treinamento. Realize a mesma tarefa com uma rede contrapropagação; compare seus resultados.
7. Selecione um padrão de entrada diferente daquele usado na Seção 11.5.2. Use o algoritmo de aprendizado hebbiano não supervisionado para reconhecer esse padrão.
8. A Seção 11.5.4 usou o algoritmo do associador linear para fazer duas associações de pares de vetores. Selecione três (novas) associações de pares de vetores e resolva a mesma tarefa. Teste se o seu associador linear é interpolativo; isto é, se ele pode associar padrões um pouco diferentes dos exemplares. Faça o seu associador linear autoassociativo.
9. Considere a memória associativa bidirecional (BAM) da Seção 11.6.3. Modifique os pares de associações dados no exemplo e crie a matriz de pesos para as associações. Selecione novos vetores e teste o seu associador BAM.
10. Descreva as diferenças entre a memória BAM e o associador linear. O que é *crosstalk* e como ele pode ser evitado?
11. Construa uma rede de Hopfield para resolver o problema do caixeiro-viajante para dez cidades.

# Aprendizado de máquina: genético e emergente

*Que limites podemos impor a esse poder, que age por longas eras e escrutina rigidamente toda a constituição, toda a estrutura e todos os hábitos de cada criatura — favorecendo o que é bom e rejeitando o que é ruim? Não vejo limites a esse poder de adaptar vagarosa e maravilhosamente cada forma às mais complexas relações de vida.*

—CHARLES DARWIN, *On the Origin of Species*

*A Primeira Lei da profecia:*

*Quando um cientista famoso, mas de idade avançada, afirma que algo é possível, ele está quase indubitavelmente certo. Quando ele afirma que algo é impossível, ele muito provavelmente está errado.*

*A Segunda Lei:*

*A única maneira de descobrir os limites do possível é se aventurar um pouco no impossível.*

*A Terceira Lei:*

*Qualquer tecnologia suficientemente avançada é indistinguível da mágica.*

—ARTHUR C. CLARKE, *Profiles of the Future*

## 12.0 Modelos de aprendizado social e emergente

Da mesma forma que as redes conexionistas receberam muito de seu suporte e inspiração iniciais do objetivo de criar um sistema neural artificial, uma série de outras analogias biológicas também influenciou a concepção de algoritmos de aprendizado de máquina. Este capítulo considera algoritmos de aprendizado inspirados nos processos subjacentes à evolução: moldar uma população de indivíduos por meio da sobrevivência de seus membros mais ajustados. O poder da seleção em uma população de indivíduos diferentes tem sido demonstrado pelo surgimento de espécies na evolução natural, bem como a partir dos processos sociais subjacentes às mudanças culturais. Ele também tem sido formalizado por meio das pesquisas em autômatos celulares, algoritmos genéticos, programação genética, vida artificial e outras formas de computação emergente.

Modelos *emergentes* de aprendizado simulam a forma mais elegante e poderosa de adaptação da natureza: a evolução de formas de vida de plantas e animais. Charles Darwin disse: "... não há limites para esse poder de adaptar vagarosa e maravilhosamente cada forma às relações mais complexas da vida..." Por meio desse processo simples de introdução de variações em gerações sucessivas e por eliminação seletiva dos indivíduos menos ajustados, *emergem* em uma população adaptações de crescente capacidade e diversidade. Evolução e surgimento ocorrem em populações de indivíduos *corporificados*, cujas ações afetam outros indivíduos, que, por sua vez, são

afetados por outros indivíduos. Assim, as pressões seletivas não surgem apenas do ambiente externo, mas também de interações entre membros de uma população. Um ecossistema tem muitos membros, cada um com papéis e habilidades apropriadas à sua própria sobrevivência, mas, mais importante que isso, cujo comportamento acumulativo molda e é moldado pelo restante da população.

Por causa da sua simplicidade, os processos subjacentes à evolução provaram que são marcadamente gerais. A evolução biológica produz espécies selecionando modificações no genoma. De modo similar, a evolução cultural produz conhecimento operando em unidades de informação transmitidas e modificadas socialmente. Os algoritmos genéticos e outras analogias evolucionárias formais produzem soluções para problemas com capacidade crescente operando em populações de soluções candidatas para o problema.

Quando o algoritmo genético é usado para solucionar um problema, ele tem três estágios distintos: primeiro, as soluções potenciais individuais do domínio do problema são codificadas em representações que suportam as variações e operações de seleção necessárias; frequentemente, essas representações são tão simples quanto uma cadeia de bits. No segundo estágio, algoritmos de acasalamento e mutação, análogos à atividade sexual de formas vivas biológicas, produzem uma nova geração de indivíduos que recombinam características de seus pais. Por fim, uma função de *aptidão* julga quais indivíduos são as “melhores” formas de vida, isto é, mais apropriados para a solução eventual do problema. Esses indivíduos são favorecidos nos processos de sobrevivência e reprodução, moldando a próxima geração de soluções potenciais. Quando necessário, uma geração de indivíduos será interpretada de volta no domínio do problema original como soluções para o problema.

Os algoritmos genéticos são também aplicados a representações mais complexas, incluindo regras de produção, para evoluir conjuntos de regras adaptadas para interagir com o ambiente. A programação genética, por exemplo, combina e aplica mutação em fragmentos de código de programas na tentativa de evoluir um programa para resolver problemas como capturar invariantes em conjuntos de dados.

Um exemplo de aprendizado como interação social que leva à sobrevivência pode ser encontrado em jogos como o Jogo da Vida, originalmente criado pelo matemático John Horton Conway e introduzido mais amplamente na comunidade por Martin Gardner em *Scientific American* (1970, 1971). Nesse jogo, o nascimento, a sobrevivência ou a morte de indivíduos é uma função de seu próprio estado e do estado de seus vizinhos. Geralmente, um pequeno número de regras, normalmente três ou quatro, é suficiente para definir o jogo. Apesar de sua simplicidade, os experimentos com o Jogo da Vida mostram que ele é capaz de evoluir estruturas de complexidade e habilidade extraordinárias, incluindo “organismos” multicelulares autorreplicáveis (Poundstone, 1985).

Uma abordagem importante para a *vida artificial* (ou *a-life*) é simular as condições da evolução biológica por meio das interações de máquinas de estados finitos, complementadas por conjuntos de estados e regras de transição. Esses autômatos são capazes de aceitar informação de fora deles mesmos e, em particular, de seus vizinhos mais próximos. As suas regras de transição incluem instruções de nascimento, de continuidade de vida e de morte. Quando uma população desses autômatos é colocada em um domínio com liberdade para atuar como agentes cooperativos assíncronos paralelos, algumas vezes testemunhamos a evolução de “formas de vida” aparentemente independentes.

Em um outro exemplo, Rodney Brooks (1986, 1987) e seus estudantes projetaram e construíram robôs simples que interagiam como agentes autônomos, resolvendo problemas em uma situação de laboratório. Não há um algoritmo de controle central; em vez disso, surge um comportamento cooperativo como um artefato das interações distribuídas e autônomas dos indivíduos. A comunidade de *a-life* tem conferências regulares e periódicos que refletem as suas pesquisas (Langton, 1995).

Na Seção 12.1, introduzimos os modelos evolucionários ou baseados na biologia por meio dos *algoritmos genéticos* (Holland, 1975), uma abordagem para o aprendizado que explora o paralelismo, as interações mútuas e, frequentemente, uma representação utilizando *bits*. Na Seção 12.2, apresentamos os *sistemas classificadores* e a *programação genética*, áreas de pesquisa relativamente novas, em que são aplicadas técnicas de algoritmos genéticos a representações mais complexas, como construir e refinar conjuntos de regras de produção (Holland et al. 1986) e criar e adaptar programas de computador (Koza, 1992). Na Seção 12.3, apresentamos a *vida artificial* (Langton, 1995). Começamos essa seção com uma introdução ao “Jogo da Vida”. Concluímos com um exemplo de comportamento emergente obtido em pesquisas realizadas no Santa Fe Institute (Crutchfield e Mitchell, 1995).

O Capítulo 13 apresenta as formas estocástica e dinâmica do aprendizado de máquina.

## 12.1 Algoritmo genético

Assim como as redes neurais, os algoritmos genéticos são baseados em uma metáfora biológica: eles veem o aprendizado como uma competição em uma população de soluções evolutivas candidatas para o problema. Uma função de “aptidão” avalia cada solução para decidir se ela contribuirá para a próxima geração de soluções. Então, por meio de operações análogas à transferência de genes na reprodução sexual, o algoritmo cria uma nova população de soluções candidatas.

Suponha que  $P(t)$  defina uma população de soluções candidatas,  $x_i^t$ , no tempo  $t$ :

$$P(t) = \{x_1^t, x_2^t, \dots, x_n^t\}$$

Apresentamos, agora, uma forma geral de algoritmo genético:

procedimento algoritmo genético;

    início

        ajuste o tempo  $t := 0$ ;

        inicialize a população  $P(t)$ ;

        enquanto a condição de parada não for satisfeita faça

            início

                avale a aptidão de cada membro da população  $P(t)$ ;

                selecione membros da população  $P(t)$  com base na aptidão;

                produza os descendentes desses pares usando operadores genéticos;

                substitua, com base na aptidão, candidatos de  $P(t)$ , por esses descendentes;

                ajuste o tempo  $t := t + 1$

            fim

    fim.

Esse algoritmo articula a estrutura básica do aprendizado genético; implementações específicas do algoritmo particularizam essa estrutura de diferentes maneiras. Qual porcentagem da população é retida? Qual porcentagem acasala e produz descendentes? Com que frequência e em quem os operadores genéticos são aplicados? O procedimento “substitua os candidatos mais fracos de  $P(t)$ ” pode ser implementado de uma forma simples, eliminando uma porcentagem fixa dos candidatos mais fracos. Abordagens mais sofisticadas podem ordenar uma população pela sua aptidão e, então, associar a cada membro uma medida de probabilidade para eliminação, onde a probabilidade de eliminação é uma função inversa da sua aptidão. Então, o algoritmo de substituição utiliza essa medida como um fator ao selecionar candidatos a serem eliminados. Embora a probabilidade de eliminação seja muito pequena para os membros mais aptos da sociedade, existe uma chance de que mesmo os melhores indivíduos sejam removidos. A vantagem desse esquema é que ele pode salvar alguns indivíduos cuja aptidão global seja pobre, mas que inclua algum componente que possa contribuir para uma solução mais poderosa. Esse algoritmo de substituição tem muitos nomes, entre os quais, *Monte Carlo*, *seleção proporcional à aptidão* e *seleção por roleta*.

Embora os exemplos da Seção 12.1.1 apresentem representações mais complexas, introduziremos as questões de representação, relacionadas com os algoritmos genéticos, usando cadeias simples de bits para representar soluções do problema. Suponha, por exemplo, que desejemos que um algoritmo genético aprenda a classificar cadeias de 1s e 0s. Podemos representar uma população de cadeias de bits como um padrão de 1s, 0s e #s, onde # é um “não importa”, que pode casar tanto com 0 quanto com 1. Assim, o padrão 1##00##1 representa todas as cadeias de oito bits que começam com 1 e que têm dois 0s no meio.

O algoritmo genético inicializa  $P(0)$  com uma população de padrões candidatos. Em geral, as populações iniciais são selecionadas aleatoriamente. A avaliação de soluções candidatas supõe uma função de aptidão,  $f(x_i^t)$ , que retorna uma medida da aptidão do candidato no tempo  $t$ . Uma medida comum de aptidão de um candidato é testá-lo sobre um conjunto de amostras de treinamento e retornar a porcentagem de classificações corretas. Usando tal função de aptidão, uma avaliação atribui a cada solução candidata o valor:

$$f(x_i^t)/m(P, t)$$

onde  $m(P, t)$  é a aptidão média de todos os membros da população. Frequentemente a medida de aptidão varia ao longo de períodos de tempo e, assim, a aptidão seria uma função do estágio atual da solução global do problema, ou  $f(x^t)$ .

Após avaliar cada candidato, o algoritmo seleciona pares para a recombinação, que utiliza *operadores genéticos* para produzir novas soluções que combinam componentes de seus genitores. Como no caso da evolução natural, a aptidão de um candidato determina em que extensão ele se reproduz, sendo que aqueles candidatos que obtêm avaliações mais altas recebem maiores probabilidades de se reproduzir. Como já observado, a seleção é frequentemente probabilística, com membros fracos recebendo menores probabilidades de reprodução, mas eles não são totalmente eliminados. É importante que alguns candidatos menos aptos sobrevivam, porque eles podem ainda conter algum componente essencial de uma solução, por exemplo, parte de um padrão de bits, e a reprodução pode extrair esse componente.

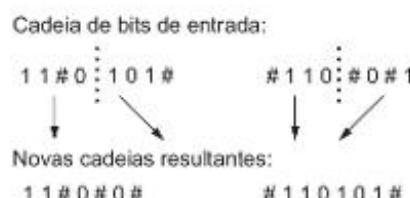
Existem vários operadores genéticos que produzem descendentes que têm características de seus genitores; o mais comum é a *recombinação* (ou *crossover*). Ela toma duas soluções candidatas e as divide, trocando seus componentes para produzir dois novos candidatos. A Figura 12.1 ilustra a recombinação sobre padrões de cadeias de bits de comprimento 8. O operador divide as cadeias no meio e forma dois filhos cujo segmento inicial se origina de um genitor e cujo segmento final vem do outro genitor. Note que a divisão da solução candidata no meio é uma escolha arbitrária. Essa divisão pode se dar em qualquer ponto na representação, e esse ponto de divisão pode até mesmo ser ajustado aleatoriamente ou mudado durante o processo de solução.

Suponha, por exemplo, que a classe-alvo seja o conjunto de todas as cadeias de bits começando e terminando com 1. As duas cadeias de genitores na Figura 12.1 teriam realizado essa tarefa relativamente bem. Entretanto, o primeiro descendente seria muito melhor que seus genitores: ele não teria nenhum falso positivo e não conseguiria reconhecer poucas sequências de bits que estariam realmente na classe solução. Note, também, que seu irmão é pior que seus genitores, e, provavelmente, será eliminado em algumas gerações.

A *mutação* é outro operador genético importante. A mutação toma um único candidato e troca de forma aleatória alguns de seus aspectos. A mutação pode, por exemplo, selecionar aleatoriamente um bit no padrão e trocá-lo, mudando um 1 para 0 ou #. A mutação é importante porque a população inicial pode excluir um componente essencial da solução. No nosso exemplo, se nenhum membro da população inicial tiver um 1 na primeira posição, então a recombinação não poderá produzir um descendente que o tenha, pois ela preserva os primeiros quatro bits do genitor para serem os primeiros quatro bits do filho. A mutação é necessária para trocar os valores desses bits. Outros operadores genéticos como, por exemplo, a *inversão*, poderiam também realizar essa tarefa e são descritos na Seção 12.1.1.

O algoritmo genético continua até que um critério de término seja satisfeito, por exemplo, se existir uma ou mais soluções candidatas cujas aptidões ultrapassem um limiar. Na próxima seção, apresentamos exemplos de codificações de algoritmos genéticos, de operadores genéticos e de avaliações de aptidão para duas situações: os problemas de satisfação de uma restrição na FNC (forma normal conjuntiva) e do caixeiro-viajante.

**Figura 12.1** Uso de recombinação de duas cadeias de bits de comprimento oito. # significa “não importa”.



### 12.1.1 Dois exemplos: satisfação de uma FNC e o caixeiro-viajante

Selecionamos, a seguir, dois problemas e discutimos questões de representação e funções de aptidão apropriadas para a sua solução. Três coisas devem ser observadas: primeiro, nenhum dos problemas são facilmente ou naturalmente codificados por representações de cadeias de bits. Segundo, os operadores genéticos devem preservar as relações cruciais dentro da população, por exemplo, a presença e a unicidade de todas as cidades em um roteiro do caixeiro-viajante. Finalmente, discutimos uma relação importante entre a função de aptidão para os estados de um problema e a codificação desse problema.

#### *Exemplo 12.1.1* O problema da satisfação de uma FNC

O problema da satisfação de uma forma normal conjuntiva (FNC) é simples: uma expressão de proposições está em uma forma normal conjuntiva quando ela for uma sequência de cláusulas ligada por uma relação e ( $\wedge$ ). Cada uma dessas cláusulas está na forma de uma disjunção, o ou ( $\vee$ ), de literais. Por exemplo, se os literais forem  $a, b, c, d, e$  e  $f$ , então a expressão

$$(\neg a \vee c) \wedge (\neg a \vee c \vee \neg e) \wedge (\neg b \vee c \vee d \vee \neg e) \wedge (a \vee \neg b \vee c) \wedge (\neg e \vee f)$$

está na FNC. Essa expressão é a conjunção de cinco cláusulas, sendo cada cláusula a disjunção de dois ou mais literais. Introduzimos proposições e sua satisfação no Capítulo 2. Discutiremos a forma FNC de expressões proposicionais e ofereceremos um método de reduzir expressões à FNC quando apresentarmos a inferência por resolução na Seção 14.2.

Satisfazer uma FNC significa que devemos encontrar uma atribuição de verdadeiro ou falso (1 ou 0) para cada um dos seis literais, de modo que a expressão FNC seja avaliada como verdadeira. O leitor deve confirmar que uma solução para a expressão na FNC é atribuir falso para  $a, b$  e  $e$ . Outra solução tem  $e$  falso e  $c$  verdadeiro.

Uma representação natural para o problema de satisfação da FNC é uma sequência de seis bits, cada bit, ordenadamente, representando verdadeiro (1) ou falso (0) para cada um dos seis literais, novamente na ordem  $a, b, c, d, e$  e  $f$ . Assim:

1 0 1 0 1 0

indica que  $a, c$  e  $e$  são verdadeiros e que  $b, d$  e  $f$  são falsos, e que o exemplo de expressão na FNC é, portanto, falso. O leitor pode explorar os resultados de outras atribuições de valores verdade para os literais da expressão.

Exigimos que as ações de cada operador genético produzam descendentes que sejam atribuições de valores verdade para a expressão FNC, assim cada operador deve produzir um padrão de seis bits de atribuições de valores verdade. Um resultado importante da nossa escolha da representação por padrões de bits para os valores verdade dos literais da expressão FNC é que qualquer um dos operadores genéticos discutidos até esse ponto produzirá um padrão de bits que é uma solução válida. Isto é, a recombinação e a mutação resultam em uma cadeia de bits que é uma solução possível. Mesmo outros operadores genéticos usados com menor frequência, como a *inversão* (inverter a ordem dos bits no padrão de seis posições) ou a *troca* (trocar dois bits diferentes do padrão), produzem um padrão de bits que é uma solução possível para o problema da FNC. Na verdade, desse ponto de vista, é difícil imaginar uma representação mais apropriada que um padrão de bits para o problema da satisfação de uma FNC.

A escolha de uma função de aptidão para essa população de cadeias de bits não é tão fácil. Por um lado, uma atribuição de valores verdade para literais tornará a expressão verdadeira ou, então, a expressão será falsa. Se uma atribuição específica tornar a expressão verdadeira, então a solução foi encontrada; caso contrário, ela não é a solução. À primeira vista parece difícil determinar uma função de aptidão que possa julgar a “qualidade” de cadeias de bits como soluções potenciais.

Entretanto, existem várias alternativas. Uma delas seria observar que a expressão FNC completa é composta pela conjunção de cinco cláusulas. Assim, podemos conceber um sistema de avaliação que nos permita ordenar as soluções de padrões de bits em um intervalo de 0 a 5, dependendo do número de cláusulas que o padrão satisfaz. Assim, o padrão:

110010 tem aptidão 1,  
010010 tem aptidão 2,  
010011 tem aptidão 3 e  
101011 tem aptidão 5 e é uma solução.

Esse algoritmo genético oferece uma abordagem racional para o problema de satisfação de uma FNC. Uma das suas propriedades mais importantes é o uso do paralelismo implícito oferecido pela população de soluções. Os operadores genéticos têm uma aptidão natural para essa representação. Por fim, a busca de uma solução parece se ajustar naturalmente a uma estratégia paralela de “dividir e conquistar”, já que a aptidão é julgada pelo número de componentes do problema que são satisfeitos. Na seção de exercícios deste capítulo, outros aspectos desse problema são considerados.

### Exemplo 12.1.2 O problema do caixeiro-viajante

O problema do caixeiro-viajante (TSP, do inglês *Traveling Salesperson Problem*) é um clássico na IA e na ciência da computação. Ele foi introduzido durante a discussão de grafos na Seção 3.1. O seu espaço de estados completo requer a consideração de  $N!$  estados, onde  $N$  é o número de cidades a serem visitadas. Já foi mostrado que esse problema é NP-difícil, já que muitos pesquisadores propuseram abordagens heurísticas para a sua solução. A descrição do problema é simples:

Um caixeiro-viajante deve visitar  $N$  cidades como parte de um roteiro de vendas. Existe um custo (por exemplo, a milhagem ou tarifa aérea) associado a cada par de cidades no roteiro. Encontre o caminho com o menor custo para que o caixeiro-viajante comece em uma cidade, visite todas as cidades exatamente uma vez e retorne ao ponto inicial.

O TSP tem algumas aplicações muito interessantes, como a perfuração de placas de circuito impresso, a cristalografia de raios X e o roteamento no processo de fabricação de circuitos VLSI. Alguns desses problemas requerem a visita a dezenas de milhares de pontos (cidades) com um caminho de custo mínimo. Uma questão muito interessante na análise da classe de problemas do TSP é se vale a pena executá-lo em uma estação de trabalho cara por muitas horas para obter uma solução próxima do ótimo, ou se é melhor executá-lo em um PC por alguns minutos para obter resultados “bons o suficiente” para essas aplicações. O TSP é um problema interessante e difícil, com muitas ramificações de estratégias de busca.

Como poderíamos utilizar um algoritmo genético para resolver esse problema? Primeiro, a escolha de uma representação para o roteiro de cidades visitadas e a criação de um conjunto de operadores genéticos para esse roteiro não são triviais. Entretanto, a especificação de uma função de aptidão é bem simples: tudo o que precisamos é avaliar o comprimento do caminho percorrido no roteiro. Poderíamos, então, ordenar os caminhos por seus comprimentos; quanto mais curto for o caminho, melhor.

Consideremos algumas representações óbvias que acabam tendo ramificações complexas. Suponha que tenhamos nove cidades a serem visitadas, 1, 2, ..., 9, de modo que representamos um caminho por uma lista ordenada desses nove inteiros. Suponha que representemos cada cidade por um padrão de quatro bits, 0001, 0010, ..., 1001. Assim, o padrão:

0001 0010 0011 0100 0101 0110 0111 1000 1001

representa uma visita a cada cidade na ordem de sua numeração. Inserimos espaços na cadeia apenas para tornar mais fácil a sua leitura. Agora, e os operadores genéticos? A recombinação está definitivamente descartada, pois a nova cadeia produzida de dois genitores diferentes muito provavelmente não representaria um caminho que visita cada cidade exatamente uma vez. Na verdade, com recombinação, algumas cidades seriam removidas, enquanto outras seriam visitadas mais de uma vez. E a mutação? Suponha que o bit mais à esquerda da sexta cidade, 0110, sofra mutação para 1? 1110, ou 14, não é mais uma cidade legítima. A inversão e a troca de cidades (os quatro bits do padrão da cidade) dentro da expressão do caminho seriam operadores genéticos aceitáveis, mas eles seriam poderosos o suficiente para obter uma solução satisfatória? Na verdade, uma forma de considerar essa busca pelo

caminho mínimo seria gerar e avaliar todas as permutações possíveis dos N elementos da lista de cidades. Os operadores genéticos devem ser capazes de produzir todas as permutações.

Outra abordagem para o TSP seria ignorar a representação por padrões de bits e dar para cada cidade um nome alfabético ou numérico, por exemplo, 1, 2, ..., 9; tornar o caminho pelas cidades uma ordenação desses nove dígitos e, então, selecionar operadores genéticos apropriados para produzir novos caminhos. A mutação, por ser uma troca aleatória de duas cidades no caminho, poderia ser usada, mas o operador de recombinação entre dois caminhos não seria útil. A troca de trechos de um caminho por outros trechos do mesmo caminho, ou qualquer outro operador que misture as letras do caminho (sem remover, adicionar ou duplicar qualquer cidade) funcionaria. Entretanto, essas abordagens tornam difícil combinar nos descendentes os “melhores” elementos dos padrões dentro dos caminhos de cidades dos dois diferentes genitores.

Vários pesquisadores (Davis, 1985; Oliver et al., 1987) criaram operadores de recombinação que superam esses problemas e nos permitem trabalhar com a lista ordenada de cidades visitadas. Davis, por exemplo, definiu um operador chamado de *recombinação ordenada*. Suponha que tenhamos nove cidades, 1, 2, ..., 9, e que a ordem dos inteiros represente a ordem das cidades visitadas.

A recombinação ordenada constrói descendentes escolhendo uma subsequência de cidades dentro do caminho de um dos genitores. Ela preserva, também, a ordenação relativa de cidades do outro genitor. Primeiro, selecione dois pontos de corte, indicados por um “|”, que são inseridos aleatoriamente no mesmo local em cada genitor. A localização dos pontos de corte é aleatória, mas, uma vez selecionados, os mesmos locais são usados em ambos os genitores. Por exemplo, para dois genitores p1 e p2, com pontos de corte após a terceira e a sétima cidade:

$$\begin{aligned} p1 &= (1 \ 9 \ 2 \ | \ 4 \ 6 \ 5 \ 7 \ | \ 8 \ 3) \\ p2 &= (4 \ 5 \ 9 \ | \ 1 \ 8 \ 7 \ 6 \ | \ 2 \ 3) \end{aligned}$$

são produzidos dois descendentes c1 e c2 do seguinte modo: primeiro, os segmentos entre os pontos de corte são copiados nos descendentes:

$$\begin{aligned} c1 &= (x \ x \ | \ 4 \ 6 \ 5 \ 7 \ | \ x \ x) \\ c2 &= (x \ x \ | \ 1 \ 8 \ 7 \ 6 \ | \ x \ x) \end{aligned}$$

A seguir, começando do segundo ponto de corte de um genitor, as cidades do outro genitor são copiadas na mesma ordem, omitindo cidades que já estejam presentes. Quando o final da cadeia é alcançado, continue do começo. Assim, a sequência de cidades de p2 é:

2 3 4 5 9 1 8 7 6

Uma vez que as cidades 4, 6, 5 e 7 forem removidas, e já que elas são parte do primeiro descendente, obtemos a lista encurtada 2, 3, 9, 1 e 8, que então constituirá as cidades remanescentes a serem visitadas por c1, preservando a ordenação encontrada em p2:

$$c1 = (2 \ 3 \ 9 \ | \ 4 \ 6 \ 5 \ 7 \ | \ 1 \ 8)$$

De modo semelhante, podemos criar o segundo descendente c2:

$$c2 = (3 \ 9 \ 2 \ | \ 1 \ 8 \ 7 \ 6 \ | \ 4 \ 5)$$

Resumindo, na recombinação ordenada, pedaços de um caminho são passados de um genitor, p1, para um filho, c1, enquanto a ordenação das cidades remanescentes do filho c1 é herdada do outro genitor, p2. Isso dá suporte à intuição óbvia de que a ordenação das cidades será importante na geração do caminho de menor custo e, portanto, é essencial que partes dessa informação ordenada sejam passadas dos genitores aptos para os filhos.

O algoritmo de recombinação ordenada garante, também, que os filhos sejam roteiros legítimos, visitando cada cidade exatamente uma vez. Se desejarmos acrescentar um operador de mutação a esse resultado, devemos ser cuidadosos, como observado anteriormente, para realizar uma troca de cidades dentro de um caminho. O operador de inversão, que simplesmente inverte a ordem de todas as cidades no roteiro, não funcionaria (não se forma um caminho novo quando a ordem de todas as cidades é invertida). Entretanto, cortar, inverter e inserir de volta um pedaço dentro de um caminho consistiria em uma utilização aceitável da inversão. Por exemplo, usando o indicador de corte | como anteriormente, o caminho:

$c1 = (2 \ 3 \ 9 | 4 \ 6 \ 5 \ 7 | 1 \ 8)$ ,

torna-se, invertendo-se a seção do meio:

$c1 = (2 \ 3 \ 9 | 7 \ 5 \ 6 \ 4 | 1 \ 8)$

Um novo operador de mutação poderia ser definido para selecionar aleatoriamente uma cidade e colocá-la em uma nova posição selecionada aleatoriamente do caminho. Esse operador de mutação poderia, também, operar em um pedaço do caminho, por exemplo, para tomar um pedaço de caminho de três cidades e colocá-lo na mesma ordem em uma nova posição dentro do caminho. Outras sugestões se encontram nos exercícios.

### 12.1.2 Avaliando o algoritmo genético (AG)

Os exemplos anteriores ilustram os problemas de representação de conhecimento, específicos do algoritmo genético, como a seleção de operadores e a criação de uma função de aptidão. A representação selecionada deve suportar os operadores genéticos. Algumas vezes, como no problema da satisfação de FNC, a representação usando bits é natural. Nessa situação, os operadores genéticos tradicionais de recombinação e mutação podem ser usados diretamente para produzir soluções potenciais. O problema do caixeiro-viajante é uma questão totalmente diferente. Primeiro, não parece haver nenhuma representação natural por cadeia de bits para esse problema. Segundo, novos operadores de recombinação e mutação tiveram que ser concebidos para preservar a propriedade dos descendentes de terem caminhos válidos passando por todas as cidades uma única vez.

Finalmente, os operadores genéticos devem passar para a nova geração pedaços de informação de soluções potenciais que contenham “significado”. Se essa informação, como no caso da satisfação de uma FNC, for uma atribuição de valor verdade, então os operadores genéticos deverão preservá-la na próxima geração. No problema do TSP, a organização do caminho é crítica e, por isso, componentes dessa informação do caminho devem ser passados para os descendentes. O sucesso dessa transferência depende tanto da representação escolhida como de operadores genéticos concebidos para cada problema.

Encerramos a discussão sobre representação com uma questão final, o problema da “naturalidade” de uma representação selecionada. Suponha, como um exemplo simples, mas um tanto artificial, que desejemos que nossos operadores genéticos percebam a diferença entre os números 6, 7, 8 e 9. Uma representação por inteiros fornece uma ordenação muito natural e uniformemente espaçada, porque, na base de inteiros decimais, o item seguinte é simplesmente um a mais que o anterior. Com a mudança para a base binária, entretanto, essa naturalidade desaparece. Considere os padrões de bits para 6, 7, 8 e 9:

0110 0111 1000 1001

Observe que entre 6 e 7, bem como entre 8 e 9, há uma mudança de 1 bit. Entretanto, entre 7 e 8 todos os bits mudam! Essa anomalia representacional pode ser enorme quando se procura gerar uma solução que requeira qualquer organização desses padrões de quatro bits. Várias técnicas, normalmente com o nome geral de *codificação de Gray*, tratam do problema da representação não uniforme. Por exemplo, uma versão de código de Gray dos primeiros dezenas números binários pode ser encontrada na Tabela 12.1. Note que cada número é diferente dos seus vizinhos em exatamente um bit. Usando a codificação de Gray, em vez dos números binários normais, as transições do operador genético entre estados vizinhos são naturais e suaves.

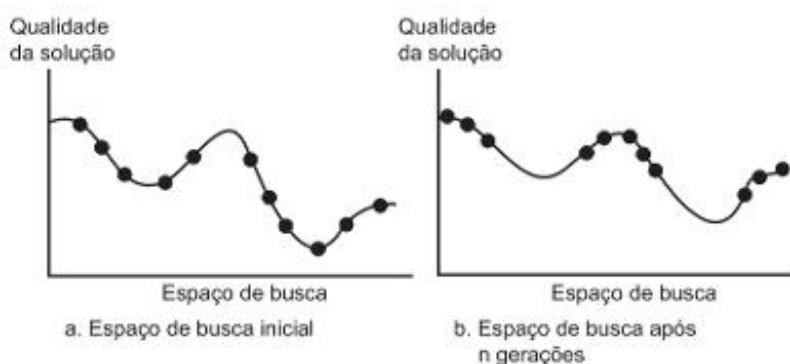
Uma importante vantagem do algoritmo genético está na natureza paralela da sua busca. Os algoritmos genéticos implementam uma forma poderosa de subida de encosta que mantém múltiplas soluções, elimina as soluções que não são promissoras e melhora as boas soluções. A Figura 12.2, adaptada de Holland (1986), mostra múltiplas soluções convergindo para pontos ideais em um espaço de busca. Nessa figura, o eixo horizontal representa os pontos possíveis em um espaço de soluções, enquanto o eixo vertical reflete a qualidade dessas soluções. Os pontos sobre a curva são membros da população atual de soluções candidatas do algoritmo genético. Inicialmente, as soluções estão espalhadas por todo o espaço de soluções possíveis. Após várias gerações, elas tendem a se agrupar em torno de soluções de alta qualidade.

**Tabela 12.1** Padrões de bits em código de Gray para os números binários 0, 1, ..., 15.

Binário	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Quando descrevemos a nossa busca genética como “subida de encosta”, aceitamos implicitamente que nos movemos ao longo de um “relevo de aptidão”, que terá seus vales e picos, com máximos e mínimos locais. Na verdade, algumas das descontinuidades do espaço de busca são artefatos da representação e dos operadores genéticos selecionados para o problema. Essa descontinuidade poderia ser causada, por exemplo, pela ausência da codificação de Gray, como já discutido. Note, também, que os algoritmos genéticos, diferentemente das formas sequenciais de subida de encosta, como na Seção 4.1, não descartam de forma imediata soluções não promissoras. Por meio dos operadores genéticos, até mesmo soluções fracas podem continuar a contribuir para a formação de futuras soluções candidatas.

Outra diferença entre algoritmos genéticos e as heurísticas de busca em espaço de estados apresentadas no Capítulo 4 é a análise da diferença estado-atual/estado-objetivo. O conteúdo de informação que dá suporte ao algoritmo A\*, como na Seção 4.2, requer uma estimativa do “esforço” para se mover entre o estado atual e um esta-

**Figura 12.2** Algoritmos genéticos visualizados como método paralelo de subida de encosta, adaptado de Holland (1986).

do objetivo. Com algoritmos genéticos, não é necessária essa estimativa, mas apenas uma medida da aptidão de cada geração atual de soluções potenciais. Não há, também, a necessidade de uma ordenação estrita de próximos estados como uma lista de estados abertos, como vimos na busca em espaço de estados; em vez disso, existe simplesmente uma população de soluções aptas para um problema, sendo cada uma potencialmente capaz de ajudar a produzir novas soluções possíveis dentro do paradigma da busca paralela.

Uma importante fonte do poder dos algoritmos genéticos é o *paralelismo implícito*, inerente nos operadores evolucionários. Em comparação com a busca em espaço de estados e com a lista ordenada de abertos, a busca se move em paralelo, operando em famílias inteiras de soluções potenciais. Restringindo a reprodução de candidatos mais fracos, os algoritmos genéticos não apenas eliminam aquela solução, mas também todos os seus descendentes. A cadeia 101#0##1, por exemplo, se quebrada no seu ponto médio, pode gerar uma família inteira de sequências da forma 101#\_\_\_\_\_. Se o genitor se mostrar inapto, a sua eliminação também removerá todos esses descendentes potenciais e, talvez, também a possibilidade de uma solução.

Conforme os algoritmos genéticos são mais amplamente usados na solução de problemas aplicados, bem como na modelagem científica, cresce o interesse em procurar compreender as suas fundamentações teóricas. Entre as várias questões que surgem, estão:

1. Podemos caracterizar tipos de problemas para os quais os AGs são bem-sucedidos?
2. Para quais tipos de problemas eles têm desempenho fraco?
3. O que “significa” para um AG ter bom ou mau desempenho em um tipo de problema?
4. Existe alguma lei que possa descrever o comportamento de alto nível dos AGs? Em particular, existe alguma previsão que possa ser feita sobre as modificações na aptidão de subgrupos da população ao longo do tempo?
5. Existe algum modo de descrever os efeitos de diferentes operadores genéticos, recombinação, mutação, inversão etc. ao longo do tempo?
6. Sob quais circunstâncias (problemas e operadores genéticos) os AGs terão melhor desempenho que os métodos de busca tradicionais da IA?

O tratamento de várias dessas questões vai além do escopo deste livro. Na verdade, como salientou Mitchell (1996), ainda há mais questões abertas sobre os fundamentos dos algoritmos genéticos que respostas amplamente aceitas. Apesar disso, desde os trabalhos iniciais em AGs, os pesquisadores, incluindo Holland (1975), têm procurado entender como eles funcionam. Embora eles tratem questões de alto nível, como as seis apresentadas anteriormente, a sua análise começa com a representação de baixo nível, no nível do bit.

Holland (1975) introduziu a noção de um *esquema* como padrão geral e um “bloco construtivo” para as soluções. Um esquema é um padrão de sequências de bits que é descrito por um molde constituído por 1, 0 e # (não importa). Por exemplo, o esquema 1 0 # # 0 1, representa a família de sequências de seis bits começando com 1 0 e terminando com 0 1. Como o padrão central # # descreve quatro padrões de bits, 0 0, 0 1, 1 0, 1 1, o esquema todo representa quatro padrões de seis 1s e 0s. Tradicionalmente, diz-se que cada esquema descreve um hiperplano (Goldberg, 1989); nesse exemplo, o hiperplano corta o conjunto de todas as possíveis representações de seis bits. O princípio central da teoria dos AGs é que os esquemas são os blocos construtivos de famílias de soluções. Considera-se que os operadores genéticos de recombinação e mutação manipulam esses esquemas procurando soluções potenciais. A especificação que descreve essa manipulação é chamada de *teorema dos esquemas* (Holland, 1975; Goldberg, 1989). De acordo com Holland, um sistema adaptativo deve identificar, testar e incorporar propriedades estruturais que hipoteticamente produzirão um melhor desempenho no ambiente. Os esquemas são concebidos para ser uma formalização dessas propriedades estruturais.

A análise de esquemas de Holland sugere que o algoritmo de seleção por aptidão foque cada vez mais a busca em subconjuntos do espaço de busca com melhor aptidão estimada; isto é, os subconjuntos são descritos por esquemas com aptidão acima da média. O operador de recombinação junta blocos construtivos de alta aptidão em uma mesma cadeia, em uma tentativa de criar cadeias cada vez mais aptas. A mutação ajuda a garantir que a diversidade (genética) não seja removida da busca; isto é, que continuamos a explorar novas partes do relevo de aptidão. O algoritmo genético pode assim ser visto como uma tensão entre desencadear

um processo de busca geral e capturar e preservar características (genéticas) importantes nesse espaço de busca. Embora a análise original de Holland da busca por AG tenha focado o nível dos bits, trabalhos mais recentes estenderam essa análise para esquemas representacionais alternativos (Goldberg, 1989). Na próxima seção, aplicaremos as técnicas de AG a representações mais complexas.

## 12.2 Sistemas classificadores e programação genética

As pesquisas iniciais em algoritmos genéticos focaram quase que exclusivamente em representações de baixo nível, como sequências de {0, 1, #}. Além de possibilitarem a implementação de operadores genéticos, cadeias de bits e outras representações similares conferem aos algoritmos genéticos muitas das vantagens de outras abordagens subsimbólicas, como as redes conexionistas. Entretanto, há problemas, como o do caixeiro-viajante, que têm uma codificação mais natural em um nível representacional mais complexo. Podemos ainda questionar se os algoritmos genéticos podem ser definidos para representações ainda mais ricas, como regras *se... então...*, ou pedaços de código de programação. Um aspecto importante de tais representações é a sua habilidade de combinar fontes distintas de conhecimento de alto nível por meio do encadeamento de regras ou de chamadas a funções para satisfazer requisitos de uma ocorrência específica de problema.

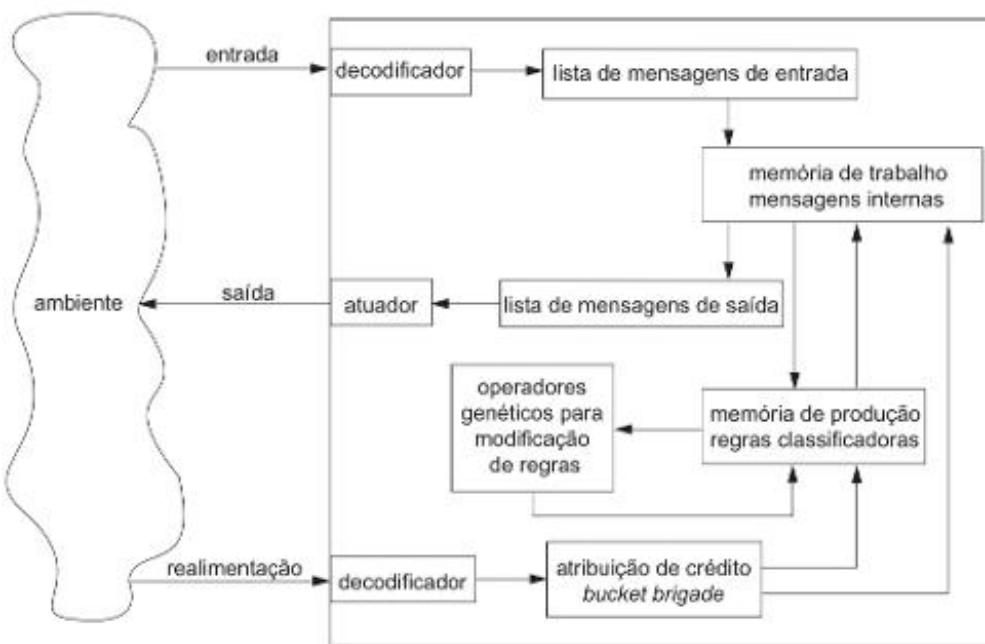
Infelizmente, é difícil definir operadores genéticos que capturem a estrutura sintática e semântica de relações lógicas e, ao mesmo tempo, permitam a aplicação de operadores como recombinação e mutação. Um modo possível de combinar o poder de raciocínio das regras com o aprendizado genético é por meio da tradução de sentenças lógicas em cadeias de bits, usando o operador de recombinação padrão. Infelizmente, em várias traduções, a maioria das sequências de bits produzidas por recombinação e mutação não corresponde mais a sentenças lógicas com sentido. Como alternativa para a representação de soluções de problemas por cadeia de bits, podemos definir variações de recombinação que podem ser aplicadas diretamente a representações de alto nível como as regras *se... então...* ou pedaços de código em uma linguagem de programação de alto nível. Esta seção discute exemplos de cada abordagem para estender o poder dos algoritmos genéticos.

### 12.2.1 Sistemas classificadores

Holland (1986) desenvolveu uma arquitetura para solução de problemas chamada *sistemas classificadores*, que aplica aprendizado genético a regras de um sistema de produção. Um sistema classificador (Figura 12.3) inclui os elementos familiares de um sistema de produção: regras de produção (aqui chamadas de classificadores), memória de trabalho, sensores de entrada (ou decodificadores) e saídas (atuadores). As características incomuns incluem o uso de propostas competitivas para resolver conflitos, algoritmos genéticos para o aprendizado e o algoritmo *bucket brigade* para atribuir crédito ou culpa a regras durante o aprendizado. A avaliação da aptidão de classificadores candidatos, necessária para o aprendizado genético, é fornecida por meio de realimentação do ambiente externo. O sistema classificador da Figura 12.3 tem os seguintes componentes principais:

1. Detectores de mensagens de entrada do ambiente.
2. Detectores de mensagens realimentadas pelo ambiente.
3. Atuadores que traduzem os resultados de aplicações de regras de volta para o ambiente.
4. Um conjunto de regras de produção constituído por uma população de classificadores. Cada classificador tem uma medida de aptidão associada.
5. Uma memória de trabalho para as regras classificadoras. Essa memória integra os resultados do disparo de regras de produção com informações de entrada.
6. Um conjunto de operadores genéticos para modificação de regras de produção.
7. Um sistema para atribuir crédito a regras envolvidas na produção de ações bem-sucedidas.

**Figura 12.3** Um sistema classificador interagindo com o ambiente, adaptado de Holland (1986).



Para resolver o problema, o classificador age como um sistema de produção tradicional. O ambiente envia uma mensagem, talvez um movimento em um jogo, aos detectores do sistema classificador. Esse evento é decodificado e colocado como um padrão na lista interna de mensagens, a memória de trabalho para o sistema de produção. Essas mensagens, na ação normal do sistema de produção guiado por dados, casam com padrões de condição das regras classificadoras. A seleção dos “classificadores mais fortemente ativados” é determinada por um esquema de lances, onde um lance é uma função tanto da aptidão acumulada do classificador como da qualidade do casamento entre o estímulo de entrada e seu padrão de condição. Os classificadores com o melhor casamento adicionam mensagens (a ação das regras disparadas) à memória de trabalho. A lista de mensagens revisada pode enviar mensagens aos atuadores que agem no ambiente ou ativar novas regras classificadoras, conforme prossegue o processamento do sistema de produção.

Os sistemas classificadores implementam uma forma de aprendizado por reforço (Seção 10.7). Com base na realimentação de um professor ou na função de avaliação de aptidão, o aprendiz calcula a aptidão de uma população de regras candidatas e adapta essa população usando uma variação de aprendizado genético. Sistemas classificadores podem aprender de duas formas. Primeiro, existe um sistema de recompensa que ajusta as medidas de aptidão das regras classificadoras, recompensando disparos de regras bem-sucedidos e penalizando erros. O algoritmo de atribuição de crédito passa parte da recompensa ou penalidade de volta para todas as regras classificadoras que contribuíram para o disparo da regra final. Essa distribuição de recompensas diferenciadas pelos classificadores interativos, bem como para aqueles que habilitaram o seu disparo, é frequentemente implementada em um algoritmo *bucket brigade*, que trata do problema de atribuir crédito ou culpa em situações em que a saída do sistema pode ser o produto de uma sequência de disparos de regras. No caso de um erro, como podemos saber qual regra foi a culpada? A responsabilidade é da última regra disparada ou foi alguma outra regra disparada anteriormente que forneceu a ela a informação errada? O algoritmo *bucket brigade* aloca tanto crédito como culpa ao longo de uma sequência de aplicações de regras, de acordo com medidas da contribuição de cada regra para a conclusão final. (Uma atribuição análoga de culpa por erros foi descrita para o algoritmo retropropagação da Seção 11.3; veja Holland (1986) para obter mais detalhes.)

A segunda forma de aprendizado modifica as próprias regras usando operadores genéticos como recombração e mutação. Isso permite que as regras mais bem-sucedidas sobrevivam e se combinem para formar novos classificadores, ao passo que classificadores com regras malsucedidas desaparecem.

Cada regra classificadora consiste em três componentes: a condição da regra que casa com os dados na memória de trabalho do mesmo modo que em sistemas de produção. No aprendizado, os operadores genéticos podem modificar tanto as condições como as ações das regras de produção. O segundo componente da regra, a ação, pode ter o efeito de modificar a lista de mensagens interna (a memória de produção). Finalmente, cada regra tem uma medida de aptidão. Esse parâmetro é modificado, como já observado anteriormente, tanto pela atividade bem-sucedida como pela atividade malsucedida. Essa medida é originalmente atribuída a cada regra na sua criação pelos operadores genéticos; por exemplo, ela pode ser fixada como a média das aptidões dos seus genitores.

Um exemplo simples ilustra as interações desses componentes de um sistema classificador. Suponha que um conjunto de objetos a ser classificado seja definido por seis atributos (condições  $c_1, c_2, \dots, c_6$ ) e suponha ainda que cada atributo possa ter cinco valores diferentes. Embora os valores possíveis de cada atributo sejam diferentes (por exemplo, o valor de  $c_3$  poderia ser cor, enquanto que o de  $c_5$  poderia descrever o tempo), daremos a cada atributo, sem perda de generalidade, um valor inteiro entre {1, 2, ..., 5}. Suponha que as condições dessas regras coloquem os objetos em uma classe dentre quatro classes possíveis: A1, A2, A3, A4.

Com base nessas restrições, cada classificador terá a forma:

$$(c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6) \rightarrow A_i, \text{ onde } i = 1, 2, 3, 4.$$

onde cada  $c_i$  no padrão da condição denota o valor {1, 2, ..., 5} do  $i$ -ésimo atributo da condição. Normalmente, as condições podem também atribuir a um atributo um valor # ou “não importa”.  $A_i$  denota a classificação A1, A2, A3 ou A4. A Tabela 12.2 apresenta um conjunto de classificadores. Note que padrões de condições diferentes podem ter a mesma classificação, como no caso das regras 1 e 2, ou os mesmos padrões, como nas regras 3 e 5, levam a diferentes classificações.

Como descrito até agora, um sistema classificador é simplesmente outra forma do onipresente sistema de produção. A única característica realmente nova das regras classificadoras nesse exemplo é o uso de sequências de dígitos e #s para representar padrões condicionais. É essa representação das condições que restringe a aplicação dos algoritmos genéticos às regras. O restante da discussão descreve o aprendizado genético em sistemas classificadores.

A fim de simplificar o restante do exemplo, consideraremos apenas o desempenho do sistema classificador ao aprender a classificação A1. Isto é, ignoraremos as outras classificações e atribuiremos a padrões de condições um valor 1 ou 0, dependendo de se eles dão ou não suporte à classificação A1. Note que não há perda de generalidade nessa simplificação; ela pode ser estendida a problemas de aprendizado de mais de uma classificação, usando para isso um vetor para indicar as classificações que casam com um padrão de condição particular. Por exemplo, os classificadores da Tabela 12.2 podem ser resumidos a:

**Tabela 12.2** Um conjunto de classificadores condição → ação a ser “aprendido”.

Condição (atributos)	Ação (classificação)	Número da regra
(1 # # # 1 #)	→ A1	1
(2 # # 3 # #)	→ A1	2
(# # 4 3 # #)	→ A2	3
(1 # # # # #)	→ A2	4
(# # 4 3 # #)	→ A3	5
etc.		

$(1 \# \# \# 1 \#) \rightarrow (1 \ 0 \ 0 \ 0)$   
 $(2 \# \# 3 \# \#) \rightarrow (1 \ 0 \ 0 \ 0)$   
 $(1 \# \# \# \# \#) \rightarrow (0 \ 1 \ 0 \ 0)$   
 $(\# \# 4 \ 3 \# \#) \rightarrow (0 \ 1 \ 1 \ 0)$

Nesse exemplo, o último desses resumos indica que os atributos da condição dão suporte às regras de classificação A2 e A3 e não o dão a A1 nem a A4. Pela substituição das atribuições de 0 ou 1 por esses vetores, o algoritmo de aprendizado pode avaliar o desempenho de uma regra por meio de múltiplas classificações.

Nesse exemplo, usaremos as regras da Tabela 12.2 para indicar as classificações corretas; essencialmente, elas funcionarão como professores ou avaliadores da aptidão das regras do sistema de aprendizado. Como acontece com a maioria dos sistemas de aprendizado genético, iniciamos com uma população aleatória de regras. A cada padrão de condição é atribuído também um parâmetro de *força* ou *aptidão* (um número real entre 0,0, força nula, e 1,0, força máxima). Esse parâmetro de força,  $s$ , é calculado a partir da aptidão dos genitores de cada regra e mede a sua aptidão histórica.

A cada ciclo de aprendizado, as regras tentam classificar as entradas e são, então, ordenadas pelo professor ou pela métrica de aptidão. Por exemplo, assuma que em um ciclo, o classificador tenha a seguinte população de regras de classificação candidatas, onde a conclusão de 1 indica que o padrão levou a uma classificação correta e 0, a uma errada:

$(\# \# \# 2 \ 1 \#) \rightarrow 1$	$s = 0,6$
$(\# \# 3 \# \# 5) \rightarrow 0$	$s = 0,5$
$(2 \ 1 \# \# \# \#) \rightarrow 1$	$s = 0,4$
$(\# 4 \# \# \# 2) \rightarrow 0$	$s = 0,23$

Suponha que uma nova mensagem de entrada venha do ambiente (1 4 3 2 1 5) e que o professor classifique (usando a primeira regra da Tabela 12.2) esse vetor de entrada como um exemplo positivo de A1. Considere o que acontece quando a memória de trabalho recebe esse padrão e as quatro regras candidatas tentam casar com ele. As regras 1 e 2 casam com ele. A resolução de conflito é feita por meio de lances competitivos entre regras candidatas. No nosso exemplo, o lance é uma função da soma dos casamentos dos valores de atributos multiplicada pela medida de força da regra. Casamentos do tipo “não importa” têm valor 0,5, enquanto os exatos têm valor 1,0. Para normalizar, dividimos esse resultado pelo comprimento do vetor de entrada. Como o vetor de entrada casa com o primeiro classificador com dois casamentos exatos e quatro “não importa”, o seu lance é de  $((4 * 0,5 + 2 * 1) * 0,6) / 6$ , ou 0,4. O segundo classificador também casa dois atributos e tem quatro “não importa” e, portanto, o seu lance será de 0,33. No nosso exemplo, apenas o classificador que tem o maior lance dispara, mas, em situações mais complexas, pode ser desejável que uma porcentagem dos lances seja aceita.

A primeira regra vence e comunica a sua ação, um 1, indicando que esse padrão é um exemplo de A1. Como essa ação é correta, a medida de aptidão da regra 1 é aumentada para um valor entre o atual e 1,0. Se a ação dessa regra fosse errada, a medida de aptidão teria sido diminuída. Se o sistema requeresse múltiplos disparos do conjunto de regras para produzir um resultado para o ambiente, todas as regras responsáveis por esse resultado receberiam uma fração da recompensa. O procedimento exato pelo qual a aptidão da regra é calculada varia de sistema para sistema e pode ser bastante complexo, envolvendo o uso do algoritmo *bucket brigade* ou outra técnica de atribuição de crédito similar. Veja Holland (1986) para obter mais detalhes.

Uma vez que a aptidão das regras candidatas tenha sido calculada, o algoritmo de aprendizado aplica operadores genéticos para criar a próxima geração de regras. Primeiro, um algoritmo de seleção decide quem são os membros mais aptos do conjunto de regras. Essa seleção é baseada na medida de aptidão, mas pode também incluir um valor aleatório adicional. O valor aleatório dá a regras com fraca aptidão a oportunidade de se reproduzir, ajudando a evitar uma eliminação muito precipitada de regras que, mesmo com desempenho global pobre, possam incorporar algum elemento da solução desejada. Suponha que as duas primeiras regras classificadoras do exemplo sejam selecionadas para sobreviver e reproduzir. A seleção aleatória de uma posição de recombinação entre o quarto e o quinto elementos,

(# ## 2   1 #) → 1	s = 0,6
(# # 3 #   # 5) → 0	s = 0,5

produz os descendentes:

(# # 3 #   1 #) → 0	s = 0,53
(# ## 2   # 5) → 1	s = 0,57

A medida de aptidão dos filhos é uma função ponderada da aptidão dos genitores. A ponderação é uma função da posição dos pontos de recombinação. O primeiro filho tem 1/3 do classificador 0,6 original e 2/3 do classificador 0,5 original. Assim, o primeiro filho tem a força de  $(1/3 * 0,6) + (2/3 * 0,5) = 0,53$ . Por um cálculo similar, a aptidão do segundo filho é 0,57. O resultado de disparar a regra classificadora, sempre 0 ou 1, acompanha a maioria dos atributos, preservando, assim, a intuição de que esses padrões são importantes nos resultados das regras. Em um sistema classificador típico, essas duas novas regras, com seus genitores, formariam o subconjunto de classificadores para a operação do sistema no próximo passo de tempo.

Poderia ser definido, também, um operador de mutação. Uma regra simples de mutação consistiria em mudar aleatoriamente qualquer padrão de atributo para outro padrão de atributo válido; por exemplo, um 5 poderia ser mudado para 1, 2, 3, 4 ou #. Novamente, como observado em nossa discussão sobre AGs, os operadores de mutação são vistos como aqueles que forçam a diversidade em uma busca por classificadores, enquanto a recombinação procura preservar e construir novos filhos a partir de pedaços bem-sucedidos dos padrões dos genitores.

Nosso exemplo foi simples e serviu basicamente para ilustrar os principais componentes do sistema de classificadores. Em um sistema real, mais de uma regra dispararia e cada uma passaria seus resultados para a memória de produção. Frequentemente, existe um esquema de taxação que evita que um classificador se torne muito proeminente no processo de solução, diminuindo a sua aptidão sempre que ele ganhar um lance. Não ilustramos, também, o algoritmo *bucket brigade*, que recompensa diferentemente regras que suportam mensagens de saída bem-sucedidas para o ambiente. Os operadores genéticos também não agem sobre os classificadores a cada operação do sistema. Antes, há um parâmetro geral para cada aplicação que decide, talvez pela análise da realimentação do ambiente, quando os classificadores devem ser avaliados e quando os operadores genéticos devem ser aplicados.

Por fim, nosso exemplo é tomado dos sistemas classificadores que Holland (1986) propôs na Universidade de Michigan. A abordagem de Michigan pode ser vista como um modelo computacional para a cognição, onde o conhecimento (os classificadores) de uma entidade cognitiva é exposto a um ambiente reativo e, como resultado, sofre modificações ao longo do tempo. Avaliamos o sucesso de todo o sistema ao longo do tempo, ao passo que a importância de um classificador individual é mínima. Foram também investigados sistemas de classificadores alternativos, incluindo o trabalho da Universidade de Pittsburgh (Michalski et al., 1983). O classificador de Pittsburgh foca no papel das regras individuais para produzir novas gerações de classificadores. Essa abordagem implementa um modelo de aprendizado indutivo proposto por Michalski.

Na próxima seção, consideramos uma aplicação diferente e particularmente excitante para AGs, a evolução de programas de computador.

### 12.2.2 Programando com operadores genéticos

Ao longo das últimas subseções, vimos os AGs aplicados a estruturas representacionais cada vez maiores. O que começou com transformações genéticas em cadeias de bits, evoluiu para operações sobre regras *se... então...* Podemos perguntar naturalmente se as técnicas genéticas e evolucionárias poderiam ser aplicadas para produzir outras ferramentas computacionais de maior escala. Há dois exemplos principais disso: a geração de programas de computador e a evolução de sistemas de máquinas de estados finitos.

Koza (1991, 1992, 2005) sugeriu que um programa de computador bem-sucedido poderia evoluir por meio de aplicações sucessivas de operadores genéticos. Na programação genética, as estruturas que são adaptadas são segmentos de programas de computador organizados hierarquicamente. O algoritmo de aprendizado mantém uma

população de programas candidatos. A aptidão de um programa é medida por sua capacidade de resolver um conjunto de tarefas e os programas são modificados pela aplicação de recombinação e mutação a subárvores de programas. A programação genética busca um espaço de programas de computadores de complexidade e tamanhos variados; na verdade o espaço de busca é o espaço de todos os programas de computador possíveis, compostos de funções e símbolos terminais apropriados para o domínio do problema. Como acontece com todos os sistemas de aprendizado genético, essa busca é aleatória, basicamente cega, mas surpreendentemente eficaz.

A programação genética começa com uma população inicial de programas gerados aleatoriamente, constituídos por pedaços de programa apropriados. Esses pedaços, adequados para um domínio de problema, podem consistir em operações aritméticas padrão, outras operações de programação e funções matemáticas relacionadas, bem como de funções lógicas e específicas do domínio. Os componentes de programa incluem itens de dados de tipos comuns: lógicos, inteiros, de ponto flutuante, vetores, simbólicos ou multivalorados.

Após a inicialização, milhares de programas de computador são gerados geneticamente. A produção de novos programas advém da aplicação de operadores genéticos. Recombinação, mutação e outros algoritmos de reprodução podem ser personalizados para a produção de programas de computador. Em breve, veremos vários exemplos disso. A aptidão de cada programa novo é, então, determinada pela observação de seu desempenho em um ambiente de problema particular. A natureza da medida de aptidão variará de acordo com o domínio do problema. Qualquer programa que se sair bem em sua tarefa sobreviverá para ajudar a produzir os filhos da próxima geração.

Em resumo, a *programação genética* inclui seis componentes, muitos deles similares aos requisitos para os AGs:

1. Um conjunto de estruturas que sofrem transformação por operadores genéticos.
2. Um conjunto de estruturas iniciais adequadas a um domínio de problema.
3. Uma medida de aptidão, dependente do domínio, para avaliar estruturas.
4. Um conjunto de operadores genéticos para transformar estruturas.
5. Parâmetros e descrições de estado que descrevem membros de cada geração.
6. Um conjunto de condições de término.

Nos próximos parágrafos, tratamos de cada um desses tópicos em mais detalhes.

A programação genética manipula módulos de programa organizados hierarquicamente. Lisp foi (e continua sendo) a representação básica para os componentes de linguagem de programação: Koza representa segmentos de programas como expressões de símbolos Lisp, ou *expressões-s*. (Veja, na Sala Virtual deste livro, uma discussão sobre as expressões-s, sua representação natural como estruturas de árvore e suas avaliações como programas.)

Os operadores genéticos manipulam expressões-s. Em particular, os operadores mapeiam estruturas de árvore de expressões-s (segmentos de programa em Lisp) para novas árvores (novos segmentos em Lisp). Embora essa expressão-s seja a base do trabalho inicial de Koza, outros pesquisadores aplicaram mais recentemente essa abordagem a diferentes linguagens e paradigmas de programação.

A programação genética construirá programas úteis, desde que estejam disponíveis pedaços atômicos e predicados avaliáveis do domínio do problema. Quando especificamos um domínio para a geração de um programa suficiente para tratar de um conjunto de problemas, devemos inicialmente analisar quais terminações são necessárias como unidades na sua solução, bem como quais funções são necessárias para produzir essas terminações. Como Koza observou (1992, p. 86) "... o usuário de programação genética deve saber... que alguma composição das funções e terminações que ele fornece pode produzir uma solução do problema".

Para inicializar as estruturas para adaptação por operadores genéticos, devemos criar dois conjuntos: F, o conjunto de funções, e T, o conjunto de valores terminais requisitados pelo domínio. F pode ser tão simples quanto {+, \*, -, /} ou pode requerer funções mais complexas, como  $\text{sen}(X)$ ,  $\text{cos}(X)$ , ou funções para operações de matrizes. T pode representar os inteiros, os reais, as matrizes ou expressões mais complexas. Os símbolos em T devem ser fechados em relação às funções definidas em F.

A seguir, é gerada uma população de "programas" iniciais, selecionando-se aleatoriamente elementos da união dos conjuntos F e T. Por exemplo, se começarmos selecionando um elemento de T, teremos uma árvore degenerada de um único nó raiz. O mais interessante é que, quando começamos com um elemento

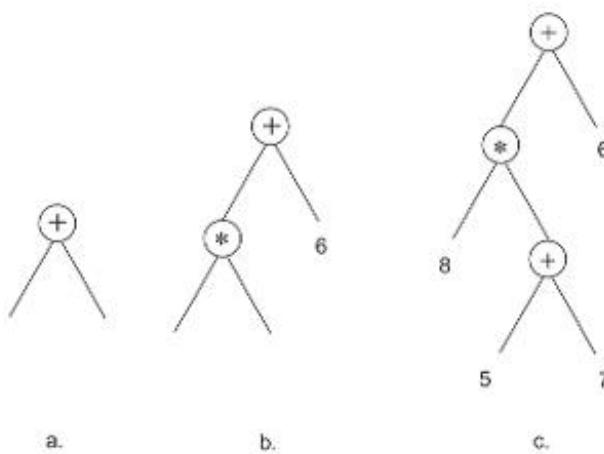
de F, digamos +, obtemos um nó raiz de uma árvore com dois filhos potenciais. Suponha que, em seguida, o inicializador selecione \* (com dois filhos potenciais) de F, como o primeiro filho e, então, o terminal 6 de T, como o segundo filho. Outra seleção aleatória poderia produzir o terminal 8 e a função + de F. Suponha que ele conclui selecionando 5 e 7 de T.

O programa que produzimos aleatoriamente é representado na Figura 12.4. A Figura 12.4(a) mostra a árvore após a primeira seleção de +, a Figura 12.4(b), após a seleção do terminal 6, e a Figura 12.4(c) mostra o programa final. Para inicializar o processo de programação genética, é criada uma população de programas similares. Conjuntos de restrições, como a profundidade máxima de programas a evoluir, podem ajudar a podar essa população. Em Koza (1992) podem ser encontradas descrições dessas restrições, bem como diferentes métodos para gerar populações iniciais.

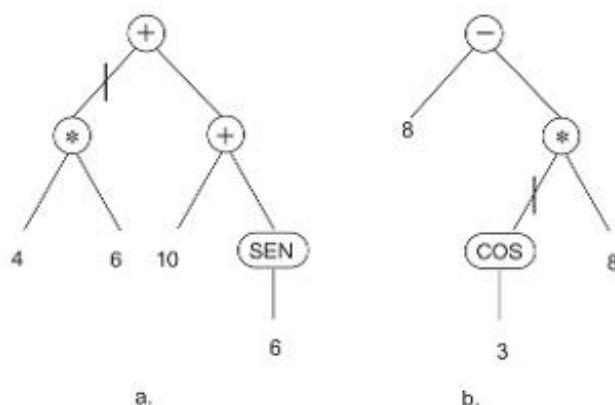
A discussão até este ponto aborda as questões de representação (expressões-s) e o conjunto de estruturas de árvore necessárias para inicializar uma situação para a evolução de programas. A seguir, precisamos de uma medida de aptidão para populações de programas. A medida de aptidão é dependente do domínio do problema e normalmente consiste em um conjunto de tarefas que o programa evoluído deve tratar. A medida de aptidão propriamente dita é uma função do desempenho de cada programa em cada tarefa. Um escore simples de *aptidão bruta* somaria as diferenças entre o que o programa produziu e os resultados que a tarefa real do domínio do problema requereria. Assim, a aptidão bruta poderia ser vista como a soma de erros ao longo de um conjunto de tarefas. Outras medidas de aptidão são possíveis, é claro. A aptidão normalizada divide a aptidão bruta pela soma total de erros possíveis, e, assim, coloca todas as medidas de aptidão dentro do intervalo entre 0 e 1. A normalização pode ser uma vantagem quando a seleção é feita a partir de uma grande população de programas. Uma medida de aptidão pode incluir, também, um ajuste para o tamanho do programa, por exemplo, para recompensar programas menores, mais parcimoniosos.

Os operadores genéticos em programas incluem tanto transformações nas próprias árvores como a troca de estrutura entre árvores. Koza (1992) descreve as transformações primárias de *reprodução* e *recombinação*. A reprodução simplesmente seleciona programas da geração atual e os copia (inalterados) na próxima geração. A recombinação troca subárvores entre as árvores que representam dois programas. Por exemplo, suponha que estejamos trabalhando com os dois programas genitores da Figura 12.5 e que os pontos aleatórios indicados por | nos genitores a e b tenham sido selecionados para recombinação. Os filhos resultantes aparecem na Figura 12.6. A recombinação pode ser usada também para transformar um único genitor trocando duas subárvores dele. Dois genitores idênticos podem criar filhos diferentes por causa da seleção aleatória dos pontos de recombinação. A raiz de um programa pode ser também selecionada como um ponto de recombinação.

**Figura 12.4** A geração aleatória de um programa para a inicialização. Os nós circulares são oriundos do conjunto de funções.



**Figura 12.5** Dois programas selecionados pela aptidão para recombinação. Os pontos I de a e b são selecionados aleatoriamente para recombinação.

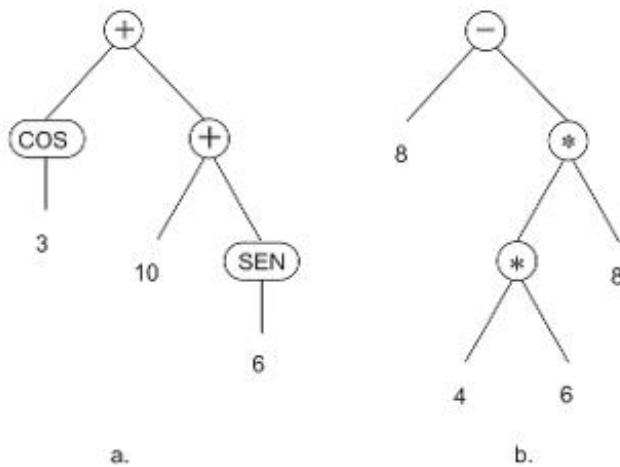


Há várias transformações genéticas secundárias, e muito menos usadas, de árvores de programas. Entre elas está a *mutação*, que simplesmente introduz modificações aleatórias nas estruturas de um programa, por exemplo, substituindo um valor terminal por outro valor ou uma subárvore funcional. A transformação de *permutação*, semelhante ao operador de inversão em cadeias de bits, age também em um único programa, por exemplo, trocando símbolos terminais ou subárvores.

O estado da solução é refletido pela geração atual de programas. Não é feito nenhum registro para fins de retrocesso, e não há nenhum outro método para sair de mínimos da superfície de aptidão. Nesse aspecto, a programação genética é muito parecida com o algoritmo de subida de encosta descrito na Seção 4.1. O paradigma da programação genética imita a natureza à medida que a evolução de novos programas é um processo contínuo. Apesar disso, pela carência de um tempo e computação infinitos, são estabelecidas condições de término. Elas normalmente são uma função da aptidão do programa e dos recursos computacionais.

O fato de a programação genética ser uma técnica para a geração computacional de programas de computador a coloca dentro da tradição de pesquisa em programação automática. Desde os tempos iniciais da IA, pesquisadores têm trabalhado para produzir automaticamente programas de computador a partir de in-

**Figura 12.6** Programas filhos produzidos por recombinação dos pontos na Figura 12.5.



formação fragmentada (Shapiro, 1992). A programação genética pode ser vista como outra ferramenta para esse importante domínio de pesquisa. Concluímos esta seção com um exemplo simples de programação genética retirado de Mitchell (1996).

### **Exemplo 12.2.1 Evoluindo um programa para a terceira lei de Kepler do movimento dos planetas**

Koza (1992, 2005) descreve muitas aplicações de programação genética para resolver problemas interessantes, mas a maioria desses exemplos é extensa e muito complexa para os nossos propósitos. Mitchell (1996), entretanto, criou um exemplo simples que ilustra muitos dos conceitos de programação genética. A terceira lei de Kepler do movimento dos planetas descreve a relação funcional entre o período orbital,  $P$ , de um planeta e a sua distância média,  $A$ , do Sol.

A função para a terceira lei de Kepler, com uma constante  $c$ , é:

$$P^2 = cA^3$$

Se assumirmos que  $P$  é expresso em unidades de anos terrestres e  $A$ , em unidades de distância média entre a Terra e o Sol, então  $c = 1$ . A expressão dessa relação é:

$$P = (\text{sqrt} (* A (* AA)))$$

Assim, o programa que queremos evoluir é representado pela árvore da Figura 12.7.

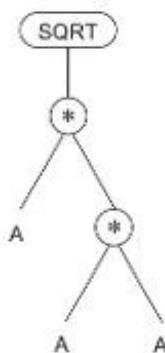
A seleção do conjunto de símbolos terminais nesse exemplo é simples; é o único valor real dado por  $A$ . O conjunto de funções poderia ser igualmente simples, digamos  $\{+, -, *, /, \text{sq}, \text{sqrt}\}$ . A seguir, criamos uma população inicial aleatória de programas. A população inicial pode incluir:

$(* A (- (* A A) (\text{sqrt} A)))$	aptidão: 1
$(/ A (/ (/ A A) (/ A A)))$	aptidão: 3
$(+ A (* (\text{sqrt} A) A))$	aptidão: 0

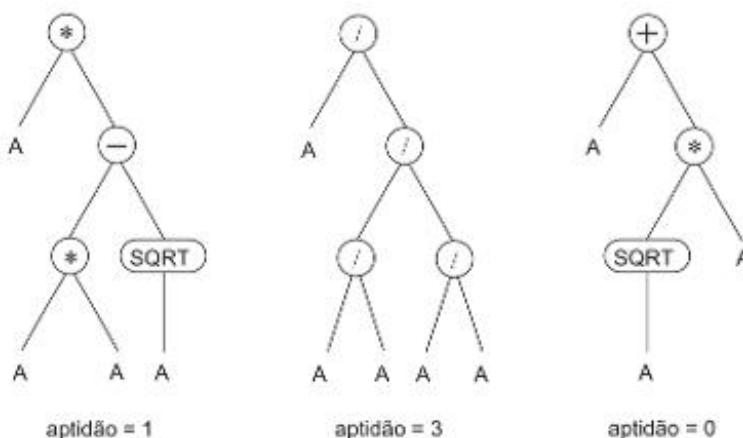
(O valor de aptidão relacionado será explicado mais adiante.) Como observado nesta seção, essa população inicial normalmente tem um limite *a priori* de tamanho e de profundidade, dado o conhecimento do domínio do problema. Esses três exemplos são descritos pelas árvores de programa da Figura 12.8.

A seguir, determinaremos um conjunto de testes para a população de programas. Suponha que conheçamos alguns dados planetários que desejamos explicar com o programa evoluído. Por exemplo, temos os dados planetários apresentados na Tabela 12.3, retirados de Urey (1952), que nos dá um conjunto de pontos de dados que os nossos programas evolutivos devem explicar.

**Figura 12.7** Programa-alvo relacionando órbita ao período, conforme a terceira lei de Kepler.



**Figura 12.8** Membros da população inicial de programas para resolver o problema do período orbital.



**Tabela 12.3** Um conjunto de casos de aptidão, com dados planetários retirados de Urey (1952). A é o eixo orbital principal da Terra e P está em unidades de anos terrestres.

Planeta	A (entrada)	P (saída)
Vênus	0,72	0,61
Terra	1,0	1,0
Marte	1,52	1,87
Júpiter	5,2	11,9
Saturno	9,53	29,4
Urano	19,1	83,5

Como a medida de aptidão é uma função dos pontos de dados que desejamos explicar, definimos a aptidão como o número de saídas do programa que estiverem afastadas em até 20% dos valores de saída corretos. Usamos essa definição para criar a medida de aptidão dos três programas da Figura 12.8. Fica a cargo do leitor criar mais membros dessa população inicial, construir operadores de recombinação e de mutação que possam produzir outras gerações de programas e determinar as condições de término.

## 12.3 Vida artificial e aprendizado social

No início deste capítulo, descrevemos uma versão simplificada do “Jogo da Vida”. Esse jogo, que pode ser mais bem observado por simulações de visualização computacional, em que as gerações sucessivas se modificam rapidamente e evoluem em um monitor, tem uma especificação bastante simples. Ele foi proposto originalmente como um jogo de tabuleiro pelo matemático John Horton Conway e se tornou famoso pela discussão de Martin Gardner sobre ele na *Scientific American* (1970, 1971). O Jogo da Vida é um exemplo simples de modelo de computação denominado *autômato celular* (AC), que são famílias de máquinas de estados finitos simples que exibem comportamentos emergentes interessantes por meio de suas interações em uma população.

### Definição

## MÁQUINA DE ESTADOS FINITOS ou AUTÔMATO CELULAR

1. Um conjunto  $I$  denominado *alfabeto de entrada*.
2. Um conjunto  $S$  de *estados* nos quais o autômato pode se encontrar.
3. Um estado denominado  $s_0$ , o *estado inicial*.
4. Uma *função de próximo estado*  $N: S \times I \rightarrow S$ , que atribui um próximo estado a cada par ordenado, consistindo em um estado atual e uma entrada atual.

A saída de uma máquina de estados finitos (do inglês *Finite State Machine*) é uma função de seu estado atual e dos valores de entrada, como apresentado anteriormente na Seção 3.1. O autômato celular torna a entrada do estado atual uma função de seus estados “vizinhos”. Assim, o estado no tempo  $(t + 1)$  é uma função de seu estado e do *estado de seus vizinhos* no tempo  $t$ . É por meio dessas interações com seus vizinhos que coleções de autômatos celulares podem apresentar comportamentos muito mais ricos do que máquinas de estados finitos simples. Como a saída de um estado é uma função de seus estados vizinhos, podemos descrever a evolução de um conjunto de máquinas de estados vizinhas como adaptação e aprendizado social.

Para as sociedades descritas nesta seção, não existe uma avaliação explícita da aptidão de membros individuais. A aptidão resulta das interações dentro da população, que podem levar à “morte” de autômatos individuais. A aptidão está implícita na sobrevivência de indivíduos de geração em geração. O aprendizado entre autômatos celulares é tipicamente não supervisionado; como ocorre na evolução natural, a adaptação é moldada pelas ações de outros membros da população em evolução.

Um ponto de vista global, ou orientado à sociedade, permite também uma perspectiva importante do aprendizado. Não precisamos mais focar exclusivamente indivíduos, mas sim observar invariâncias e regularidades que emergem dentro da sociedade como um todo. Esse é um aspecto importante da pesquisa de Crutchfield-Mitchell apresentada na Seção 12.3.2.

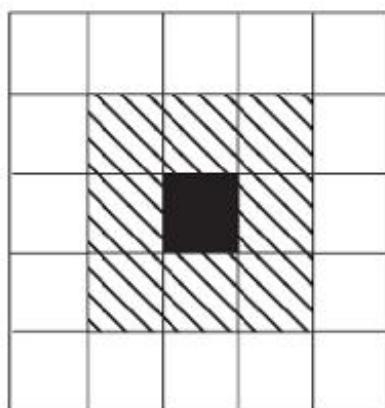
Por fim, diferentemente do aprendizado supervisionado, a evolução não precisa ser “intencional”. Isto é, a sociedade de agentes não precisa ser vista como “indo para algum lugar”, digamos um ponto “ômega”. Tínhamos um viés de convergência ao usarmos medidas de aptidão explícitas nas seções anteriores deste capítulo. Porém, como salientado por Stephen Jay Gould (1977, 1996), a evolução não precisa ser vista como algo que faz “melhor”; ela apenas favorece a sobrevivência. O único sucesso é continuar existindo, e os padrões que emergem são os padrões de uma sociedade.

### 12.3.1 O “Jogo da Vida”

Considere a grade bidimensional simples, ou tabuleiro de jogo, da Figura 12.9. Temos, aqui, um quadrado “ocupado” (em preto, com o valor de bit 1), com seus oito vizinhos mais próximos indicados por hachura. O tabuleiro é transformado ao longo de períodos de tempo, onde o estado de cada quadrado no tempo  $t + 1$  é uma função de seu estado e do estado desses vizinhos indicados, no tempo  $t$ . Três regras simples podem guiar a evolução no jogo: primeiro, se qualquer quadrado, ocupado ou não, tiver exatamente três de seus vizinhos ocupados, ele estará ocupado no próximo período de tempo. Segundo, se qualquer quadrado ocupado tiver exatamente dois de seus vizinhos mais próximos ocupados, ele estará ocupado no próximo período de tempo. Por fim, para todas as outras situações, o quadrado não estará ocupado no próximo período de tempo.

Uma interpretação dessas regras é que, para cada geração ou período de tempo, a vida em qualquer lugar, isto é, se o quadrado está ou não ocupado (tem o valor 1), é um resultado de sua própria vida e a de seus vizinhos durante a geração anterior. Especificamente, uma população muito densa (mais que três) ou muito esparsa (menos que dois) de vizinhos próximos em qualquer período de tempo não permitirá a vida para a próxima geração.

**Figura 12.9** A região hachurada indica o conjunto de vizinhos para o “Jogo da Vida”.

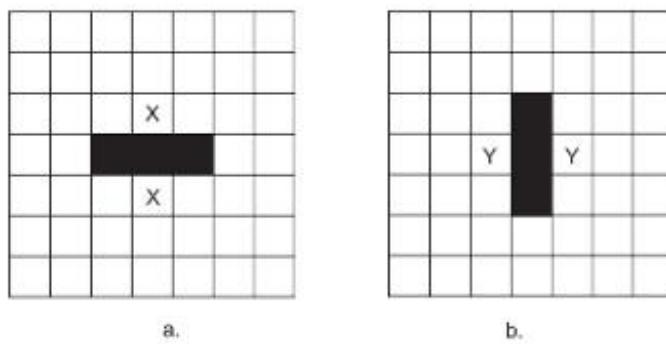


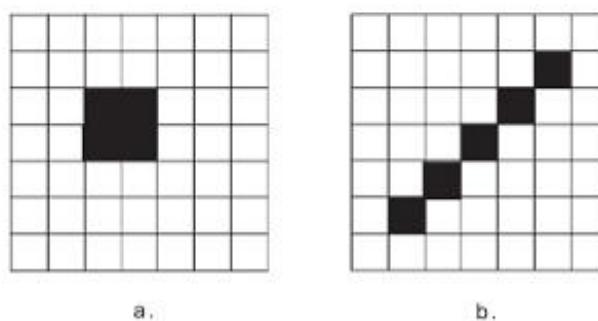
Considere, por exemplo, o estado de vida para a Figura 12.10(a). Aqui, exatamente dois quadrados, indicados por um X, têm exatamente três vizinhos ocupados. No próximo ciclo de vida, será produzida a Figura 12.10(b). Aqui há, exatamente, dois quadrados, indicados por Y, com exatamente três vizinhos ocupados. Podemos ver que o estado do mundo ficará circulando para a frente e para trás entre as figuras 12.10(a) e (b). O leitor poderá determinar qual será o próximo estado para as figuras 12.11(a) e (b) e examinar outras configurações de “mundo” possíveis. Poundstone (1985) descreve a extraordinária variedade e a riqueza das estruturas que podem emergir no Jogo da Vida, como os *planadores* (*gliders*), padrões de células que se movem pelo mundo por meio de ciclos repetidos de mudanças de forma, como vemos na Figura 12.12.

Por causa da sua capacidade de produzir comportamentos coletivos ricos por meio da interação de células simples, os autômatos celulares têm se mostrado uma ferramenta poderosa para o estudo da matemática do surgimento de vida a partir de componentes simples, inanimados. A *vida artificial* é definida como *a vida feita pelo esforço humano, e não pela natureza*. Como pode ser visto no exemplo anterior, a vida artificial tem uma forte natureza “de baixo para cima”; isto é, os átomos de um sistema de vida artificial são definidos e montados, e suas interações físicas “emergem”. As regras da máquina de estados finitos capturam regularidades dessa forma de vida.

Mas como as construções de vida artificial podem ser usadas? Na biologia, por exemplo, o conjunto de entidades vivas fornecido pela natureza, tão complexo e diverso como ele possa ser, é dominado pela contingência acidental e histórica. Cremos que existam regularidades lógicas que agem na criação desse conjunto, mas elas não precisam existir e é improvável que venhamos a descobrir muitas de todas as regularidades possíveis, quando restringimos nossa visão ao conjunto de entidades biológicas que a natureza realmente

**Figura 12.10** Um conjunto de vizinhos gerando o fenômeno da luz “piscante”.



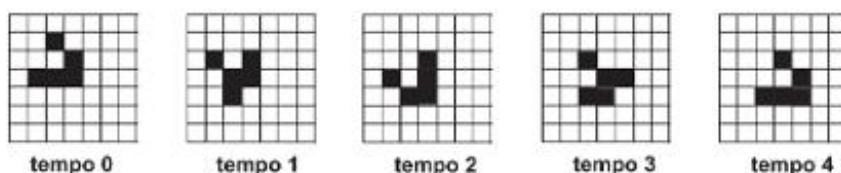
**Figura 12.11** O que acontece a esses padrões no próximo ciclo de tempo?

fornecce. É essencial explorar o conjunto completo de regularidades biológicas possíveis, algumas das quais podem ter sido eliminadas por acidentes históricos. Podemos sempre imaginar como seria o mundo atual se a existência dos dinossauros não tivesse sido bruscamente interrompida. Para se ter uma teoria do real, é necessário entender os limites do possível.

Além do esforço determinado de antropólogos e outros cientistas para completar as lacunas no conhecimento da nossa evolução real, continua a especulação sobre a reconstrução da própria história da evolução. O que aconteceria se a evolução começasse a partir de condições iniciais diferentes? O que aconteceria se houvesse "acidentes" que interviessem alternativamente dentro de nosso ambiente físico e biológico? O que surgiria de novo? O que se manteria constante? O caminho evolucionário que realmente ocorreu na Terra é apenas uma das muitas trajetórias possíveis. Algumas dessas questões poderiam ser tratadas se pudéssemos gerar algumas das várias biologias que são possíveis. (Veja, por exemplo, o PACE — *Programmable Artificial Cell Evolution* —, na Seção 12.3.3.)

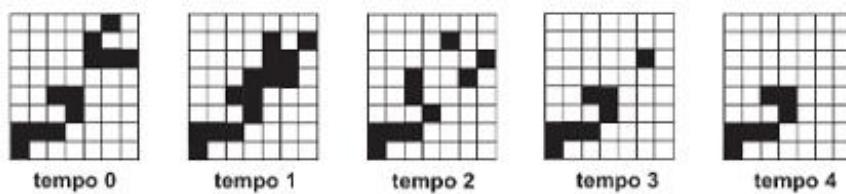
A tecnologia da vida artificial não é apenas um artefato dos domínios computacional ou biológico. Os cientistas de áreas tão diversas, como a química e a farmacologia, construíram artefatos sintéticos, muitos deles relacionados ao conhecimento das entidades reais existentes em nosso mundo. Por exemplo, no campo da química, as pesquisas sobre a constituição da matéria e os diversos compostos que a natureza oferece levaram à análise desses compostos, suas partes constituintes e suas ligações. Essa análise e sua recombinação levaram à criação de diversos compostos que não existem naturalmente. O nosso conhecimento dos blocos construtivos da natureza nos levou às nossas próprias versões sintéticas, juntando componentes da realidade em novos e diferentes padrões. É por meio dessa análise cuidadosa dos componentes químicos naturais que chegamos a um entendimento do conjunto de compostos possíveis.

Uma ferramenta para entender os mundos possíveis é a simulação e a análise dos movimentos sociais e os efeitos de interações. Temos exemplos simples disso no Jogo da Vida. A sequência de ciclos de tempo demonstrada na Figura 12.12 implementa o *planador* que foi mencionado anteriormente. O planador se movimenta ao longo do espaço do jogo passando sucessivamente por um pequeno número de padrões. A sua ação é simples, conforme ele se movimenta em quatro períodos de tempo para uma nova localização, uma coluna mais à direita e uma linha mais abaixo na grade.

**Figura 12.12** Um planador se move através da tela.

Um aspecto interessante do Jogo da Vida é que entidades como o planador persistem até elas interagirem com outros membros da sua sociedade; o que acontece, então, é difícil de entender e prever. Na Figura 12.13, por exemplo, vemos a situação em que dois planadores emergem e se entrelaçam. Após quatro períodos de tempo, o planador que se movimenta para baixo e para a esquerda é “consumido” pela outra entidade. É interessante notar que as nossas descrições ontológicas, isto é, o uso de termos como “entidade”, “luz piscante”, “planador”, “consumido”, refletem nossos próprios vieses antropocêntricos ao visualizar formas de vida e interações, quer sejam elas artificiais ou não. É uma característica humana dar nomes a regularidades, conforme elas aparecem em nossas estruturas sociais.

**Figura 12.13** Um planador é consumido por outra entidade.



### 12.3.2 Programação evolucionária

O “Jogo da Vida” é um exemplo intuitivo, altamente descritivo de autômatos celulares. Podemos generalizar nossa discussão dos autômatos celulares caracterizando-os como máquinas de estados finitos. Discutimos, agora, sociedades de máquinas de estados finitos conectadas e as analisamos como entidades emergentes. Esse estudo é, algumas vezes, denominado *programação evolucionária*.

A história da programação evolucionária remonta ao início dos próprios computadores. John von Neumann, em uma série de palestras em 1949, explorou a questão sobre qual nível de complexidade organizacional seria necessário para a ocorrência de autorreplicação (Burks, 1970). Burks cita o objetivo de von Neumann como “... ele não tentava simular a autorreprodução de um sistema natural ao nível da genética e da bioquímica. Ele desejava abstrair do problema da autorreprodução natural a sua forma lógica”.

Removendo detalhes químicos, biológicos e mecânicos, von Neumann foi capaz de representar os requisitos essenciais para a autorreplicação. Ele concebeu (mas nunca construiu) um autômato autorreprodutivo, consistindo em um arranjo celular bidimensional contendo um grande número de autômatos individuais, de 29 estados, em que o próximo estado de cada autômato era uma função de seu estado atual e dos estados dos seus quatro vizinhos mais próximos (Burks, 1970, 1987).

É interessante notar que von Neumann concebeu o seu autômato autorreprodutivo, contendo pelo menos 40 mil células, para ter a funcionalidade de uma Máquina de Turing Universal. Esse dispositivo computacional universal era também um *construtor universal*, no sentido de ser capaz de ler uma fita de entrada, interpretar os dados contidos na fita e, por meio do uso de um braço construtivo, construir a configuração descrita na fita em uma parte desocupada do espaço celular. Ao colocar na fita uma descrição do próprio autômato, von Neumann criou um autômato autorreprodutivo (Arbib, 1966).

Mais tarde, Codd (1968) reduziu o número de estados necessários para um autômato computacionalmente universal, autorreprodutivo, de 29 para 8, mas aumentou para 100 milhões a quantidade de células estimadas para o projeto completo. Depois, Devore simplificou a máquina de Codd para que ela ocupasse cerca de 87.500 células. Em tempos mais modernos, Langton criou um autômato autorreprodutivo, sem a universalidade computacional, em que cada célula tinha apenas oito estados e ocupava apenas 100 células (Langton, 1986; Hightower, 1992; Codd, 1992). As descrições atuais desses esforços investigativos podem ser encontradas nos anais das conferências sobre vida artificial (Langton, 1989; Langton et al., 1992).

Assim, a análise formal de máquinas autorreprodutivas tem raízes profundas na teoria da computação. Talvez resultados até mesmo mais excitantes estejam implícitos em estudos empíricos de formas de vida artificial. O sucesso desses programas não é indicado por uma função de aptidão *a priori*, mas pelo simples fato de poderem sobreviver e se replicar. O seu sinal de sucesso é que eles sobrevivem. O lado escuro desses desenvolvimentos é que experimentamos o legado de vírus e *worms* de computador, que são capazes de forçar seu caminho entre hospedeiros externos, replicando-se (normalmente destruindo qualquer informação na memória necessária para a replicação) e infectando outros hospedeiros.

Concluimos esta seção discutindo vários projetos de pesquisa que podem ser vistos como exemplos de computação de vida artificial. A última seção do Capítulo 12 é um exemplo detalhado da computação emergente por Melanie Mitchell e seus colegas do Santa Fe Institute. Primeiro, apresentamos dois projetos já discutidos em capítulos anteriores, o de Rodney Brooks, do MIT, e o de Nils Nilsson e seus alunos, de Stanford. Nas apresentações anteriores, o trabalho de Brooks foi apresentado no contexto geral de representações alternativas, na Seção 6.3, e o de Nilsson, no tópico de planejamento, na Seção 7.4.3. No contexto deste capítulo, recolocamos essas duas apresentações anteriores no contexto de vida artificial e de fenômenos emergentes.

Rodney Brooks (1991a, b) desenvolveu no MIT um programa de pesquisa baseado na premissa da vida artificial, ou seja, em que a inteligência emerge a partir das interações de vários agentes autônomos simples. Brooks descreve sua técnica como “inteligência sem representação”, construindo uma série de robôs capazes de detectar obstáculos e de se mover através de salas e corredores do MIT. Com base na *arquitetura de subsunção*, esses agentes são capazes de vagar, explorar e evitar outros objetos. A inteligência desse sistema é um artefato de organização simples e de interações incorporadas com o seu ambiente. Brooks descreve que “... conectamos máquinas de estados finitos formando camadas de controle. Cada camada é construída no topo de camadas previamente existentes. Camadas em níveis mais baixos não dependem da existência de camadas em níveis mais altos”. Outras referências se encontram em McGonigle (1990, 1998), Brooks (1987, 1991a); Lewis e Luger (2000).

Nils Nilsson e seus estudantes (Nilsson, 1994; Benson, 1995; Benson e Nilsson, 1995) desenvolveram um programa *teleorreativo* (TR) para controle de agentes, que dirige um agente até um objetivo de um modo que leva em conta, continuamente, circunstâncias ambientais variáveis. Esse programa opera de forma similar a um sistema de produção, mas dá suporte também a *ações duradouras*, ou ações que ocorrem ao longo de períodos arbitrários de tempo, tais como vá *em frente até...*. Assim, diferentemente dos sistemas de produção usuais, as condições são continuamente avaliadas e a ação que é executada é aquela associada à condição verdadeira atual de nível mais alto. Para obter mais detalhes, o leitor interessado deve consultar Nilsson (1994), Benson (1995), Benson e Nilsson (1995) e Klein et al. (2000).

A comissão europeia tem dado suporte ao projeto PACE. Essa pesquisa tem produzido uma nova geração de células químicas sintéticas. O foco é que a vida evolui em torno de sistemas computacionais vivos, que são auto-organizáveis e também autorreparáveis. As células artificiais completas também são autorreprodutivas, capazes de realizar evolução, e podem manter sua estrutura complexa usando os recursos mais simples do ambiente. Para obter mais detalhes, consulte o site do projeto PACE (<<http://www.istpace.org>>).

O esforço de pesquisa de John Koza em programação genética na Stanford University continua a publicar seus muitos sucessos. Os avanços recentes incluem o desenvolvimento automático de circuitos e controladores, a síntese de topologia de circuitos, dimensionamento, posicionamento e roteamento, além de outros artefatos da engenharia. As referências podem ser encontradas em Koza (2005).

Esses quatro esforços de pesquisa são amostras de uma grande população de projetos de pesquisa baseados em autômatos, que são basicamente experimentais. Eles fazem questionamentos sobre o mundo natural. O mundo natural responde com sobrevivência e crescimento para os algoritmos bem-sucedidos e com a aniquilação de um sistema incapaz de se adaptar.

Finalmente, consideraremos as pesquisas do Santa Fe Institute: um estudo de caso em emergência.

### 12.3.3 Um estudo de caso em emergência (Crutchfield e Mitchell, 1995)

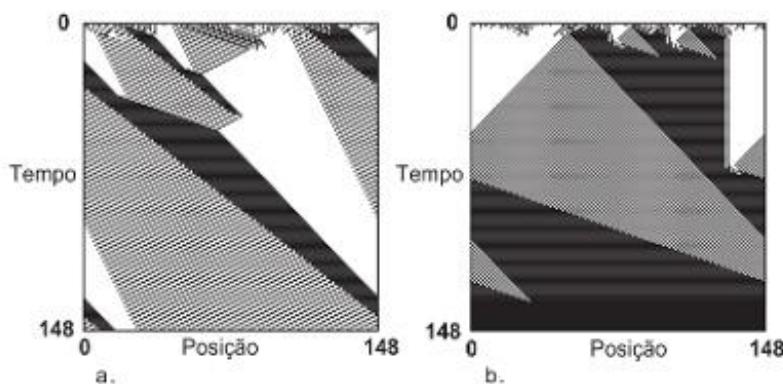
Crutchfield e Mitchell exploram a capacidade de evolução e interação em sistemas simples para criar relações de processamento de informação coletivas de alto nível. A sua pesquisa oferece um exemplo da emergência de computação global ao longo de um sistema espacialmente organizado, constituído por processadores distribuídos e localmente interativos. O termo *computação emergente* descreve o surgimento de estruturas de processamento de informação global dentro desses sistemas. O objetivo da pesquisa de Crutchfield e Mitchell é descrever uma arquitetura e mecanismos que sejam suficientes para evoluir e que suportem métodos para a computação emergente.

Especificamente, um autômato celular (AC) é constituído por diversas células individuais; na verdade, são 149 células em cada autômato dos exemplos apresentados. Essas células de estados binários estão distribuídas em um espaço unidimensional sem uma coordenação global. Cada célula troca de estado em função de seu próprio estado e dos estados de seus dois vizinhos imediatos. O AC forma um reticulado bidimensional conforme ele evolui ao longo de períodos de tempo. O reticulado começa com um conjunto de  $N$  células geradas aleatoriamente. No exemplo da Figura 12.14, há 149 células representadas por meio dos primeiros 149 passos de tempo da sua evolução. (Há um período de tempo zero e as células são numeradas como 0, 1, ..., 148). Dois exemplos de comportamento desses autômatos celulares podem ser vistos nos diagramas de espaço-tempo da Figura 12.14. Nesses diagramas, os 1s são indicados como células pretas e os 0s como células brancas. É claro que regras diferentes para as vizinhanças das células produzem padrões diferentes no diagrama de espaço-tempo do AC.

Descrevemos, em seguida, o conjunto de regras que determina a atividade das células que constituem cada AC. A Figura 12.15 apresenta um AC de estado binário, unidimensional, de vizinho mais próximo, com  $N = 11$  células. É apresentado tanto o reticulado como a tabela de regras para atualizar o reticulado. O reticulado é mostrado ao longo de um passo de tempo. O reticulado é, na verdade, um cilindro, onde as pontas esquerda e direita tornam-se vizinhas a cada período de tempo (isso é importante para aplicar o conjunto de regras). A tabela de regras implementa a regra do *voto majoritário* local: se uma vizinhança de três células tem uma maioria de uns, então a célula central se torna um no próximo passo de tempo; caso contrário, ela se torna zero.

Crutchfield e Mitchell desejam encontrar um AC que realize a seguinte computação coletiva, aqui denominada *a maioria vence*: se o reticulado inicial contiver uma maioria de uns, o AC deve evoluir ao longo do tempo

**Figura 12.14** Diagramas de espaço-tempo mostrando o comportamento de dois ACs descobertos pelo algoritmo genético em diferentes rodadas. Eles empregam partículas incorporadas para a computação não local, ou padrões emergentes gerais apresentados. Cada diagrama de espaço-tempo repete certo número de passos de tempo, com 1s mostrados como células pretas e 0s como células brancas; o tempo cresce para baixo no diagrama (Crutchfield e Mitchell, 1995).

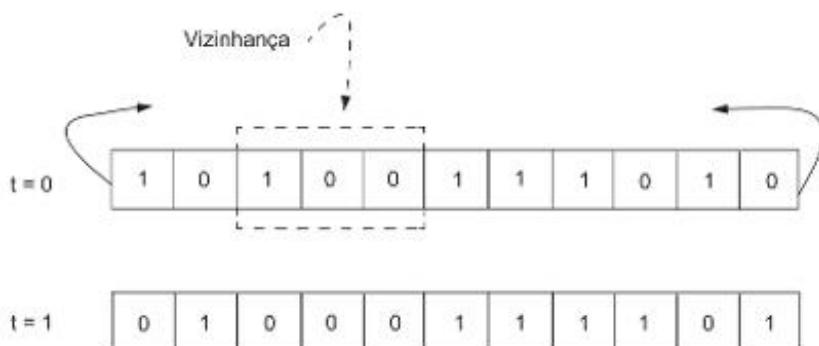


**Figura 12.15** Ilustração de um autômato celular unidimensional, de estado binário, de vizinhos mais próximos, com  $N = 11$ . O reticulado e a tabela de regras para atualizá-lo são mostrados na ilustração. A configuração do reticulado é mostrada ao longo de um passo de tempo. O autômato celular é circular, sendo os dois valores extremos vizinhos.

Tabela de regras:

vizinhança:	000	001	010	011	100	101	110	111
bit de saída:	0	0	0	1	0	1	1	1

Reticulado



para todos os 1s; caso contrário, ele deve evoluir para zeros. Eles usaram ACs com vizinhanças contendo sete células, com uma célula central com três vizinhos de cada lado. Um aspecto interessante dessa pesquisa é que é difícil se projetar uma regra de AC que execute a computação “a maioria vence”. Na verdade, em Mitchell et al. (1996), eles mostram que a regra simples do “voto majoritário” de sete vizinhos não realiza a computação “a maioria vence”. O AG é usado para buscar uma regra que o faça.

O AG, na Seção 12.1, é usado para criar as tabelas de regras para diferentes experimentos com os ACs. Especificamente, um AG é usado para evoluir as regras para a população unidimensional de células de estado binário que constitui cada AC. Foi concebida uma função de aptidão para recompensar aquelas regras que dão suporte ao resultado “a maioria vence” para o AC. Assim, ao longo do tempo, o AG construiu um conjunto de regras cuja aptidão era uma função de seu sucesso, reforçando regras globalmente majoritárias. As regras com maior aptidão na população foram selecionadas para sobreviver e foram combinadas aleatoriamente por recombinação para produzirem descendentes, com cada filho sujeito a uma pequena probabilidade de mutação. Esse processo repetiu-se por 100 gerações, com a aptidão sendo estimada para um novo conjunto de células iniciais a cada geração. Mais detalhes podem ser encontrados em Crutchfield e Mitchell (1995).

Como podemos quantificar a computação emergente que os ACs mais bem-sucedidos suportam? Como acontece em muitos processos naturais que se estendem espacialmente, as configurações de células com frequência se organizam ao longo do tempo em regiões espaciais que são dinamicamente homogêneas. Idealmente, a análise e a determinação de regularidades subjacentes deveriam ser um processo automático. Na realidade, Hanson e Crutchfield (1992) criaram uma linguagem para autômatos finitos determinísticos minimais e a usaram para descrever as bacias de atração dentro de cada autômato celular. Essa linguagem pode ser usada para descrever nosso exemplo.

Algumas vezes, como na Figura 12.14(a), essas regiões são óbvias para o observador humano como domínios invariantes, isto é, regiões nas quais o mesmo padrão ocorre de modo recorrente. Rotulamos esses domínios como valores  $\wedge^0$  e, então, eliminamos os elementos invariantes por filtragem para melhor descrever as interações ou computações realizadas pelas interseções desses domínios. A Tabela 12.4 descreve três regiões  $\wedge$ :  $\wedge^0$ , os 0s repetidos;  $\wedge^1$ , os 1s repetidos; e  $\wedge^2$ , o padrão repetido 10001. Há outras regiões  $\wedge$  na Figura 12.14(a), mas discutiremos apenas esse subconjunto.

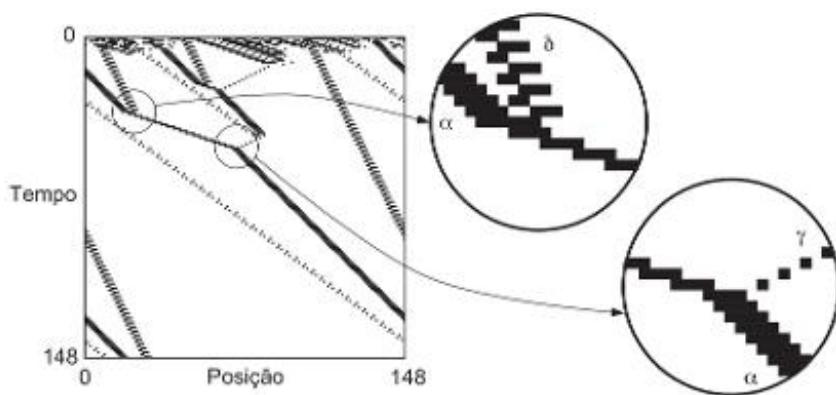
**Tabela 12.4** Catálogo de domínios regulares, partículas (fronteiras de domínios), velocidades de partículas (entre parênteses) e interações de partículas, do comportamento no espaço-tempo do AC da Figura 12.14(a). A notação  $p \sim \wedge^x \wedge^y$  significa que  $p$  é a partícula que forma a fronteira entre os domínios regulares  $\wedge^x$  e  $\wedge^y$ .

Domínios regulares		
$\wedge^0 = 0^*$	$\wedge^1 = 1^*$	$\wedge^2 = (10001)^*$
Partículas (velocidades)		
$\alpha \sim \wedge^1 \wedge^0 (1)$		$\beta \sim \wedge^0 \wedge^1 (0)$
$\gamma \sim \wedge^2 \wedge^0 (-2)$		$\delta \sim \wedge^0 \wedge^2 (1/2)$
$\eta \sim \wedge^2 \wedge^1 (4/3)$		$\mu \sim \wedge^1 \wedge^2 (3)$
Interações		
decaimento	$\alpha \rightarrow \gamma + \mu$	
reação	$\alpha + \delta \rightarrow \mu, \eta + \alpha \rightarrow \gamma, \mu + \gamma \rightarrow \alpha$	
aniquilação	$\eta + \mu \rightarrow \emptyset_1, \gamma + \delta \rightarrow \emptyset_0$	

Com a eliminação por filtragem dos elementos invariantes dos domínios  $\wedge$ , podemos ver as interações desses domínios. Na Tabela 12.4, descrevemos a interação de seis áreas  $\wedge$ , por exemplo, as partículas nas fronteiras dos domínios  $\wedge^1$  e  $\wedge^0$ . A fronteira, onde todos os domínios 1 encontram todos os domínios 0, é denominada *partícula α incorporada*. Crutchfield e Mitchell afirmam que a coleção de partículas incorporadas é um mecanismo primário para carregar informação (ou sinais) por longos intervalos de espaço-tempo. As operações lógicas nessas partículas, ou sinais, ocorrem quando elas colidem. Assim, a coleção de domínios, paredes de domínio, partículas e interações de partículas de um AC representam os elementos básicos de processamento de informação incorporados no comportamento do AC, isto é, a computação intrínseca do AC.

Como exemplo, a Figura 12.16 descreve a lógica emergente da Figura 12.14(a). As áreas de domínio  $\wedge$  foram filtradas em relação ao seu conteúdo invariante para permitir que as partículas de parede sejam facilmente observadas. Cada região ampliada da Figura 12.16 demonstra a lógica de duas partículas incorporadas que interagem. A interação de partícula  $\alpha + \delta \rightarrow \mu$ , mostrada à direita na figura, implementa a lógica de mapear uma configuração

**Figura 12.16** Análise da lógica emergente para a classificação de densidades da Figura 12.14(a). Esse AC tem três domínios, seis partículas e seis interações de partículas, como observado na Tabela 12.4. Os domínios foram filtrados usando um transdutor não linear de 18 estados. Adaptado de Crutchfield e Mitchell (1995).



espacial representando os sinais  $\alpha$  e  $\delta$  para um sinal  $\mu$ . Um detalhamento similar é mostrado para a interação de partículas  $\mu + \gamma \rightarrow \alpha$ , que mapeia uma configuração representando  $\mu$  e  $\gamma$  para o sinal  $\alpha$ . Uma listagem mais completa das interações de partículas da Figura 12.16 pode ser encontrada na Tabela 12.4.

Em resumo, um resultado importante da pesquisa de Crutchfield-Mitchell é a descoberta de métodos para descrever a computação emergente dentro de um sistema espacialmente distribuído, consistindo de processadores celulares interagindo localmente. A localização da comunicação dentro das células impõe uma restrição de comunicação global. O papel dos AGs é descobrir regras locais para as células, cujo efeito é realizar processamento de informação sobre “grandes” distâncias no espaço-tempo. Crutchfield e Mitchell usaram ideias emprestadas da teoria de linguagens formais para caracterizar esses padrões de espaço-tempo.

Para Crutchfield e Mitchell, o resultado do autômato evolutivo reflete um nível totalmente novo de comportamento, que é distinto das interações de baixo nível das células distribuídas. As interações globais baseadas em partículas demonstram como pode emergir uma coordenação complexa dentro de uma coleção de ações individuais simples. O resultado do AG operando sobre regras de células locais mostrou como um processo evolucionário, por tirar vantagem de certas ações de células formadoras de padrões não lineares, produziu um novo nível de comportamento e, também, o delicado equilíbrio necessário para a computação emergente efetiva.

Os resultados da pesquisa de Crutchfield-Mitchell são importantes, pois, com a ajuda dos AGs, eles demonstraram a emergência de invariâncias de alto nível dentro de um autômato celular. Além disso, eles apresentam ferramentas computacionais, adaptadas da teoria das linguagens formais, que podem ser usadas para descrever essas invariâncias. A continuação dessas pesquisas poderá elucidar a emergência da complexidade, que é a característica definitiva dos seres vivos, fundamental para entender as origens da mente, das espécies e dos ecossistemas.

## 12.4 Epílogo e referências

A pesquisa em algoritmos genéticos e em aprendizado baseado na biologia começou com a concepção dos algoritmos genéticos por John Holland. Sua pesquisa inicial inclui *Adaptation in Natural and Artificial Systems* (1975), um estudo sobre fenômenos emergentes em sistemas autorreplicáveis (1976), e *Escaping Brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems* (1986), introduzindo os classificadores. Alguns exemplos do trabalho na análise de sistemas genéticos podem ser encontrados em Forrest e Mitchell (Forrest, 1990; Mitchell, 1996; Forrest e Mitchell, 1993a, 1993b). Outros pesquisadores, especialmente Goldberg (1989), Mitchell (1996) e Koza (1992, 1994), continuaram a análise formal dos algoritmos genéticos e de seu aprendizado.

Como observado anteriormente, Holland (1986) foi também responsável pelo projeto original dos sistemas classificadores, que criam uma visão macro do aprendizado, do ponto de vista de todo o sistema. Outra visão similar é representada pelo projeto SOAR (Laird et al., 1986a, 1986b; Rosenbloom e Newell, 1987; Rosenbloom et al., 1993).

Diversos livros populares, incluindo *Chaos*, de James Gleick (1987) ajudaram a estabelecer a importância de sistemas emergentes e caóticos. Os conceitos desses sistemas, como atratores de estranhos, tiveram um papel importante nas arquiteturas conexionistas (Capítulo 11).

John Koza é o projetista principal da área de pesquisa em programação genética. Suas principais contribuições são descritas em: *Genetic Programming: On the programming of computers by means of natural selection* (1992) e *Genetic Programming II: Automatic discovery of reusable programs* (1994). Kosa (2005) usa a programação genética para montar ferramentas de engenharia, incluindo sistemas de controle. O exemplo do uso de programas genéticos para aprender a terceira lei de Kepler, Seção 12.2.2, foi sugerido por Mitchell (1996).

O Jogo da Vida foi originalmente apresentado pelo matemático John Horton Conway, mas se tornou famoso pela abordagem de Martin Gardner na *Scientific American* (1970, 1971). A pesquisa sobre o poder computacional das máquinas de estados finitos remonta ao projeto dos primeiros computadores digitais. John von Neumann foi muito ativo nessas pesquisas e foi, de fato, o primeiro a mostrar que uma máquina de estados finitos tinha o poder

computacional da Máquina Universal de Turing. A maior parte das pesquisas iniciais de von Neumann foi apresentada nos textos de Arthur Burks (1966, 1970, 1987). Outros pesquisadores (Hightower, 1992; Koza, 1992) descrevem como a pesquisa em vida artificial evoluiu desde esse primeiro trabalho em máquinas de estados finitos. Outros importantes pesquisadores em vida artificial são Langton (1986) e Ackley e Littmann (1992). Os anais das primeiras conferências sobre vida artificial foram editados por Langton (1989, 1990).

Dennett, *Darwin's Dangerous Ideas* (1995) e *Sweet Dreams* (2006), e outros filósofos abordaram a importância de conceitos evolucionários no pensamento filosófico. Recomendamos, também, *Full House: The Spread of Excellence from Plato to Darwin* (Gould, 1996).

Além das breves descrições das pesquisas em agentes na Seção 12.3 (Brooks, 1986, 1987, 1991a, 1991b; Nilsson, 1994; Benson e Nilsson, 1995), há muitos outros projetos nesse domínio, incluindo o modelo de Maes (1989, 1990) do espalhamento de ativação em redes comportamentais e a extensão da arquitetura de quadro-negro por Hayes-Roth et al. (1993). Os anais da AAAI, IJCAI e conferências sobre vida artificial contêm vários artigos desse importante domínio de pesquisa. Crutchfield e Mitchell (1995) deram suporte à nossa apresentação da Seção 12.3.3.

## 12.5 Exercícios

1. O algoritmo genético visa dar suporte à busca pela diversidade genética com a sobrevivência de habilidades importantes (representadas pelos padrões genéticos) para um domínio de problema. Descreva como operadores genéticos diferentes podem dar suporte simultaneamente a dois desses objetivos.
2. Discuta o problema de conceber representações para operadores genéticos visando realizar a busca por soluções em domínios diferentes. Qual é o papel aqui do *viés induutivo*?
3. Considere o problema de satisfação de FNC da Seção 12.1.1. Como o papel do número de termos disjuntivos na expressão FNC influencia (fornecendo um viés) o espaço de solução? Considere outras representações e operadores genéticos possíveis para o problema de satisfação de FNC. Você consegue conceber outra medida de aptidão?
4. Construa um algoritmo genético para solucionar o problema de satisfação de FNC.
5. Considere o problema do caixeleiro-viajante da Seção 12.1.1. Discuta o problema de selecionar uma representação apropriada para esse problema. Projete outros operadores genéticos e medidas de aptidão apropriados para esse problema.
6. Construa um algoritmo genético para uma solução do problema do caixeleiro-viajante.
7. Discuta o papel das técnicas representacionais, como a codificação de Gray, para moldar o espaço de busca para o algoritmo genético. Discuta dois domínios de problema em que técnicas similares sejam importantes.
8. Leia o teorema dos esquemas de Holland (Mitchell, 1996; Koza, 1992). Como a teoria dos esquemas de Holland descreve a evolução do espaço de solução do AG? O que ele tem a dizer sobre problemas que não são codificados como cadeias de bits?
9. Como o algoritmo *bucket brigade* (Holland, 1986) se relaciona com o algoritmo retropropagação (Seção 11.3)?
10. Escreva um programa para resolver a terceira lei de Kepler do problema do movimento dos planetas, descrita com uma representação preliminar na Seção 12.2.2.
11. Discuta as restrições (apresentadas na Seção 12.2.2) ao uso de técnicas de programação genética para resolver problemas. Por exemplo, quais componentes de uma solução não podem ser evoluídos no paradigma da programação genética?
12. Leia a discussão inicial do Jogo da Vida na coluna escrita por Gardner na *Scientific American* (1970, 1971). Discuta outras estruturas de vida artificial, semelhantes aos *planadores*, apresentados na Seção 12.3.1.
13. Escreva um programa de vida artificial que implemente a funcionalidade das figuras 12.10 a 12.13.

14. A área de pesquisa em agentes foi introduzida na Seção 12.3. Recomendamos a leitura adicional de qualquer um dos projetos mencionados, mas especialmente a pesquisa de Brooks, Nilsson e Benson, ou Crutchfield e Mitchell. Escreva uma breve monografia sobre um desses tópicos.
15. Discuta o papel do viés indutivo nas representações, estratégias de busca e operadores usados nos modelos de aprendizado apresentados no Capítulo 12. Essa questão pode ser resolvida? Isto é, o modelo de aprendizado genético funciona apenas por causa de suas suposições representacionais ou ele pode ser traduzido em domínios mais amplos?
16. Para considerações mais aprofundadas sobre evolução e emergência de complexidade, leia e discuta *Darwin's Dangerous Idea* (Dennett, 1995) ou *Full House: The Spread of Excellence from Plato to Darwin* (Gould, 1996).

# Aprendizado de máquina: probabilístico

*A teoria da probabilidade nada mais é do que o senso comum reduzido a cálculo...*

—LAPLACE

*A finalidade do cálculo é a percepção, e não os números...*

—RICHARD HAMMING

## 13.0 Modelos estocásticos e dinâmicos do aprendizado

Como observado nos capítulos anteriores, há dois motivos principais para se usar as ferramentas probabilísticas para entender e prever os fenômenos no mundo: primeiro, os eventos podem ser genuinamente relacionados um ao outro de forma probabilística, e, segundo, as relações causais determinísticas entre situações no mundo em mudança podem ser tão complexas que suas interações são mais bem capturadas por modelos estocásticos. A comunidade de IA adotou e implantou modelos probabilísticos por essas duas razões, e essas tecnologias estocásticas tiveram uma influência muito importante no projeto, no poder e na flexibilidade dos algoritmos de aprendizado de máquina.

A regra de Bayes, apresentada inicialmente na Seção 5.3, é a base para os modelos probabilísticos do aprendizado de máquina. Os métodos Bayesianos têm suporte à interpretação de novas experiências com base nas relações aprendidas anteriormente: entender eventos atuais é uma função das expectativas de aprendizado dos eventos anteriores. De modo semelhante, os modelos de Markov e suas variantes oferecem uma base para os métodos estocásticos de aprendizado. Em uma cadeia de Markov, a probabilidade de um evento em qualquer ponto no tempo é uma função da probabilidade com a qual os eventos ocorreram em períodos de tempo anteriores. Em uma *cadeia de Markov de primeira ordem*, introduzida na Seção 9.3, a probabilidade de um evento no tempo  $t$  é uma função apenas da probabilidade de seu predecessor no tempo  $t - 1$ .

O Capítulo 13 tem três seções, primeiro a Seção 13.1 continua a apresentação dos modelos de Markov introduzindo os *modelos ocultos de Markov* (ou MOMs) e diversas variantes e extensões importantes. A Seção 13.2

estende a apresentação das redes Bayesianas iniciada na Seção 9.3, focalizando especialmente as *redes Bayesianas dinâmicas* (ou RBDs) e diversas extensões. A Seção 13.3 continua a apresentação do aprendizado por reforço iniciada na Seção 10.7, com o acréscimo de medidas probabilísticas para reforço.

## 13.1 Modelos ocultos de Markov (MOMs)

No início do século XX, o matemático russo Andrey Markov propôs uma matemática para modelar eventos de degrau discreto relacionados probabilisticamente. Em comparação com uma sequência determinística de eventos, como se poderia esperar quando um computador executa um conjunto fixo de instruções, os formalismos de Markov apoiaram a ideia de que cada evento pode estar probabilisticamente relacionado aos eventos que o precedem, como nos padrões de fenômenos ou palavras em uma sentença falada. Em nossas tentativas de programar computadores para aprender fenômenos em um mundo em mutação, as percepções de Markov provaram ser extremamente importantes.

### 13.1.1 Introdução e definição dos modelos ocultos de Markov

Um *modelo oculto de Markov* (ou *MOM*) é uma generalização da *cadeia* (ou *processo*) de Markov tradicional, introduzida na Seção 9.3. Nas cadeias de Markov vistas até esse ponto, cada estado correspondia a um evento físico — e observável — discreto, como o clima em determinado horário do dia. Essa classe de modelos é bastante limitada e agora a generalizamos para uma classe mais ampla de problemas. Nesta seção, estendemos o modelo de Markov para situações em que as observações são, por si sós, funções probabilísticas dos estados ocultos atuais. Assim, o modelo resultante, denominado *modelo oculto de Markov*, é um processo estocástico duplamente embutido.

Um MOM pode ser descrito como um processo estocástico observável, com suporte de outro processo estocástico não observável, ou oculto. Um exemplo de uso para um MOM seria o suporte para o reconhecimento de palavras baseado em computador, por meio da interpretação de sinais acústicos ruidosos. Os próprios padrões fonéticos, ou seja, os fonemas que o orador pretende usar como parte da produção de palavras em uma linguagem, compõem o nível oculto do modelo de Markov. As observações, ou seja, os sinais acústicos ruidosos que são ouvidos, são uma função estocástica desses fonemas intencionados. Os fonemas que o orador intencionou podem ser vistos apenas probabilisticamente por meio do fluxo de nível superior (observável) dos sinais acústicos ruidosos. Definimos um MOM:

#### Definição

#### MODELO OCULTO DE MARKOV

Um modelo gráfico é denominado *modelo oculto de Markov (MOM)* se for um modelo de Markov cujos estados não são diretamente observáveis, mas ocultos por mais um sistema estocástico interpretando sua saída. Mais formalmente, dado um conjunto de estados  $S = s_1, s_2, \dots, s_n$  e dado um conjunto de probabilidades de transição de estado  $A = a_{11}, a_{12}, \dots, a_{1n}, a_{21}, a_{22}, \dots, a_{nn}$ , há um conjunto de probabilidades de observação,  $O = p_i(o_t)$ , cada um expressando a probabilidade de uma observação  $o_t$  (no tempo  $t$ ) ser gerada por um estado  $s_t$ .

Agora, damos dois exemplos de MOMs, adaptados de Rabiner (1989). Primeiro, considere a situação do lançamento de uma moeda. Suponha que haja uma pessoa em uma sala lançando moedas, uma por vez. Não sabemos o que está acontecendo na sala; pode haver várias moedas, selecionadas aleatoriamente para o lançamento, e cada uma delas pode ter seus próprios resultados viésados, ou seja, elas podem ser moedas “viciadas”. Tudo o que ob-

servamos fora da sala é uma série de resultados de lançamento, por exemplo, O observável = H, H, T, H, T, H, ... . Nossa tarefa é criar um modelo de moedas lançadas dentro da sala que produza a sequência observável resultante.

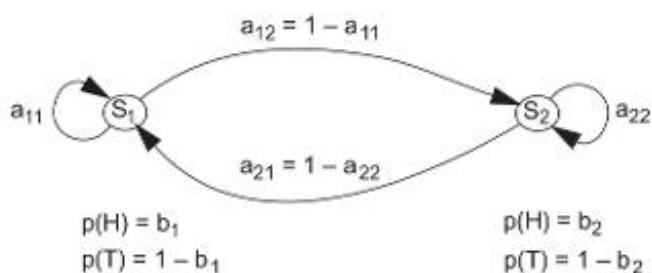
Dada essa situação, agora tentamos modelar o conjunto resultante de observações emanando da sala. Começamos com um modelo simples; suponha que haja apenas uma moeda sendo lançada. Nesse caso, só temos que determinar o viés da moeda enquanto ela é lançada com o tempo, produzindo o conjunto de observações de cara (H) e coroa (T). Essa técnica resulta em um modelo de Markov observável diretamente (ordem zero), que é simplesmente o conjunto de tentativas de Bernoulli. Na verdade, esse modelo pode ser muito simples para capturar, de modo confiável, o que ocorre no problema do lançamento de moedas.

Em seguida, consideramos um modelo de duas moedas, com dois estados ocultos  $s_1$  e  $s_2$ , como visto na Figura 13.1. Uma matriz de transição probabilística, A, controla o estado em que o sistema se encontra, ou seja, qual moeda é lançada, a qualquer momento. Cada moeda tem um viés de H/T diferente,  $b_1$  e  $b_2$ . Suponha que observemos a seguinte sequência de lançamentos: O = H, T, T, H, T, H, H, H, T, T, H. Esse conjunto de observações poderia ser devido ao seguinte caminho através do diagrama de transição de estados da Figura 13.1:  $s_2, s_1, s_1, s_2, s_2, s_2, s_1, s_2, s_2, s_1, s_2$ .

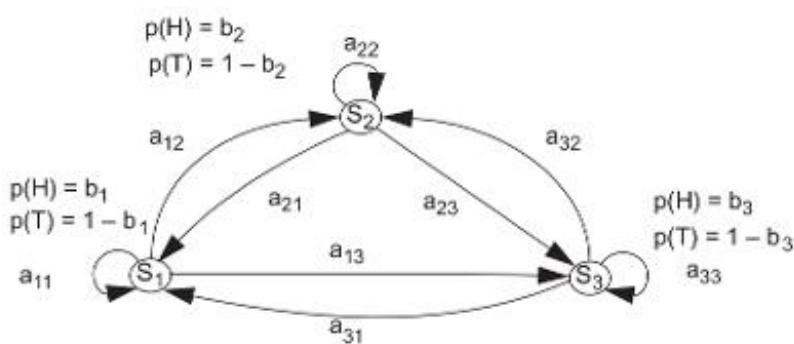
Um terceiro modelo é apresentado na Figura 13.2, onde três estados (três moedas) são usados para capturar a saída do lançamento de moeda. Novamente, cada moeda/estado,  $s_i$ , tem seu próprio viés,  $b_i$ , e o estado do sistema (qual moeda é lançada) é uma função de um evento probabilístico, como o lançamento de um dado e a matriz de transição, A. A máquina de três estados poderia considerar a saídas H/T com a sequência de estados:  $s_3, s_1, s_2, s_3, s_3, s_1, s_1, s_2, s_3, s_1, s_3$ .

A questão difícil para o problema do lançamento de moeda é a escolha de um modelo ótimo. O modelo mais simples tem um parâmetro, o viés de uma única moeda. O modelo de dois estados tem quatro parâmetros, as tran-

**Figura 13.1** Um diagrama de transição de estados de um modelo oculto de Markov de dois estados preparados para o problema do lançamento de moeda. Os  $a_{ij}$  são determinados pelos elementos da matriz de transição  $2 \times 2$ , A.



**Figura 13.2** Diagrama de transição de estados para um modelo oculto de Markov do lançamento de moeda. Cada moeda/estado,  $s_i$ , tem seu viés,  $b_i$ .



sições de estado e os vieses de cada moeda. O modelo de três estados da Figura 13.2 tem nove parâmetros. Com mais graus de liberdade, o modelo maior é mais poderoso para capturar uma situação (possivelmente) mais complexa, porém, as questões de complexidade para determinar os parâmetros para o modelo maior podem arruinar o exercício. Por exemplo, os observáveis reais poderiam ser produzidos por uma situação mais simples, em que o modelo maior seria incorreto, subespecificado e exigiria muito mais dados para estabelecer qualquer grau de confiança. Problemas semelhantes, como o *sobreajuste*, também assombrariam os modelos menores. Por fim, em cada modelo existe uma suposição implícita de *estado estacionário*, ou seja, que as possibilidades de transição e viés do sistema não mudam com o tempo.

Para um segundo exemplo, considere o problema de  $N$  urnas, cada uma contendo uma coleção de  $M$  bolas com diferentes cores. O processo físico de obter observações é, de acordo com um processo aleatório, escolher uma das  $N$  urnas. Quando uma urna é selecionada, uma bola é removida e sua cor é registrada no fluxo de saída. A bola é então recolocada e o processo aleatório associado à urna atual seleciona a próxima (que poderia ser a mesma) urna para continuar o processo. Esse processo gera uma sequência de observações contendo as cores das bolas. É óbvio que o MOM mais simples correspondente a esse processo é o modelo em que cada estado corresponde a uma urna específica, os valores da matriz de transição para esse estado produzem a próxima escolha de estado e, por fim, um modelo em que a probabilidade de cor da bola é definida pelo número e cor das bolas para cada estado (urna). Porém, a menos que o modelador tenha alguma informação anterior, seria uma tarefa realmente muito difícil determinar todos os parâmetros exigidos para selecionar um MOM ideal.

Na próxima seção, consideraremos diversas formas variantes dos MOMs.

### 13.1.2 Variantes importantes dos modelos ocultos de Markov

Na seção anterior, observamos que os MOMs são modelos extremamente poderosos, embora simples, que podem representar domínios incertos. A simplicidade computacional dos MOMs vista até aqui segue o princípio Markoviano de um sistema de primeira ordem: que o estado atual depende apenas do estado anterior. Nesta seção, exploramos como podemos modificar os princípios básicos de um MOM para conseguir capacidades de representação ainda mais ricas. Os MOMs regulares da Seção 13.1.1 têm limitações em domínios específicos, e, em seguida, demonstramos as variantes do MOM que podem tratar de muitas dessas limitações.

Uma extensão dos MOMs bastante utilizada é o *MOM autorregressivo*, em que uma observação prévia também pode influenciar o valor da observação atual. Essa extensão é usada para capturar mais informações do passado e fatorá-las na interpretação do estado atual. Para modelar processos complexos, compostos de combinações de conjuntos de subprocessos mais simples, frequentemente se utilizam *MOMs fatoriais*. Uma extensão dos MOMs, os *MOMs hierárquicos* (ou *MOHM*), considera que cada estado de um sistema é, por si próprio, um MOM. Em seguida, consideraremos estas e outras extensões dos MOMs com mais detalhes.

#### **MOMs autorregressivos (AR-MOM)**

Nos modelos ocultos de Markov, a propriedade de primeira ordem formula uma hipótese em que o estado atual depende apenas do estado anterior imediato, e que as variáveis observadas não estão correlacionadas. Isso pode ser uma limitação, em que o estado atual não captura adequadamente as informações disponíveis nos estados anteriores. No raciocínio por diagnóstico, por exemplo, isso pode levar a generalizações inferiores, especialmente ao modelar dados da série temporal. O fato é que, em ambientes complexos, os valores observáveis em um período de tempo frequentemente se correlacionam a observações subsequentes. Isso deve ser fatorado na estrutura do MOM.

Contornamos esse aspecto da generalização limitada permitindo que a observação anterior influencie a interpretação da observação atual, com as influências MOM normais do estado anterior. Esse MOM é denominado *modelo oculto de Markov autorregressivo*, cujo modelo de emissão é definido pelo modelo de probabilidade não zero:

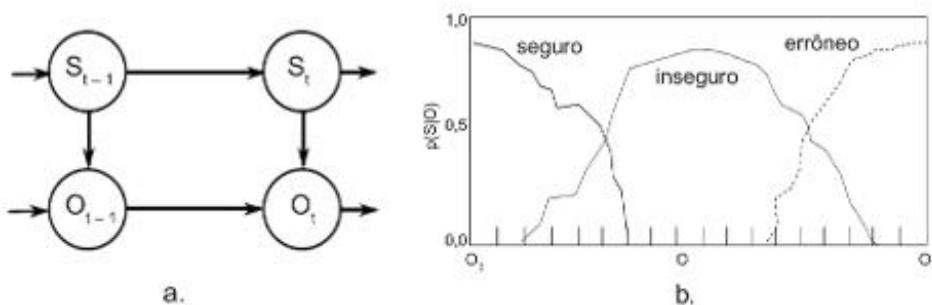
$$p(O_t | S_t, O_{t-1}).$$

onde  $O$  é o estado de emissão ou observação e  $S$  representa a camada oculta, como vemos na Figura 13.3(a). A diferença entre o MOM tradicional e o AR-MOM é a existência de um arco (influência) entre  $O_{t-1}$  e  $O_t$ . A motivação para essa influência é direta: o estado de observação em um período de tempo será um indicador de como será a próxima observação. Em outras palavras, os valores de saída observados de uma análise de série temporal mudarão de acordo com uma distribuição subjacente.

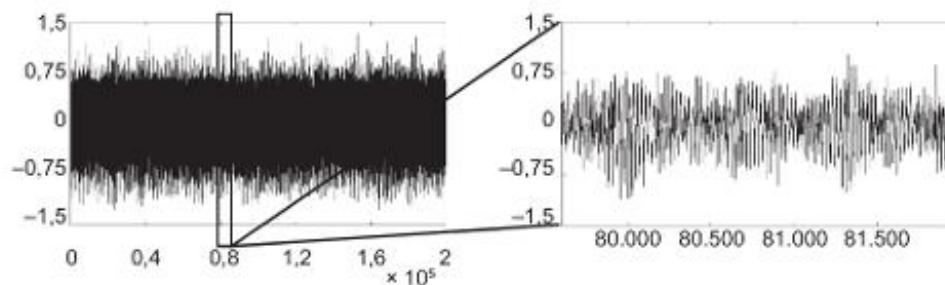
O AR-MOM normalmente leva a modelos com maior probabilidade de convergência do que os MOMs regulares, especialmente ao modelar dados de série temporal extremamente complexos e ruidosos. Demonstramos o AR-MOM com um exemplo adaptado do trabalho de Chakrabarti et al. (2007). A tarefa é monitorar o estado de funcionamento de um sistema de rotor de helicóptero. A Figura 13.4 representa uma amostra dos dados que é observada a partir do sistema em funcionamento. Os dados são a saída de uma série de sensores, incluindo informações de medições de calor e vibração em múltiplos componentes. A Figura 13.5 mostra os resultados do processamento dos dados da Figura 13.4 usando uma transformação rápida de Fourier (FFT, do inglês *Fast Fourier Transform*) para produzir dados equivalentes no domínio de frequência. Esses pontos de dados são então usados para treinar um AR-MOM. Os três estados do nó oculto ( $S_t$ ) na Figura 13.3 são seguro, inseguro e errôneo.

O objetivo desse projeto era investigar técnicas para a detecção de falhas e diagnóstico em sistemas mecânicos. Usando um conjunto de dados de séries temporais de sensores monitorando processos mecânicos, a tarefa foi elaborar um modelo quantitativo do processo inteiro e treiná-lo em conjuntos de dados (falhas disseminadas como um eixo rachado) para uso posterior no diagnóstico de tempo real e precisão de falhas futuras. Depois que o AR-MOM foi treinado, ele foi liberado para trabalhar em tempo real. Embora a precisão relatada (Chakrabarti et al., 2006) estivesse um pouco acima de 75%, é importante ver como essas ferramentas de prognóstico podem ser elaboradas.

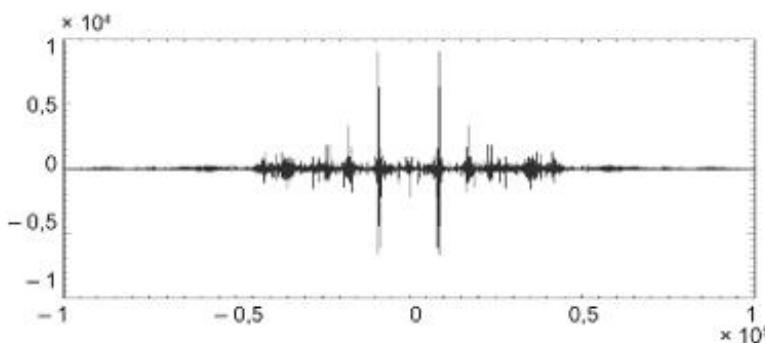
**Figura 13.3** (a) Os estados oculto,  $S$ , e observável,  $O$ , do AR-MOM, onde  $p(O_t | S_t, O_{t-1})$ . (b) Os valores do  $S_t$  oculto do AR-MOM de exemplo: seguro, inseguro e errôneo.



**Figura 13.4** Uma seleção dos dados de tempo real por vários períodos de tempo, com uma fatia de tempo expandida, para o AR-MOM.



**Figura 13.5** Os dados da série temporal da Figura 13.4 processados por uma transformação rápida de Fourier no domínio de frequência. Esses foram os dados submetidos ao AR-MOM para cada período de tempo.



### MOMs fatoriais

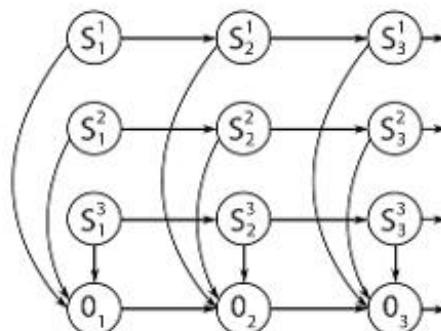
Em nossas discussões anteriores sobre MOMs e suas variantes, descrevemos sistemas cujo espaço de estados era parcialmente oculto. Só poderíamos obter dicas para o espaço de estados por meio de uma série de observações. Tentamos deduzir a sequência de estados ocultos subjacentes usando informações da sequência observada de dados emitidos pelo sistema.

Mas o que acontece se o processo subjacente for muito complexo e não puder ser representado de modo eficiente como um simples conjunto de estados? O que acontece se o processo subjacente for, na realidade, uma combinação de vários subprocessos? Nossa MOM regular não pode capturar suficientemente as informações sobre os processos nessas situações. Precisamos de um modelo mais rico para representar situações mais complexas.

Os MOMs fatoriais são uma solução possível. A Figura 13.6 mostra um MOM fatorial autorregressivo. Vemos que o processo subjacente pode ser dividido em  $n$  subprocessos, onde cada um desses  $n$  processos influencia a observação atual,  $O_t$ . Além disso, cada subprocesso segue a propriedade de Markov de primeira ordem, ou seja, seu estado atual depende somente do(s) seu(s) estado(s) anterior(es). Essa relação com a variável aleatória observável pode ser definida pela DPC (distribuição de probabilidade condicional):

$$p(O_t | O_{t-1}, S_1^1, S_1^2, \dots, S_1^l)$$

**Figura 13.6** Um MOM fatorial autorregressivo, onde o estado observável  $O_t$ , no tempo  $t$ , depende do subprocesso múltiplo ( $S_t$ ).  $S_1^1$  é  $O_{t-1}$ .



### MOMs hierárquicos (MOHMs)

MOMs hierárquicos são usados para modelar sistemas extremamente complexos. No *modelo oculto hierárquico de Markov* (ou *MOHM*), cada estado individual é considerado um modelo probabilístico autocontido. Mais precisamente, cada estado do MOHM pode ser um MOHM por si só. Isso implica que os estados do MOHM emitem sequências de observações, em vez de apenas observações isoladas, como é o caso para os MOMs padrão e as variantes vistas até este ponto.

Quando um estado em um MOHM for alcançado, ele ativará seu próprio modelo probabilístico, ou seja, ativará um dos estados do MOHM subjacente, que, por sua vez, poderá ativar seu MOHM subjacente, e assim por diante, sendo o caso básico o MOM tradicional. O processo é repetido até que um estado, denominado *estado de produção*, seja ativado. Somente estados de produção emitem símbolos de observação no sentido usual do modelo oculto de Markov. Quando o estado de produção tiver emitido um símbolo, o controle retorna ao estado que ativou o estado de produção. Os estados que não emitem diretamente símbolos de observação são denominados *estados internos*. A ativação de um estado em um MOHM sob um estado interno é denominada *transição vertical*. Após uma transição vertical ser concluída, ocorre uma transição *horizontal* para um estado no mesmo nível. Quando uma transição horizontal leva a um estado terminal, o controle é retornado ao estado no MOHM mais acima na hierarquia, que produziu a última transição vertical.

Observe que uma transição vertical pode resultar em mais transições verticais antes de alcançar uma sequência de estados de produção e finalmente retornar ao nível superior. Assim, os estados de produção visitados fazem surgir uma sequência de símbolos de observação que são produzidos pelo estado no nível superior.

### MOMs n-grama

Como já observamos, nos modelos de Markov tradicionais de primeira ordem, o estado atual, dado o estado imediatamente anterior, é independente de todos os estados anteriores. Embora essa regra reduza a complexidade computacional para uso dos processos de Markov, pode acontecer de o estado anterior apenas não poder capturar completamente informações importantes, disponíveis no domínio do problema. Tratamos explicitamente dessa situação com o *modelo oculto de Markov n-grama* (ou *MOM n-grama*). Esses modelos, principalmente em suas formas *bigrama* e *trigrama*, são especialmente importantes em métodos estocásticos para a compreensão da linguagem natural, apresentados com mais detalhes na Seção 15.4.

Com os modelos de Markov bigrama, a interpretação do estado atual observável depende apenas da interpretação do estado anterior, como  $p(O_t | O_{t-1})$ ; isso é equivalente à cadeia de primeira ordem tradicional de Markov. Embora a suposição de primeira ordem de Markov seja óbvia, é interessante pensar a respeito do que significa um processamento de bigrama, por exemplo, na compreensão da linguagem natural. Ele formula a hipótese de que, para o reconhecimento de palavras ou fonemas, dada uma sequência de palavras anteriores, a probabilidade de uma palavra em particular ocorrer é mais bem estimada quando usamos apenas a interpretação da palavra imediatamente anterior. Assim, o bigrama é uma hipótese para dar melhores resultados do que o conhecimento de nenhuma informação de palavra anterior ou conhecimento de uma sequência maior de palavras anteriores. Por exemplo, dada uma coleção de dados da linguagem,  $p(\text{cordeiro} | \text{pequeno})$  é um previsor melhor da palavra observada atual ser cordeiro do que  $p(\text{cordeiro} | \text{um pequeno})$ ,  $p(\text{cordeiro} | \text{tinha um pequeno})$  ou  $p(\text{cordeiro} | \text{Maria tinha um pequeno})$ .

De modo semelhante, modelos de trigrama são modelos de Markov em que a interpretação do estado atual observável depende dos dois estados anteriores. O modelo de trigrama é representado por  $P(O_t | O_{t-1}, O_{t-2})$ . Seguindo com o exemplo de linguagem natural, o uso de modelos de trigrama implica que  $p(\text{cordeiro} | \text{um pequeno})$  é um previsor melhor da palavra ser cordeiro do que  $p(\text{cordeiro} | \text{pequeno})$ , ou de mais palavras anteriores, incluindo  $p(\text{cordeiro} | \text{Maria tinha um pequeno})$ .

Podemos estender o modelo n-grama de Markov para qualquer tamanho  $n$  que capture as invariâncias presentes nos dados que tentamos modelar. Um modelo n-grama de Markov é uma cadeia de Markov em que a probabilidade do estado atual depende exatamente dos  $n-1$  estados anteriores.

Como veremos com mais detalhes no Capítulo 15, modelos n-grama são especialmente úteis para capturar sequências de fonemas, sílabas ou palavras na compreensão da fala ou outras tarefas de reconhecimento de linguagem. Podemos definir o valor de  $n$  dependendo da complexidade desejada do domínio que queremos representar. Se  $n$  for muito pequeno, podemos ter poder representativo mais fraco e perder a capacidade de capturar o po-

der preditivo de sequências maiores. Por outro lado, se o valor de  $n$  for muito grande, então o custo computacional para validar o modelo pode ser muito alto ou mesmo impossível. O motivo principal para isso é que as expectativas anteriores para combinações de fonema ou palavra são tomadas de grandes bancos de dados (geralmente, bancos de dados combinados) de fenômenos de linguagem coletados, denominados *corpo*. Simplesmente podemos não ter dados suficientes para validar um  $n$ -grama com um  $n$  muito grande. Por exemplo, podemos não ter informação alguma sobre  $p(\text{cordeiro} | \text{Maria tinha um pequeno})$ , enquanto temos muita informação de  $p(\text{cordeiro} | \text{pequeno})$ ,  $p(\text{cordeiro} | \text{um pequeno})$  ou  $p(\text{cordeiro} | \text{o})$ . Normalmente, em tarefas de reconhecimento de fala ou texto, usamos  $n$ -gramas com um valor de  $n$  não superior a 3.

Há diversas outras variantes de MOMs que não descrevemos aqui, incluindo *MOMs de memória mista*, que usam vários modelos de Markov de baixa ordem para contornar questões de complexidade. Outra extensão dos MOMs autorregressivos, chamada *modelos enterrados de Markov*, permite dependências não lineares entre os nós observáveis. *MOMs de entrada-saída* são usados para o mapeamento entre sequências de entradas e uma sequência de saídas. Em situações em que as interações entre subprocessos de um processo complexo são mais complicadas do que apenas uma composição, um *MOM acoplado* normalmente é usado. Um *modelo oculto semi-Markov* generaliza um MOM para evitar o efeito de decaimento geométrico dos MOMs, fazendo com que a probabilidade de transições entre estados ocultos dependa da quantidade de tempo que se passou desde a última transição de estado. Veja as referências na Seção 13.4.

Na próxima seção, aplicamos a tecnologia MOM de bigrama ao problema de interpretar combinações de fonemas em inglês e usamos a programação dinâmica com o algoritmo de Viterbi para implementar uma busca MOM.

### 13.1.3 Usando MOMs e Viterbi para decodificar sequências de fonemas

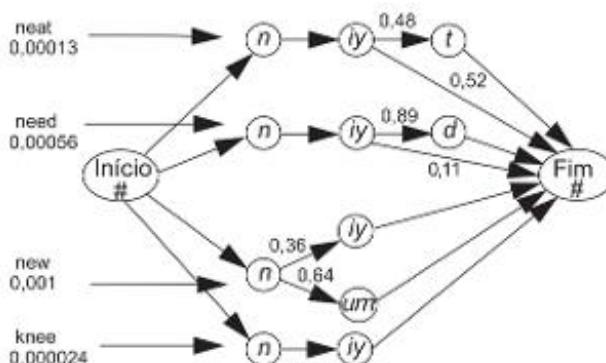
Nossa seção final sobre modelos ocultos de Markov demonstra um uso importante da tecnologia MOM: identificar padrões na linguagem natural falada. Supomos a presença de um corpo de linguagem grande que dê suporte às expectativas anteriores sobre combinações de fonemas. Por fim, usamos algoritmos de programação dinâmica, introduzidos na Seção 4.1, para implementar a inferência MOM. Quando a programação dinâmica é usada para achar a probabilidade máxima de uma sequência *a posteriori*, isso normalmente é denominado *algoritmo de Viterbi*.

O exemplo a seguir, de análise computacional de padrões de fala humana e uso do algoritmo de Viterbi para interpretar sequências de fonemas, é adaptado de Jurafsky e Martin (2008). A entrada nessa máquina probabilística é uma sequência de fonemas, ou sons básicos da fala. Estes são derivados da decomposição do sinal acústico produzido pelo uso da linguagem falada. É incomum na fala automatizada compreender que os sinais acústicos seriam capturados de forma não ambígua como uma sequência de fonemas. Em vez disso, os sinais da fala são interpretados como fonemas específicos probabilisticamente. Consideraremos a interpretação não ambígua de sinais para simplificar nossa apresentação do processamento do algoritmo de Viterbi do MOM.

A Figura 13.7 apresenta um segmento de um banco de dados de palavras, relacionadas pela proximidade dos conjuntos de fonemas que compõem seus componentes acústicos. Embora esse conjunto de palavras, *neat*, *new*, *need* e *knee*, seja apenas uma pequena parte do vocabulário completo do inglês, pode-se imaginar que um número muito grande desses agrupamentos relacionados possa dar suporte a um sistema de compreensão da fala. A Figura 13.7 é um exemplo de uma máquina de estados finitos probabilística, conforme introduzida inicialmente na Seção 5.3. Consideraremos nosso corpo de linguagem como uma coleção de máquinas de estados finitos probabilísticas similares que podem dar medidas de probabilidade prévias de diferentes combinações possíveis de fonemas e, por fim, como esses fonemas podem ser vistos como partes de palavras.

O objetivo da análise é determinar qual palavra em inglês, a partir do nosso banco de dados de palavras, representa melhor a entrada de um sinal acústico. Isso requer o uso da tecnologia MOM, pois a representação de palavras possíveis é estocástica por si só. Na máquina de estados finitos não determinística da Figura 13.7, a sequência de fonemas nos dá o conjunto de observações para interpretar. Suponha que a sequência de observações seja composta dos fonemas #, n, iy, #, onde # indica uma pausa de quebra entre os sons. Usamos o algoritmo de Viterbi para ver qual caminho pela máquina de estados finitos probabilística melhor captura essas observações. No modo direto, Viterbi acha iterativamente o melhor estado seguinte e seu valor, e depois define um ponteiro

**Figura 13.7** Uma máquina de estados finitos probabilística representando um conjunto de palavras em inglês relacionadas foneticamente. A probabilidade de cada palavra ocorrer está abaixo dessa palavra. Adaptado de Jurafsky e Martin (2008).



para ele. No modo inverso, repercorremos esses ponteiros para obter o melhor caminho. Assim, a saída de Viterbi é um dos caminhos ideais dos estados a partir do grafo associado à probabilidade desse caminho.

Usando Viterbi, cada estado da busca é associado a um valor. O valor para estar no estado  $s_i$  no tempo  $t$  é  $viterbi[s_i, t]$ . O valor associado ao próximo estado  $s_j$  na máquina de estados no tempo  $t + 1$  é  $viterbi[s_j, t + 1]$ . O valor para o próximo estado é calculado como o produto da pontuação do estado atual,  $viterbi[s_i, t]$ , vezes a probabilidade de transição de ir do estado atual para o estado seguinte,  $caminho[s_i, s_j]$ , vezes a probabilidade de observação  $s_j$  dado  $s_i$ ,  $p(s_j | s_i)$ . A probabilidade de transição,  $caminho[s_i, s_j]$ , é tomada da máquina de estados finitos não determinística e  $p(s_j | s_i)$  é tomada de probabilidades de observação conhecidas de pares de fonemas que ocorrem no idioma inglês.

A seguir, apresentamos o pseudocódigo para o algoritmo de Viterbi. Observe sua semelhança com a programação dinâmica, na Seção 4.1. A matriz para armazenar e coordenar valores precisa ter suporte para iteração por  $R$  linhas — igual ao número de fonemas na máquina de estados finitos probabilística mais dois, para lidar com cada estado mais os estados início e fim. Ela também precisa percorrer  $C$  colunas — igual ao número de observações mais dois — para lidar com o uso do fonema vazio #. As colunas também indicam a sequência de tempo para observações e cada estado precisa estar ligado ao fonema observado apropriado, como vemos na Figura 13.8.

```

função Viterbi(observações de tamanho T, MEF probabilística)
    início
        número := número de estados na MEF
        cria matriz de probabilidades viterbi[R = N + 2, C = T + 2];
        viterbi[0, 0] := 1,0;
        para cada passo (observação) t de 0 a T faça
            para cada estado  $s_i$  de i = 0 ao número faça
                para cada transição de  $s_i$  a  $s_j$  na MEF probabilística faça
                    início
                        novo-contador := viterbi[ $s_i, t$ ] x caminho[ $s_i, s_j$ ] x  $p(s_j | s_i)$ ;
                        se ((viterbi[ $s_j, t + 1$ ] = 0) ou (novo-contador > viterbi[ $s_j, t + 1$ ]))
                            então
                                início
                                    viterbi[ $s_j, t + 1$ ] := novo-contador
                                    anexa ponteiro [ $s_j, t + 1$ ] à lista de ponteiros
                                fim;
                            fim;
                    retorna viterbi[R, C];
                    retorna lista de ponteiros
                fim.
            fim;
        fim.
    fim.

```

**Figura 13.8** Um rastreamento do algoritmo de Viterbi em vários dos caminhos da Figura 13.7. As linhas informam o valor máximo para Viterbi em cada palavra para cada valor de entrada (linha superior). Adaptado de Jurafsky e Martin (2008).

Início = 1,0	#	$\pi$	$iy$	#	fim
neat 0,00013 2 caminhos	1,0	$1,0 \times 0,00013 = 0,00013$	$0,00013 \times 1,0 = 0,00013$	$0,00013 \times 0,52 = 0,000067$ (2 caminhos)	
need 0,00056 2 caminhos	1,0	$1,0 \times 0,00056 = 0,00056$	$0,00056 \times 1,0 = 0,00056$	$0,00056 \times 0,11 = 0,00062$ (2 caminhos)	
new 0,001 2 caminhos	1,0	$1,0 \times 0,001 = 0,001$	$0,001 \times 0,36 = 0,00036$ (2 caminhos)	$0,00036 \times 1,0 = 0,00036$	
knee 0,000024 1 caminho	1,0	$1,0 \times 0,000024 = 0,000024$	$0,000024 \times 1,0 = 0,000024$	$0,000024 \times 1,0 = 0,000024$	
Melhor total					0,00036

A Figura 13.8, adaptada de Jurafsky e Martin (2008), apresenta um rastreamento do algoritmo de Viterbi processando um componente da máquina de estados finitos probabilística da Figura 13.7 e a sequência de fonemas #, n, iy, # (as observações de tamanho  $T = 4$ ). As ligações de rastreamento de volta indicam o caminho ideal através da máquina de estados finitos probabilística. Esse caminho indica que a interpretação mais provável da sequência de fonemas observados é a palavra new.

Na próxima seção, apresentamos outro modelo gráfico, uma generalização dos modelos de Markov, denominada rede Bayesiana dinâmica (ou RBD).

## 13.2 Redes Bayesianas dinâmicas e aprendizado

Frequentemente, encontramos problemas na modelagem em que o processo subjacente a ser caracterizado é mais bem descrito como uma sequência ou progressão de estados. Os dados coletados dos processos sequenciais normalmente são indexados por um parâmetro como a estampa de hora da aquisição de dados ou uma posição na sequência de texto. Essas dependências sequenciais podem ser determinísticas e estocásticas. Os processos dinâmicos que são inherentemente estocásticos exigem modelos capazes de modelar seu não determinismo.

Embora as sequências determinísticas possam ser caracterizadas muito bem por máquinas de estados finitos (MEFs, vistas na Seção 3.1), algumas sequências importantes podem ser arbitrariamente longas. Portanto, a modelagem de cada ocorrência dessas sequências como conjuntos de estados individuais torna-se problemática, pois as quantidades de transições de estado crescem exponencialmente com relação ao número de estados em determinado instante. A modelagem probabilística dessas sequências, portanto, permite-nos manter modelos tratáveis, capturando ainda a informação de sequência relevante ao problema sendo tratado.

Na Seção 13.2, descrevemos duas técnicas diferentes de aprendizado, o aprendizado de estrutura e o aprendizado de parâmetro, no contexto dos modelos gráficos. Descrevemos rapidamente as técnicas para aprendizado de estrutura, incluindo o uso da amostragem Monte Carlo da cadeia de Markov (Markov chain Monte Carlo, MCMC).

Também detalhamos um meta-algoritmo popular, denominado *maximização de expectativa* (ou *Expectation Maximization, ME*) para aprendizado de parâmetro. Mostramos como um algoritmo recursivo de programação dinâmica, denominado *algoritmo de Baum-Welch*, adapta o meta-algoritmo ME para uso com RBDs e MOMs.

Na Seção 13.3, explicamos como os modelos de Markov podem ser usados para apoio à decisão, introduzindo o *processo de decisão de Markov* (ou *Markov Decision Processes, PDM*) e o *PDM parcialmente observado* (ou *Partially Observable Markov Decision Process, PDMPO*). Depois, apanhamos o aprendizado por reforço (conforme apresentado originalmente na Seção 10.7) e o suplementamos com modelos de estado probabilístico e realimentação. Por fim, encerramos o Capítulo 13 usando o aprendizado por reforço por meio de um PDM para projetar um sistema de navegação por robô.

### 13.2.1 Redes Bayesianas dinâmicas

Na Seção 9.3, introduzimos a rede Bayesiana de crença, um formalismo muito popular e bem-sucedido para modelagem probabilística. Porém, para a modelagem de processos dinâmicos ou sequenciais, RBCs são bastante inflexíveis. Elas não têm provisão para modelar dados de série temporal, incluindo os processos causais ou correlações de dados sequenciais que são frequentemente encontrados em tarefas complexas, incluindo a compreensão da linguagem natural e os diagnósticos em tempo real. O problema é que as dependências condicionais de uma RBC são todas consideradas ativas no mesmo instante.

Na Seção 9.3, introduzimos a *rede Bayesiana dinâmica* (ou *RBD*), embora tenhamos deixado uma discussão mais detalhada para este capítulo. Em resumo, uma rede Bayesiana dinâmica, às vezes denominada *temporal*, é uma sequência de redes Bayesianas idênticas, cujos nós estão ligados na dimensão (direcionada) do tempo. Redes Bayesianas dinâmicas são uma sequência de modelos gráficos direcionados de processos estocásticos. Elas são generalizações de ferramentas estocásticas populares, como modelos ocultos de Markov (MOMs) e sistemas lineares dinâmicos (SLDs) que representam estados ocultos (e observados) como variáveis de estado, que podem ter interdependências complexas através dos períodos de tempo. A estrutura gráfica oferece um modo de especificar dependências condicionais e, portanto, oferece uma parametrização compacta para o modelo.

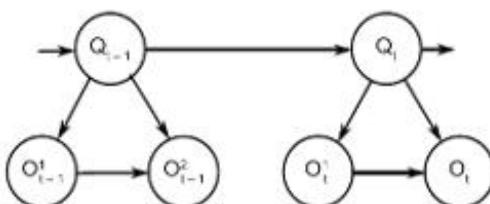
As redes Bayesianas individuais que compõem as RBDs não são dinâmicas no sentido de que elas próprias não mudam com o tempo, a suposição *estacionária*, mas ainda dão suporte à modelagem de processos dinâmicos. Assim, as RBDs funcionam sob a hipótese de que seus parâmetros de definição são invariantes no tempo. Porém, como veremos, os nós ocultos podem ser adicionados para representar as condições de operação atuais, criando assim misturas de modelos para capturar não invariâncias periódicas na dimensão do tempo.

Em cada período de tempo (fatia), uma RBD é semelhante a uma RBC estática que tem um conjunto  $Q_t$ , de  $N_h$  variáveis aleatórias representando os estados ocultos, e um conjunto  $O_t$ , de  $N_o$  variáveis aleatórias representando os estados que são observáveis. Cada variável aleatória pode ser discreta ou contínua. Temos que definir um modelo que representa as distribuições de probabilidade condicionais entre os estados em uma única fatia de tempo e um modelo de transição que representa a relação entre as RBCs para fatias de tempo consecutivas. Portanto, se  $Z_t = \{Q_t \cup O_t\}$  for a união dos dois conjuntos de variáveis, os modelos de transição e observação podem ser definidos como o produto das distribuições de probabilidade condicional em uma RBC de dois períodos de tempo sem ciclos, denominado  $B_t$ . A notação  $(1:N)$  significa para todas as variáveis de 1 a N:

$$p(Z_t(1:N) | Z_{t-1}(1:N)) = \prod_{i=1}^N p(Z_t(i) | \text{Gen}(Z_t(i))),$$

onde  $Z_t(i)$  é o i-ésimo nó na fatia de tempo t,  $\text{Gen}(Z_t(i))$  são os genitores de  $Z_t(i)$  e  $N = N_h + N_o$ . Observe que aqui, para simplificar, restringimos a definição a processo de Markov de primeira ordem, onde as variáveis no tempo t são influenciadas apenas por seu predecessor no tempo t - 1. Podemos representar a distribuição de estado inicial incondicional,  $p(Z_1(1:N))$ , como uma rede Bayesiana incondicional  $B_1$ .  $B_1$  e  $B_t$ , para todo t, definem a RBD. Mostramos duas fatias de tempo de uma RBD simples na Figura 13.9.

**Figura 13.9** Um exemplo de RBD com duas fatias de tempo. O conjunto  $Q$  representa variáveis aleatórias ocultas, o conjunto  $O$ , variáveis observadas, e  $t$  indica tempo.



Como deve ser óbvio, a rede Bayesiana dinâmica é uma generalização dos modelos ocultos de Markov apresentados na Seção 13.1. Na Figura 13.9, as redes Bayesianas são ligadas entre os tempos  $t$  e  $t + 1$ . Pelo menos um nó das duas redes precisa estar ligado ao longo do tempo.

### 13.2.2 Aprendizado em redes Bayesianas

O aprendizado é fundamental para a ação inteligente. Agentes inteligentes geralmente registram eventos observados no passado, na expectativa de que essa informação possa ter alguma utilidade no futuro. Por todo o seu tempo de vida, um agente inteligente encontra muitos eventos, talvez mais do que ele tenha recursos para armazenar. Portanto, torna-se necessário codificar a informação do evento com uma abstração concisa, em vez de uma documentação completa de cada observação. Além disso, a compactação de informações auxilia a recuperação mais rápida. O aprendizado, portanto, pode ser visto como uma combinação de compactação de dados, indexação e armazenamento para recuperação eficiente.

Qualquer observação nova por um agente pode ser ligeiramente diferente das observações anteriores, mas ainda pertencer a uma classe geral de eventos anteriores. Nesse caso, é desejável aprender esse evento como uma instância da classe anterior, ignorando os detalhes menos importantes do evento específico. O novo evento também pode ser muito diferente de todos os eventos anteriores, e então o agente precisará aprendê-lo como uma nova classe ou como uma anomalia específica sem classe. Um agente precisa criar um número suficiente de classes que caracterize o espaço de parâmetros, além de ser capaz de tratar com sucesso novos eventos anômalos. Assim, o bom aprendizado é um equilíbrio da generalidade e da especificidade.

No contexto dos modelos gráficos probabilísticos, encontramos dois problemas de aprendizado gerais que se encaixam na discussão anterior, o *aprendizado de estrutura*, em que novas relações inteiras (modelos gráficos) são aprendidas, e o *aprendizado de parâmetro*, em que componentes de um modelo existente são recalibrados.

#### Aprendizado de estrutura e busca

O aprendizado de estrutura enfatiza o problema de determinar a existência de dependências estatísticas entre variáveis. Por exemplo, as respostas às questões a seguir ajudariam a determinar a estrutura de um modelo gráfico probabilístico do tempo. As nuvens escuras estão relacionadas às chances de chuva? Existe uma correlação entre a posição de Saturno e o número de furacões em um dia qualquer? O aquecimento global está relacionado à presença de tufões? O clima no Ártico está relacionado a mudanças de fronteiras do deserto do Saara? Como é evidente, em um modelo probabilístico, uma resposta positiva a essas questões garantiria uma dependência condicional entre as variáveis em questão e uma resposta negativa nos permitiria tratar essas variáveis como independentes.

No projeto de um modelo gráfico probabilístico, cada variável seria dependente de outra variável, enquanto preserva a restrição de que a rede resultante é um grafo aciclico direcionado (GAD). No pior dos casos, então, ficamos com um GAD totalmente conectado como modelo. Porém, isso é muito raro na maior parte das aplicações de qualquer tamanho, pois a criação das tabelas de probabilidade condicional torna-se intratável. Uma percepção im-

portante no suporte ao projeto de modelos gráficos é que frequentemente podemos determinar que diversas variáveis exibam independências uma em relação à outra no contexto do problema sendo modelado. Como a complexidade da inferência nas redes Bayesianas de crença depende do número de dependências entre as variáveis no modelo, queremos ter o mínimo de dependências possível, preservando ainda a validade do modelo. Em outras palavras, queremos podar o máximo possível de dependências (desnecessárias) a partir do grafo totalmente conectado.

O aprendizado de estrutura pode, assim, ser considerado como um problema da busca para a estrutura ideal no espaço de todas as estruturas possíveis para determinado conjunto de variáveis representando um domínio de aplicação. Conforme observado por Chickering et al. (1994), uma solução ideal para essa busca pelos conjuntos de dados existentes é NP-difícil. A estrutura ideal é aquela que descreve bem os dados enquanto permanece a mais simples possível.

No entanto, essa estrutura aprendida precisa descrever os dados já observados e também se ajustar a novas informações, possivelmente relevantes. Assim, quando criamos (aprendemos) uma estrutura complexa que caracteriza os dados acumulados exatamente, podemos acabar com uma estrutura que seja muito específica e que, como resultado, deixa de generalizar bem os novos dados. Na literatura sobre otimização, essa situação às vezes é chamada de *sobreajuste*.

Embora a busca no espaço exponencial de possíveis estruturas de grafo acíclico direcionado por métodos de força bruta seja um tanto irrealista para aplicações práticas com muitas variáveis, existem alguns princípios que realmente ajudam a enfrentar esse problema. A primeira regra prática é a navalha de Occam, ou o *princípio da parcimônia*. Entre duas estruturas concorrentes que funcionam de modo comparável, a estrutura mais simples deverá ser preferível à mais complexa. A navalha de Occam afirma que, em alguns casos, pode ser uma boa ideia buscar estruturas possíveis indo da simples à complexa, terminando a busca quando acharmos uma estrutura que seja simples o bastante, porém bem-sucedida por ser um modelo adequado para a tarefa em mãos. O uso dessa heurística da simplicidade impõe uma crença Bayesiana anterior na busca pela estrutura.

Porém geralmente é difícil definir *a priori* uma medida da complexidade para uma estrutura. A escolha de uma medida razoável de complexidade que imponha uma ordem total sobre possíveis estruturas não é óbvia. Pode haver classes de estruturas que sejam igualmente complexas, quando poderíamos impor uma ordem parcial no espaço da estrutura. Contudo, até mesmo entre estruturas da mesma classe, uma busca exaustiva por todas as possibilidades normalmente é impraticável.

Assim como em muitos problemas difíceis em IA, a indução da estrutura no caso geral simplesmente não é possível em situações razoavelmente complexas. Como resultado, aplicamos heurísticas em uma tentativa de tornar tratável a busca pela estrutura. Com um modelo funcional, embora insatisfatório, de uma situação, um método heurístico consiste em realizar uma busca local gulosa perto dos componentes do modelo que falharam. Como o objetivo é achar apenas uma otimização local dentro da estrutura gráfica, isso poderia oferecer uma solução boa e suficiente para o problema de modelagem. Outra heurística seria contar com o especialista humano no domínio de problema para obter ajuda sobre como repensar a estrutura do modelo. Novamente, o especialista humano poderia focar nos pontos de falha do modelo atual. Um método mais geral para o problema de indução do modelo consiste na amostragem do espaço de modelos possíveis. Apresentamos, a seguir, uma técnica popular para esse tipo de amostragem, Monte Carlo da cadeia de Markov.

### ***Abordagem Monte Carlo da cadeia de Markov (MCCM)***

*Monte Carlo da cadeia de Markov* (ou MCCM) é uma classe de algoritmos para amostragem a partir de distribuições de probabilidade. A MCCM é baseada na construção de uma cadeia de Markov que tenha a distribuição desejada como distribuição de equilíbrio. O estado da cadeia após um grande número de etapas, às vezes chamadas *tempo de aquecimento*, é usado então como uma amostra para as distribuições desejadas de uma possível estrutura GAD para o domínio da aplicação. Naturalmente, a qualidade da amostra da cadeia de Markov melhora em função do número de amostras tomadas.

No contexto da busca de estrutura, consideramos que as estruturas possíveis para determinado conjunto de variáveis formam o espaço de estados de uma cadeia de Markov. A matriz de transição para esses estados é preenchida por uma técnica baseada em pesos. Os pesos podem começar como uniformes, em que um percurso aleatório de Monte Carlo entre essas estruturas pesaria todas as modificações estruturais de um modo igualmente provável.

Com o tempo, é claro, os pesos maiores são usados para transições entre estruturas relacionadas mais de perto. Se a estrutura A for diferente da estrutura B em apenas uma ligação, então um percurso de Monte Carlo desse

espaço é uma busca nos vizinhos heuristicamente organizados dessas estruturas. Cada estrutura, quando amostrada, é associada a uma pontuação, normalmente baseada em *máximo a posteriori* (ou *MAP*), o  $p(\text{estrutura} | \text{dados})$ . A técnica de amostragem de Monte Carlo atravessa o espaço de estruturas, escolhendo com uma probabilidade mais alta as transições que melhoram a verossimilhança dos dados, em vez de transições que tornam a pontuação pior. Depois de várias iterações, a técnica de MCCM converge para estruturas melhores, aperfeiçoando a distribuição posterior de  $p(\text{estrutura} | \text{dados})$  em comparação com seu ponto de partida. Novamente, há o problema de descobrir os máximos locais no espaço amostrado; esse problema pode ser tratado por técnicas como *reinício aleatório* ou *témpora simulada* (*simulated annealing*).

### Aprendizado de parâmetro e maximização de expectativa (ME)

O aprendizado de parâmetro pode ser contrastado com o aprendizado de estrutura pelo espaço dos modelos gráficos. Dado um modelo existente de um domínio de problema, o aprendizado de parâmetro tenta deduzir a distribuição conjunta ideal para um conjunto de variáveis, dado um conjunto de observações. Esse aprendizado é frequentemente utilizado como uma sub-rotina dentro de uma busca de estrutura geral. Quando existe um conjunto de dados completo, o aprendizado de parâmetro para uma distribuição se reduz à contagem.

Entretanto, em muitas aplicações interessantes dos modelos gráficos, podemos encontrar o problema de conjuntos incompletos de dados para uma variável. Outro problema (relacionado) é a existência de possíveis variáveis ocultas ou latentes dentro do modelo. Em casos como esses, a *maximização de expectativa* (ou *ME*) serve como uma ferramenta poderosa para deduzir as estimativas prováveis para distribuições conjuntas. O algoritmo ME foi explicado inicialmente (e recebeu seu nome) em um artigo clássico de A. Dempster, N. Laird e D. Rubin (1977). Esse artigo generalizou a metodologia e desenvolveu a teoria por trás dele.

A ME provou ser um dos métodos mais populares de aprendizado nas modelagens estatística e probabilística. O algoritmo ME é usado em estatística para achar estimativas de máxima verossimilhança dos parâmetros em modelos probabilísticos, em que o modelo depende de possíveis variáveis latentes não observadas. A ME é um algoritmo iterativo, que alterna entre realizar uma etapa de expectativa (E), que calcula uma expectativa da verossimilhança de uma variável, incluindo variáveis latentes como se fossem observadas, e a maximização (M), que calcula as estimativas de máxima verossimilhança dos parâmetros a partir da verossimilhança esperada encontrada na etapa E. Os parâmetros achados na etapa M são então utilizados para iniciar outra etapa E, e o processo é repetido até a convergência, ou seja, até que reste muito pouca oscilação entre os valores nas etapas E e M. Apresentamos um exemplo do algoritmo de maximização de expectativa na Seção 13.2.3.

### Maximização de expectativa para MOMs (Baum-Welch)

Uma ferramenta importante para o aprendizado de parâmetro é o Baum-Welch, uma variante do algoritmo ME. Ele calcula as estimativas de máxima verossimilhança e estimativas do modo posterior para os parâmetros (probabilidades de transmissão e emissão) de um MOM ou outro modelo gráfico temporal, quando dados apenas os conjuntos de dados de treinamento de emissão (observáveis). O algoritmo Baum-Welch tem duas etapas:

1. Calcular as probabilidades para a frente e para trás de cada estado temporal;
2. Com base em 1, determinar a frequência dos valores do par transição-emissão e dividi-la pela probabilidade da sequência inteira.

A etapa 1 do algoritmo Baum-Welch utiliza o *algoritmo para a frente e para trás*. Esse é o algoritmo que dá suporte à programação dinâmica, como vimos inicialmente na Seção 4.1 e vimos várias vezes desde então, incluindo a Seção 13.1.3, quando foi usado para interpretar sequências de fonemas.

A etapa 2 do algoritmo Baum-Welch é usada para calcular a probabilidade de uma sequência de saída em particular, dados os parâmetros do modelo, no contexto do modelo oculto de Markov ou outros modelos temporais. A etapa 2 equivale a calcular a contagem esperada de pares transição-emissão particulares. Toda vez que uma transição é achada, o valor da razão da transição sobre a probabilidade da sequência inteira aumenta e esse valor pode então se tornar o novo valor da transição.

Um procedimento por força bruta para resolver o problema de aprendizado de parâmetro MOM é a geração de todas as sequências possíveis de eventos observados e estados ocultos com suas probabilidades, usando duas ma-

trizes de transição. A probabilidade conjunta de duas sequências, dado o modelo, é calculada multiplicando as probabilidades correspondentes nas matrizes. Esse procedimento tem uma complexidade de tempo de  $O(2TN^T)$ , onde  $T$  é o tamanho das sequências de eventos observados, e  $N$ , o número total de símbolos no alfabeto de estados. O custo da complexidade de tempo é intratável para problemas realísticos, de modo que o método escolhido para implementar o Baum-Welch utiliza a programação dinâmica.

Em seguida, demonstramos um exemplo simples de aprendizado de parâmetro usando a propagação de crença cíclica (Pearl, 1988) com o algoritmo de maximização de expectativa (Dempster et al., 1977).

### 13.2.3 Maximização de expectativa: um exemplo

Ilustramos a maximização de expectativa (ME) com um exemplo e uso do algoritmo de propagação de crença cíclica de Pearl (1988) para mostrar as etapas de inferência. Esse exemplo foi implementado em IBAL, uma linguagem de modelagem estocástica de primeira ordem discutida rapidamente na Seção 9.3, e a representação do campo aleatório de Markov é tirada da lógica cíclica (Pless et al., 2006). Embora métodos mais simples estejam disponíveis para resolver as restrições do exemplo de alarme contra roubo que apresentamos, nosso objetivo é demonstrar um esquema de inferência importante para RBCs, propagação de crença cíclica e o algoritmo ME, em um ambiente inteligível.

Considere a rede Bayesiana de crença da Figura 13.10, uma RBC com 3 nós, alarme  $A_t$ , roubo  $R_t$  e terremoto  $T_t$ . Esse modelo descreve um sistema de alarme contra roubo,  $A_t$ , ativo em um tempo de observação  $t$ , com dependência causal de haver um roubo  $R_t$  ou um terremoto  $T_t$ . Esse modelo, com algumas medidas de probabilidade assumidas, pode ser descrito na lógica cíclica pelo programa:

$$A_t | R_t, T_t = [[[0.999; 0.001], [0.7; 0.3]], [[0.8; 0.2], [0.1; 0.9]]]$$

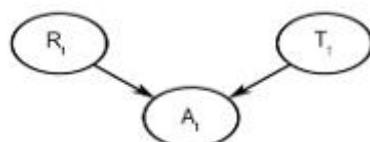
Simplificamos a sintaxe lógica cíclica real para tornar a apresentação mais clara.

Nesse programa, consideramos que todas as variáveis são lógicas. Suponha também que a RBC da Figura 13.10 contenha a distribuição de probabilidade condicional (DPC) especificada anteriormente. Por exemplo, a tabela específica que a probabilidade para o alarme não disparar, dado que existe um roubo e um terremoto, é de 0,001; a probabilidade para o alarme disparar, não havendo roubo nem um terremoto, é de 0,1.

Note também que esse programa especifica uma classe geral de RBCs, ou então uma RBC para cada tempo  $t$ . Isso é semelhante ao método representativo tomado por Kersting e DeRaedt (2000). Porém a lógica cíclica (Pless et al., 2006) estende a técnica tomada por Kersting e DeRaedt de várias maneiras. Por exemplo, a lógica cíclica converte as sentenças da lógica probabilística, como aquela anterior, em um campo aleatório de Markov para aplicar o algoritmo de inferência, com propagação de crença cíclica (Pearl, 1988).

Um *campo aleatório de Markov*, ou *rede de Markov*, é um modelo gráfico probabilístico (ver Seção 9.3) cuja estrutura é um grafo não direcionado (em comparação com os modelos gráficos direcionados das redes Bayesianas de crença). Os nós do grafo de estruturas do campo aleatório de Markov correspondem a variáveis aleatórias e as ligações entre os nós correspondem às dependências entre as variáveis aleatórias conectadas. Na lógica cíclica, o campo aleatório de Markov é usado como uma representação intermediária e equivalente para o modelo gráfico original (assim como a árvore de cliques foi usada na Seção 9.3). O campo aleatório de Markov admite inferência probabilística, em nosso caso, a propagação de crença cíclica (Pearl, 1988).

**Figura 13.10** Uma rede Bayesiana de crença para o exemplo de alarmes contra roubo, terremoto e roubo.



Os componentes quantitativos do campo aleatório de Markov são especificados por um conjunto de funções potenciais, cujo domínio é uma clique da estrutura de grafo do campo aleatório de Markov, e que definem uma correspondência entre o conjunto de todas as atribuições possíveis para as variáveis da clique e o conjunto de números reais não negativos.

No sistema lógico cíclico, é usada uma versão especial de um campo aleatório de Markov. Uma função potencial é atribuída a cada dependência entre variáveis aleatórias. Consequentemente, o campo aleatório de Markov pode ser representado por um grafo bipartido com dois tipos de nós, *nós de variável* e *nós de agrupamento*. Os nós de variável correspondem a variáveis aleatórias, enquanto os nós de agrupamento correspondem a funções potenciais sobre variáveis aleatórias especificadas pelos vizinhos do nó de agrupamento. A Figura 13.11 ilustra o campo aleatório de Markov bipartido que captura as funções potenciais da RBC da Figura 13.10.

Na lógica cíclica, podemos especificar fatos sobre o modelo e fazer as consultas; por exemplo, podemos declarar:

$R_1 = v$   
 $T_1 = f$   
 $A_1 = ?$

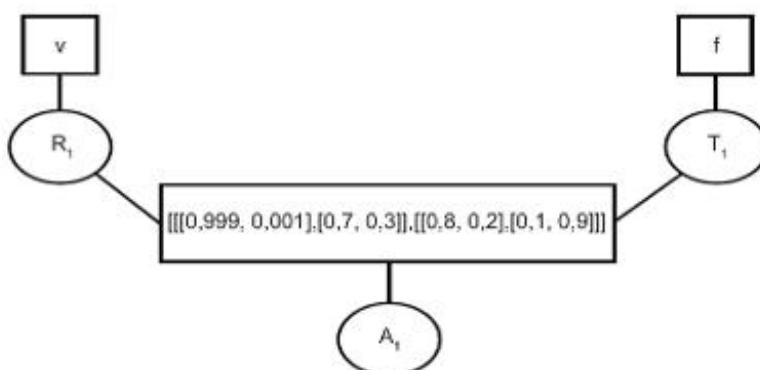
Uma interpretação dessas três declarações é que sabemos que, no tempo 1, houve um roubo, que não houve terremoto e que gostaríamos de saber a probabilidade de o alarme disparar. Usando essas três declarações com  $A_t | R_t, T_t = [[[0,999, 0,001], [0,7, 0,3]], [[0,8, 0,2], [0,1, 0,9]]]$ , o sistema de lógica cíclica constrói o campo aleatório de Markov com os fatos e funções em potencial incorporadas como na Figura 13.11.

Observe que, na Figura 13.11, os fatos do programa são incorporados no campo aleatório de Markov como nós de agrupamento de folha, e a tabela de distribuição de probabilidade condicional (DPC) é codificada no nó de agrupamento conectando os três nós de variáveis. Para deduzir a resposta para essa consulta, o sistema aplica o algoritmo de propagação de crença cíclica ao campo aleatório de Markov. Como especificamos fatos determinísticos simples e o modelo gráfico original não tem laços, o algoritmo de inferência é exato (Pearl, 1988) e converge em uma iteração retornando a distribuição para alarme, [0,8, 0,2].

Além disso, a lógica cíclica admite aprendizado permitindo que seus usuários atribuam distribuições que possam ser aprendidas para regras selecionadas de um programa lógico cíclico. Esses parâmetros podem ser estimados usando um algoritmo de passagem de mensagem derivado da maximização de expectativa (Dempster et al., 1977). Para demonstrar isso, em seguida supomos não ter conhecimento da distribuição de probabilidade condicional da RBC da Figura 13.10. Na lógica cíclica, para obter valores para essa distribuição, o usuário define a DPC como sendo uma distribuição que pode ser aprendida em um tempo arbitrário t. Para indicar isso, usamos a variável L no programa:

$A_t | R_t, T_t = L$

**Figura 13.11** Um campo aleatório de Markov refletindo as funções potenciais das variáveis aleatórias na RBC da Figura 13.10, com as duas observações sobre o sistema.



Podemos agora apresentar um conjunto coletado de observações ao interpretador de lógica cíclica durante um conjunto de períodos de tempo usando a representação de fatos a seguir. Para simplificar, coletamos apenas três pontos de dados em cada um dos três intervalos de tempo:

```

 $T_1 = f$ 
 $R_1 = v$ 
 $A_1 = v$ 
 $T_2 = v$ 
 $R_2 = f$ 
 $A_2 = v$ 
 $T_3 = f$ 
 $R_3 = f$ 
 $A_3 = f$ 

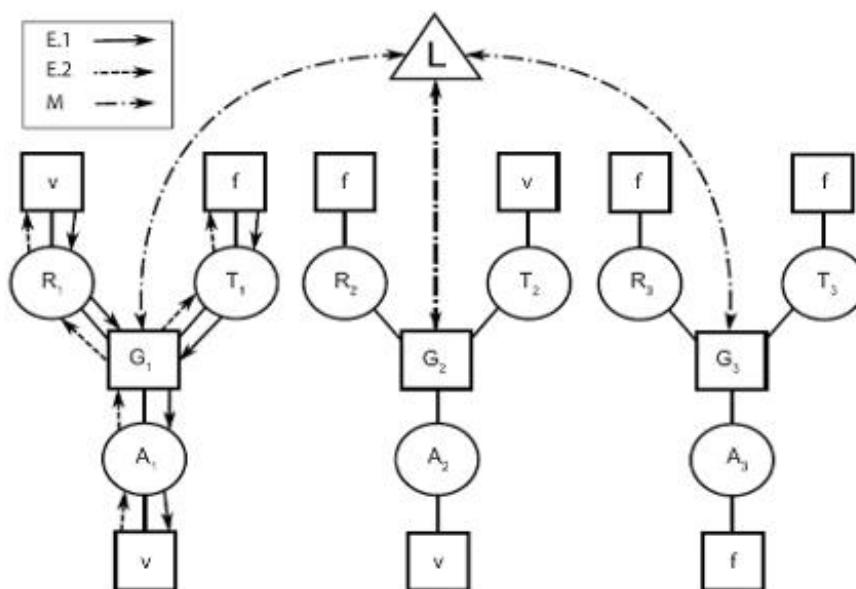
```

A lógica cíclica usa uma combinação do esquema de inferência de propagação de crença cíclica com a maximização de expectativa para estimar os parâmetros de um modelo que podem ser aprendidos. Para fazer isso, o interpretador constrói o campo aleatório de Markov para o modelo, como antes. Para a RBC de alarme da Figura 13.10, seu campo aleatório de Markov correspondente, com o parâmetro que pode ser aprendido  $L$  ligado a cada nó de agrupamento, cuja regra é unificada com a regra que pode ser aprendida, apresentada anteriormente ( $A_v | R_v, T_v = L$ ), é apresentado na Figura 13.12.

Em seguida, preenchemos as distribuições conhecidas da RBC e fazemos uma inicialização aleatória para os nós de agrupamento  $G_1$ ,  $G_2$  e  $G_3$ , bem como para o nó que pode ser aprendido  $L$ . Por fim, esse sistema é apresentado com os dados representando os três períodos de tempo, onde fatos conhecidos são incorporados nos nós de agrupamento de folha, como antes. A Figura 13.12 mostra o resultado.

O algoritmo de maximização de expectativa é um processo iterativo implementado a partir da passagem de mensagem. Após cada iteração, a aproximação atual da tabela de probabilidade condicional é a mensagem do nó que pode ser aprendido a cada um de seus nós de agrupamento ligados. Quando o nó a ser aprendido é atualizado, cada nó de agrupamento vizinho envia uma mensagem a  $L$  (setas com traço e ponto rotuladas com  $M$  na Figura 13.12). Essa mensagem é o

**Figura 13.12** Um nó que pode ser aprendido  $L$  é adicionado ao campo aleatório de Markov da Figura 13.11. O campo aleatório de Markov percorre três períodos de tempo. Para simplificar, a iteração ME só é indicada no instante 1.



produto de todas as mensagens vindo para esse nó de agrupamento a partir dos nós de variável vizinhos. Essas tabelas (não normalizadas) são uma estimativa da probabilidade conjunta em cada nó do agrupamento. Essa é uma distribuição em todos os estados das variáveis condicionadas e condicionantes. O nó que pode ser aprendido toma a soma de todas essas mensagens de agrupamento e as converte em uma tabela de probabilidade condicional normalizada.

Fazendo a inferência (propagação de crença cíclica) sobre os nós de agrupamento e variável, calculamos a mensagem para os nós que podem ser aprendidos. Refletindo a estrutura bipartida do campo aleatório de Markov subjacente, a propagação da crença cíclica é feita com dois estágios da passagem de mensagem: enviar mensagens de todos os nós de agrupamento aos seus nós de variável vizinhos, as setas sólidas rotuladas com E.1 na Figura 13.12, e depois novamente de volta, as setas tracejadas rotuladas com E.2 na figura.

A aplicação desse algoritmo de propagação de crença até a convergência resulta em uma aproximação dos valores esperados. Isso é equivalente à etapa de expectativa no algoritmo ME. A média que ocorre em todos os nós de agrupamento que podem ser aprendidos corresponde à etapa de maximização retornando uma estimativa de máxima verossimilhança dos parâmetros em um nó que pode ser aprendido. A iteração que suporta a convergência nos nós de variáveis e agrupamentos seguida da atualização dos nós que podem ser aprendidos e depois continuando a iteração é equivalente ao algoritmo ME completo.

O resumo do algoritmo, conforme exibido na Figura 13.12, é:

Na etapa de expectativa, ou E:

1. Todos os nós de agrupamento enviam suas mensagens aos nós de variáveis.
2. Todos os nós de variáveis enviam suas mensagens de volta aos nós de agrupamento atualizando a informação dos nós de agrupamento.

Na etapa de maximização, ou M:

1. Os nós de agrupamento enviam mensagens, atualizadas na etapa E, para os nós que podem ser aprendidos, onde sua média é calculada.

No algoritmo descrito, os nós podem ser mudados em qualquer ordem, e as atualizações dos nós de agrupamentos e variáveis podem ser sobrepostas às atualizações dos nós de aprendizado. Esse processo iterativo de atualização representa uma família de algoritmos de estilo ME, alguns dos quais podem ser mais eficientes que o ME padrão (Pless et al., 2006; Dempster et al., 1977) para certos domínios. Uma extensão algorítmica que essa estrutura também suporta é o algoritmo de *propagação de crença generalizada*, proposto por Yedidia et al. (2000).

### 13.3 Extensões estocásticas ao aprendizado por reforço

Na Seção 10.7, introduzimos inicialmente o aprendizado por reforço, onde, com reforço por recompensa, um agente aprende a tomar diferentes conjuntos de ações em um ambiente. O objetivo do agente é maximizar sua recompensa a longo prazo. De modo geral, o agente deseja aprender uma política ou um mapeamento entre estados de recompensa do mundo e suas próprias ações no mundo.

Para ilustrar os conceitos do aprendizado por reforço, consideramos o exemplo de um robô reciclagem (adaptado de Sutton e Barto, 1998). Considere um robô móvel cuja tarefa seja coletar latas de refrigerante vazias de escritórios comerciais. O robô tem uma bateria recarregável. Ele é equipado com sensores para achar latas, além de um braço para coletá-las. Supomos que o robô tenha um sistema de controle para interpretar informações sensoriais para navegação e para mover seu braço a fim de coletar as latas. O robô usa o aprendizado por reforço, com base no nível de carga atual de sua bateria, para procurar latas. O agente deve tomar uma dentre três decisões:

1. O robô pode procurar ativamente uma lata durante certo intervalo de tempo;
2. O robô pode parar e esperar alguém para lhe trazer uma lata; ou
3. O robô pode retornar à estação base para recarregar suas baterias.

O robô toma decisões em algum período de tempo fixo quando uma lata vazia é achada ou quando ocorre algum outro evento. Consequentemente, o robô tem três ações, enquanto seu estado é determinado pelo nível de

carga da bateria. A recompensa é definida em zero por quase todo o tempo, mas se torna positiva quando uma lata é coletada e negativa se a bateria perder sua carga. Note que, nesse exemplo, o aprendizado baseado em reforço é uma parte do robô que monitora o estado físico do robô e também a situação (estado) de seu ambiente externo: o agente do robô monitora o estado do robô e também seu ambiente externo.

Há uma limitação importante da estrutura do aprendizado por reforço tradicional que acabamos de descrever: ele é determinístico por natureza. Para contornar essa limitação e ser capaz de usar o aprendizado por reforço em uma gama maior de tarefas complexas, estendemos a estrutura determinística com um componente estocástico. Nas próximas seções, consideraremos dois exemplos do aprendizado por reforço estocástico: o *processo de decisão de Markov* e o *processo de decisão de Markov parcialmente observável*.

### 13.3.1 Processo de decisão de Markov (ou PDM)

Os exemplos de aprendizado por reforço até aqui, tanto na Seção 10.7 quanto na introdução da Seção 13.3, foram determinísticos por natureza. Com efeito, quando o agente executa uma ação de um estado específico no tempo  $t$ , ele sempre resulta em um novo estado especificado deterministicamente no tempo  $t + 1$ : o sistema é completamente determinístico.

Porém, em muitas aplicações de aprendizado por reforço, essa pode não ser a situação: o agente não tem conhecimento ou controle completo sobre o próximo estado do mundo. Especificamente, tomar uma ação do estado  $s_t$  poderia levar a mais de um estado resultante possível no tempo  $t + 1$ . Por exemplo, no jogo de xadrez, quando fazemos certo movimento, normalmente não sabemos qual será o movimento do nosso oponente. Não temos conhecimento completo do estado em que o mundo estará como resultado do nosso movimento. De fato, essa incerteza é verdadeira para muitos jogos que envolvem mais de uma pessoa. Um segundo exemplo é o jogo de loteria ou outros jogos de azar. Selecionamos um movimento sem certeza da recompensa provável. Um terceiro exemplo consiste na tomada de decisão de agente único (não uma disputa contra um ou mais outros agentes), quando o mundo é tão complexo que uma resposta determinística para a escolha de um estado não é computável. Novamente, nessa situação, uma resposta baseada nas probabilidades poderia ser a melhor que podemos justificar. Em todas essas situações, precisamos de uma estrutura mais poderosa para estabelecer o problema do aprendizado por reforço. Uma estrutura assim é o *processo de decisão de Markov* (ou PDM).

Os PDMs são baseados na propriedade de primeira ordem de Markov, que declara que uma transição para um estado seguinte é representada por uma distribuição de probabilidade que depende somente do estado atual e de possíveis ações. Isso é independente de todos os estados anteriores ao estado atual. Os PDMs são processos estocásticos de tempo discreto, consistindo em um conjunto de estados, um conjunto de ações que podem ser executadas a partir de cada estado e uma função de recompensa associada a cada estado. Por conseguinte, PDMs oferecem uma estrutura muito poderosa, que pode ser usada para modelar uma grande classe de situações de aprendizado por reforço. Em seguida, definimos o PDM:

#### *Definição*

#### PROCESSO DE DECISÃO DE MARKOV, ou PDM

Um processo de decisão de Markov é uma tupla  $\langle S, A, P, R \rangle$ , onde:

$S$  é um conjunto de estados e

$A$  é um conjunto de ações.

$p_a(s_t, s_{t+1}) = p(s_{t+1} | s_t, a_t = a)$  é a probabilidade de que, se o agente executa a ação  $a \in A$  a partir do estado  $s_t$  no tempo  $t$ , isso resulta no estado  $s_{t+1}$  no tempo  $t+1$ . Como a probabilidade,  $p_a \in P$  é definida no espaço de estados inteiro das ações, ela geralmente é representada com uma matriz de transição.

$R(s)$  é a recompensa recebida pelo agente quando no estado  $s$ .

Devemos observar que a única diferença entre a estrutura do PDM para o aprendizado por reforço e a apresentação do aprendizado por reforço da Seção 10.7 é que a função de transição agora é substituída por uma função de distribuição de probabilidade. Essa é uma modificação importante em nossa teoria, que nos permite capturar a incerteza no mundo, quando o mundo é computacionalmente complexo ou é de fato estocástico. Nesse casos, a resposta da recompensa a uma ação do agente é mais bem representada como uma distribuição de probabilidade.

### 13.3.2 Processo de decisão de Markov parcialmente observável (PDMPO)

Observamos que os PDMs podem ser usados para modelar um jogo como o xadrez, em que o próximo estado do mundo pode estar fora do controle determinístico de um agente. Quando fazemos um movimento no xadrez, o estado resultante do jogo depende do movimento do nosso oponente. Assim, estamos cientes do estado atual em que estamos, estamos cientes de qual ação (movimento) estamos tomando, mas o estado resultante não nos é imediatamente acessível. Temos apenas uma distribuição de probabilidade no conjunto de estados possíveis em que provavelmente estaremos, dependendo do movimento do nosso oponente.

Agora, consideremos o jogo de pôquer. Aqui, nem ao menos temos certeza de nosso estado atual! Sabemos as cartas que temos, mas não temos perfeito conhecimento das cartas do nosso oponente. Só podemos supor, talvez com conhecimento de apostas, quais cartas nosso oponente tem. O mundo é ainda mais incerto que o mundo dos PDMs. Não apenas não sabemos em que estado estamos, mas também devemos executar uma ação com base na nossa suposição do estado real do mundo. Como resultado, nossa suposição do novo estado do mundo induzido por ação será ainda mais incerta. Logo, precisamos de uma estrutura mais rica do que o PDM para modelar essa situação duplamente incerta. Uma estrutura que realiza essa tarefa é o *processo de decisão de Markov parcialmente observável* (ou PDMPO), que é chamado assim porque só podemos observar parcialmente o estado em que estamos, bem como a resposta do nosso oponente. Em vez de um conhecimento perfeito do nosso estado atual, só temos uma distribuição de probabilidade sobre o conjunto possível de estados em que poderíamos estar. Em seguida, definimos o PDMPO:

#### Definição

#### PROCESSO DE DECISÃO DE MARKOV PARCIALMENTE OBSERVÁVEL, ou PDMPO

Um processo de decisão de Markov parcialmente observável é uma tupla  $\langle S, A, O, P, R \rangle$ , onde:

$S$  é um conjunto de estados e

$A$  é um conjunto de ações.

$O$  é o conjunto de observações indicando o que o agente pode ver em seu mundo. Como o agente não pode observar diretamente seu estado atual, as observações são probabilisticamente relacionadas ao estado real do mundo.

$P_a(s_t, o, s_{t+1}) = p(s_{t+1} | s_t, o_t = o, a_t = a)$  é a probabilidade de que, quando o agente executar a ação  $a$  do estado  $s_t$  no tempo  $t$ , isso resultará em uma observação  $o$  que levará a um estado subjacente  $s_{t+1}$  no tempo  $t+1$ .

$R(s_t, a, s_{t+1})$  é a recompensa recebida pelo agente quando ele executa a ação  $a$  no estado  $s_t$  e passa para o estado  $s_{t+1}$ .

Em resumo, o PDM pode ser considerado como um caso especial do PDMPO quando a observação o nos dá uma indicação precisa do estado atual  $s$ . Uma analogia pode ser útil na compreensão dessa distinção: o PDM está relacionado à cadeia de Markov, assim como o PDMPO está relacionado ao modelo oculto de Markov.

Existem diversos algoritmos para a criação de PDMs e PDMPOs. A complexidade desses algoritmos é obviamente maior do que aquela encontrada nos casos puramente determinísticos apresentados anteriormente. De fato,

os algoritmos para solucionar PDMPOs são computacionalmente indecidíveis, ou seja, pode ser impossível achar uma solução ideal para um problema usando um PDMPO em tempo polinomial, e uma solução pode realmente não ser computável. Como resultado, tratamos desses problemas com algoritmos que aproximam uma solução ideal. As referências para algoritmos de solução para PDMs e PDMPOs incluem Sutton e Barto (1998) e Puterman (1998). Em seguida, apresentamos uma implementação de exemplo de um PDM.

### 13.3.3 Exemplo de implementação do processo de decisão de Markov

Para ilustrar um PDM simples, usamos novamente o exemplo do robô reciclagem (adaptado de Sutton e Barto, 1998) introduzido no início da Seção 13.3. Lembre-se de que, toda vez que esse agente de aprendizado por reforço encontra um evento externo, ele toma uma decisão de buscar ativamente uma lata de refrigerante para reciclagem (chamamos isso de busca), esperar que alguém lhe traga uma lata (chamamos isso de espera) ou retornar à estação base para recarregar sua bateria (recarga). Essas decisões são tomadas pelo robô com base apenas no estado do nível de energia da bateria. Para simplificar, dois níveis de bateria são distinguidos, baixo e alto: o espaço de estados de  $S = \{\text{baixo}, \text{alto}\}$ . Se o estado atual da bateria for alto, a bateria está carregada; caso contrário, seu estado é baixo. Portanto, os conjuntos de ações do agente são  $A(\text{baixo}) = \{\text{busca, espera, recarga}\}$  e  $A(\text{alto}) = \{\text{busca, espera}\}$ . Se  $A(\text{alto})$  incluisse recarga, esperaríamos que o agente aprendesse que uma política usando essa ação seria subótima!

O estado da bateria é determinado independentemente das ações tomadas. Se ele for alto, então esperamos que o robô seja capaz de concluir um período de busca ativa sem o risco de esgotar sua bateria. Assim, se o estado for alto, então o agente permanecerá nesse estado com probabilidade  $a$  e mudará seu estado para baixo com probabilidade  $1 - a$ . Se o agente decide realizar uma busca ativa enquanto está no estado baixo, o nível de energia da bateria permanece igual com probabilidade  $b$ , e a bateria esgota sua carga de energia com probabilidade  $1 - b$ .

Em seguida, criamos um sistema de recompensa: quando a bateria se acaba,  $S = \text{baixo}$ , o robô é resgatado, sua bateria é recarregada para alto e ele recebe uma recompensa  $-3$ . Toda vez que o robô acha uma lata, ele recebe uma recompensa de  $1$ . Indicamos a quantidade esperada de latas que o robô coletará (a recompensa esperada que o agente receberá) enquanto busca ativamente e enquanto espera com  $R_{\text{busca}}$  e  $R_{\text{espera}}$ , e supomos que  $R_{\text{busca}} > R_{\text{espera}}$ . Também supomos que o robô não pode coletar quaisquer latas enquanto retorna para recarregar e nem quando o nível da bateria está baixo. O sistema que descrevemos pode ser representado por um PDM finito cujas probabilidades de transição ( $p_a(s_t, s_{t+1})$ ) e recompensas esperadas são dadas na Tabela 13.1.

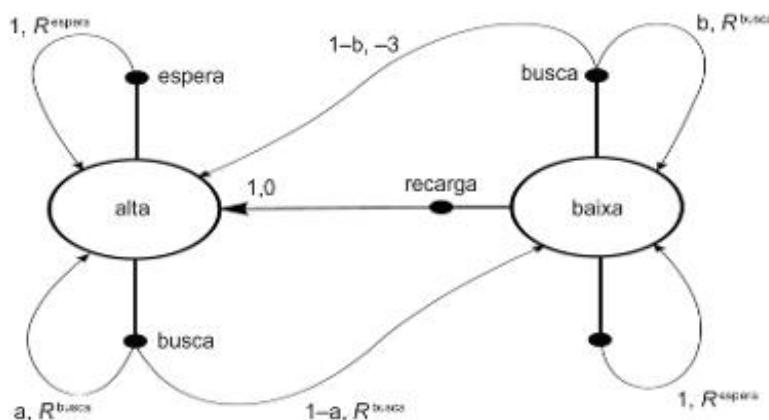
**Tabela 13.1** Probabilidades de transição e recompensas esperadas para um PDM finito do exemplo de robô reciclagem. A tabela contém todas as combinações possíveis do estado atual,  $s_t$ , o próximo estado,  $s_{t+1}$ , as ações e recompensas possíveis a partir da ação  $a_t$ .

$s_t$	$s_{t+1}$	$a_t$	$p_a(s_t, s_{t+1})$	$R_a(s_t, s_{t+1})$
alto	alto	busca	$a$	$R_{\text{busca}}$
alto	baixo	busca	$1 - a$	$R_{\text{busca}}$
baixo	alto	busca	$1 - b$	$-3$
baixo	baixo	busca	$b$	$R_{\text{busca}}$
alto	alto	espera	$1$	$R_{\text{espera}}$
alto	baixo	espera	$0$	$R_{\text{espera}}$
baixo	alto	espera	$0$	$R_{\text{espera}}$
baixo	baixo	espera	$1$	$R_{\text{espera}}$
baixo	alto	recarga	$1$	$0$
baixo	baixo	recarga	$0$	$0$

Para representar a dinâmica desse PDM finito, podemos usar um grafo, como na Figura 13.13, o de transição do PDM para o robô reciclagem. Um grafo de transição tem dois tipos de nós: os nós de estado e os nós de ação. Cada estado do agente é representado por um nó de estado representado como um grande círculo com o nome do estado,  $s$ , dentro dele. Cada par estado-ação é representado por um nó de ação conectado ao nó de estado indicado por um pequeno círculo preto. Se o agente começa no estado  $s_t$  e toma a ação  $a$ , ele se move pela linha do nó de estado  $s_t$  para o nó de ação ( $s_t, a$ ). Consequentemente, o ambiente responde com uma transição para o nó do próximo estado por meio de uma das setas saindo do nó de ação ( $s_t, a$ ), onde cada seta corresponde a uma tripla ( $s_t, a, s_{t+1}$ ), onde  $s_{t+1}$  é o estado seguinte. A seta é rotulada com a probabilidade da transição,  $p_a(s_t, s_{t+1})$ , e a recompensa esperada para a transição,  $R_a(s_t, s_{t+1})$ , é representada pela seta. Note que as probabilidades de transição associadas às setas saindo de um nó de ação sempre somam 1.

Veremos várias das técnicas de aprendizado estocástico descritas neste capítulo novamente no Capítulo 15, “Compreensão da linguagem natural”.

**Figura 13.13** Grafo de transição para o robô reciclagem. Os nós de estado são os círculos grandes, e os nós de ação são os pequenos estados pretos.



## 13.4 Epílogo e referências

Este capítulo apresentou os algoritmos de aprendizado de máquina dos pontos de vista dinâmico e estocástico. O âmbito do aprendizado de máquina probabilístico são os algoritmos de Bayes e de Markov. Essas duas ferramentas foram simplificadas (a rede Bayesiana de crença e as cadeias de primeira ordem Markov) para tornar as técnicas estocásticas de aprendizado poderosas, porém fáceis de manejar. O teorema de Bayes foi apresentado inicialmente na Seção 5.3, e a rede Bayesiana de crença, na Seção 9.3. As cadeias de Markov, bem como a suposição de primeira ordem de Markov, foram apresentadas na Seção 9.3. Seus usos e extensões compõem o material do Capítulo 13.

Existem diversos textos introdutórios e tutoriais em técnicas estocásticas para modelagem e aprendizado. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, de J. Pearl (1988), é uma antiga apresentação inicial que focalizou claramente o campo de modelos gráficos. Também recomendamos *Bayesian Networks and Decision Graphs*, de F. V. Jensen (1996), *Probabilistic Networks and Expert Systems*, de R. G. Cowell et al. (1999), *Graphical Models*, de S. Lauritzen (1996), *An Introduction to Bayesian Networks*, de F. Jensen (2001), e *Learning in Graphical Models*, de M. I. Jordan (1998).

Vários outros autores apresentam o aprendizado probabilístico de um ponto de vista mais estatístico, incluindo J. Whittaker (1990), *Graphical Models in Applied Multivariate Statistics*, e D. Edwards (2000), *Introduction to*

*Graphical Modeling.* Um ponto de vista da engenharia/comunicação é usado por B. Frey (1998), *Graphical Models for Machine Learning and Digital Communication*.

*Causality*, de J. Pearl (2000), é uma contribuição importante à filosofia que dá suporte aos modelos gráficos. Além de tentar esclarecer a noção da própria causalidade, ele delinea sua importância e utilidade dentro do projeto dos modelos probabilísticos.

As introduções tutoriais aos modelos gráficos incluem *Bayesian Networks without Tears*, de E. Charniak (1991), e *Belief Networks, Hidden Markov Models, and Markov Random Fields: A Unifying View*, de P. Smyth (1997). Excelentes introduções on-line ao campo incluem *A Brief Introduction to Graphical Models and Bayesian Networks*, de K. Murphy [[www.cs.ubc.ca/~murphyk/Bayes/bnintro.html#appl](http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html#appl)], além de *A Brief Introduction to Bayes' Rule*, de Murphy em [[www.cs.ubc.ca/~murphyk/Bayes/bayesrule.html](http://www.cs.ubc.ca/~murphyk/Bayes/bayesrule.html)]. Uma listagem de tutoriais mais completa pode ser encontrada em [[www.autonlab.org/tutorials/bayesnet.html](http://www.autonlab.org/tutorials/bayesnet.html)].

O algoritmo ME foi explicado e recebeu seu nome em um artigo clássico de 1977, de A. Dempster, N. Laird e D. Rubin. Eles apontaram que o método foi “proposto muitas vezes em circunstâncias especiais” por outros autores, mas seu artigo de 1977 generalizou o método e desenvolveu a teoria por trás dele.

A inferência em redes de Markov e Bayesiana dinâmica vem em duas formas, exata, onde os resultados podem ser computacionalmente dispendiosos, mas são demonstravelmente corretos, e aproximada, onde seu custo é menor, porém os resultados são aproximados e podem ser máximos locais. Para um raciocínio exato, consulte *The Generalized Distributive Law*, de S. M Aji e R. McEliece (2000), e *Factor Graphs and the Sum Product Algorithm*, de F. Kschischang et al. (2000). Para obter algoritmos aproximados, consulte *An Introduction to Variational Methods for Graphical Models*, de M. I. Jordan et al. (1999), *Variational Probabilistic Inference and the QMR-DT Database*, de T. Jaakkola e M. I. Jordan (1998), e *Generalized Belief Propagation*, de J. Yedidia et al. (2000). A propagação de crença ciclica, um algoritmo de aproximação para modelos gráficos, é descrita em *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, de J. Pearl (1988).

Em anos recentes, a criação de diversas formas de linguagens de raciocínio estocásticas de primeira ordem tem oferecido uma importante área de pesquisa. Sugeremos *Probabilistic Frame-Based Systems*, de D. Koller e A. Pfeffer (1998), *Learning Probabilistic Relational Models*, de N. Friedman et al. (1999), *Bayesian Logic Programs* de K. Kersting e L. DeRaedt (2000), *Probabilistic Logic Programming*, de R. Ng e V. Subrahmanian (1992), e *The Design and Testing of a First-Order Stochastic Modeling Language*, de Pless et al. (2006).

Contribuições importantes para aprendizado por reforço incluem *Learning to Predict by the method of Temporal Differences*, de R. S. Sutton (1988), *Reinforcement Learning: An Introduction*, de R. S. Sutton e A. G. Barto (1998), e *Markov Decision Processes*, de M. L. Puterman (1994).

O autor deseja agradecer a diversos alunos de graduação atuais e recentes, especialmente Dan Pless, o criador da lógica cíclica de linguagem de inferência estocástica de primeira ordem (Pless et al., 2006), Chayan Chakrabarti (Chakrabarti et al., 2006), Roshan Rammohan e Nikita Sakhanenko (Sakhanenko et al., 2007) por muitas das ideias, figuras, exemplos e exercícios para este capítulo.

## 13.5 Exercícios

---

1. Crie um modelo oculto de Markov para representar a sequência de pontuação de um jogo de futebol americano onde os *touchdowns* valem 6 pontos, que podem ser seguidos por uma tentativa extra de 0, 1 ou 2 pontos. Naturalmente, existem dois times e qualquer um pode pontuar. Se um time tem a bola e não pontua, eles emitem um 0. Assim, supondo que o fluxo de emissão de escores é 6, 1, 6, 0, 3, 0, 6, 1, 0, 3, 0, como seria o seu MOM?
  - a. Discuta esse problema e qual seria o melhor MOM.
  - b. Teste o seu MOM em mais dois fluxos de escores inventados.
  - c. Acompanhe um jogo real e veja como o seu sistema pode prever o fluxo de pontuação.

2. Crie um modelo oculto de Markov para prever a sequência de pontuação de *half inning* (meio tempo de ataque) de um jogo de beisebol americano. Suponha que essa sequência seja 0, 0, 0, 1, 0, 1, 1, 2, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0. Para simplificar, restrinja a pontuação de cada equipe a 0, 1 ou 2 corridas durante seu *half inning* no batedor.
  - a. Como você mudaria o modelo para permitir qualquer número de corridas por *half inning*?
  - b. Como você treinaria seu sistema para obter valores de pontuação mais realistas?
3. Crie uma figura para representar o modelo oculto hierárquico de Markov da Seção 13.1.2. Para que tipo de situação de problema seu MOHM poderia ser apropriado? Discuta a questão de ajuste das estruturas dos MOMs aos domínios de aplicação.
4. Dado o exemplo do algoritmo de Viterbi processando a máquina de estados finitos probabilística das figuras 13.7 e 13.8:
  - a. Por que *new* é visto como uma interpretação melhor que *knee* para os fonemas observados?
  - b. Como os estados alternativos na máquina de estados finitos probabilística são tratados pelo algoritmo de Viterbi, por exemplo, a escolha dos fonemas *uw* e *iy* na palavra *new*?
5. Dado o modelo oculto de Markov e o algoritmo de Viterbi da Seção 9.3.6, realize um rastreamento completo, incluindo a inicialização apropriada dos ponteiros, que mostre como a observação #, n, iy, t, # seria processada.
6. Manuseie o robô descrito no exemplo de processo de decisão de Markov da Seção 13.3.3. Use o mesmo mecanismo de recompensa e selecione valores probabilísticos para a e b para o processamento de decisão.
  - a. Use o robô novamente com diferentes valores para a e b. Quais políticas dão ao robô as melhores chances de recompensa?
7. Programe o robô com suporte do PDM da Seção 13.3.3 na linguagem à sua escolha. Experimente diferentes valores de a e b que possam otimizar a recompensa. Existem várias políticas interessantes possíveis: se recarga fosse uma política de A(alto), seu robô aprenderia que essa política está abaixo da ideal? Sob quais circunstâncias o robô sempre procuraria latas vazias, ou seja, a política para A(baixo) = recarga é abaixo da ideal?
8. Jack tem uma revenda de automóveis e está buscando um modo de maximizar seus lucros. Toda semana, ele pede um estoque de carros, ao custo de  $d$  dólares por carro. Esses carros são entregues instantaneamente. Os novos carros são acrescentados ao seu estoque. Então, durante a semana, ele vende um número aleatório de carros,  $k$ , a um preço de  $c$  cada. Jack também possui um custo  $u$  para cada carro não vendido que ele precisa manter no estoque. Formule esse problema como um processo de decisão de Markov. Quais são os estados e as ações? Quais são as recompensas? Quais são as probabilidades de transição? Descreva o retorno a longo prazo.
9. Considere o domínio geral das tarefas de navegação do mundo da grade, onde há um estado alvo, obstáculos e um fator de desconto  $\gamma < 1$ . As ações são estocásticas, de modo que o agente pode deslizar para uma célula diferente ao tentar se mover. Há cinco ações possíveis: ir para norte, sul, leste, oeste ou ficar o mesmo local. Considere a situação em que os custos negativos são recebidos quando se bate nas paredes. Você pode desenhar um ambiente de exemplo  $3 \times 3$  em que a melhor ação em pelo menos um estado é ficar? Nesse caso, especifique as ações, recompensas e probabilidades de transição. Se não puder, explique por quê.
10. Quando saímos para jantar, sempre gostamos de estacionar o mais perto possível do restaurante. Suponha que o restaurante esteja situado em uma rua muito longa, que corre de leste a oeste, e que permita estacionamento em apenas um lado. A rua é dividida em seções com tamanho para um carro. Chegamos ao restaurante pelo leste, começando a  $D$  unidades de distância. A probabilidade de uma vaga na distância  $s$  do restaurante não estar ocupada é  $p_s$ , independentemente de todas as outras vagas. Formule esse problema como um processo de decisão de Markov. Especifique todos os elementos do seu PDM! (Adaptado de Puterman, 1994.)
11. Como você mudaria a representação do PDM da Seção 13.3 para um PDMPO? Use o problema do robô simples e sua matriz de transição de Markov criada na Seção 13.3.3 e mude-o para um PDMPO. Dica: pense em usar uma matriz de probabilidade para os estados parcialmente observáveis.
12. Discutimos o jogo de cartas de pôquer rapidamente na Seção 13.3.2. Dado que o estado atual (probabilístico) de um jogador é aposta boa, aposta questionável ou aposta ruim, desenvolva um PDMPO para representar essa situação. Você poderia discutir isso usando as probabilidades que certas mãos possíveis do pôquer podem ser distribuídas.
13. Desenvolva o custo de complexidade para achar uma política ideal para o problema do PDMPO exaustivamente.



# Tópicos avançados para a solução de problemas por IA

*Precisão não é o mesmo que verdade...*

—HENRI MATISSE

*Tempo presente e tempo passado serão todos, quem sabe, presentes no tempo futuro e o tempo futuro está contido no tempo passado...*

—T. S. ELIOT, “Burnt Norton”

*... cada padrão maior acaba sendo o produto final de uma longa sequência de pequenos atos.*

— CHRISTOPHER ALEXANDER

## Raciocínio automatizado e linguagem natural

A Parte V examina duas aplicações importantes da inteligência artificial: a compreensão de linguagem natural e o raciocínio automatizado. Se algum dia reivindicarmos a criação de uma inteligência artificial, deveremos abordar linguagem, raciocínio e aprendizado. As nossas soluções para esses problemas se baseiam nas ferramentas e nas técnicas introduzidas nas seções anteriores deste livro. Por causa da sua importância, esses problemas exercearam uma influência profunda no desenvolvimento dessas ferramentas e no direcionamento geral da própria IA.

Na introdução da Parte III, discutimos as vantagens e desvantagens dos resolvidores de problemas por método fraco. Entre os problemas do uso de métodos fracos se incluem a complexidade dos espaços de busca e as dificuldades de se representar conhecimento específico do mundo com representações genéricas. Apesar do sucesso dos sistemas especialistas e outros sistemas similares por métodos fortes, muitos domínios requerem métodos gerais; na verdade, as estratégias de controle dos próprios sistemas especialistas dependem de bons métodos fracos para resolver problemas. Muitos trabalhos promissores em resolvidores de problemas por método fraco continuam sendo feitos pela comunidade de *raciocínio automatizado* e *provadores de teoremas*. Essas técnicas encontraram aplicação em várias áreas importantes, incluindo projeto e verificação de circuitos integrados, provas da correção de programas e, indiretamente, criação da linguagem de programação Prolog. No Capítulo 14, abordamos questões sobre o raciocínio automatizado.

Há diversas razões para o fato de a *compreensão de linguagem natural* ter se mostrado uma tarefa difícil para a comunidade de IA. Entre as razões mais importantes estão a grande quantidade de conhecimento, de habilida-

des e de experiência necessárias para dar suporte ao uso da linguagem. A compreensão bem-sucedida da linguagem requer um entendimento do mundo natural, da psicologia humana e de convenções sociais. Ela se vale de habilidades tão variadas quanto o raciocínio lógico e a interpretação de metáforas. Por causa da complexidade e da ambiguidade da linguagem humana, a compreensão de linguagem natural tem motivado, em grande parte, a pesquisa em representação de conhecimento. Até agora, esses esforços têm sido apenas parcialmente bem-sucedidos: usando uma abordagem baseada em conhecimento, os pesquisadores desenvolveram com sucesso programas que comprehendem linguagem natural em domínios específicos. Se essas técnicas eventualmente solucionarão o próprio problema da compreensão de linguagem natural continua sendo assunto de debates.

As técnicas de representação apresentadas no Capítulo 7, como as redes semânticas, roteiros e quadros, dominaram os trabalhos iniciais da IA em linguagem natural. Mais recentemente, a análise de correlação de padrões de linguagem tem exercido papel importante na compreensão da linguagem. As articulações da linguagem não são criações aleatórias de sons ou palavras, mas ocorrem em padrões. Técnicas Bayesianas podem modelar essas construções de linguagem. O Capítulo 13, sobre aprendizado de máquina probabilístico, oferece diversos modelos, incluindo bigramas e trigramas, importantes para a compreensão de combinações de fonemas e palavras da linguagem. No Capítulo 15, examinamos técnicas sintáticas, semânticas e estocásticas, baseadas no conhecimento, para a compreensão de linguagem natural.

Com o advento da *World Wide Web* e o uso extensivo de máquinas de busca pela população em geral, as questões sobre o processamento da linguagem natural se tornaram ainda mais importantes. A *mineração de textos*, ou a busca geral de informação útil em textos não estruturados, e a *sumarização de informações*, ou a capacidade de “compreender” textos e extrair as suas questões críticas, são tecnologias novas e importantes. Os fundamentos dessas ferramentas de software, o reconhecimento de padrões simbólico e estocástico, são apresentados no Capítulo 15.

Finalmente, na Sala Virtual deste livro, construímos muitas estruturas de dados que dão suporte à compreensão de linguagem natural, incluindo sistemas de redes semânticas e quadros, em Prolog, Lisp e Java. Também criamos um analisador sintético de rede semântica descendente recursivo, bem como um analisador sintético baseado no algoritmo de Earley e na programação dinâmica em Prolog e Java. No epílogo e no Capítulo 16, discutimos algumas limitações atuais na compreensão da linguagem natural, no aprendizado e na solução de problemas complexos.

# Raciocínio automatizado

*Como é possível, diz o homem arguto, que, quando me dão um conceito, consigo ir além dele e conectá-lo a outro que não está contido nele, como se este último necessariamente pertencesse ao primeiro?*

—IMMANUEL KANT, “Prologomena to a Future Metaphysics”

*Qualquer decisão racional pode ser vista como uma conclusão alcançada, a partir de certas premissas... Portanto, o comportamento de uma pessoa racional pode ser controlado se o valor e as premissas factuais, nas quais ele baseia suas decisões, forem a ela especificados.*

—SIMON, *Decision-Making and Administrative Organization*, 1944

*Raciocinar é uma arte, e não uma ciência...*

—WOS ET AL., *Automated Reasoning*, 1984

## 14.0 Introdução aos métodos fracos para a prova de teoremas

Wos et al. (1984) descrevem um programa de *raciocínio automatizado* como aquele que “emprega uma notação não ambígua e exata para representar a informação, regras de inferência precisas para tirar conclusões e estratégias cuidadosamente delineadas para controlar essas regras de inferência”. Eles acrescentam que aplicar estratégias para que as regras de inferência deduzam informações novas é uma arte: “Uma boa escolha para a representação inclui uma notação que aumente a chance de solução de um problema e inclua informação, que, embora não seja necessária, seja útil. Uma boa escolha de regras de inferência é aquela que se entrosa bem com a representação escolhida. Uma boa escolha de estratégias é aquela que controla as regras de inferência, de modo a aumentar sensivelmente a eficácia do programa de raciocínio”.

O raciocínio automatizado, como descrito anteriormente, utiliza métodos fracos de solução de problemas. Ele usa uma representação uniforme, como o cálculo de predicados de primeira ordem (Capítulo 2), o cálculo de cláusulas de Horn (Seção 14.3) ou a forma de cláusulas usada para resolução (Seção 14.2). As suas regras de inferência são consistentes e, sempre que possível, completas. Ele usa estratégias gerais, como busca em amplitude, em profundidade ou pela melhor escolha, e, como veremos neste capítulo, heurísticas, como *conjunto de suporte e preferência por unidade*, para combater a explosão combinatória da busca exaustiva. O projeto de estratégias de busca e especialmente estratégias de busca heurísticas é como uma arte; não podemos garantir que elas encontrarão uma solução útil para um problema usando quantidades razoáveis de tempo e memória.

A solução de problemas por método fraco é uma ferramenta importante por si só, mas também é uma base essencial para a solução de problemas por método forte. Os sistemas de produção e os ambientes (*shells*) de sistemas especialistas são dois exemplos de resolvedores de problemas por método fraco. Muito embora as regras do sistema de produção ou do sistema especialista baseados em regras codifiquem heurísticas fortes de solução de problema, a sua aplicação é baseada em estratégias de inferência gerais (método fraco).

As técnicas para a solução de problemas por método fraco têm sido o foco de pesquisas em IA desde o início. Frequentemente, essas técnicas aparecem sob o título de *prova de teoremas*, embora prefiramos o título mais genérico de *raciocínio automatizado*. Começamos este capítulo (Seção 14.1) com um exemplo inicial de raciocínio automatizado, o *Resolvedor Geral de Problemas* e seu uso de *análise de meios e fins* e *tabelas de diferenças* para controlar a busca.

Na Seção 14.2, apresentamos um importante produto da pesquisa em raciocínio automatizado, o *sistema de resolução por refutação*. Discutimos a linguagem de representação, a regra de inferência da resolução, as estratégias de busca e os processos de extração de respostas usados na prova de teoremas por resolução. Como exemplo de raciocínio por cláusulas de Horn, na Seção 14.3, descrevemos o motor de inferência para o Prolog e mostramos como essa linguagem contribui para uma filosofia de programação declarativa com um interpretador baseado em um provador de teoremas por resolução. Concluímos este capítulo (Seção 14.4) com alguns breves comentários sobre *dedução natural*, tratamento de igualdade e regras de inferência mais sofisticadas.

## 14.1 Resolvedor Geral de Problemas e as tabelas de diferenças

O *Resolvedor Geral de Problemas* (GPS, do inglês *General Problem Solver*) (Newell e Simon, 1963b; Ernst e Newell, 1969) resultou das pesquisas de Allen Newell e Herbert Simon na Carnegie Mellon University, então chamada Carnegie Institute of Technology. As suas raízes estão em um programa de computador anterior denominado *Logic Theorist* (LT) de Newell, Shaw e Simon (Newell e Simon, 1963a). O programa LT provou vários dos teoremas apresentados no *Principia Mathematica*, de Russell e Whitehead (Whitehead e Russell, 1950).

Como ocorre com todos os resolvedores de problemas de método fraco, o *Logic Theorist* empregava um meio de representação uniforme e regras de inferência consistentes e adotava várias estratégias, ou métodos heurísticos, para guiar o processo de solução. Ele usava cálculo proposicional (Seção 2.1) como seu meio de representação. As regras de inferência eram *substituição, troca e separação*.

A substituição permite que qualquer expressão seja substituída para todas as ocorrências de um símbolo em uma proposição que seja um axioma ou um teorema já sabidamente verdadeiro. Por exemplo,  $(B \vee B) \rightarrow B$  pode ter a expressão  $\neg A$  substituída por  $B$  para produzir  $(\neg A \vee \neg A) \rightarrow \neg A$ .

A troca permite que um conectivo seja trocado por sua definição ou por uma forma equivalente. Por exemplo, a equivalência lógica de  $\neg A \vee B$  e  $A \rightarrow B$  pode levar à troca de  $(\neg A \vee \neg A)$  por  $(A \rightarrow \neg A)$ .

A separação é a regra de inferência que denominamos *modus ponens* (Capítulo 2).

O LT aplica essas regras de inferência de uma maneira guiada por objetivo e em amplitude ao teorema a ser provado, tentando encontrar uma série de operações que levem a axiomas ou teoremas sabidamente verdadeiros. A estratégia do LT consiste em quatro métodos organizados em uma *rotina executiva*:

Primeiro, aplicar o método de substituição diretamente ao objetivo atual, tentando casá-lo com todos os axiomas e teoremas conhecidos.

Segundo, se isso não levar à prova, aplicar todas as separações e trocas ao objetivo e testar cada um desses resultados usando substituição. Se a substituição não casar qualquer um deles com o objetivo, então eles serão adicionados a uma *lista de subproblemas*.

Terceiro, usar o método de encadeamento, empregando a transitividade da implicação, para encontrar um novo subproblema que, se resolvido, forneceria a prova. Assim, se  $a \rightarrow c$  é o problema e  $b \rightarrow c$  for encontrado, então  $a \rightarrow b$  é colocado como um novo subproblema.

Quarto, se os primeiros três métodos falharem no problema original, vá à lista de subproblemas e selecione o próximo subproblema não investigado.

A rotina executiva continua a aplicar esses quatro métodos até que uma solução seja encontrada, ou até que não haja mais problemas na lista de subproblemas, ou, ainda, se a memória e o tempo alocados para encontrar a prova estiverem esgotados. Dessa forma, o *Logic Theorist* executa uma busca guiada por objetivos e em amplitude do espaço do problema.

A parte da rotina executiva que habilita as regras de inferência de substituição, troca e separação é denominada *processo de casamento*. Suponha que desejemos provar  $p \rightarrow (q \rightarrow p)$ . O processo de casamento identifica primeiro um dos axiomas,  $p \rightarrow (q \vee p)$ , como sendo mais apropriado que os outros — isto é, ele tem um casamento mais próximo em termos de uma diferença definida para o domínio — porque o conectivo principal, aqui  $\rightarrow$ , é o mesmo em ambas as expressões. Segundo, o processo de casamento confirma que as expressões à esquerda do conectivo principal são idênticas. Finalmente, o casamento identifica a diferença entre as expressões à direita do conectivo principal. Essa diferença final, entre  $\rightarrow$  e  $\vee$ , sugere a troca óbvia para provar o teorema. O processo de casamento ajuda a controlar a busca (exhaustiva) que seria necessária para aplicar todas as substituições, trocas e separações. De fato, o casamento reduziu suficientemente o processo de tentativa e erro, tornando o LT um resolvidor de problemas bem-sucedido.

O exemplo de uma prova pelo LT demonstra o poder do processo de casamento. O Teorema 2.02 do *Principia Mathematica* é  $p \rightarrow (q \rightarrow p)$ . O casamento encontra o axioma  $p \rightarrow (q \vee p)$  conforme apropriado para troca. A substituição de  $\neg q$  por  $q$  prova o teorema. O casamento, controlando as regras de substituição e troca, provou esse teorema diretamente, sem qualquer busca por outros axiomas ou teoremas.

Em outro exemplo, suponha que desejemos que o LT prove:

$$(p \rightarrow \neg p) \rightarrow \neg p.$$

1. ( $A \vee A$ )  $\rightarrow A$  O casamento identifica o melhor axioma dos cinco disponíveis.
2. ( $\neg A \vee \neg A$ )  $\rightarrow \neg A$  Substituição de  $\neg A$  por  $A$  para aplicar
3. ( $A \rightarrow \neg A$ )  $\rightarrow \neg A$  troca de  $\rightarrow$  por  $\vee$  e  $\neg$ , seguido de
4. ( $p \rightarrow \neg p$ )  $\rightarrow \neg p$  substituição de  $p$  por  $A$ .

CQD

O LT original demonstrou esse teorema em cerca de 10 segundos usando cinco axiomas. A prova efetiva ocorreu em dois passos e sem busca. O casamento selecionou o axioma apropriado para o primeiro passo porque a sua forma era muito parecida com a conclusão que ele tentava estabelecer: (expressão)  $\rightarrow$  proposição. Então,  $\neg A$  foi substituído para  $A$ . Isso permitiu a troca do segundo e último passo, ele próprio motivado pelo objetivo requerer um  $\rightarrow$  em vez de um  $\vee$ .

O *Logic Theorist* não foi apenas o primeiro exemplo de um sistema de raciocínio automatizado, mas ele demonstrou, também, a importância de estratégias de busca e de heurísticas em um programa de raciocínio. Em muitas ocasiões, o LT encontrou, em poucos passos, soluções que a busca exaustiva jamais encontraria, mas não conseguiu demonstrar alguns teoremas, e Newell et al. sugeriram melhorias que poderiam tornar a sua solução possível.

Nessa mesma ocasião, alguns pesquisadores de Carnegie e Yale (Moore e Anderson, 1954) começaram a examinar protocolos verbais de raciocínio de pessoas que resolviam problemas lógicos. Embora o objetivo principal deles fosse identificar processos humanos que pudessem resolver essa classe de problemas, os pesquisadores começaram a comparar a solução de problemas por humanos com a de programas de computador como o Teórico Lógico. Isso se tornou o primeiro exemplo do que hoje é conhecido como *psicologia de processamento de informação*, em que uma explicação para o comportamento observado de um organismo é fornecida por um programa constituído de processos de informação primitivos que gera esse comportamento (Newell et al., 1958). Essa pesquisa foi também um dos primeiros trabalhos que fundamentaram a disciplina moderna *Ciência Cognitiva* (ver Seção 16.2; Luger, 1994).

Um exame minucioso desses primeiros protocolos mostrou as diversas formas pelas quais as soluções do LT diferiam daquelas de um ser humano. O comportamento humano mostrava fortes evidências de um mecanismo de casamento de padrões e redução de diferenças, denominado *análise de meios e fins*, na qual os métodos (os *meios*) de redução de diferenças eram fortemente ligados às diferenças específicas a serem reduzidas (os *fins*): os operadores para redução de diferenças eram indexados pelas diferenças que eles poderiam reduzir.

Em um exemplo muito simples, se a expressão inicial fosse  $p \rightarrow q$  e o objetivo fosse  $\neg p \vee q$ , as diferenças incluiriam o símbolo  $\rightarrow$  na expressão inicial e  $\vee$  no objetivo, bem como a diferença de  $p$  na primeira e  $\neg p$  no objetivo. A tabela de diferenças conteria as diferentes maneiras como um  $\rightarrow$  poderia ser substituído por um  $\vee$  e como  $\neg$  poderia ser removido. Essas transformações seriam tentadas, uma por vez, até que as diferenças fossem removidas e o teorema fosse provado.

Na maioria dos problemas interessantes, as diferenças entre a expressão inicial e o objetivo não podiam ser reduzidas diretamente. Nesse caso, um operador (da tabela) seria procurado para reduzir parcialmente essa diferença. O procedimento completo era aplicado recursivamente a esses resultados até que não existissem diferenças. Isso poderia exigir, também, a sequência de diferentes caminhos de busca, representados por diferentes aplicações de reduções.

A Figura 14.1(a), de Newell e Simon (1963b), apresenta doze regras de transformação, a coluna do meio, para a solução de problemas lógicos. A coluna da direita fornece diretrizes de quando as transformações são usadas.

A Figura 14.1(b) apresenta uma prova, de Newell e Simon (1963b), gerada por um ser humano. Antes da prova, as regras de transformação da Figura 14.1(a) são disponibilizadas para uma pessoa que, sem experiência em lógica formal, é solicitada a modificar a expressão  $(R \supset -P) \cdot (\neg R \supset Q)$  para  $\neg(\neg Q \cdot P)$ . Na notação do Capítulo 2,  $\sim$  é  $\neg$ ,  $\cdot$  é  $\wedge$ , e  $\supset$  é  $\rightarrow$ . O  $\rightarrow$  ou  $\leftrightarrow$  na Figura 14.1(a) indica uma troca válida. A coluna mais à direita da Figura 14.1(b) indica a regra da Figura 14.1(a) que é aplicada a cada passo da prova.

Newell e Simon (1963b) chamaram as estratégias de solução de problemas dos seres humanos de *redução de diferenças*, e o processo geral de usar transformações apropriadas para reduzir diferenças específicas do problema.

**Figura 14.1(a)** Regras de transformação para problemas lógicos, de Newell e Simon (1961).

R 1.	$A \cdot B \rightarrow B \cdot A$ $A \vee B \rightarrow B \vee A$	
R 2.	$A \supset B \rightarrow \neg B \supset \neg A$	
R 3.	$A \cdot A \leftrightarrow A$ $A \vee A \leftrightarrow A$	
R 4.	$A \cdot (B \cdot C) \leftrightarrow (A \cdot B) \cdot C$ $A \vee (B \vee C) \leftrightarrow (A \vee B) \vee C$	
R 5.	$A \vee B \leftrightarrow \neg(\neg A \cdot \neg B)$	
R 6.	$A \supset B \leftrightarrow \neg A \vee B$	
R 7.	$A \cdot (B \vee C) \leftrightarrow (A \cdot B) \vee (A \cdot C)$ $A \vee (B \cdot C) \leftrightarrow (A \vee B) \cdot (A \vee C)$	
R 8.	$A \cdot B \rightarrow A$ $A \cdot B \rightarrow B$	Se aplica apenas em expressões principais.
R 9.	$A \rightarrow A \vee X$	Se aplica apenas em expressões principais.
R 10.	$A \} \rightarrow A \cdot B$ $B \}$	A e B são duas expressões principais.
R 11.	$A \supset B \} \rightarrow B$	A e $A \supset B$ são duas expressões principais.
R 12.	$A \supset B \} \rightarrow A \supset C$ $B \supset C \}$	$A \supset B$ e $B \supset C$ são duas expressões principais.

**Figura 14.1(b)** Uma prova de um teorema no cálculo proposicional, de Newell e Simon (1961).

1.	$(R \supset \neg P) \cdot (\neg R \supset Q)$	$\neg(\neg Q \cdot P)$
2.	$(\neg R \vee \neg P) \cdot (R \vee Q)$	Regra 6 é aplicada à direita e à esquerda de 1.
3.	$(\neg R \vee \neg P) \cdot (\neg R \supset Q)$	Regra 6 é aplicada à esquerda de 1.
4.	$R \supset \neg P$	Regra 8 é aplicada a 1.
5.	$\neg R \vee \neg P$	Regra 6 é aplicada a 4.
6.	$\neg R \supset Q$	Regra 8 é aplicada a 1.
7.	$R \vee Q$	Regra 6 é aplicada a 6.
8.	$(\neg R \vee \neg P) \cdot (R \vee Q)$	Regra 10 é aplicada a 5 e a 7.
9.	$P \supset \neg R$	Regra 2 é aplicada a 4.
10.	$\neg Q \supset R$	Regra 2 é aplicada a 6.
11.	$P \supset Q$	Regra 12 é aplicada a 6 e a 9.
12.	$\neg P \vee Q$	Regra 6 é aplicada a 11.
13.	$\neg(P \cdot \neg Q)$	Regra 5 é aplicada a 12.
14.	$\neg(\neg Q \cdot P)$	Regra 1 é aplicada a 13. CQD.

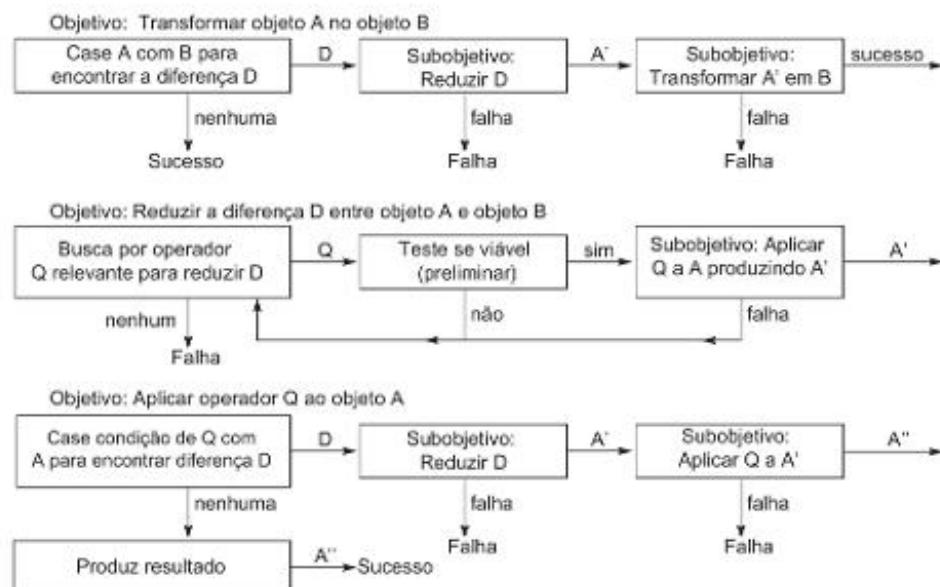
ma, de *análise de meios e fins*. O algoritmo que aplica a análise de meios e fins usando reduções de diferenças é o *Resolvedor Geral de Problemas* (GPS, do inglês *General Problem Solver*).

A Figura 14.2 apresenta o diagrama de controle e a tabela de conexões do GPS. O objetivo é transformar a expressão A na expressão B. O primeiro passo é localizar uma diferença D entre A e B. O subobjetivo, reduzir D, é identificado no segundo bloco da primeira linha; o terceiro bloco indica que a redução de diferenças é recursiva. A “redução” é a segunda linha, onde é identificado um operador Q para a diferença D. Na realidade, uma lista de operadores é identificada a partir da tabela de conexões. Essa lista fornece alternativas ordenadas para a redução de diferenças, caso o operador escolhido não seja aceitável, por exemplo, por não passar no *teste de viabilidade*. Na terceira linha da Figura 14.2, o operador é aplicado e D é reduzida.

O modelo GPS de solução de problemas requer dois componentes. O primeiro é o procedimento geral descrito anteriormente para comparar duas descrições de estado e reduzir suas diferenças. O segundo componente do GPS é a *tabela de conexões*, que fornece as ligações entre diferenças de problemas e as transformações específicas que as reduzem, apropriada para uma área de aplicação. A Figura 14.2 fornece diferenças e suas reduções [as doze transformações da Figura 14.1(a)] para expressões do cálculo proposicional. Poderia haver outras tabelas de conexões para reduzir diferenças em formas algébricas, ou para tarefas como a torre de Hanói, ou para jogos mais complexos como xadrez. Devido à modularidade da tabela de diferenças, ou seja, às tabelas serem mudadas para diferentes aplicações, o resolvedor de problemas era chamado *geral*. Várias áreas de aplicações diferentes da técnica GPS foram descritas por Ernst e Newell (1969).

A estruturação real das diferentes reduções de um domínio de problema ajuda a organizar a busca para aquele domínio. Uma heurística, ou ordem de prioridade para redução de diferentes classes de diferenças, está implícita na ordem das transformações dadas na tabela de redução. Essa ordem de prioridade coloca as transformações mais genericamente aplicáveis antes das especializadas, ou então determina a ordenação, qualquer que seja ela, que um especialista no domínio julgue mais apropriada.

**Figura 14.2** Fluxograma e tabela de redução de diferenças para o Resolvedor Geral de Problemas, de Newell e Simon (1963b).



Para a tarefa lógica do texto:

Teste de viabilidade (preliminar)

O conectivo principal é o mesmo (por exemplo,  $A \cdot B \rightarrow B$  falha para  $P \vee Q$ )?

O operador é muito grande (por exemplo,  $(A \vee B) \cdot (A \vee C) \rightarrow A \vee (B \cdot C)$  falha para  $P \cdot Q$ )?

O operador é muito fácil (por exemplo,  $A \rightarrow A$  se aplica a qualquer coisa)?

As condições secundárias são satisfatórias (por exemplo, R8 se aplica apenas a expressões principais)?

Tabela de conexões

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
Acrescentar termos		X					X	X	X	X	X	X
Retirar termos			X					X	X		X	X
Trocando conectivo				X	X	X						
Trocando sinal					X							
Trocando sinal inferior						X	X					
Trocando agrupamento							X					
Trocando posição	X	X										

X significa que uma variante da regra é relevante.

O GPS selecionará a variante apropriada.

Uma série de direções de pesquisa evoluiu do trabalho do resolvedor geral de problemas. Uma delas é o uso de técnicas de IA para analisar o comportamento humano na solução de problemas. Em particular, o sistema de produção substituiu os métodos de meios e fins do GPS como a forma preferida para modelar o processamento de informação humana (Capítulo 16). As regras de produção em sistemas especialistas baseados em regras modernos substituíram as posições específicas na tabela de diferenças do GPS (Capítulo 8).

Em outra evolução interessante do GPS, a própria tabela de diferenças evoluiu de outra maneira, tornando-se a *tabela de operadores para planejamento*, como em STRIPS e ABSTRIPS. O planejamento é importante na solução de problemas de robótica. Para realizar uma tarefa, como ir até a próxima sala e voltar trazendo um objeto, o computador precisa desenvolver um *plano*. Esse plano orquestra as ações do robô: descarregar o que ele estiver segurando, ir até a porta da sala em que ele se encontra, atravessar a porta, encontrar a sala requerida, atravessar a porta, ir até o objeto, e assim por diante. A formação de planos para o STRIPS, o Stanford Research Institute Problem Solver (Fikes e Nilsson, 1971; Fikes et al., 1972; Sacerdotti, 1974), utiliza uma tabela de operadores parecida com a tabela de diferenças do GPS. Cada operador (ação primitiva do robô) nessa tabela tem um conjunto associado de *pré-condições* que são parecidas com os testes de viabilidade da Figura 14.2. A tabela de operadores contém, também, *listas acrescenta* e *remove*, que atualizam o modelo do “mundo”

uma vez que o operador é aplicado. Apresentamos um planejador do tipo STRIPS na Seção 7.4 e o construímos em Prolog na Sala Virtual do livro.

Em resumo, os primeiros modelos de raciocínio automatizado em IA foram concebidos para o Teórico Lógico e para o Resolvedor Geral de Problemas desenvolvidos no Carnegie Institute. Esses programas já ofereciam todos os pré-requisitos para a solução de problemas por método fraco: um meio de representação uniforme, um conjunto de regras consistentes e um conjunto de métodos ou estratégias para aplicação dessas regras. Os mesmos componentes constituem os *procedimentos de prova por resolução*, uma base moderna e mais poderosa para raciocínio automatizado.

## 14.2 Prova de teoremas por resolução

### 14.2.1 Introdução

A resolução é uma técnica para provar teoremas no cálculo proposicional ou de predicados que tem sido parte das pesquisas em solução de problemas por IA desde meados de 1960 (Bledsoe, 1977; Robinson, 1965; Kowalski, 1979b). A resolução é uma regra de inferência consistente que, quando usada para produzir uma *refutação* (Seção 14.2.3), é também completa. Em uma aplicação prática importante, a prova de teoremas por resolução, em particular o sistema de resolução por refutação, tem tornado possível a geração atual dos interpretadores Prolog (Seção 14.3).

O princípio da resolução, introduzido em um importante artigo por Robinson (1965), descreve uma maneira de encontrar contradições em uma base de dados de cláusulas, com uso mínimo de substituições. A resolução por refutação prova um teorema negando a expressão a ser provada e acrescentando esse objetivo negado ao conjunto de axiomas que são sabidamente (que foram supostos como) verdadeiros. A regra de inferência da resolução é, então, usada para mostrar que isso leva a uma contradição. Já que o provador de teoremas mostra que o objetivo negado é inconsistente com o conjunto de axiomas dado, resulta que o objetivo original deve ser consistente. Isso prova o teorema.

Uma prova de resolução por refutação envolve os seguintes passos:

1. Colocar as premissas ou axiomas na *forma clausular* (14.2.2).
2. Acrescentar a negação do que deve ser provado, na forma clausular, ao conjunto de axiomas.
3. *Resolver* essas cláusulas conjuntamente, produzindo novas cláusulas que resultem logicamente delas (14.2.3).
4. Produzir uma contradição gerando a cláusula vazia.
5. As substituições usadas para produzir a cláusula vazia são aquelas sob as quais o oposto do objetivo negado (o que deveria ser provado originalmente) é verdadeiro (14.2.4).

A resolução é uma regra de inferência consistente no sentido do Capítulo 2. Entretanto, ela não é completa. A resolução é *completa em relação à refutação*; isto é, a cláusula vazia ou nula sempre pode ser gerada quando existir uma contradição no conjunto de cláusulas. Na Seção 14.2.4, será dito mais sobre esse tópico quando apresentarmos as estratégias para refutação.

As provas de resolução por refutação requerem que os axiomas e a negação do objetivo estejam expressos em uma forma normal denominada *forma clausular*, que representa a base de dados lógica como um conjunto de disjunções de *literais*. Um literal é uma expressão atômica ou a negação de uma expressão atômica.

A forma mais comum de resolução, denominada *resolução binária*, é aplicada a duas cláusulas quando uma contém um literal e a outra, a sua negação. Se esses literais contiverem variáveis, eles devem ser unificados para se tornarem equivalentes. Uma nova cláusula é produzida consistindo nos termos disjuntivos de todos os predicados nas duas cláusulas, menos o literal e a sua ocorrência negativa, que dizemos que foram “resolvidos”. A cláusula resultante recebe a substituição de unificação, sob a qual o predicho e a sua negação se tornaram “equivalentes”.

Antes de tornarmos isso mais preciso nas subseções subsequentes, apresentaremos um exemplo simples. A resolução produz uma prova similar a que já foi produzida com o *modus ponens*. Não queremos mostrar que essas regras de inferência são equivalentes (a resolução é, na verdade, mais geral que o *modus ponens*), mas sim dar ao leitor uma ideia do processo.

Desejamos provar que “Fido morrerá” a partir das declarações “Fido é um cão”, “todo cão é um animal” e “todo animal morrerá”. Escrevendo essas três premissas como predicados e aplicando o *modus ponens*, teremos:

1. Todo cão é um animal:  $\forall(X) (\text{cão}(X) \rightarrow \text{animal}(X))$ .
  2. Fido é um cão:  $\text{cão}(\text{fido})$ .
  3. *Modus ponens* e  $\{\text{fido}/X\}$  resulta:  $\text{animal}(\text{fido})$ .
  4. Todo animal morrerá:  $\forall(Y) (\text{animal}(Y) \rightarrow \text{morre}(Y))$ .
  5. *Modus ponens* e  $\{\text{fido}/Y\}$  resulta:  $\text{morre}(\text{fido})$ .

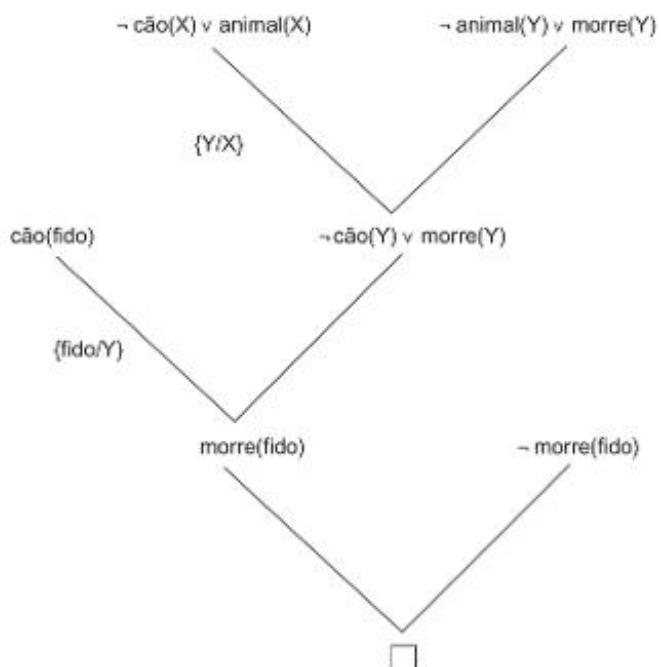
O raciocínio por resolução equivalente converte esses predicados em formas clausulares:

<b>Forma de predicado</b>	<b>Forma clausular</b>
1. $\forall(X) (\text{cão}(X) \rightarrow \text{animal}(X))$	$\neg \text{cão}(X) \vee \text{animal}(X)$
2. $\text{cão}(\text{fido})$	$\text{cão}(\text{fido})$
3. $\forall(Y) (\text{animal}(Y) \rightarrow \text{morre}(Y))$	$\neg \text{animal}(Y) \vee \text{morre}(Y)$
Negue a conclusão de que Fido morrerá:	
4. $\neg \text{morre}(\text{fido})$	$\neg \text{morre}(\text{fido})$

Resolva cláusulas que têm literais opostos, produzindo novas cláusulas por resolução como na Figura 14.3. Esse processo é normalmente denominado *colisão* (*clashing*).

O símbolo  $\square$  na Figura 14.3 indica que a cláusula vazia é produzida e a contradição foi encontrada. O símbolo  $\square$  representa a colisão de um predicado com a sua negação; a situação em que duas declarações mutuamente contradizem-se.

**Figura 14.3** Prova por resolução para o problema do "cão morto".



tórias estão presentes no espaço de cláusulas. Elas colidem para produzir a cláusula vazia. A sequência de substituições (unificações) usadas para tornar os predicados equivalentes também nos dá o valor das variáveis sob as quais um objetivo é verdadeiro. Por exemplo, se perguntassemos se algo morreria, o nosso objetivo negado seria  $\neg(\exists Z) \text{morre}(Z)$ , em vez de  $\neg\text{morre}(\text{fido})$ . A substituição {fido/Z} na Figura 14.3 determinaria que fido é uma ocorrência de um animal que morrerá. As questões implícitas nesse exemplo são esclarecidas no restante da Seção 14.2.

### 14.2.2 Produção da forma clausular para resolução por refutações

O procedimento de prova por resolução requer que todas as declarações na base de dados que descrevam uma situação sejam convertidas em uma forma padrão denominada forma *clausular*. Isso é motivado pelo fato de que a resolução é um operador sobre pares de termos disjuntivos para produzir novos termos disjuntivos. A forma que a base de dados assume é referida como *conjunção de disjunções*. Ela é uma *conjunção* porque todas as cláusulas que constituem a base de dados são consideradas verdadeiras ao mesmo tempo. Ela é uma *disjunção*, uma vez que cada cláusula individual é expressa com uma disjunção (ou  $\vee$ ) como conectivo. Assim, a base de dados inteira da Figura 14.3 pode ser representada na forma clausular como:

$$(\neg\text{cão}(X) \vee \text{animal}(X)) \wedge (\neg\text{animal}(Y) \vee \text{morre}(Y)) \wedge (\text{cão}(\text{fido}))$$

A essa expressão acrescentamos (por conjunção) a negação do que desejamos provar, nesse caso  $\neg\text{morre}(\text{fido})$ . Geralmente, a base de dados é escrita como um conjunto de disjunções, e os operadores  $\wedge$  são omitidos.

Apresentamos, agora, um algoritmo constituído por uma sequência de transformações para reduzir qualquer conjunto de declarações do cálculo de predicados à forma clausular. Foi mostrado (Chang e Lee, 1973) que essas transformações podem ser usadas para reduzir qualquer conjunto de expressões do cálculo de predicados a um conjunto de cláusulas que são inconsistentes, se, e somente se, o conjunto original de expressões for inconsistente. A forma clausular não será estritamente equivalente ao conjunto original de expressões do cálculo de predicados, uma vez que certas interpretações podem ser perdidas. Isso ocorre porque a skolemização restringe as substituições possíveis para as variáveis quantificadas existencialmente (Chang e Lee, 1973). Entretanto, ela preserva a insatisfabilidade. Isto é, se houver uma contradição (uma refutação) no conjunto original de expressões do cálculo de predicados, existirá uma contradição na forma clausular. As transformações não sacrificam a completude para provas por refutação.

Demonstramos esse processo de redução para a forma normal conjuntiva por meio de um exemplo e oferecemos uma breve descrição discutindo cada passo. Não pretendemos que estas sejam provas da equivalência dessas transformações para todas as expressões do cálculo de predicados.

Na expressão a seguir, de acordo com as convenções do Capítulo 2, letras maiúsculas indicam variáveis ( $W, X, Y$  e  $Z$ ); letras minúsculas do meio do alfabeto indicam constantes ou variáveis ligadas ( $l, m$  e  $n$ ); letras minúsculas do início do alfabeto indicam os nomes de predicados ( $a, b, c, d$  e  $e$ ). Para aumentar a legibilidade das expressões, usamos parênteses () e colchetes [], e removemos os que forem redundantes. Como exemplo, considere a seguinte expressão, onde  $X, Y$  e  $Z$  são variáveis e  $l$  é uma constante:

$$(i) (\forall X)([a(X) \wedge b(X)] \rightarrow [c(X,l) \wedge (\exists Y)((\exists Z)[c(Y,Z)] \rightarrow d(X,Y))]) \vee (\forall X)(e(X))$$

1. Primeiro eliminamos o  $\rightarrow$  usando a forma equivalente provada no Capítulo 2:  $a \rightarrow b \equiv \neg a \vee b$ . Essa transformação reduz a expressão em (i) anterior:

$$(ii) (\forall X)(\neg[a(X) \wedge b(X)] \vee [c(X,l) \wedge (\exists Y)(\neg[c(Y,Z)] \vee d(X,Y))] \vee (\forall X)(e(X)))$$

2. A seguir, reduzimos o escopo da negação. Isso pode ser realizado usando várias das transformações do Capítulo 2. Entre elas se incluem:

$$\neg(\neg a) \equiv a$$

$$\neg(\exists X) a(X) \equiv (\forall X) \neg a(X)$$

$$\neg(\forall X) b(X) \equiv (\exists X) \neg b(X)$$

$$\neg(a \wedge b) \equiv \neg a \vee \neg b$$

$$\neg(a \vee b) \equiv \neg a \wedge \neg b$$

Usando a quarta equivalência (ii) se torna:

$$(iii) (\forall X)([\neg a(X) \vee \neg b(X)] \vee [c(X, l) \wedge (\exists Y)((\exists Z)[\neg c(Y, Z)] \vee d(X, Y))]) \vee (\forall X)(e(X))$$

3. A seguir, padronizamos a expressão trocando nomes de todas as variáveis, de modo que as variáveis ligadas por diferentes quantificadores tenham nomes distintos. Como indicado no Capítulo 2, como os nomes das variáveis são “fictícios” ou “marcadores de lugar”, o nome particular escolhido para uma variável não afeta o seu valor verdade nem a generalidade da cláusula. As transformações usadas nesse passo são da forma:

$$((\forall X)a(X) \vee (\forall X)b(X)) \equiv (\forall X)a(X) \vee (\forall Y)b(Y)$$

Como (iii) tem duas ocorrências da variável X, reescrevemos:

$$(iv) (\forall X)([\neg a(X) \vee \neg b(X)] \vee [c(X, l) \wedge (\exists Y)((\exists Z)[\neg c(Y, Z)] \vee d(X, Y))]) \vee (\forall W)(e(W))$$

4. Mova todos os quantificadores para a esquerda sem modificar a sua ordem. Isso é possível porque o passo 3 removeu a possibilidade de qualquer conflito entre nomes de variáveis. (iv) torna-se agora:

$$(v) (\forall X)(\exists Y)(\exists Z)(\forall W)([\neg a(X) \vee \neg b(X)] \vee [c(X, l) \wedge (\neg c(Y, Z) \vee d(X, Y))]) \vee e(W)$$

Após o passo 4, dizemos que a cláusula está na forma *normal prenex*, porque todos os quantificadores estão na frente como um *prefixo* e a expressão, ou *matriz*, os segue.

5. Nesse ponto, todos os quantificadores existenciais são eliminados por um processo denominado *skolemização*. A expressão (v) tem um quantificador existencial para Y. Quando uma expressão contém uma variável quantificada existencialmente, por exemplo,  $(\exists Z)(\text{foo}(\dots, Z, \dots))$ , pode-se concluir que existe uma atribuição para Z sob a qual foo é verdadeiro. A skolemização identifica esse valor. Ela não necessariamente mostra *como* produzir esse valor; ela é apenas um método para dar um nome a uma atribuição que *deve* existir. Se k representa essa atribuição, então temos  $\text{foo}(\dots, k, \dots)$ . Assim:

$$(\exists X)(\text{cão}(X)) \text{ pode ser substituído por } \text{cão}(\text{fido})$$

onde o nome fido foi selecionado do domínio de definição de X para representar o indivíduo X. fido é denominado uma *constante de Skolem*. Se o predicado tiver mais que um argumento e a variável quantificada existencialmente estiver dentro do escopo de variáveis quantificadas universalmente, a variável existencial deve ser uma função dessas outras variáveis. Isso é representado no processo de skolemização:

$$(\forall X)(\exists Y)(\text{mãe}(X, Y))$$

Essa expressão indica que toda pessoa tem uma mãe. Toda pessoa é um X e a mãe existente será uma função da pessoa X em particular que é escolhida. Assim, a skolemização resulta em:

$$(\forall X)\text{mãe}(X, m(X))$$

que indica que cada X tem uma mãe (o m daquele X). Em outro exemplo:

$$(\forall X)(\forall Y)(\exists Z)(\forall W)(\text{foo}(X, Y, Z, W))$$

é skolemizado para:

$$(\forall X)(\forall Y)(\forall W)(\text{foo}(X, Y, f(X, Y), W))$$

As variáveis Y e Z, quantificadas existencialmente, estão dentro do escopo (à direita) da variável X, quantificada universalmente, mas não dentro do escopo de W. Assim, cada uma será substituída por uma função de Skolem de X. Substituindo Y pela função de Skolem f(X) e Z por g(X), (v) se torna:

$$(vi) (\forall X)(\forall W)([\neg a(X) \vee \neg b(X)] \vee [c(X, l) \wedge (\neg c(f(X), g(X)) \vee d(X, f(X)))] \vee e(W))$$

Após a skolemização, o passo 6 pode ser seguido, o que simplesmente retira os prefixos.

6. Retire todas as quantificações universais. Nesse ponto existem apenas variáveis quantificadas universalmente (passo 5), sem nenhum conflito de variáveis (passo 3). Assim, todos os quantificadores podem ser retirados, e qualquer procedimento de prova empregado supõe que todas as variáveis sejam quantificadas universalmente. A fórmula (vi) torna-se agora:

$$(vii) [\neg a(X) \vee \neg b(X)] \vee [c(X,I) \wedge (\neg c(f(X),g(X)) \vee d(X,f(X)))] \vee e(W)$$

7. A seguir, converta a expressão para a forma de conjunção de disjunções. Isso requer o uso das propriedades associativa e distributiva de  $\wedge$  e  $\vee$ . Relembrando o Capítulo 2, em que

$$a \vee (b \vee c) = (a \vee b) \vee c$$

$$a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

que indica que  $\wedge$  ou  $\vee$  podem ser agrupados de qualquer maneira desejada. A propriedade distributiva do Capítulo 2 também é usada, quando necessário. Visto que

$$a \wedge (b \vee c)$$

já está na forma clausal,  $\wedge$  não será distribuída. Entretanto,  $\vee$  deve ser distribuída em relação a  $\wedge$  usando:

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

A forma final de (vii) é:

$$(viii) [\neg a(X) \vee \neg b(X) \vee c(X,I) \vee e(W)] \wedge$$

$$[\neg a(X) \vee \neg b(X) \vee \neg c(f(X),g(X)) \vee d(X,f(X)) \vee e(W)]$$

8. Agora faça com que cada conjunção seja uma cláusula separada. No exemplo (viii) há duas cláusulas:

$$(ixa) \neg a(X) \vee \neg b(X) \vee c(X,I) \vee e(W)$$

$$(ixb) \neg a(X) \vee \neg b(X) \vee \neg c(f(X),g(X)) \vee d(X,f(X)) \vee e(W)$$

9. O último passo é novamente *padronizar as variáveis separadamente*. Isso requer que sejam dados nomes diferentes para as variáveis em cada cláusula gerada pelo passo 8. Esse procedimento surge da equivalência estabelecida no Capítulo 2 que

$$(\forall X) (a(X) \wedge b(X)) \equiv (\forall X) a(X) \wedge (\forall Y) b(Y)$$

que resulta da natureza dos nomes de variáveis como marcadores de lugar, (ixa) e (ixb) se tornam agora, usando novos nomes de variáveis, U e V:

$$(xa) \neg a(X) \vee \neg b(X) \vee c(X,I) \vee e(W)$$

$$(xb) \neg a(U) \vee \neg b(U) \vee \neg c(f(U),g(U)) \vee d(U,f(U)) \vee e(V)$$

A importância dessa padronização final se torna aparente apenas quando apresentamos os passos de unificação da resolução. Encontramos a unificação mais geral que toma dois predicados dentro de duas cláusulas equivalentes e, então, essa substituição é feita em todas as variáveis do mesmo nome dentro de cada cláusula. Assim, se algumas variáveis (desnecessariamente) compartilharem nomes com outras, elas podem ser renomeadas pelo processo de unificação com uma subsequente (possível) perda de generalidade da solução.

Esse processo em nove passos é usado para transformar qualquer conjunto de expressões do cálculo de predicados em forma clausal. A propriedade da completude da resolução por refutações não é perdida. A seguir, demonstramos o procedimento de resolução para gerar provas a partir dessas cláusulas.

### 14.2.3 Procedimento de prova por resolução binária

O procedimento de prova por *resolução por refutação* responde a uma questão ou deduz um novo resultado por meio da redução do conjunto de cláusulas a uma contradição, representada pela cláusula nula ( $\square$ ). A contradi-

ção é produzida resolvendo pares de cláusulas da base de dados. Se uma resolução não produzir diretamente uma contradição, então a cláusula produzida pela resolução, o *resolvente*, é acrescentada à base de dados de cláusulas e o processo continua.

Antes de mostrarmos como o processo de resolução funciona no cálculo de predicados, apresentamos um exemplo do cálculo proposicional, sem variáveis. Considere duas cláusulas *pais*  $p_1$  e  $p_2$  do cálculo proposicional:

$$p_1: a_1 \vee a_2 \vee \dots \vee a_n$$

$$p_2: b_1 \vee b_2 \vee \dots \vee b_m$$

que têm dois literais  $a_i$  e  $b_j$ , onde  $1 < i \leq n$  e  $1 \leq j \leq m$ , tal que  $\neg a_i = b_j$ . A resolução binária produz a cláusula:

$$a_1 \vee \dots \vee a_{i-1} \vee a_{i+1} \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_{j-1} \vee b_{j+1} \vee \dots \vee b_m.$$

A notação anterior indica que o resolvente é constituído da disjunção de todos os literais das duas cláusulas pais, exceto os literais  $a_i$  e  $b_j$ .

Um argumento simples pode fornecer a intuição por trás do princípio da resolução. Suponha que

$$a \vee \neg b \text{ e } b \vee c$$

sejam duas declarações verdadeiras. Observe que uma das declarações,  $b$  ou  $\neg b$ , deve ser sempre verdadeira e a outra, sempre falsa ( $b \vee \neg b$  é uma tautologia). Portanto, uma das variáveis em

$$a \vee c$$

deve ser sempre verdadeira,  $a \vee c$  é o resolvente das duas cláusulas pais  $a \vee \neg b$  e  $b \vee c$ .

Considere, agora, um exemplo do cálculo proposicional, onde queremos provar  $a$  a partir dos seguintes axiomas (é claro que  $I \leftarrow m \equiv m \rightarrow I$ , para todas as proposições  $I$  e  $m$ ):

$$a \leftarrow b \wedge c$$

$$b$$

$$c \leftarrow d \wedge e$$

$$e \vee f$$

$$d \wedge \neg f$$

Reduzimos o primeiro axioma à forma clausular:

$$a \leftarrow b \wedge c$$

$$a \vee \neg(b \wedge c) \quad \text{por } I \rightarrow m \equiv \neg I \vee m$$

$$a \vee \neg b \vee \neg c \quad \text{pela lei de de Morgan}$$

Os axiomas restantes são reduzidos e temos as seguintes cláusulas:

$$a \vee \neg b \vee \neg c$$

$$b$$

$$c \vee \neg d \vee \neg e$$

$$e \vee f$$

$$d$$

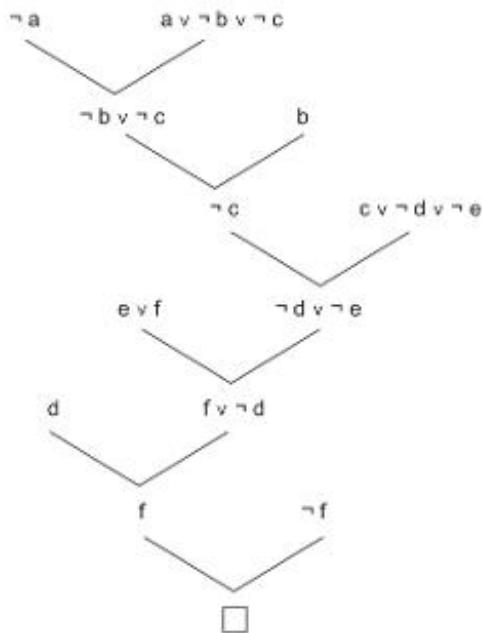
$$\neg f$$

A prova por resolução está na Figura 14.4. Primeiro, o objetivo a ser provado,  $a$ , é negado e acrescentado ao conjunto de cláusulas. A derivação de  $\square$  indica que a base de cláusulas é inconsistente.

Para usar resolução binária no cálculo de predicados, onde cada literal pode conter variáveis, deve existir um processo pelo qual dois literais com nomes de variáveis diferentes, ou um com um valor constante, pode ser visto como equivalente. A unificação foi definida na Seção 2.3.2 como o processo para determinar as substituições mais gerais e consistentes, que tornam dois predicados equivalentes.

O algoritmo da resolução para o cálculo de predicados é muito parecido com o do cálculo proposicional, exceção que:

**Figura 14.4** Uma prova por resolução para um exemplo do cálculo proposicional.



1. Um literal e a sua negação nas cláusulas pais produzem um resolvente apenas se eles forem unificados para uma substituição  $\sigma$ .  $\sigma$  é, então, aplicada ao resolvente antes de acrescentá-lo ao conjunto de cláusulas. Exigimos que  $\sigma$  seja o unificador mais genérico das cláusulas pais.
2. As substituições de unificação usadas para encontrar a contradição oferecem ligações de variáveis pelas quais a consulta inicial é verdadeira. Explicitamos esse processo, denominado *extração de resposta*, na Seção 14.2.5.

Ocasionalmente, dois ou mais literais em uma cláusula têm uma substituição unificadora. Quando isso acontece, pode ser que não exista uma refutação para um conjunto de cláusulas contendo essa cláusula, muito embora o conjunto seja contraditório. Por exemplo, considere as cláusulas:

$$\begin{aligned} p(X) \vee p(f(Y)) \\ \neg p(W) \vee \neg p(f(Z)) \end{aligned}$$

O leitor deve notar que, com a resolução simples, essas cláusulas podem ser reduzidas apenas a formas equivalentes ou tautológicas, mas não a uma contradição, isto é, nenhuma substituição pode torná-las inconsistentes.

Essa situação pode ser tratada aplicando-se *fatoração* a essas cláusulas. Se um subconjunto dos literais em uma cláusula tiver um unificador mais geral (Seção 2.3.2), então ela é substituída por uma nova cláusula denominada *fator* daquela cláusula. O fator é a cláusula original modificada pela aplicação da substituição unificadora mais geral, removendo-se, então, os literais redundantes. Por exemplo, os dois literais da cláusula  $p(X) \vee p(f(Y))$  serão unificados pela substituição  $\{f(Y)/X\}$ . Fazemos a substituição nos dois literais para obter a cláusula  $p(f(Y)) \vee p(f(Y))$  e, então, substituímos essa cláusula pelo seu fator:  $p(f(Y))$ . Qualquer sistema de resolução por refutação que inclua fatoração é completo para a refutação. A padronização das variáveis por separação, Seção 14.2.2, passo 3, pode ser interpretada como uma aplicação trivial da fatoração. A fatoração pode ser tratada, também, como parte do processo de inferência na *hiper-resolução* descrita na Seção 14.4.2.

Apresentamos, agora, um exemplo de uma resolução por refutação para o cálculo de predicados. Considere a seguinte história do “estudante feliz”:

Qualquer um que passe no seu exame de história e que ganhe na loteria é feliz. Mas quem estuda ou tem sorte pode passar em todos os exames. John não estudou, mas ele tem sorte. Quem tem sorte ganha na loteria. John é feliz?

Primeiro, passe as sentenças para a forma de predicados:

Qualquer um que passe no exame de história e ganhe na loteria é feliz.

$\forall X (\text{passa}(X, \text{história}) \wedge \text{ganha}(X, \text{loteria}) \rightarrow \text{feliz}(X))$

Quem estuda ou tem sorte pode passar em todos os seus exames.

$\forall X \forall Y (\text{estuda}(X) \vee \text{tem\_sorte}(X) \rightarrow \text{passa}(X, Y))$

John não estudou, mas ele tem sorte.

$\neg \text{estuda}(\text{john}) \wedge \text{tem\_sorte}(\text{john})$

Qualquer um que tem sorte ganha na loteria.

$\forall X (\text{tem\_sorte}(X) \rightarrow \text{ganha}(X, \text{loteria}))$

Esses quatro predicados são transformados agora para a forma clausular (Seção 14.2.2):

1.  $\neg \text{passa}(X, \text{história}) \vee \neg \text{ganha}(X, \text{loteria}) \vee \text{feliz}(X)$
2.  $\neg \text{estuda}(Y) \vee \text{passa}(Y, Z)$
3.  $\neg \text{tem\_sorte}(W) \vee \text{passa}(W, V)$
4.  $\neg \text{estuda}(\text{john})$
5.  $\text{tem\_sorte}(\text{john})$
6.  $\neg \text{tem\_sorte}(U) \vee \text{ganha}(U, \text{loteria})$

A essas cláusulas é acrescentada, na forma clausular, a negação da conclusão:

7.  $\neg \text{feliz}(\text{john})$

O grafo de resolução por refutação da Figura 14.5 mostra uma derivação da contradição e, consequentemente, prova que John é feliz.

Como exemplo final nessa subseção, apresentamos o problema da “vida excitante”; suponha que:

Quem não é pobre e for esperto é feliz. Aqueles que sabem ler são espertos. John sabe ler e não é pobre. Pessoas felizes têm vida excitante. Podemos encontrar alguém com uma vida excitante?

Supomos:  $\forall X (\text{esperto}(X) \equiv \neg \text{estúpido}(X)) \wedge \forall Y (\text{rico}(Y) \equiv \neg \text{pobre}(Y))$  e obtemos:

$\forall X (\neg \text{pobre}(X) \wedge \text{esperto}(X) \rightarrow \text{feliz}(X))$

$\forall Y (\text{lê}(Y) \rightarrow \text{esperto}(Y))$

$\text{lê}(\text{john}) \wedge \neg \text{pobre}(\text{john})$

$\forall Z (\text{feliz}(Z) \rightarrow \text{excitante}(Z))$

A negação da conclusão é:

$\neg \exists W (\text{excitante}(W))$

Essas expressões do cálculo de predicados para o problema da “vida excitante” são transformadas nas seguintes cláusulas:

$\text{pobre}(X) \vee \neg \text{esperto}(X) \vee \text{feliz}(X)$

$\neg \text{lê}(Y) \vee \text{esperto}(Y)$

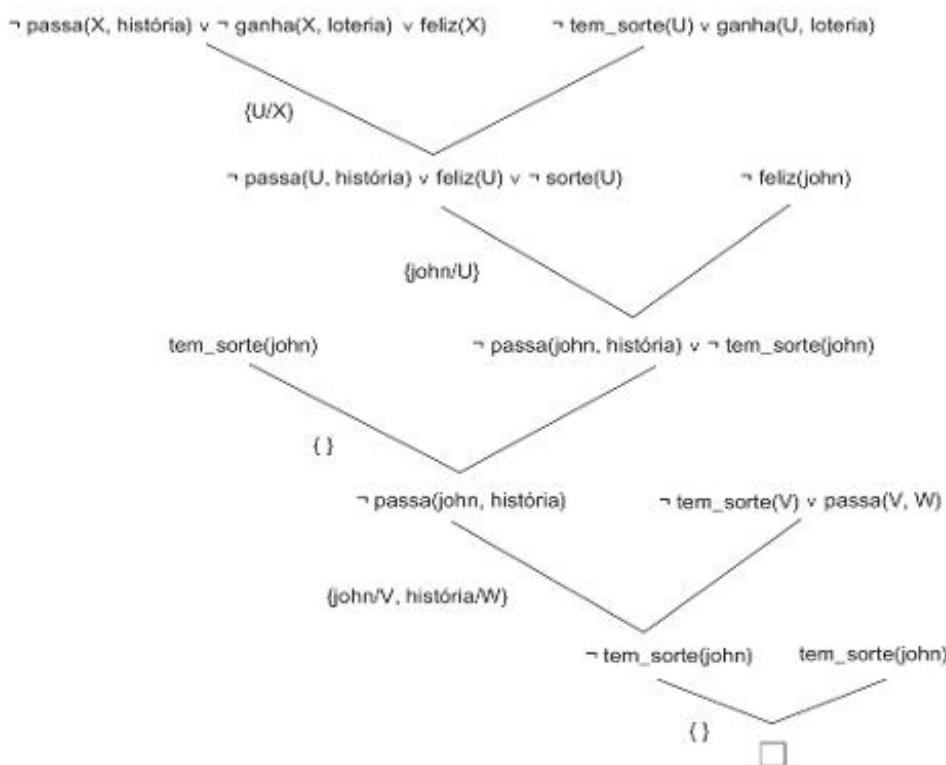
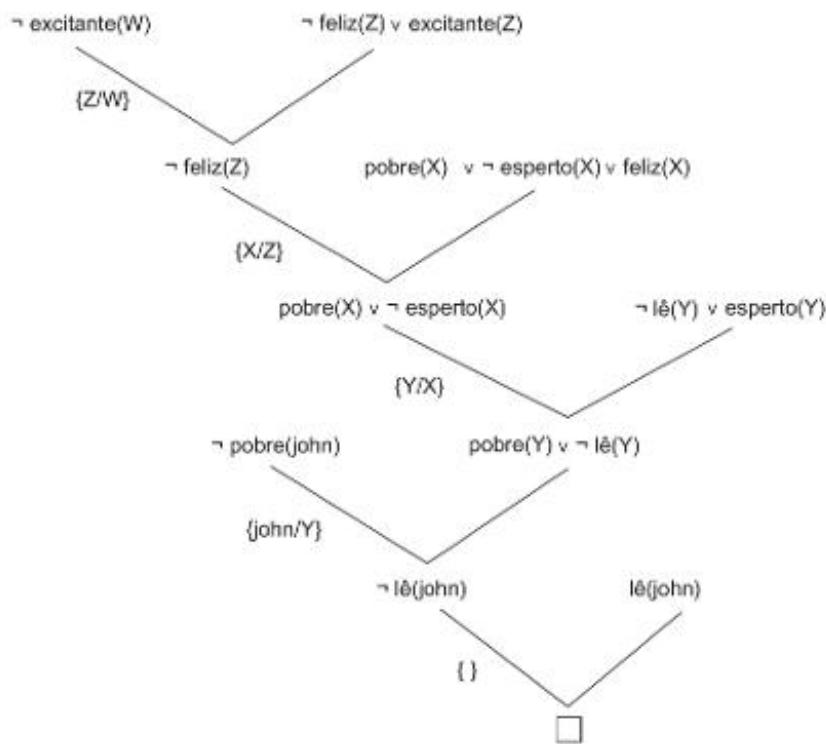
$\text{lê}(\text{john})$

$\neg \text{pobre}(\text{john})$

$\neg \text{feliz}(Z) \vee \text{excitante}(Z)$

$\neg \text{excitante}(W)$

A resolução por refutação para esse exemplo se encontra na Figura 14.6.

**Figura 14.5** Uma resolução por refutação para o problema do “estudante feliz”.**Figura 14.6** A prova por resolução para o problema da “vida excitante”.

#### 14.2.4 Estratégias e técnicas de simplificação para a resolução

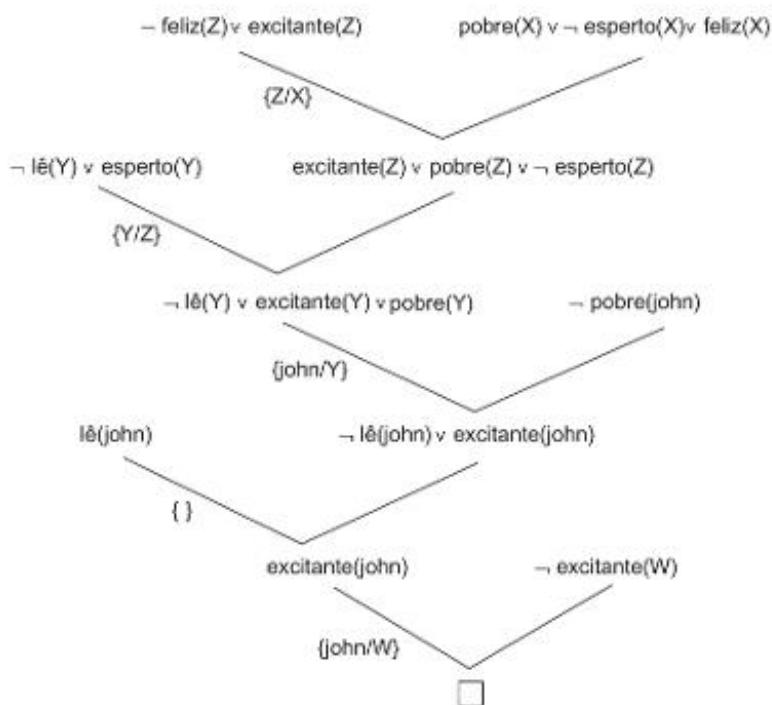
Uma árvore de prova diferente para o espaço de busca do problema da Figura 14.6 aparece na Figura 14.7. Há algumas semelhanças nessas provas; por exemplo, ambas levam cinco passos de resolução. Também para as duas provas, a aplicação associativa das substituições unificadoras encontrou que john era a ocorrência de pessoa com “vida excitante”.

Entretanto, mesmo essas duas semelhanças não necessariamente teriam que ocorrer. Quando o sistema de prova por resolução foi definido (Seção 14.2.3), não foi imposta nenhuma ordem de combinação de cláusulas. Essa é uma questão crítica: quando há  $N$  cláusulas no espaço de cláusulas, existem  $N^2$  formas de combiná-las ou verificar se elas podem ser combinadas, apenas no primeiro nível! O conjunto de cláusulas resultante dessa comparação é também bastante grande; mesmo se 20% delas produzissem novas cláusulas, a próxima rodada de resoluções possíveis conteria mais combinações que a primeira. Em um problema grande, esse crescimento exponencial sairá rapidamente de controle.

Por essa razão, heurísticas de busca são muito importantes nos procedimentos de prova por resolução, bem como em todas as soluções de problema por método fraco. Como ocorreu com as heurísticas que consideramos no Capítulo 4, não existe um método que possa determinar a melhor estratégia para um problema em particular. Apesar disso, algumas estratégias gerais podem lidar com a explosão combinatória exponencial.

Antes de descrevermos as nossas estratégias, precisamos esclarecer vários pontos. Primeiro, com base na definição de insatisfabilidade de uma expressão do Capítulo 2, um *conjunto de cláusulas é insatisfazível* se não existir uma interpretação que estabeleça o conjunto como satisfazível. Segundo, uma regra de inferência é *completa para a refutação* se, dado um conjunto de cláusulas insatisfazíveis, for possível estabelecer, pelo uso apenas dessa regra de inferência, que o conjunto é insatisfazível. A resolução com fatoração tem essa propriedade (Chang e Lee, 1973). Por fim, uma *estratégia é completa* se, por meio de seu uso com uma regra de inferência completa para a refutação, tivermos a garantia de poder encontrar uma refutação sempre que um conjunto de cláusulas for insatisfazível. A *busca em amplitude* é um exemplo de uma estratégia completa.

**Figura 14.7** Outra resolução por refutação para o problema da Figura 14.6.

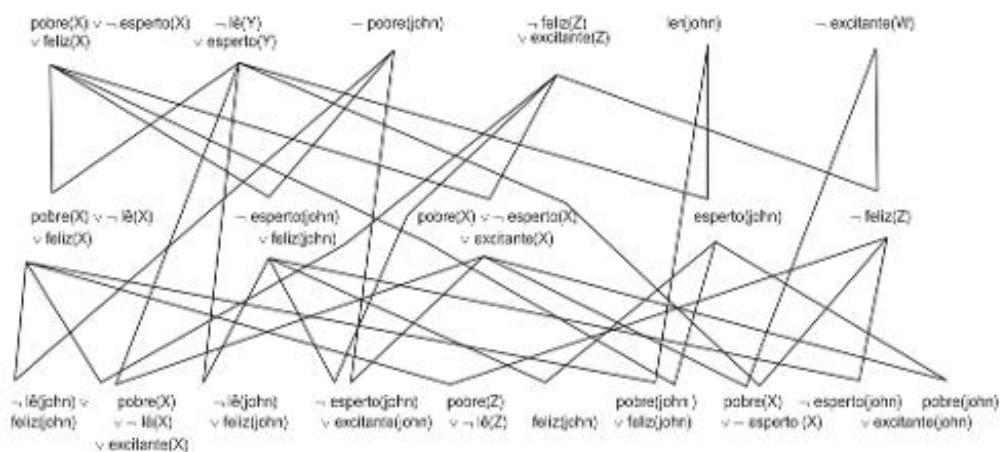


### Estratégia de busca em amplitude

A análise da complexidade da comparação exaustiva de cláusulas foi baseada na busca em amplitude. Na primeira rodada, cada cláusula no espaço de cláusulas é comparada com todas as cláusulas no espaço de cláusulas para verificar a sua resolução. As cláusulas no segundo nível do espaço de busca são geradas resolvendo as cláusulas produzidas no primeiro nível para todas as cláusulas originais. Geramos as cláusulas no nível  $n$  resolvendo todas as cláusulas no nível  $n - 1$  para os elementos do conjunto de cláusulas original e para todas as cláusulas previamente produzidas.

Essa estratégia pode rapidamente sair de controle para problemas grandes. Entretanto, ela tem uma propriedade interessante. Como ocorre em qualquer busca em amplitude, ela garante encontrar o caminho-solução mais curto, porque ela gera todos os estados da busca para cada nível antes de passar para um nível mais profundo. Ela é, também, uma estratégia completa, já que, se for executada por tempo suficiente, garante encontrar uma refutação, caso exista. Assim, quando o problema é pequeno, como o são os apresentados nos nossos exemplos, a estratégia de busca em amplitude pode ser uma boa opção. A Figura 14.8 aplica a estratégia de busca em amplitude ao problema da “vida excitante”.

**Figura 14.8** Espaço de estados completo para o problema da “vida excitante” gerado por busca em amplitude (para dois níveis).



### Estratégia do conjunto de suporte

Uma excelente estratégia para espaços de cláusulas grandes é o chamado conjunto de suporte (Wos e Robinson, 1968). Para um conjunto de cláusulas de entrada,  $S$ , podemos especificar um subconjunto,  $T$  de  $S$ , denominando o conjunto de suporte. A estratégia requer que um dos resolventes em cada resolução tenha um ancestral no conjunto de suporte. Pode ser provado que se  $S$  é insatisfazível e  $S - T$  é satisfazível, então a estratégia do conjunto de suporte é completa para a refutação (Wos et al., 1984).

Se o conjunto de cláusulas original for consistente, então qualquer conjunto de suporte que inclua a negação da consulta original satisfaz esses requisitos. Essa estratégia é baseada na visão de que a negação do que queremos provar fará com que o espaço de cláusulas seja contraditório. O conjunto de suporte força resoluções entre cláusulas das quais ao menos uma é a cláusula objetivo negada, ou então uma cláusula produzida pelas resoluções para o objetivo negado.

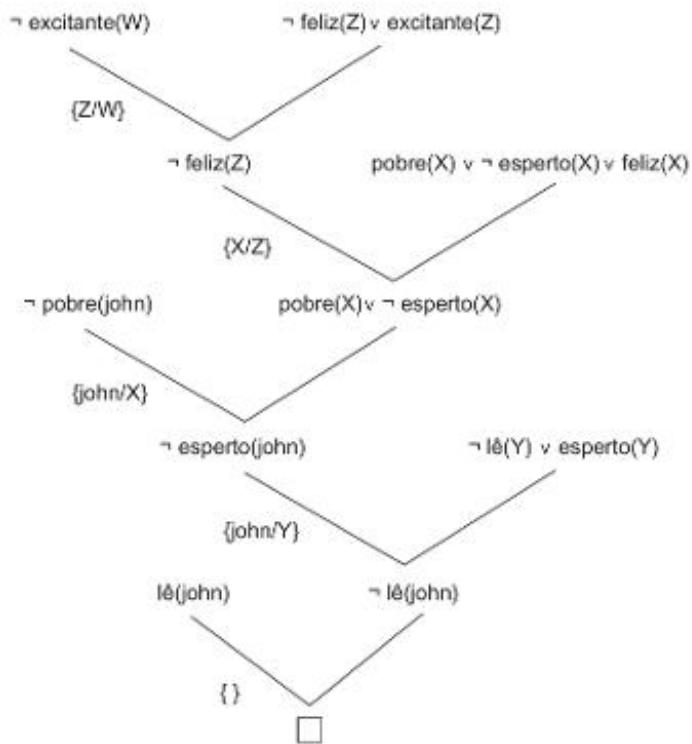
A Figura 14.6 é um exemplo da estratégia do conjunto de suporte aplicada ao problema da vida excitante. Como uma refutação por conjunto de suporte existe sempre que uma refutação existir, o conjunto de suporte pode ser tomado como a base de uma estratégia completa. Uma maneira de se fazer isso é realizar uma busca em amplitude sobre todas as refutações por conjunto de suporte possíveis. É claro que isso será muito mais eficiente que a busca em amplitude em todas as cláusulas. Precisamos apenas que todos os resolventes do objetivo negado sejam examinados com todos os seus descendentes.

### Estratégia da preferência por unidade

Observe que nos exemplos de resolução vistos até agora, a derivação da contradição é indicada pela cláusula que não contém literais. Assim, sempre que produzimos uma cláusula resultante que tem menos literais que as cláusulas que foram usadas para criá-la, estaremos mais perto de produzir a cláusula sem literais. Em particular, resolvendo uma cláusula com um literal, denominada cláusula *unitária*, garantiremos que o resolvente seja menor que a maior cláusula pai. A estratégia da preferência por unidade usa unidades, ou seja, cláusulas unitárias, na resolução, sempre que elas estiverem disponíveis. A Figura 14.9 usa a estratégia da preferência por unidade com o conjunto de suporte para produzir uma estratégia completa mais eficiente.

A resolução por unidade é uma estratégia correlacionada que requer que um dos resolventes seja sempre uma cláusula unitária. Esse é um requisito mais forte que a estratégia da preferência por unidade. Podemos mostrar que a resolução por unidade não é completa usando o mesmo exemplo que mostra a incompletude da forma de entrada linear.

**Figura 14.9** Usando a estratégia da preferência por unidade para o problema da “vida excitante”.



### Estratégia da forma de entrada linear

A estratégia da forma de entrada linear é um uso direto do objetivo negado e dos axiomas originais: tome o objetivo negado e resolva-o com um dos axiomas, obtendo uma nova cláusula. Esse resultado é resolvido com um dos axiomas para obter outra cláusula, que é resolvida novamente com um dos axiomas. Esse processo continua até que seja produzida a cláusula vazia.

A cada estágio, resolvemos a cláusula que foi obtida mais recentemente com um axioma derivado das declarações originais do problema. Nunca usamos uma cláusula previamente derivada, nem resolvemos dois axiomas ao mesmo tempo. A forma de entrada linear não é uma estratégia completa, como pode ser visto a partir da sua aplicação ao seguinte conjunto de quatro cláusulas (que são obviamente insatisfazíveis). Independ-

dentemente de que cláusula for tomada como a negação do objetivo, a estratégia de entrada linear não é capaz de produzir uma contradição:

```
¬a ∨ ¬b  
a ∨ ¬b  
¬a ∨ b  
a ∨ b
```

### Outras estratégias e técnicas de simplificação

Não pretendemos apresentar um conjunto exaustivo de estratégias ou mesmo as técnicas mais sofisticadas para provar teoremas usando inferência por resolução. Essas estão disponíveis na literatura, como, por exemplo Wos et al. (1984) e Wos (1988). O nosso objetivo é, em vez disso, introduzir as ferramentas básicas para essa área de pesquisa e descrever como essas ferramentas podem ser usadas na solução de problemas. Como já observamos, o procedimento de prova por resolução é apenas outra técnica de solução de problemas pelo método fraco.

Nesse sentido, a resolução pode servir como um motor de inferência para o cálculo de predicados, mas um motor que requer muita análise e aplicação cuidadosa de estratégias para ser bem-sucedido. Em um problema suficientemente grande para ser interessante, a colisão aleatória de expressões, junto à resolução, é tão decepcionante quanto usar aleatoriamente as teclas em um teclado de computador e esperar que daí resulte um artigo científico de qualidade.

Os exemplos usados neste capítulo são muito pequenos e têm todas as cláusulas necessárias (e apenas aquelas necessárias) para a sua solução. Isso é raramente verdade em problemas interessantes. Apresentamos várias estratégias simples para combater essas complexidades combinatórias e concluímos essa subseção descrevendo algumas outras considerações importantes para o projeto de um resolvedor de problemas baseado em resolução. Mais adiante (na Seção 14.3), mostramos como um sistema de resolução por refutação, com uma interessante combinação de estratégias de busca, fornece uma “semântica” para *programação em lógica*, especialmente para o projeto de interpretadores Prolog.

Uma combinação de estratégias pode ser bastante efetiva para controlar a busca — por exemplo, o uso de conjunto de suporte aliado à preferência por unidade. As heurísticas de busca podem ser também incorporadas no projeto de regras (criando uma ordenação da esquerda para a direita dos literais para a resolução). Essa ordenação pode ser muito efetiva para podar o espaço de busca. Esse uso implícito de estratégia é importante na programação Prolog (Seção 14.3).

A generalidade das conclusões pode ser um critério para projetar uma estratégia de solução. Por um lado, poderia ser importante armazenar soluções intermediárias, tão gerais quanto possível, pois isso permite que elas sejam usadas mais livremente na solução. Assim, a introdução de qualquer resolução para cláusulas que requerem especialização por variáveis de ligação, como {john/X}, deveria ser evitada tanto quanto possível. Por outro lado, se uma solução requerer ligações de variáveis específicas, como na análise se John tem uma infecção por estafilococos, as substituições {john/Pessoa} e {estafilococos/Infecção} podem restringir o espaço de busca e aumentar a probabilidade e a velocidade para encontrar uma solução.

Uma questão importante na seleção de uma estratégia é a noção de completude. Pode ser muito importante, em algumas aplicações, saber que uma solução será encontrada (se existir uma solução). Isso pode ser garantido se forem usadas apenas estratégias completas.

Podemos, também, aumentar a eficiência acelerando o processo de casamento. Podemos eliminar unificações desnecessárias (e custosas) entre cláusulas que não podem produzir novos resolventes, indexando cada cláusula com os literais que ela contém, e se eles são positivos ou negativos. Isso nos permite encontrar diretamente potenciais resolventes para uma cláusula qualquer. Deveríamos, também, eliminar certas cláusulas tão logo elas fossem produzidas. Primeiro, qualquer cláusula tautológica não precisa ser considerada; elas nunca se tornarão falsas e, assim, não são úteis na busca por uma solução.

Outro tipo de cláusula que não fornece informação é aquela que está *inclusa* em outra, isto é, quando uma nova cláusula já tiver uma ocorrência mais geral no espaço de cláusulas. Por exemplo, se p(john) for deduzida para um espaço que já contenha  $\forall X (p(X))$ , então p(john) pode ser descartada sem perdas; na verdade, há uma economia por-

que há menos cláusulas no espaço de cláusulas. De modo similar,  $p(X)$  está incluída na cláusula  $p(X) \vee q(X)$ . Informação menos genérica não acrescenta nada à informação mais geral quando ambas estão no espaço de cláusulas.

Por fim, a *associação procedural* avalia, ou processa sem continuar a busca, qualquer cláusula que puder produzir informação nova. Ela realiza operações aritméticas, comparações entre átomos ou cláusulas, ou “executa” qualquer outro procedimento determinístico que possa acrescentar informação concreta à solução do problema, ou que restrinja, de uma maneira qualquer, o processo de solução. Por exemplo, podemos usar um procedimento para calcular uma ligação para uma variável quando informação suficiente para isso estiver presente. Essa ligação de variável restringe possíveis resoluções e poda o espaço de busca.

A seguir, mostramos como podemos extrair respostas do processo de resolução por refutação.

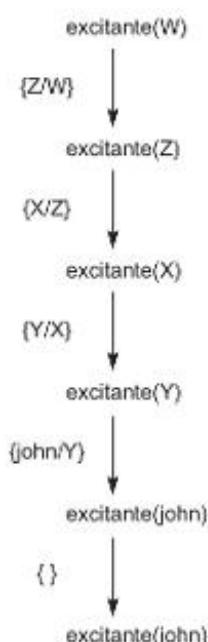
### 14.2.5 Extração de resposta de resolução por refutações

Os casos para os quais uma hipótese é verdadeira incluem o conjunto de substituições pelas quais a refutação é encontrada. Portanto, a retenção da informação sobre as substituições de unificação feitas na resolução por refutação fornece informação sobre a resposta correta. Nesta subseção, daremos três exemplos disso e introduziremos um método de contabilização para extrair respostas de uma resolução por refutação.

O método de registro de respostas é simples: reter a conclusão original a ser provada e, nessa conclusão, introduzir cada unificação que for feita no processo de resolução. Assim, a conclusão original é a “contabilização” de todas as unificações que são feitas como parte da refutação. Na busca computacional por resolução por refutações, isso poderia requerer ponteiros extras, como quando existe mais que uma escolha possível na busca por uma refutação. Pode ser necessário um mecanismo de controle, como o retrocesso (*backtracking*), para produzir caminhos de soluções alternativos. Mas, ainda assim, com um pouco de cuidado, essa informação adicional pode ser retida.

Vejamos alguns exemplos desse processo. Na Figura 14.6, onde foi encontrada uma prova para a existência de uma pessoa com uma vida excitante, foram feitas as unificações da Figura 14.10. Se retivermos o objetivo original e aplicarmos todas as substituições da refutação a essa cláusula, encontraremos a resposta sobre qual é a pessoa que tem uma vida excitante.

**Figura 14.10** Substituições de unificação da Figura 14.6 aplicadas à consulta original.



A Figura 14.10 mostra como uma resolução por refutação não só pode mostrar que “ninguém vive uma vida excitante” é falso, como também, no processo dessa demonstração, pode produzir uma pessoa com vida excitante, John. Esse é um resultado geral, onde as unificações que produzem uma refutação são as mesmas que produzem os casos para os quais a consulta original é verdadeira.

Um segundo exemplo é a história simples:

Fido, o cão, vai a todo lugar que John, seu dono, for. John está na biblioteca. Onde está Fido?

Primeiro, representamos essa história em expressões do cálculo de predicados e, então, reduzimos essas expressões à forma clausular. Os predicados:

$\text{em}(\text{john}, X) \rightarrow \text{em}(\text{fido}, X)$   
 $\text{em}(\text{john}, \text{biblioteca})$

As cláusulas:

$\neg \text{em}(\text{john}, Y) \vee \text{em}(\text{fido}, Y)$   
 $\text{em}(\text{john}, \text{biblioteca})$

A conclusão negada:

$\neg \text{em}(\text{fido}, Z)$ , Fido não está em parte alguma!

A Figura 14.11 apresenta o processo de extração da resposta. O literal que rastreia as unificações é a consulta original (onde está Fido?):

$\text{em}(\text{fido}, Z)$

Mais uma vez, as unificações para as quais a contradição é encontrada informam que a consulta original é verdadeira: Fido está na biblioteca.

O exemplo final mostra como o processo de skolemização pode fornecer o caso para o qual a resposta pode ser extraída. Considere a seguinte situação:

Todas as pessoas têm pais. Os pais dos pais são os avós. Dada a pessoa John, prove que John tem avós.

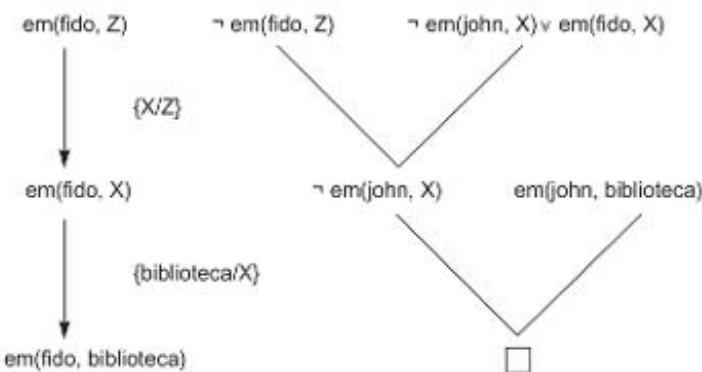
As seguintes sentenças representam os fatos e relacionamentos na situação anterior. Primeiro, “Todas as pessoas têm pais”:

$(\forall X)(\exists Y) p(X, Y)$

Os pais dos pais são os avós:

$(\forall X)(\forall Y)(\forall Z) p(X, Y) \wedge p(Y, Z) \rightarrow \text{avô}(X, Z)$

Figura 14.11 Processo de extração de resposta para o problema “procurando fido”.



O objetivo é encontrar  $W$  tal que  $\text{avô}(\text{john}, W)$  ou  $\exists (W)(\text{avô}(\text{john}, W))$ . A negação do objetivo é  $\neg \exists (W)(\text{avô}(\text{john}, W))$  ou:

$$\neg \text{avô}(\text{john}, W)$$

No processo de colocar os predicados anteriores na forma clausular para a resolução por refutação, o quantificador existencial no primeiro predicado (todas as pessoas têm pais) requer uma função de Skolem. Essa função de Skolem seria a função óbvia: tome o  $X$  dado e encontre um dos pais de  $X$ . Denominemos essa função  $\text{ap}(X)$ , “encontre um ancestral parental de  $X$ ”. Para John, isso seria o seu pai ou a sua mãe. A forma clausular para os predicados desse problema é:

$$\begin{aligned} & p(X, \text{ap}(X)) \\ & \neg p(W, Y) \vee \neg p(Y, Z) \vee \text{avô}(W, Z) \\ & \neg \text{avô}(\text{john}, V) \end{aligned}$$

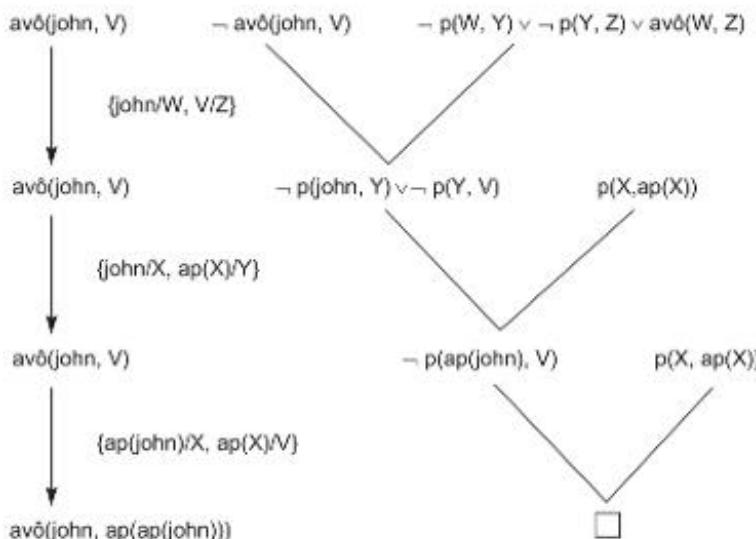
A resolução por refutação e o processo de extração da resposta para esse problema são apresentados na Figura 14.12. Note que as substituições de unificação na resposta são:

$$\text{avô}(\text{john}, \text{ap}(\text{ap}(\text{john})))$$

A resposta à questão se John tem avós é “encontre o ancestral parental do ancestral parental de John”. A função de Skolem nos permite calcular esse resultado.

O processo geral para a extração da resposta, descrito anteriormente, pode ser usado em todas as resoluções por refutação, quer elas se deem pelas unificações gerais, como nas figuras 14.10 e 14.11, quer elas ocorram pela avaliação da função de Skolem, como na Figura 14.12. O processo produz uma resposta. O método é realmente bastante simples: os casos (unificações) para os quais a contradição é encontrada são exatamente aqueles para os quais o oposto da conclusão negada (a consulta original) é verdadeiro. Embora esta subseção não tenha demonstrado matematicamente como isto é verdade para todos os casos, mostramos vários exemplos de como o processo funciona; uma discussão mais detalhada pode ser encontrada na literatura (Nilsson, 1980; Wos et al., 1984).

**Figura 14.12** Skolemização como parte do processo de extração de resposta.



## 14.3 Prolog e raciocínio automatizado

### 14.3.1 Introdução

Apenas compreendendo a implementação de uma linguagem de programação, podemos guiar adequadamente o seu uso, seus efeitos colaterais e ter confiança nos seus resultados. Nesta seção, descrevemos a semântica de Prolog e a relacionamos com as questões de raciocínio automatizado apresentadas na seção anterior.

Uma séria crítica ao procedimento de prova por resolução, visto na Seção 14.2, é que ele requer uma base de dados totalmente homogênea para representar o problema. Quando descritores do cálculo de predicados são reduzidos ou transformados em forma clausular, uma parte importante da informação para solucionar o problema é deixada de fora. A informação omitida não é a verdade ou a falácia de uma parte do problema, mas sim as informações de controle ou as descrições procedimentais sobre como *usar* a informação. Por exemplo, uma cláusula objetivo negada, em um formato para resolução, poderia ser da forma:

$a \vee \neg b \vee c \vee \neg d$

onde  $a$ ,  $b$ ,  $c$  e  $d$  são literais. O mecanismo de inferência por resolução aplica uma estratégia de busca para deduzir a cláusula vazia. Todos os literais estão abertos para a estratégia e aquele que será usado depende da estratégia selecionada. As estratégias usadas para guiar a prova de teoremas por resolução são heurísticas fracas; elas não incorporam conhecimento profundo de um domínio específico de problema.

Por exemplo, a cláusula objetivo negada no exemplo de resolução anterior poderia ser uma transformação da declaração do cálculo de predicados:

$a \leftarrow b \wedge \neg c \wedge d$

Isso pode ser compreendido como “para verificar se  $a$  é verdadeiro, verifique se  $b$  é verdadeiro e  $c$  é falso e  $d$  é verdadeiro”. A regra foi concebida como um *procedimento* para solucionar  $a$  e implementa informação heurística específica para esse uso. De fato, o subobjetivo  $b$  poderia oferecer o caminho mais fácil para tornar todo o predicado falso, de modo que a ordem “tente  $b$ , depois veja se  $c$  é falso e, então, teste  $d$ ” poderia economizar muito tempo de solução do problema. A heurística implícita diz “teste primeiro o caminho mais fácil para tornar o problema falso, então, se isso já foi feito, vá em frente e gere a parte restante (talvez bem mais difícil) da solução”. Os especialistas humanos projetam procedimentos e relações que não apenas são verdadeiros, mas que também contêm informações críticas para o *uso* dessa verdade. Nas situações mais interessantes de solução de problemas, não podemos prescindir dessas heurísticas (Kowalski, 1979b).

Na próxima seção, introduzimos as cláusulas de Horn e usamos a sua interpretação procedural como uma estratégia explícita que preserva essa informação heurística.

### 14.3.2 Programação em lógica e Prolog

Para entender as fundamentações matemáticas do Prolog, primeiro definimos a *programação em lógica*. Uma vez que tenhamos feito essa definição, acrescentaremos uma estratégia de busca explícita para programação em lógica para implementarmos a estratégia de busca, algumas vezes denominada *semântica procedural*, do Prolog. Para obtermos o Prolog completo, discutimos, também, o uso da *negação* e da *suposição de mundo fechado*.

Considere a base de dados de cláusulas preparada para resolução por refutação, como vimos na Seção 14.2. Se restringirmos esse conjunto a cláusulas que tenham, no máximo, um literal positivo (zero ou mais literais negativos), temos um espaço de cláusulas com algumas propriedades interessantes. Primeiro, os problemas que podem ser descritos por esse conjunto de cláusulas permanecem insatisfazíveis para resoluções por refutação, ou são completos para a refutação (Seção 14.2). Segundo, um importante benefício em restringir a nossa representação a essa subclasse de todas as cláusulas é uma estratégia de busca muito eficiente para refutações: uma redução de forma de entrada linear, baseada em preferência por unidade, da esquerda para a direita e em profundidade. Com uma recursão bem fundamentada (chamadas recursivas que têm um término) e verificação de ocorrências, essa

estratégia garante encontrar refutações se o espaço de cláusula for insatisfazível (van Emden e Kowalski, 1976). Uma cláusula de Horn contém, no máximo, um literal positivo, o que significa que ela é da forma:

$$a \vee \neg b_1 \vee \neg b_2 \vee \dots \vee \neg b_n$$

onde  $a$  e todos os  $b_i$ s são literais positivos. Para enfatizar o papel chave do único literal positivo na resolução, geralmente escrevemos as cláusulas de Horn como implicações com o literal positivo como a conclusão:

$$a \leftarrow b_1 \wedge b_2 \wedge \dots \wedge b_n$$

Antes de continuarmos a discussão da estratégia de busca, definimos formalmente esse subconjunto de cláusulas, denominadas *cláusulas de Horn*. Essas cláusulas, junto a uma estratégia *não determinística* de redução de objetivo, constituem o que denominamos *programa em lógica*.

### Definição

## PROGRAMA EM LÓGICA

Um *programa em lógica* é um conjunto de expressões quantificadas universalmente em cálculo de predicados de primeira ordem, da forma:

$$a \leftarrow b_1 \wedge b_2 \wedge b_3 \wedge \dots \wedge b_n$$

onde  $a$  e os  $b_i$ s são literais positivos, algumas vezes denominados objetivos atômicos. O  $a$  é a *cabeça* da cláusula; a conjunção dos  $b_i$ , o *corpo*.

Essas expressões são as *cláusulas de Horn* do cálculo de predicados de primeira ordem. Elas aparecem em três formas: primeiro, quando a cláusula original não tem literais positivos; segundo, quando ela não tem literais negativos; e terceiro, quando ela tem um literal positivo e um ou mais literais negativos. Esses são os casos 1, 2 e 3, respectivamente:

1.  $\leftarrow b_1 \wedge b_2 \wedge \dots \wedge b_n$   
denominada cláusula *sem cabeça* ou *objetivos* a serem tentados:  $b_1$  e  $b_2$  e ... e  $b_n$ .
2.  $a_1 \leftarrow$   
 $a_2 \leftarrow$   
...  
 $a_n \leftarrow$   
denominados *fatos*.
3.  $a \leftarrow b_1 \wedge \dots \wedge b_n$   
denominada uma relação de *regra*.

O cálculo de cláusulas de Horn permite apenas as formas apresentadas anteriormente; pode haver apenas um literal à esquerda de  $\leftarrow$ , e esse literal deve ser positivo. Todos os literais à direita de  $\leftarrow$  são também positivos.

A redução de cláusulas que têm, no máximo, um literal positivo para a forma de Horn requer três passos. Primeiro, selecione o literal positivo da cláusula, se houver um literal positivo, e mova-o para o extremo esquerdo (usando a propriedade comutativa de  $\vee$ ). Esse único literal positivo se torna a *cabeça* da cláusula de Horn, como definido anteriormente. Segundo, transforme toda a cláusula para a forma de Horn pela regra:

$$a \vee \neg b_1 \vee \neg b_2 \vee \dots \vee \neg b_n \equiv a \leftarrow \neg (\neg b_1 \vee \neg b_2 \vee \dots \vee \neg b_n)$$

Finalmente, use a lei de de Morgan para transformar essa especificação em:

$$a \leftarrow b_1 \wedge b_2 \wedge \dots \wedge b_n$$

onde pode ser usada a propriedade comutativa de  $\wedge$  para ordenar os subobjetivos  $b_i$ .

Deve-se notar que pode não ser possível transformar cláusulas de um espaço de cláusulas arbitrário na forma de Horn. Algumas cláusulas, como  $p \vee q$ , não têm a forma de Horn. Para criar uma cláusula de Horn, só pode haver no máximo um literal positivo na cláusula original. Se esse critério não for satisfeito, pode ser necessário repensar a especificação original do problema no cálculo de predicados. A vantagem para a representação na forma de Horn é uma estratégia de refutação eficiente, como veremos em breve.

O algoritmo de computação para programas em lógica funciona por redução não determinista de objetivos. A cada passo da computação em que exista um objetivo da forma:

$$\leftarrow a_1 \wedge a_2 \wedge \dots \wedge a_n$$

o interpretador escolhe *arbitrariamente* um  $a_i$  para  $1 \leq i \leq n$ . Então, ele escolhe uma cláusula de modo *não determinista*:

$$a^1 \leftarrow b_1 \wedge b_2 \wedge \dots \wedge b_n$$

tal que o  $a^1$  se unifique com  $a_i$  pela substituição  $\xi$  e use essa cláusula para reduzir o objetivo. O novo objetivo se torna, então:

$$\leftarrow (a_1 \wedge \dots \wedge a_{i-1} \wedge b_1 \wedge b_2 \wedge \dots \wedge b_n \wedge a_{i+1} \wedge \dots \wedge a_n) \xi$$

Esse processo de redução não determinista de objetivos continua até que a computação termine com o conjunto de objetivos vazio.

Se eliminarmos o não determinismo impondo uma ordem para a redução de subobjetivos, não mudaremos o resultado da computação. Todos os resultados que podem ser encontrados de maneira não determinista podem ser encontrados, também, por uma busca exaustiva ordenada. Entretanto, reduzindo a quantidade de não determinismo, podemos definir estratégias que podam os ramos desnecessários do espaço. Assim, uma das maiores preocupações das linguagens de programação em lógica práticas é fornecer ao programador facilidades para controlar e, quando possível, reduzir a quantidade de não determinismo. Essas facilidades permitem que o programador influencie não só a ordem pela qual os objetivos são reduzidos, mas também o conjunto de cláusulas que são usadas para reduzir cada objetivo. (Como em qualquer busca em grafo, devem ser tomadas precauções para evitar ciclos infinitos na prova.)

A especificação abstrata de um programa em lógica tem uma semântica clara, aquela do sistema de resolução por refutação. van Emden e Kowalski (1976) mostram que a menor interpretação pela qual um programa em lógica é verdadeiro é a interpretação do programa. O preço pago pelas linguagens de programação práticas, como Prolog, é que a execução de programas nessas linguagens pode computar apenas um subconjunto de suas interpretações associadas (Shapiro, 1987).

O Prolog sequencial é uma aproximação de um interpretador para o modelo de programação em lógica, projetado para ser executado eficientemente em computadores von Neumann. Esse é o interpretador que usamos até agora neste livro. O Prolog sequencial usa tanto a ordem de objetivos em uma cláusula como a ordem de cláusulas no programa para controlar a busca por uma prova. Quando vários objetivos estiverem disponíveis, o Prolog sempre os persegue da esquerda para a direita. Na busca por uma cláusula unificável para um objetivo, as cláusulas possíveis são verificadas na ordem em que elas foram apresentadas pelo programador. Quando é feita uma seleção, um apontador para retrocesso é colocado junto com a unificação registrada, o que permite que outras cláusulas sejam usadas (de novo, na ordem do programador), caso falhe a seleção original de uma cláusula unificável. Se essas tentativas falharem para todas as cláusulas possíveis do espaço de cláusulas, então a computação falha. Com o corte, que é uma tentativa de usar eficientemente a busca em profundidade com retrocesso, o interpretador pode não visitar todas as combinações (interpretações) de cláusulas do espaço de busca.

Mais formalmente, dado um objetivo:

$$\leftarrow a_1 \wedge a_2 \wedge a_3 \dots \wedge a_n$$

e um programa  $P$ , o interpretador Prolog busca, sequencialmente, a primeira cláusula em  $P$  cuja cabeça se unifica com  $a_1$ . Essa cláusula é, então, usada para reduzir os objetivos. Se:

$$a^1 \leftarrow b_1 \wedge b_2 \wedge \dots \wedge b_n$$

for a cláusula a ser reduzida com a unificação  $\xi$ , a cláusula objetivo se torna então:

$\leftarrow (b_1 \wedge b_2 \wedge \dots \wedge b_n \wedge a_2 \wedge a_3 \wedge \dots \wedge a_n) \xi$

O interpretador Prolog continua tentando reduzir o objetivo mais à esquerda,  $b_1$  nesse exemplo, usando a primeira cláusula no programa P que se unifica com  $b_1$ . Suponha que ela seja:

$b^1 \leftarrow c_1 \wedge c_2 \wedge \dots \wedge c_p$

sob a unificação  $\phi$ . O objetivo se torna, então:

$\leftarrow (c_1 \wedge c_2 \wedge \dots \wedge c_p \wedge b_2 \wedge \dots \wedge b_n \wedge a_2 \wedge a_3 \wedge \dots \wedge a_n) \xi \phi$

Note que a lista de objetivos é tratada como uma pilha forçando a busca em profundidade. Se eventualmente o interpretador Prolog não conseguir encontrar uma unificação que resolva um objetivo, ele então retrocede para o seu ponto de escolha de unificação mais recente, restaura todas as ligações feitas desde aquele ponto de escolha e escolhe a próxima cláusula que será unificada (pela ordem em P). Desse modo, o Prolog implementa a sua busca do espaço de cláusula em profundidade, da esquerda para a direita.

Se o objetivo for reduzido à cláusula nula ( $\square$ ), então a composição de unificações que fizeram as reduções:

$\leftarrow (\square) \xi \phi \dots \omega$

(aqui,  $\xi \phi \dots \omega$ ) fornece uma interpretação para a qual a cláusula objetivo original era verdadeira.

Além do retrocesso, segundo a ordem das cláusulas em um programa, o Prolog sequencial permite o corte ou “!””. Um corte pode ser colocado em uma cláusula como um objetivo. Quando encontra um corte, o interpretador fica comprometido com o caminho de execução atual e, em particular, com aquele subconjunto de unificações feitas desde a escolha da cláusula que contém o corte. Ele também compromete o interpretador com a escolha daquela cláusula como o único método para reduzir o objetivo. Caso seja encontrada uma falha dentro da cláusula após o corte, a cláusula inteira falha.

De um modo procedural, o corte torna desnecessária a retenção de apontadores de retrocesso para a cláusula que é reduzida e todos os seus componentes *antes* do corte. Assim, o corte pode significar que apenas algumas das possíveis interpretações do modelo sejam computadas.

Resumimos a nossa discussão do Prolog sequencial comparando-o ao modelo de resolução por refutação da Seção 14.2.

1. O espaço de cláusulas para resolução é um superconjunto de expressões em cláusula de Horn na programação em lógica. Cada cláusula deve ter, no máximo, um literal positivo para estar na forma de Horn.
2. As seguintes estruturas representam o problema na forma de Horn:
  - a. Os objetivos,

$\leftarrow b_1 \wedge b_2 \wedge \dots \wedge b_n,$

formam uma lista de declarações por cláusulas que constituem os objetivos a serem testados pela resolução por refutação. Por sua vez, cada  $a_i$  é negado, unificado e reduzido até que a cláusula vazia seja encontrada (se for possível).

- b. Os fatos,

$a_1 \leftarrow$

$a_2 \leftarrow$

.

.

.

$a_n \leftarrow,$

são cláusulas separadas para fins de resolução. Finalmente,

- c. As regras em cláusulas de Horn, ou axiomas,

$a \leftarrow b_1 \wedge b_2 \wedge \dots \wedge b_n,$

nos permitem reduzir os subobjetivos que são casados.

3. Com uma estratégia de *forma de entrada linear* e preferência por unidade (sempre preferindo cláusulas de fatos e usando o objetivo negado e seus resolventes descendentes; ver Seção 14.2.4) e aplicando a ordem da esquerda para a direita em profundidade (com retrocesso) para selecionar cláusulas para resolução, o provador de teoremas por resolução age como um interpretador Prolog. Como essa estratégia é completa para a refutação, o seu uso garante que a solução será encontrada (desde que a parte do conjunto de interpretações não seja podada pelo uso do corte).
4. Finalmente, a composição de unificações na prova fornece a resposta (interpretação) para a qual o objetivo é verdadeiro. Isso é exatamente equivalente ao processo de extração de respostas da Seção 14.2.5. O registro da composição de unificações no literal objetivo produz a interpretação da resposta.

Uma questão importante do Prolog é que queremos usar literais negativos no lado direito das regras da cláusula de Horn. Isso exige que os interpretadores imponham a *suposição de mundo fechado*. No cálculo de predicados, a prova de  $\neg p(X)$  é exatamente a prova que  $p(X)$  seja logicamente falso. Isto é,  $p(X)$  é falso sob todas as interpretações que tornam o conjunto de axiomas verdadeiro. O interpretador Prolog, baseado no algoritmo de unificação do Capítulo 2, oferece um resultado mais restrito que a resolução por refutação geral da Seção 14.2. Em vez de tentar todas as interpretações, ele examina apenas aquelas explícitas na base de dados. Agora, axiomatizamos essas restrições para ver exatamente as restrições implícitas nos ambientes atuais de Prolog.

Para todo predicado  $p$  e toda variável  $X$  pertencente a  $p$ , suponha que  $a_1, a_2, \dots, a_n$  constituam o domínio de  $X$ . O interpretador Prolog, usando unificação, impõe:

1. O axioma do *nome único*. Para todos os átomos do domínio  $a_i \neq a_j$ , a menos que eles sejam idênticos. Isso implica que átomos com nomes distintos sejam distintos.
2. O axioma de *mundo fechado*.

$$p(X) \rightarrow p(a_1) \vee p(a_2) \vee \dots \vee p(a_n).$$

Isso significa que os únicos casos possíveis de uma relação são aqueles implicados pelas cláusulas presentes na especificação do problema.

3. O axioma do *fechamento do domínio*.

$$(X = a_1) \vee (X = a_2) \vee \dots \vee (X = a_n).$$

Isso garante que os átomos que ocorrem na especificação do problema constituam todos os únicos átomos.

Esses três axiomas estão implícitos na ação do interpretador Prolog. Eles podem ser vistos como algo que é acrescentado ao conjunto de cláusulas de Horn e que constitui uma descrição de problema, com isso restringindo o conjunto de interpretações possíveis a uma consulta Prolog.

Intuitivamente, isso significa que o Prolog supõe como falsos todos os objetivos que ele não consegue provar como verdadeiros. Isso pode introduzir anomalias: se um valor verdade de um objetivo é desconhecido em relação à base de dados atual, o Prolog suporá que é falso.

Outras limitações estão implícitas em Prolog, assim como estão implícitas em todas linguagens de programação. As mais importantes delas, além do problema da negação como falha, representam violações do modelo semântico para a programação em lógica. Em particular, há uma falta de verificação de ocorrências prévias (ver Seção 2.3; isso permite que uma cláusula seja unificada com um subconjunto de si mesma) e o uso do corte. A atual geração de interpretadores Prolog deve ser vista pragmaticamente. Alguns problemas surgem porque “atualmente não se conhece uma maneira” de contornar a questão (da verificação de ocorrências); outros surgem das tentativas para otimizar o uso da busca em profundidade com retrocesso (o corte). Muitas das anomalias do Prolog são resultado da tentativa de implementar, em um computador sequencial, a semântica não determinista da programação em lógica pura. Isso inclui os problemas introduzidos pelo corte.

Na seção final do Capítulo 14, introduzimos esquemas de inferência alternativos para o raciocínio automatizado.

## 14.4 Outras questões em raciocínio automatizado

Descrevemos os resolvidores de problemas de método fraco como aqueles que usam (a) um *meio de representação uniforme* para (b) *regras de inferência consistentes* que focam nas características sintáticas da representação e são guiados por (c) *métodos ou estratégias* para combater a explosão combinatória da busca exaustiva. Concluímos este capítulo com alguns comentários sobre cada um desses aspectos do processo de solução por método fraco.

### 14.4.1 Representações uniformes para soluções por método fraco

O procedimento de prova por resolução requer que coloquemos todos os nossos axiomas na forma de cláusulas. Essa representação uniforme nos permite, então, resolver cláusulas e simplifica o projeto de heurísticas de solução de problemas. Uma grande desvantagem dessa abordagem é que muita informação heurística valiosa pode ser perdida nessa codificação uniforme.

O formato se ... então de uma regra carrega mais informação para o uso de *modus ponens* ou busca em sistemas de produção do que uma de suas variantes sintáticas. Ele também nos oferece um modo eficiente de usar a regra. Por exemplo, suponha que queiramos representar a inferência abdutiva, que vimos na Seção 8.0, Se o motor não pega e as luzes não acendem, então a bateria pode estar descarregada. Essa regra sugere como se deve testar a bateria.

A forma disjuntiva da mesma regra obscurece essa informação heurística sobre como a regra deve ser aplicada. Se expressarmos essa regra no cálculo de predicados — pega  $\wedge \neg$  luz  $\rightarrow$  bateria, a forma clausular dessa regra é: pega  $\vee$  luz  $\vee$  bateria. Essa cláusula pode ter várias formas equivalentes, que podem, cada uma, representar uma implicação diferente:

$$\begin{aligned} & (\neg \text{pega} \wedge \neg \text{luz}) \rightarrow \text{bateria} \\ & (\neg \text{pega} \rightarrow (\text{bateria} \vee \text{luz})) \\ & (\neg \text{bateria} \wedge \neg \text{luz}) \rightarrow \text{pega} \\ & (\neg \text{bateria} \rightarrow (\text{pega} \vee \text{luz})) \end{aligned}$$

e assim por diante.

Para reter a informação heurística no processo de raciocínio automatizado, vários pesquisadores, inclusive Nilsson (1980) e Bundy (1988), defendem métodos de raciocínio que codificam heurísticas, formando regras de acordo com o modo como o especialista humano projetou os relacionamentos entre as regras. Já propusemos essa abordagem no nosso raciocínio por grafo e/ou na Seção 3.3 e na forma Prolog de raciocínio automatizado da Seção 14.3. Os sistemas especialistas baseados em regras também permitem que o programador controle a busca por meio da estrutura de regras. Desenvolvemos mais essa ideia nos próximos dois exemplos, um guiado por dados e o outro, por objetivo. Ambos retêm a forma de implicações e usam essa informação para guiar a busca por meio de um grafo e/ou.

Considere, para uso em um raciocínio guiado por dados, os seguintes fatos, regras (axiomas) e objetivo:

Fato:

$$(a \vee (b \wedge c))$$

Regras (ou axiomas):

$$\begin{aligned} & (a \rightarrow (d \wedge e)) \\ & (b \rightarrow f) \\ & (c \rightarrow (g \vee h)) \end{aligned}$$

Objetivo:

$$\neg e \vee f$$

A prova de  $e \vee f$  está no grafo e/ou da Figura 14.13. Note o uso dos conectores e nas relações  $\vee$  e os conectores ou nas relações  $\wedge$  no espaço de busca guiado por dados. Se nos for dado que  $a$  ou  $b \wedge c$  é verdadeiro, então devemos raciocinar com os dois termos disjuntivos para garantir que o nosso argumento preserve a verdade; assim, esses dois caminhos são unidos pela conjunção. Por outro lado, quando  $b$  e  $c$  são verdadeiros, podemos continuar a explorar qualquer um desses termos conjuntivos. O casamento de regras toma qualquer estado intermediário, por exemplo  $o$  e  $c$ , e o substitui pela conclusão da regra, por exemplo  $(g \vee h)$ , cuja premissa casa com esse estado. A descoberta dos dois estados,  $e$  e  $f$  na Figura 14.13, indica que o objetivo ( $e \vee f$ ) foi estabelecido.

De modo similar, podemos usar o casamento de regras em grafos e/ou para o raciocínio guiado por objetivo. Quando uma descrição de objetivo inclui um  $\vee$ , como no exemplo da Figura 14.14, então qualquer uma das alternativas pode ser explorada independentemente para estabelecer o objetivo. Se o objetivo é uma conjunção, então, evidentemente, ambos os termos conjuntivos devem ser estabelecidos.

### Objetivo:

$$(a \times (b \wedge c))$$

Regras (ou axiomas):

$$(f \wedge d) \rightarrow a$$

$$(e \rightarrow (B \wedge C))$$

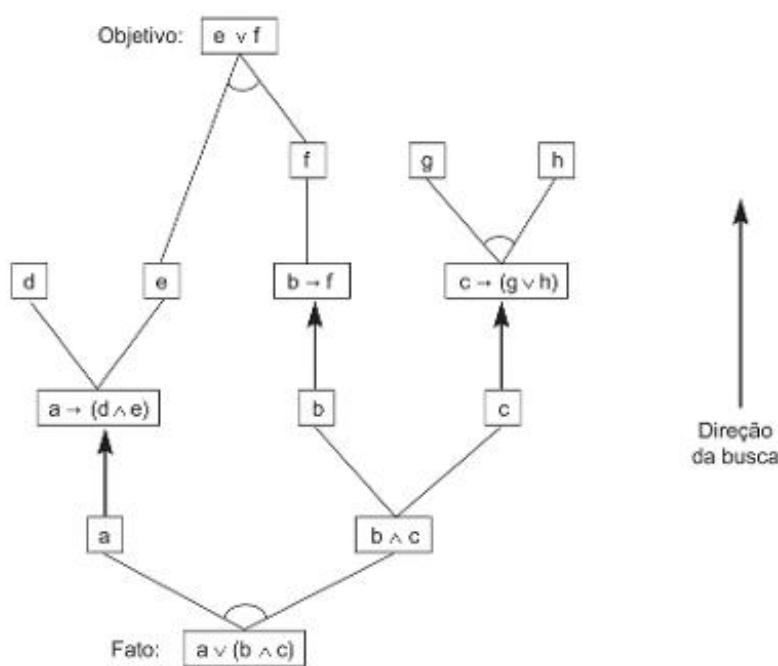
(a = d)

Fato\*

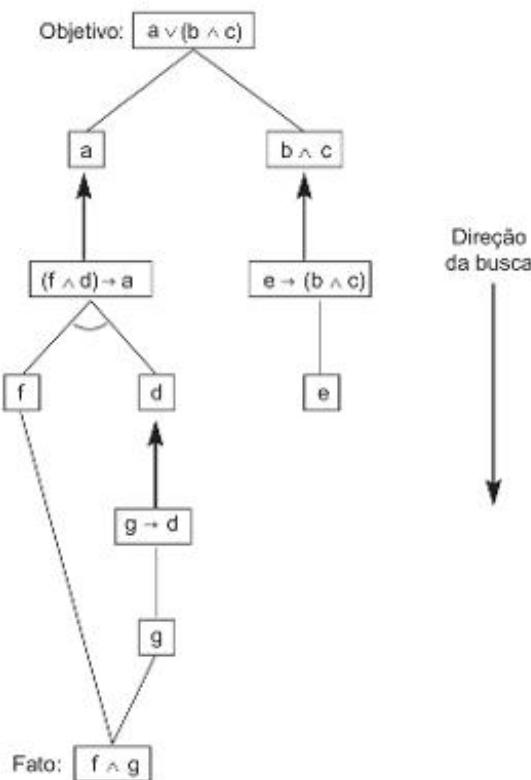
148

Embora esses exemplos sejam tirados do cálculo proposicional, uma busca similar é gerada usando fatos e regras do cálculo de predicados. A unificação torna os literais compatíveis para a aplicação das regras de inferência por meio de ramos diferentes do espaço de busca. É claro que as unificações devem ser consistentes (isto é, unificáveis) a partir de diferentes ramos e do espaço de busca.

**Figura 14.13** Raciocínio guiado por dados com um grafo e/ou no cálculo proposicional.



**Figura 14.14** Raciocínio guiado por objetivo com um grafo E/OU no cálculo proposicional.



Essa subseção sugeriu métodos de solução para ajudar a preservar a informação heurística contida no meio de representação para a solução de problemas por método fraco. Essa é essencialmente a maneira como o motor de inferência do sistema especialista permite que o programador especifique informação heurística e de controle em uma regra. Os sistemas especialistas se baseiam na forma de regras, como a ordenação de regras ou a ordenação das premissas de uma regra, para controlar a busca, em vez de depender totalmente de métodos fracos de solução de problemas. O que se perde nessa abordagem é a capacidade de aplicar procedimentos uniformes de busca, como a resolução, por meio do conjunto completo de regras. Entretanto, como pode ser notado nos exemplos das figuras 14.13 e 14.14, o *modus ponens* ainda pode ser usado. O controle de sistemas de produção usando busca em amplitude, em profundidade ou pela melhor escolha oferece uma arquitetura de raciocínio por método fraco para implementar sistemas de regras (ver exemplos nos capítulos 4, 6 e 8).

#### 14.4.2 Regras de inferência alternativas

A resolução é a regra de inferência consistente mais geral que apresentamos até agora. Várias regras de inferência mais sofisticadas foram criadas em uma tentativa de tornar a resolução mais eficiente. Consideraremos brevemente duas delas: a *hiper-resolução* e a *paramodulação*.

A resolução, como a apresentamos, é, na realidade, uma variante especial denominada *resolução binária*: exatamente duas cláusulas pais são colididas. Uma aplicação bem-sucedida de hiper-resolução substitui uma sequência de resoluções binárias para produzir uma cláusula. A hiper-resolução colide, em um único passo, uma cláusula com alguns literais negativos, denominados *núcleo*, e certo número de cláusulas com literais todos positivos, denominados *satélites*. Esses satélites devem ter um literal positivo que casará com um literal negativo do

núcleo. Deve haver, também, um satélite para cada literal negativo do núcleo. Assim, o resultado de uma aplicação de hiper-resolução é uma cláusula com todos os literais positivos.

Uma vantagem da hiper-resolução é que uma cláusula de literais positivos é produzida em cada inferência por hiper-resolução, e o próprio espaço de cláusulas se mantém mais reduzido porque não são produzidos resultados intermediários. As unificações sobre todas as cláusulas no passo de inferência devem ser consistentes.

Como exemplo de hiper-resolução, considere o seguinte conjunto de cláusulas:

```
¬ casado(X,Y) ∨ ¬ mãe(X,Z) ∨ pai(Y,Z)
casado(kate, george) ∨ gosta(george, kate)
mãe(kate, sarah)
```

Encontramos uma conclusão em um passo usando hiper-resolução:

```
pai(george, sarah) ∨ gosta(george, kate)
```

A primeira cláusula no exemplo é o núcleo; a segunda são dois satélites. Os satélites são todos positivos e há um para cada literal negativo do núcleo. Note como o núcleo é apenas a forma clausular da implicação:

```
casado(X,Y) ∧ mãe(X,Z) → pai(Y,Z)
```

A conclusão dessa regra é parte do resultado final. Note que não há resultados intermediários, tal como:

```
¬ mãe(kate,Z) ∨ pai(george,Z) ∨ gosta(george,kate)
```

que seria encontrada em uma prova por resolução binária aplicada ao mesmo espaço de cláusulas.

A hiper-resolução é consistente e completa quando usada sozinha. Quando combinada com outras estratégias, como o conjunto de suporte, a completude pode ser comprometida (Wos et al., 1984). São necessárias estratégias de busca especiais para organizar as cláusulas núcleo e satélites, embora na maioria dos ambientes onde a hiper-resolução é usada, as cláusulas sejam normalmente indexadas pelo nome e pelas propriedades positiva ou negativa de cada literal. Isso faz com que seja eficiente preparar as cláusulas núcleo e satélites para a inferência por hiper-resolução.

Uma questão importante e difícil no projeto de mecanismos de prova de teoremas é o controle da igualdade. Áreas como a matemática são especialmente complexas, pois a maioria dos fatos e relacionamentos têm múltiplas representações, como resultado da aplicação das propriedades associativa e comutativa. Para ver isso com um exemplo muito simples, considere as múltiplas maneiras que a expressão  $3 + (4 + 5)$  pode ser representada, incluindo  $3 + ((4 + 0) + 5)$ . Essa é uma questão complexa, pois as expressões precisam ser substituídas, unificadas e verificadas por igualdade com outras expressões durante a solução automatizada do problema matemático.

A *demodulação* é o processo de reescrever expressões de modo que elas tomem automaticamente uma forma canônica escolhida. As cláusulas unitárias usadas para produzir essa forma canônica são os *demoduladores*, que especificam a igualdade de expressões diferentes, permitindo a substituição de uma expressão por sua forma canônica. Com o uso adequado da demodulação, toda informação nova produzida é reduzida a uma forma especificada antes de ser colocada no espaço de cláusulas. Por exemplo, poderíamos ter o demodulador:

```
igual(pai(pai(X)), avô(X))
```

e a nova cláusula:

```
idade(pai(pai(sarah)), 86).
```

Antes de acrescentar essa nova cláusula ao espaço de cláusula, aplicamos o demodulador e, em vez de acrescentar a cláusula original, acrescentamos:

```
idade(avô(sarah), 86).
```

Aqui, o problema da igualdade é o da troca de nome. Desejamos classificar uma pessoa como  $pai(pai(X))$  ou  $avô(X)$ ? De modo similar, podemos encontrar nomes canônicos para todas as relações familiares: um  $irmão(pai(Y))$  é o  $tio(Y)$  etc. Uma vez que escolhemos os nomes canônicos para armazenar informação, projetamos demodulador

res, como a cláusula igual, para reduzir toda informação nova a essa forma determinada. Note que os demoduladores são sempre cláusulas unitárias.

A *paramodulação* é uma generalização da substituição por igualdade ao nível do termo. Por exemplo, dada a expressão:

mais\_velha(mãe(Y), Y)

e a relação de igualdade:

igual(mãe(sarah), kate)

podemos concluir pela paramodulação que:

mais\_velha(kate, sarah)

Note o casamento e a substituição ao nível de termo de {sarah/Y} e mãe(sarah) por kate. Uma diferença vital entre demodulação e paramodulação é que a última permite uma substituição não trivial de variáveis nos dois argumentos, do predicado de igualdade e do predicado no qual a substituição é realizada. A demodulação realiza as substituições com base no demodulador. Para produzir uma expressão na sua forma final, é possível usar demoduladores múltiplos; a paramodulação é usada apenas uma vez em qualquer situação.

Apresentamos exemplos simples desses poderosos mecanismos de inferência. Eles devem ser vistos como técnicas gerais para serem usadas no espaço de cláusulas da resolução. Como todas as outras regras de inferência que vimos, essas técnicas estão fortemente ligadas à representação escolhida e devem ser controladas por estratégias apropriadas.

#### 14.4.3 Resposta a perguntas oferecida pela resolução por refutação

Na Seção 14.2.5, demonstramos um ajuste natural entre o processo de resolução por refutação e a geração de respostas para a consulta original (o teorema a ser provado). Stuart Shapiro e seus colegas (Burhans e Shapiro, 2005) demonstram como as resoluções por refutação também oferecem um paradigma útil para a resposta a perguntas.

Burhans e Shapiro observaram que o processo de refutação divide as cláusulas geradas durante as resoluções por refutação em três classes, cada uma apropriada para responder a diferentes tipos de questões sobre o teorema em consideração. Em suas pesquisas, as cláusulas que são relevantes estão identificadas como possíveis respostas, onde a relevância é determinada de acordo com uma questão (o teorema a ser provado) e uma base de conhecimento (o conjunto de cláusula usado na prova), onde qualquer cláusula descendente da forma de cláusula do objetivo negado é considerada relevante.

A prova por refutação é dividida em três classes de resposta: *específica*, *genérica* e *hipotética*. Essas classes são distinguidas pelo modo como os literais compõem as variáveis de compartilhamento de cláusula. O resultado independe do contexto, ou seja, ele pode ser usado com qualquer refutação baseada em conjunto de suporte, pragmática para resposta a perguntas.

Respostas específicas, às vezes denominadas respostas *extencionais* (Green, 1969a, b), são associadas ao conjunto de termos básicos correspondentes às constantes substituídas por valores variáveis no processo de produção de uma refutação. Por exemplo, a consulta “tem alguém em casa?” seria substituída pelo objetivo negado “não há ninguém em casa” e levaria a uma refutação quando fosse mostrado que “Amy está em casa”. Esse conjunto de indivíduos que pode ser produzido por meio do processo de refutação provê as respostas específicas.

Respostas genéricas, também denominadas respostas *intencionais* (Green, 1969a, b), correspondem a todas as instâncias não básicas de responder a cláusulas que podem ser associadas com a refutação. Por exemplo, para a consulta “tem alguém em casa?”, quaisquer cláusulas baseadas em variável, como “todas as crianças estão em casa”, compõem o conjunto de respostas genéricas, desde que essas cláusulas estejam contidas no espaço de cláusulas ou sejam dedutíveis dele.

Por fim, respostas hipotéticas, às vezes denominadas *condicionais* ou *qualificadas*, tomam a forma de uma regra cujo antecedente tem algumas variáveis básicas e cujo consequente casa com a consulta da resposta negada.

Por exemplo, “tem alguém em casa?” poderia casar com a cláusula “se Allen está em casa, então Amy está em casa”, com a busca subsequente para determinar se Allen está em casa. Para obter mais detalhes, consulte Burhans e Shapiro (2005).

#### 14.4.4 Estratégias de busca e seu uso

Algumas vezes, o domínio de aplicação coloca demandas especiais às regras de inferência e às heurísticas para guiar o seu uso. Já vimos o uso de demoduladores para ajudar na substituição por igualdade. Bledsoe, no seu *sistema de dedução natural*, identifica duas estratégias importantes para preparar teoremas para prova por resolução. Ele chama essas estratégias de *dividir* e *reduzir* (Bledsoe, 1971).

Bledsoe concebeu as suas estratégias para serem usadas em matemática e, em particular, para serem aplicadas à *teoria dos conjuntos*. O efeito dessas estratégias é quebrar um teorema em partes para facilitar a prova por métodos convencionais como a resolução. A estratégia de dividir toma várias formas matemáticas e as divide em partes apropriadas. A prova de  $A \wedge B$  é equivalente à prova de  $A$  e à prova de  $B$ . Da mesma forma, a prova de  $A \leftrightarrow B$  é a prova de  $A \rightarrow B$  e a prova de  $A \leftarrow B$ .

A heurística de reduzir tenta, também, quebrar provas grandes em seus componentes. Por exemplo, a prova de  $s \in A \cap B$  pode ser decomposta nas provas de  $s \in A$  e  $s \in B$ . Outro exemplo poderia ser provar a verdade de uma propriedade de  $\neg(A \cup B)$  provando a propriedade para  $\neg A$  e para  $\neg B$ . Com a quebra de provas maiores em pedaços menores, Bledsoe espera conter o espaço de busca. As suas heurísticas incluem, também, um uso limitado de substituição por igualdade.

Como mencionado ao longo de todo este livro, o uso apropriado de heurísticas é como uma arte, que leva em consideração a área de aplicação, bem como a representação e as regras de inferência utilizadas. Encerramos este capítulo citando alguns provérbios gerais, que algumas vezes são falsos, mas que, com uso cuidadoso, podem ser muito eficazes. Esses provérbios resumem os pensamentos dos pesquisadores da área (Bledsoe, 1971; Nilsson, 1980; Wos et al., 1984; Dallier, 1986; Wos, 1988), bem como as nossas próprias reflexões sobre resolvidores de problemas por método fraco. Eles são expressados sem outros comentários.

Use, sempre que possível, cláusulas com o menor número de literais.

Quebre a tarefa em subtarefas antes de empregar mecanismos gerais de inferência.

Use predicados de igualdade sempre que isso for apropriado.

Use demoduladores para criar formas canônicas.

Use paramodulação quando realizar inferência com predicados de igualdade.

Use estratégias que mantenham a completude.

Use estratégias de conjunto de suporte, por estas conterem a potencial contradição.

Use unidades durante a resolução, pois estas encurtam a cláusula resultante.

Realize verificações de subsunção em cláusulas novas.

Use um mecanismo de ordenação sobre cláusulas e literais dentro de cláusulas que reflitam as suas intuições e a sua experiência na solução do problema.

### 14.5 Epílogo e referências

Os resolvidores por método fraco requerem a escolha de um meio representacional, de mecanismos de inferência e de estratégias de busca. Essas três escolhas estão entrelaçadas de modo intrincado e não podem ser reali-

zadas isoladamente. O domínio de aplicação também afeta a escolha da representação, das regras de inferência e das estratégias. Os “provérbios” no final da última seção devem ser considerados ao se fazer essas escolhas.

A resolução é o processo de restringir interpretações possíveis até que se observe que o espaço de cláusulas, com a inclusão do objetivo negado, é inconsistente. Este livro não entra na discussão da consistência da resolução ou da completude das resoluções por refutação. As provas são baseadas no teorema de Herbrand (Chang e Lee, 1973) e na noção de interpretações possíveis do conjunto de cláusulas. O leitor interessado é encorajado a procurar as referências para essas provas.

Várias outras referências são apropriadas: *Automated Theorem Proving: A Logical Basis* oferece uma abordagem formal (Loveland, 1978). Vários artigos clássicos da área foram coletados na série *The Automation of Reasoning: Collected Papers, 1957 to 1970* (Siekmann e Wrightson, 1983a, b). Nilsson (1980), Weyhrauch (1980), Genesereth e Nilsson (1987), Kowalski (1979b), Lloyd (1984), Wos et al. (1984) e Wos (1988) oferecem resumos valiosos de conceitos importantes em raciocínio automatizado. Robinson (1965) e Bledsoe (1977) realizaram contribuições fundamentais para a área. Uma contribuição importante foi feita por Boyer e Moore (1979). O trabalho inicial em prova de teoremas de Newell e Simon e seus colegas está relatado em *Computers and Thought* (Feigenbaum e Feldman, 1963) e em *Human Problem Solving* (Newell e Simon, 1972). Respostas a questões baseadas em refutações são descritas por Burhans e Shapiro (2005).

A CADE (do inglês *Conference on Automated Deduction*) é o fórum principal para apresentação de novos resultados em raciocínio automatizado. Verificação de modelo, sistemas de verificação e sistemas escaláveis de representação de conhecimento são questões de pesquisa atuais (McAllester, 1999; Ganzinger et al., 1999; Veroff e Spinks, 2006). Importantes pesquisas continuam sendo feitas por Wos e seus colegas no Argonne National Laboratory. Veroff (1997) publicou um conjunto de ensaios sobre raciocínio automatizado em homenagem a Wos. O trabalho de Bundy (1983, 1988) em raciocínio automatizado na University of Edinburgh é também importante. Também continuam as pesquisas para estender o provador de teorema de Boyer-Moore na University of Texas, em Austin; veja *Computer-Aided Reasoning: ACL2 Case Studies*, Kaufmann et al. (2000).

## 14.6 Exercícios

1. Tome o consultor financeiro baseado em lógica da Seção 2.4, coloque os predicados que descrevem o problema na forma de cláusulas e use resoluções por refutação para responder a consultas como se um investidor em particular devesse aplicar em investimento(combinação).
2. Use resolução para provar a declaração de Wirth no Exercício 12 do Capítulo 2.
3. Use resolução para responder a consulta do Exemplo 3.3.4.
4. No Capítulo 6, apresentamos uma forma simplificada do percurso do cavalo. Tome a regra caminho3, coloque-a na forma clausular e use resolução para responder a consultas como caminho3(3,6). A seguir, use a chamada recursiva de caminho, na forma de cláusula, para responder a consultas.
5. Como você poderia usar resolução para implementar uma busca para um “sistema de produção”?
6. Como você usaria o raciocínio guiado por dados com resolução? Use isso para tratar do espaço de busca do Exercício 1. Quais problemas podem surgir em um grande espaço de problema?
7. Use resolução para consultas no problema do fazendeiro, do lobo, da cabra e do repolho, da Seção 3.5.
8. Use resolução para resolver o problema seguinte de Wos et al. (1984). Há quatro pessoas: Roberta, Thelma, Steve e Pete. Os quatro receberam oito tarefas diferentes. Cada pessoa tem exatamente duas tarefas. As tarefas são chefe, guarda, enfermeiro, telefonista, policial, professor, ator e boxeador. O enfermeiro é masculino. O esposo da chefe é o telefonista. Roberta não é uma boxeadora. Pete não passou da oitava série. Roberta, a chefe, e o policial foram jogar golfe juntos. Quais são as tarefas de cada um? Mostre como a adição de uma preferência sexual para cada tarefa pode modificar o problema.
9. Desenvolva dois exemplos para hiper-resolução onde o núcleo tenha, ao menos, quatro literais.

- 10.** Escreva um demodulador para soma que reduziria cláusulas da forma igual(res, soma(5, soma(6, menos(6)))) a igual(res, soma(5, 0)). Especifique outro demodulador para reduzir este último resultado a igual(res, 5).
- 11.** Escreva um “conjunto canônico” para seis relações familiares. Especifique demoduladores para reduzir formas alternativas de relações desse conjunto. Por exemplo, o “irmão da mãe” é o “tio”.
- 12.** Tome o problema do estudante feliz da Figura 14.5 e aplique três das estratégias de refutação da Seção 14.2.4 para a sua solução.
- 13.** Coloque a seguinte expressão do cálculo de predicados na forma clausular:  
$$\forall (X)(p(X) \rightarrow (\forall (Y)[p(Y) \rightarrow p(f(X, Y))]) \wedge \neg \forall (Y)[q(X, Y) \rightarrow p(Y)]})$$
- 14.** Crie o grafo e/ou para a seguinte dedução guiada por dados do cálculo de predicados.  
Fato:  $\neg d(f) \vee [b(f) \wedge c(f)]$ .  
Regras:  $\neg d(X) \rightarrow \neg a(X)$  e  $b(Y) \rightarrow e(Y)$  e  $g(W) \leftarrow c(W)$ .  
Prova:  $\neg a(Z) \vee e(Z)$ .
- 15.** Prove que a estratégia da forma de entrada linear não é completa para a refutação.
- 16.** Crie o grafo e/ou para o seguinte problema: Por que é impossível concluir o objetivo:  
 $r(Z) \vee s(Z)?$   
Fato:  $p(X) \vee q(X)$ .  
Regras:  $p(a) \rightarrow r(a)$  e  $q(b) \rightarrow s(b)$ .
- 17.** Use fatoração e resolução para produzir uma refutação para as seguintes cláusulas:  $p(X) \vee p(f(Y))$  e  $\neg p(W) \vee \neg p(f(Z))$ . Tente produzir uma refutação sem fatoração.
- 18.** Derive uma prova por resolução do teorema da Figura 14.1.
- 19.** Um modelo semântico alternativo para a programação em lógica é aquele do *Prolog Plano Concorrente*. Compare a semântica do Prolog vista na Seção 14.3 com aquela do *Prolog Plano Concorrente* (Shapiro, 1987).

# Compreensão da linguagem natural

*Mas deve-se reconhecer que a noção de "probabilidade de uma sentença" é inteiramente inútil sob qualquer interpretação conhecida desse termo...*

—NOAM CHOMSKY (1965)

*Eu percebo uma fúria nas suas palavras,  
Mas não comprehendo as palavras.*

—WILLIAM SHAKESPEARE, *Othello*

*Eles estiveram numa grande festa de linguagens  
e roubaram as sobras.*

—WILLIAM SHAKESPEARE, *Love's Labour's Lost*

*Eu gostaria que alguém me dissesse o que "Ditty wah ditty" significa.*

—ARTHUR BLAKE

## 15.0 O problema da compreensão da linguagem natural

Comunicar-se por meio de linguagem natural, quer seja como texto ou como um ato de fala, depende enormemente das nossas habilidades na língua, do nosso conhecimento e expectativas dentro do domínio do discurso. A compreensão de linguagem não é meramente a transmissão de palavras: ela também requer inferências sobre objetivo, conhecimento e suposições do locutor, bem como sobre o contexto da interação. A implementação de um programa para compreender linguagem natural requer que representemos conhecimento e expectativas do domínio e raciocinemos efetivamente sobre eles. Precisamos considerar questões como não monotonicidade, revisão de crença, metáfora, planejamento, aprendizado e as complexidades práticas da interação humana. Mas esses são os problemas centrais da própria inteligência artificial!

Considere, por exemplo, as seguintes linhas do *Soneto XVIII* de Shakespeare:

Devo comparar-te a um dia de verão?  
És mais encantadora e mais delicada:  
O vento forte de maio agita a flor em botão,  
E a despedida do verão tem data marcada.

Não podemos compreender essas linhas por meio de um tratamento literal simplista do seu significado. Precisamos, sim, lidar com questões como:

1. Qual era a intenção de Shakespeare ao escrevê-lo? Devemos saber um bocado sobre o amor humano e as suas convenções sociais para começar a compreender essas linhas. Ou ele apenas tentava entregar alguma coisa ao seu editor para receber o seu pagamento?
2. Por que Shakespeare compara a sua amada a um dia de verão? Será que ele quer dizer que ela dura 24 horas e pode causar queimaduras ou será que ela o faz sentir o calor e a beleza do verão?
3. Que inferências são requeridas nessa passagem? A intenção de Shakespeare não se encontra explicitamente no texto; ela precisa ser inferida usando metáforas, analogias e conhecimento prévio. Por exemplo, como fazemos para interpretar as referências aos ventos tempestuosos e à brevidade do verão como lamentos pela brevidade da vida humana e do amor?
4. Como as metáforas moldam o nosso entendimento? As palavras não são meras referências a objetos explícitos, como blocos sobre uma mesa: o significado do poema está na atribuição seletiva de propriedades de um dia de verão à amada. Que propriedades são atribuídas e quais não são e, sobretudo, por que algumas propriedades são importantes e outras são ignoradas?
5. Será que um sistema computacional para transcrição de texto para fala (*text-to-speech*) sabe alguma coisa sobre pentâmetro iâmbico? Como um computador poderia resumir o “assunto” desse poema ou recuperá-lo inteligentemente de um *corpus* de poesias?

Não podemos meramente encadear os significados dicionarizados das palavras de Shakespeare e denominar o resultado compreensão. Em vez disso, devemos empregar um processo complexo de capturar padrões de palavras, analisar sentenças, construir uma representação do significado semântico e interpretar esse significado à luz do conhecimento do domínio do problema.

Nosso segundo exemplo é parte de uma oferta de emprego para um cargo de professor em ciência da computação.

O Departamento de Ciência da Computação da University of New Mexico... está realizando uma seleção para preencher duas vagas para estágio probatório para cargos permanentes. Estamos interessados em contratar pessoas com interesses em:

Software, incluindo ferramentas de análise, de projeto e de desenvolvimento...

Sistemas, incluindo arquitetura, compiladores, redes...

...  
Os candidatos devem ter completado um curso de doutorado em...

O departamento tem programas de pesquisa reconhecidos internacionalmente em computação adaptativa, inteligência artificial,... e mantém intensas colaborações científicas com o Santa Fe Institute e vários laboratórios nacionais...

Várias questões surgem ao se tentar compreender esse anúncio:

1. Como o leitor sabe que o anúncio é para uma vaga de professor, já que o que está explicitamente escrito é apenas que são duas vagas para “estágio probatório para cargos permanentes”? Por quanto tempo as pessoas são contratadas em estágio probatório para cargos permanentes?
2. Que software e quais ferramentas de software são exigidos para o trabalho em um ambiente universitário, já que eles não são mencionados explicitamente? Cobol, Prolog, UML? Uma pessoa precisaria ter bastante conhecimento sobre lecionar e pesquisar em uma universidade para compreender essas expectativas.
3. O que programas internacionalmente reconhecidos e colaborações com instituições interessantes têm a ver com um anúncio de emprego universitário?
4. Como um computador poderia resumir o assunto desse anúncio? O que ele deve saber para recuperar inteligentemente este anúncio da *web* para um candidato com doutorado que está em busca de emprego?

Há (no mínimo) três questões principais envolvidas na compreensão de uma linguagem. Primeiro, presume-se uma grande quantidade de conhecimento humano. Os atos de linguagem descrevem relacionamentos em um mundo normalmente complexo. O conhecimento desses relacionamentos deve ser parte de qualquer sistema de compreensão de linguagem. Segundo, uma linguagem é baseada em padrões: fonemas são componentes de palavras e palavras constituem frases e sentenças. A ordenação de fonemas, palavras e sentenças não é aleatória. Não é possível haver comunicação sem uma grande restrição quanto ao uso desses componentes. Finalmente, os atos de

linguagem são o produto de agentes, tanto de humanos quanto de um computador. Os agentes estão incorporados em um ambiente complexo com dimensões individual e sociológica. Os atos de linguagem são intencionais.

Este capítulo oferece uma introdução aos problemas da compreensão de linguagem natural e às técnicas computacionais desenvolvidas para a sua solução. Embora neste capítulo foquemos na compreensão de texto, os sistemas de compreensão e geração de fala também necessitam resolver esses problemas, bem como as dificuldades adicionais associadas ao reconhecimento e à desambiguação de palavras inseridas em um contexto particular.

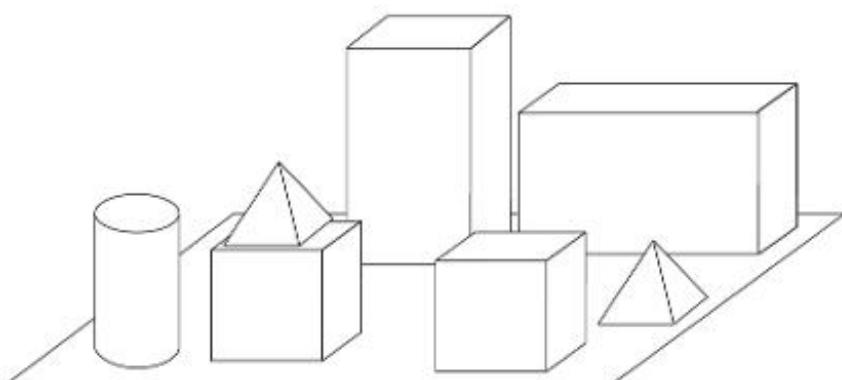
Os primeiros programas de IA, devido ao conhecimento necessário à compreensão irrestrita de linguagem, fizeram progressos restringindo o seu foco a *micromundos*, aplicações limitadas que requeriam um conhecimento mínimo do domínio. Um dos primeiros programas que seguiu essa abordagem foi o SHRDLU, de Terry Winograd (Winograd, 1972), que podia conversar sobre um *mundo de blocos*, constituído de blocos de diferentes formas e cores e de uma garra para movimentá-los, como vemos na Figura 15.1. Veja também a Seção 8.4.

O SHRDLU podia responder a consultas em inglês como, por exemplo, “O que está sobre o bloco vermelho?”, “Qual é a forma do bloco azul que está sobre a mesa?” ou ainda “Coloque a pirâmide verde sobre o tijolo vermelho”. Ele podia lidar com referências pronominais como “Há algum bloco vermelho? Pegue-o”. Ele podia também compreender elipses, tais como “Qual é a cor do bloco sobre o tijolo azul? Sua forma?” Por causa da simplicidade do mundo de blocos, era possível dotar o sistema de conhecimento adequado. Como o mundo de blocos não envolvia os problemas mais difíceis de raciocínio de senso comum, como a compreensão de noções como tempo, causalidade, possibilidades ou crenças, as técnicas para representar esse conhecimento eram relativamente simples. Apesar do seu domínio limitado, o SHRDLU estabeleceu um modelo para a integração de sintaxe e semântica e demonstrou que um programa com suficiente conhecimento de um domínio de discurso poderia se comunicar com significação usando linguagem natural.

Complementarmente ao componente de conhecimento intensivo da compreensão de linguagem, descrito anteriormente, é necessário também modelar os padrões e as expectativas relativas às próprias expressões de linguagem. As *cadeias de Markov* oferecem uma ferramenta poderosa para capturar essas regularidades. Na língua inglesa, por exemplo, os artigos e adjetivos geralmente precedem os substantivos, e certos substantivos e verbos tendem a ocorrerem juntos. Modelos de Markov podem, também, capturar os relacionamentos entre padrões de linguagem e os mundos que eles descrevem.

Na Seção 15.1, apresentamos uma análise de alto nível para a compreensão de linguagem natural. A Seção 15.2 apresenta uma análise sintática; a Seção 15.3 combina sintaxe e semântica usando uma análise por rede de transição estendida. A Seção 15.4 apresenta a abordagem estocástica para capturar regularidades em expressões de linguagem. Finalmente, na Seção 15.5, consideraremos várias aplicações em que programas para compreensão de linguagem natural são úteis: resposta a consultas, acesso a informação em bancos de dados, consultas na web e resumo de textos.

**Figura 15.1** Um mundo de blocos, adaptado de Winograd (1972).



## 15.1 Desconstruindo a linguagem: uma análise

A linguagem é um fenômeno complicado, que envolve processos tão variados como o reconhecimento de sons ou de letras impressas, análise sintática, inferências semânticas de alto nível e mesmo a comunicação de conteúdo emocional a partir de ritmo e inflexão. Para lidar com essa complexidade, os linguistas definiram diferentes níveis de análise para a linguagem natural:

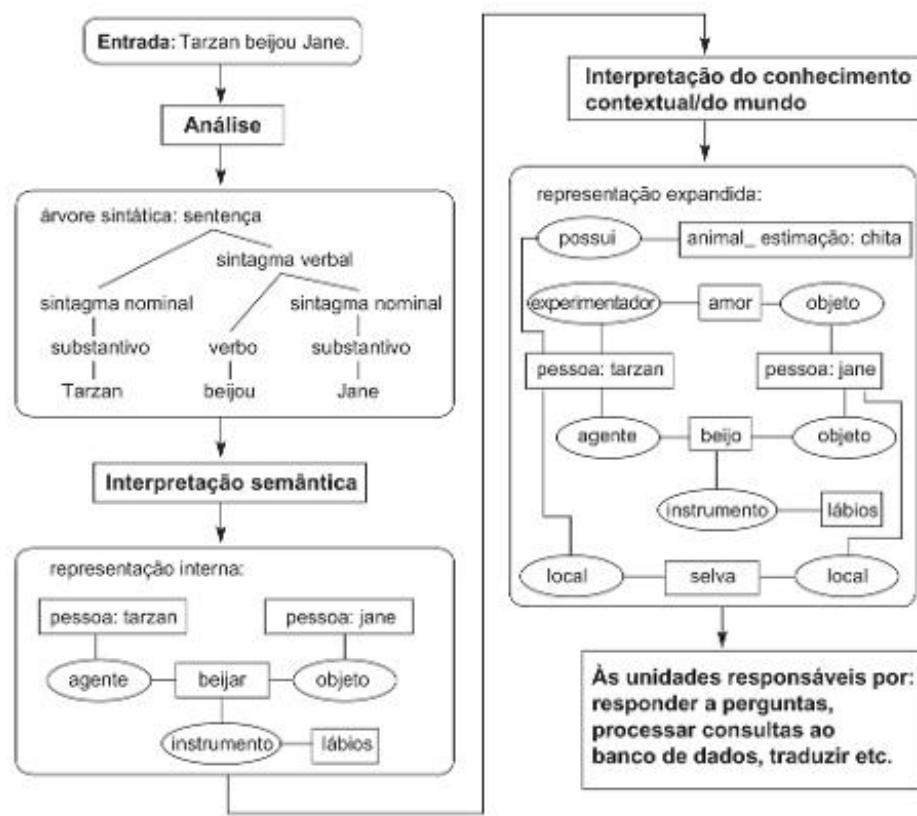
1. A *prosódia* trata do ritmo e da entonação de linguagem. Esse nível de análise é difícil de ser formalizado e normalmente é negligenciado; entretanto, a sua importância é evidente no efeito poderoso da poesia ou de cantos religiosos, bem como no papel desempenhado pelo ritmo no mundo infantil e no balbuciar dos bebês.
2. A *fonologia* examina os sons que são combinados para formar a linguagem. Esse ramo da linguística é importante para o reconhecimento e para a geração computadorizada da fala.
3. A *morfologia* se preocupa com os componentes (morfemas) que constituem palavras. Entre eles se incluem as regras que governam a formação de palavras, por exemplo, o efeito de prefixos (in-, não-, anti- etc.) e sufixos (-ndo, -mente, -ção etc.) que modificam o significado dos radicais de palavras. A análise morfológica é importante para determinar o papel de uma palavra em uma sentença, incluindo o seu tempo, número e sua parte do discurso.
4. A *sintaxe* estuda as regras para combinar palavras em frases e sentenças válidas e o uso dessas regras para analisar e gerar sentenças. Esse é o componente da análise linguística que foi mais bem formalizado e cuja automatização foi mais bem-sucedida.
5. A *semântica* considera o significado de palavras, frases e sentenças e os modos pelos quais o significado é transmitido em expressões em linguagem natural.
6. A *pragmática* é o estudo das formas de uso da linguagem e de seus efeitos sobre o interlocutor. Por exemplo, a pragmática trataria do motivo de “Sim” ser *normalmente* uma resposta inadequada à pergunta: “Você tem horas?”
7. O *conhecimento do mundo* inclui o conhecimento do mundo físico, do mundo da interação social humana e do papel de objetivos e intenções na comunicação. Este conhecimento geral de fundo é essencial para o entendimento do significado completo de um texto ou conversação.

Embora esses níveis de análise pareçam naturais e sejam apoiados por evidências psicológicas, eles são, em certo grau, divisões artificiais que foram impostas à linguagem. Todos eles interagem extensamente, e até mesmo entonações de baixo nível e variações rítmicas têm efeito sobre o significado de uma elocução, por exemplo, o uso de sarcasmo. Essa interação é evidente na relação entre sintaxe e semântica, e, embora uma divisão entre essas linhas pareça essencial, a fronteira exata é difícil de ser caracterizada. Por exemplo, sentenças como “Eles estão comendo maçãs” podem ser analisadas de diferentes formas, que são resolvidas apenas por atenção ao significado no contexto. A sintaxe afeta, também, a semântica, como pode ser visto pelo papel da estrutura da frase na interpretação do significado de uma sentença. Embora a natureza exata da distinção entre sintaxe e semântica seja frequentemente debatida, as evidências psicológicas e a sua utilidade para lidar com a complexidade do problema são argumentos a favor da sua manutenção. No Capítulo 16, abordamos novamente essas questões mais profundas de compreensão e interpretação da linguagem.

Embora a organização específica de programas para a compreensão de linguagem natural varie de acordo com diferentes filosofias e aplicações, por exemplo, uma interface para um banco de dados, um sistema de tradução automática ou um programa para compreensão de histórias, todos eles precisam traduzir a sentença original em uma representação interna do seu significado. Geralmente, a compreensão de linguagem natural baseada em símbolos segue os estágios apresentados na Figura 15.2.

O primeiro estágio é a *análise (parsing)*, que analisa a estrutura sintática de sentenças. Ela verifica se as sentenças são sintaticamente bem formadas e determina, também, uma estrutura linguística. Identificando as principais relações linguísticas como sujeito-verbo, verbo-objeto e substantivo-modificador, o analisador fornece um arcabouço para a interpretação semântica, que normalmente é representado por uma árvore sintática.

**Figura 15.2** Estágios na produção de uma representação interna de uma sentença.



O analisador (*parser*) da linguagem emprega conhecimento sobre a sintaxe da linguagem, a morfologia e um pouco de semântica.

O segundo estágio é a *interpretação semântica*, que produz uma representação do significado do texto. Na Figura 15.2 isso é mostrado como um grafo conceitual. Outras representações usadas frequentemente são dependências conceituais, quadros e representações baseadas em lógica. A interpretação semântica usa conhecimento sobre o significado das palavras e a estrutura linguística, como papéis de substantivos ou a transitividade de verbos. Na Figura 15.2, o programa usou conhecimento do significado de beijar para acrescentar o valor padrão (atribuído por omissão) lábios ao instrumento de beijar. Esse estágio realiza também verificações de consistência. Por exemplo, a definição do verbo beijar pode incluir restrições para que o objeto seja uma pessoa se o agente for uma pessoa, isto é, Tarzan beija Jane e (normalmente) não beija Chita.

No terceiro estágio, as estruturas da base de conhecimento são acrescentadas à representação interna da sentença para produzir uma representação expandida do significado da sentença. Isso acrescenta o conhecimento do mundo necessário para a compreensão total, tal como os fatos de que Tarzan ama Jane, Jane e Tarzan moram na selva e Chita é o animal de estimação do Tarzan. Essa estrutura resultante representa o significado do texto em linguagem natural e é usada pelo sistema para o resto do processamento.

Em uma interface de banco de dados, por exemplo, a estrutura expandida combinaria a representação do significado das consultas com o conhecimento sobre a organização do banco de dados. Isso poderia, então, ser traduzido em uma consulta apropriada na linguagem de banco de dados (ver Seção 15.5.2). Em um programa para compreensão de histórias, essa estrutura expandida representaria o significado da história e seria usada para responder a questões sobre ela (veja a discussão de roteiros no Capítulo 7 e de resumo de textos na Seção 15.5.3).

Esses estágios existem na maioria dos sistemas (não probabilísticos) de compreensão de linguagem natural, embora eles possam ou não corresponder a módulos de software distintos. Por exemplo, muitos programas não

produzem uma árvore sintática explícita, mas geram diretamente a representação semântica interna. Entretanto, a árvore está implícita na análise da sentença. A *análise incremental* (Allen, 1987) é uma técnica comumente usada, na qual um fragmento da representação interna é produzido tão logo uma parte significativa da sentença é analisada. Esses fragmentos são combinados em uma estrutura completa à medida que a análise prossegue. Eles são também usados para resolver ambiguidades e guiar a análise.

## 15.2 Sintaxe

### 15.2.1 Especificação e análise utilizando gramáticas livres de contexto

O Capítulo 3 introduziu o uso de *regras de reescrita* para especificar uma gramática. As regras listadas abaixo definem uma gramática para sentenças transitivas simples como “O homem gosta do cão”. As regras estão numeradas para referência.

1. sentença  $\leftrightarrow$  sintagma\_nominal sintagma\_verbal
2. sintagma\_nominal  $\leftrightarrow$  substantivo
3. sintagma\_nominal  $\leftrightarrow$  artigo substantivo
4. sintagma\_verbal  $\leftrightarrow$  verbo
5. sintagma\_verbal  $\leftrightarrow$  verbo sintagma\_nominal
6. artigo  $\leftrightarrow$  um
7. artigo  $\leftrightarrow$  o
8. substantivo  $\leftrightarrow$  homem
9. substantivo  $\leftrightarrow$  cão
10. verbo  $\leftrightarrow$  gosta
11. verbo  $\leftrightarrow$  morde

As regras de 6 a 11 têm palavras em português no lado direito; essas regras formam um dicionário de palavras que podem aparecer em sentenças. Essas palavras são os *terminais* da gramática e definem um *léxico* da linguagem. Termos que descrevem conceitos linguísticos de mais alto nível (sentença, sintagma\_nominal etc.) são denominados *não terminais*, e aparecem em uma fonte destacada. Note que os terminais não aparecem no lado esquerdo das regras.

Uma sentença válida é qualquer cadeia de terminais que pode ser *derivada* usando essas regras. Uma derivação começa com o símbolo não terminal sentença e produz uma cadeia de terminais a partir de uma série de substituições definidas pelas regras da gramática. Uma substituição válida substitui um símbolo que casa com o lado esquerdo de uma regra pelos símbolos do lado direito dessa regra. Em estágios intermediários da derivação, a cadeia pode conter tanto terminais como não terminais e é denominada *forma sentencial*.

Uma derivação da sentença “O homem morde o cão” é dada por:

CADEIA	APLIQUE REGRA N°
sentença	1
sintagma_nominal sintagma_verbal	3
artigo substantivo sintagma_verbal	7
O substantivo sintagma_verbal	8
O homem sintagma_verbal	5
O homem verbo sintagma_nominal	11
O homem morde sintagma_nominal	3
O homem morde artigo substantivo	7
O homem morde o substantivo	9
O homem morde o cão	

Esse é um exemplo de derivação de cima para baixo (*top-down*): ela começa com o símbolo sentença e age para baixo até uma cadeia de terminais. Uma derivação de baixo para cima (*bottom-up*) começa com uma cadeia de terminais e substitui padrões do lado direito por aqueles do lado esquerdo, terminando quando restar apenas o símbolo sentença.

Uma derivação pode ser representada como uma estrutura de árvore, conhecida como uma *árvore sintática*, na qual cada nó é um símbolo do conjunto de regras da gramática. Os nós interiores da árvore são não terminais; cada nó e seus filhos correspondem, respectivamente, ao lado esquerdo e direito de uma regra da gramática. Os nós folhas são terminais e o símbolo sentença é a raiz da árvore. A árvore sintática para “O homem morde o cão” aparece na Figura 15.3.

A existência de uma derivação ou árvore sintática não apenas prova que uma sentença é válida na gramática, como também determina a estrutura da sentença. A *estrutura frasal* da gramática define a organização linguística mais profunda da linguagem. Por exemplo, a quebra de uma sentença em sintagma\_nominal e sintagma\_verbal especifica a relação entre uma ação e seu agente. Essa estrutura frasal desempenha um papel essencial na interpretação semântica, definindo estágios intermediários em uma derivação nos quais o processamento semântico pode ocorrer, como vimos na Seção 14.3.

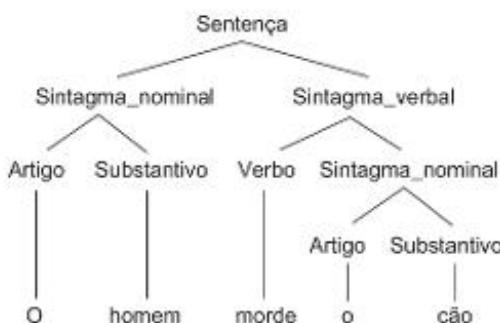
A análise sintática é o problema de construir uma derivação ou uma *árvore sintática* para uma cadeia de entrada a partir de uma definição formal de uma gramática. Os algoritmos de análise são agrupados em duas classes: os *analisadores de cima para baixo*, que começam com o símbolo de alto nível sentença e tentam construir uma árvore cujas folhas casam com a sentença-alvo, e os *analisadores de baixo para cima*, que começam com as palavras da sentença (os terminais) e tentam encontrar uma série de reduções que produz o símbolo sentença.

Uma dificuldade que pode acrescentar enorme complexidade ao problema de análise é a determinação de qual regra, dentre as várias regras potencialmente aplicáveis, deve ser usada em cada passo da derivação. Se for feita uma escolha errada, o analisador pode não conseguir reconhecer uma sentença válida. Por exemplo, ao se tentar analisar a sentença “O cão morde”, em um modo de baixo para cima, as regras 7, 9 e 11 produzem artigo substantivo verbo. Nesse ponto, uma aplicação errada da regra 2 produziria artigo sintagma\_nominal verbo; isso não poderia ser reduzido ao símbolo sentença. O analisador deveria ter usado a regra 3. Problemas similares podem ocorrer em uma análise de cima para baixo.

O problema de selecionar a regra correta em qualquer estágio da análise é tratado permitindo-se que o analisador coloque ponteiros para retrocesso e que retorne à situação do problema, se for feita uma escolha incorreta (ver Seção 3.2), ou então usando antecipação para procurar na cadeia de entrada características que ajudarão a determinar a regra apropriada para ser aplicada. Em qualquer uma dessas abordagens, devemos atentar para o controle da complexidade da execução, ao mesmo tempo garantindo uma análise correta.

O problema inverso é a *geração*, ou a produção de sentenças válidas a partir de uma representação interna. A geração inicia com uma representação de um conteúdo com significação (tal como uma rede semântica ou um grafo de dependência conceitual) e constrói uma sentença gramaticalmente correta que comunica esse significado.

**Figura 15.3** Árvore sintática para a sentença “O homem morde o cão”.



do. Entretanto, a geração não é meramente o inverso da análise; ela encontra dificuldades particulares e requer metodologias distintas. Apresentamos analisadores descendentes recursivos livres de contexto e sensíveis ao contexto na Sala Virtual deste livro.

Como a análise é particularmente importante no processamento tanto de linguagens de programação como de linguagem natural, os pesquisadores desenvolveram uma série de algoritmos de análise diferentes. Entre eles se incluem as estratégias de cima para baixo e de baixo para cima. Embora uma discussão completa dos algoritmos de análise esteja fora do escopo deste capítulo, consideraremos, com algum detalhe, algumas das abordagens para os analisadores. Na próxima seção, mostramos o analisador de Earley, um importante analisador de tempo polinomial baseado na programação dinâmica. Na Seção 15.3, introduzimos as restrições semânticas aos analisadores, com os analisadores por *rede de transição*. Embora esses analisadores por si só não sejam suficientemente poderosos para a análise de linguagem natural, eles formam a base para as *redes de transição estendidas*, que têm se mostrado uma ferramenta útil e poderosa no tratamento da linguagem natural.

### 15.2.2 O analisador de Earley: revisão de programação dinâmica

A programação dinâmica (PD) foi proposta inicialmente por Richard Bellman (1956) e apresentada com vários exemplos na Seção 4.1. A ideia é simples: ao tratar de um problema complexo que possa ser desmembrado em vários subproblemas inter-relacionados, diversas soluções parciais geradas são armazenadas de modo que possam ser reutilizadas no processo de solução continua. Essa abordagem, às vezes, é chamada de memorização dos resultados do subproblema para reúso.

Há muitos exemplos de programação dinâmica no casamento de padrões, por exemplo, na determinação de uma medida de diferença entre duas sequências de bits ou caracteres. A diferença geral entre as sequências será uma função das diferenças entre seus componentes específicos. Um exemplo disso é um corretor ortográfico sugerindo palavras que são “aproximações” da palavra malsoletrada (Seção 4.1). Outro exemplo de PD é reconhecer palavras na compreensão da fala como uma função dos possíveis fonemas de um fluxo de entrada. À medida que os fonemas são reconhecidos (com probabilidades associadas), a palavra mais apropriada normalmente é uma função das medidas probabilísticas conjuntas dos fonemas individuais (ver Seção 13.1). Nesta seção, usamos a PD para mostrar o analisador de Earley determinando se sequências de palavras criam sentenças sintaticamente corretas. Nosso pseudocódigo é adaptado daquele de Jurafsky e Martin (2008).

Os algoritmos de análise da Seção 15.2.1 são frequentemente implementados com uma busca recursiva, em profundidade, da esquerda para a direita, das estruturas sintáticas aceitáveis. Essa técnica de busca pode significar que muitas das análises parciais aceitáveis dos primeiros componentes (mais à esquerda) da sequência são geradas repetidamente. Esse retorno das primeiras soluções parciais dentro da estrutura de análise completa é resultado dos requisitos de retrocesso posteriores da busca e pode se tornar exponencialmente dispendioso em análises maiores. A programação dinâmica oferece uma alternativa eficiente em que análises parciais, uma vez geradas, são salvas para reúso na análise final geral de uma sequência de palavras. O primeiro analisador baseado em PD foi criado por Earley (1970).

#### **Memorização e pares pontuados**

Na análise com o algoritmo de Earley, a memorização de soluções parciais (análises parciais) é feita com uma estrutura de dados denominada *tabela*. É por isso que a técnica de análise de Earley frequentemente é chamada de *análise de tabela*. A tabela é gerada pelo uso de regras gramaticais pontuadas.

A regra gramatical pontuada oferece uma representação que indica, na tabela, o estado do processo de análise em determinado momento. Cada regra pontuada se encontra em uma das três categorias, dependendo de se a posição do ponto está no início, em algum lugar no meio ou no final do lado direito, RHS (do inglês *Right Hand Side*), da regra gramatical. Referimo-nos a essas três categorias como estágios de análise *inicial*, *parcial* e *concluída*, respectivamente:

- |                    |                                     |
|--------------------|-------------------------------------|
| Previsão inicial:  | Símbolo → • RHS_não_visto           |
| Análise parcial:   | Símbolo → RHS_visto • RHS_não_visto |
| Análise concluída: | Símbolo → RHS_visto •               |

Além disso, há uma correspondência natural entre os estados contendo diferentes regras pontuadas e as arestas da(s) árvore(s) de análise produzida(s) pela análise. Considere a seguinte gramática muito simples, onde símbolos terminais são cercados de aspas, como em "mary":

$\text{Sentença} \rightarrow \text{Substantivo Verbo}$   
 $\text{Substantivo} \rightarrow \text{"mary"}$   
 $\text{Verbo} \rightarrow \text{"corre"}$

Ao realizarmos uma análise de cima para baixo e da esquerda para direita dessa sentença, a seguinte sequência de estados é produzida:

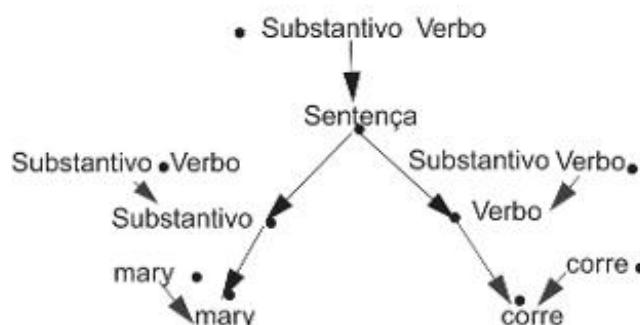
$\text{Sentença} \rightarrow \cdot \text{ Substantivo Verbo}$	previsto: Substantivo seguido por um Verbo
$\text{Substantivo} \rightarrow \cdot \text{ mary}$	previsto: mary
$\text{Substantivo} \rightarrow \text{ mary} \cdot$	varrido: mary
$\text{Sentença} \rightarrow \text{ Substantivo} \cdot \text{ Verbo}$	concluído: Substantivo; previsto: Verbo
$\text{Verbo} \rightarrow \cdot \text{ corre}$	previsto: corre
$\text{Verbo} \rightarrow \text{ corre} \cdot$	varrido: corre
$\text{Sentença} \rightarrow \text{ Substantivo Verbo} \cdot$	concluído: Verbo, concluído: sentença

Observe que os procedimentos de varredura e conclusão produzem um resultado de forma determinística. O procedimento de previsão descreve as regras de análise possíveis que se aplicam à situação atual. Varredura e previsão criam os estados na árvore de análise da Figura 15.4.

O algoritmo de Earley opera gerando previsões de cima para baixo e da esquerda para direita de como analisar determinada entrada. Cada previsão é registrada como um estado contendo todas as informações relevantes sobre a previsão, onde o componente chave de cada estado é uma regra pontuada. (Um segundo componente de implementação será introduzido na próxima seção.) Todas as previsões geradas após o exame de uma palavra em particular da entrada são referenciadas coletivamente como o *conjunto de estados*.

Para determinada sentença de entrada com  $n$  palavras,  $p_1$  a  $p_n$ , um total de  $n + 1$  conjuntos de estados são gerados:  $[S_0, S_1, \dots, S_n]$ . O conjunto de estados inicial  $S_0$  contém aquelas previsões que são feitas antes do exame de quaisquer palavras de entrada,  $S_1$  contém previsões feitas após o exame de  $p_1$ , e assim por diante. Referimo-nos à coleção inteira de conjuntos de estados como a *tabela* produzida pelo analisador. A Figura 15.4 ilustra a relação entre a geração do conjunto de estados e o exame das palavras de entrada. Embora, tradicionalmente, os conjuntos de estados que compõem cada componente da análise sejam chamados de conjuntos de estados, a ordem da geração desses estados é importante. Assim, chamaremos cada componente da tabela de *lista de estados*, e o descreveremos como  $[\text{Estado}_1, \text{Estado}_2, \dots, \text{Estado}_n]$ . Isso também funciona bem com a implementação Prolog na Sala Virtual deste livro, onde as listas de estados serão mantidas como listas em Prolog. Por fim, descrevemos cada estado da lista de estados como uma sequência de símbolos específicos, delimitados por parênteses, por exemplo,  $(\$ \rightarrow \cdot S)$ .

Figura 15.4 A relação das regras pontuadas com a geração de uma árvore de análise.



Agora, considere a análise de algoritmo de Earley da sentença simples *mary corre*, usando a gramática anterior. O algoritmo começa criando um estado inicial fictício,  $(\$ \rightarrow \cdot S)$ , que é o primeiro membro da lista de estados  $S_0$ . Esse estado representa a previsão de que a sequência de entrada pode ser analisada como uma sentença, e é inserida em  $S_0$  antes do exame de quaisquer palavras de entrada. Uma análise bem-sucedida produz uma lista de estado final  $S_n$ , que contém o estado  $(\$ \rightarrow S \bullet)$ .

Começando com  $S_0$ , o analisador executa um laço em que cada estado,  $S_i$ , na lista de estados atual, é examinado em ordem e usado para gerar novos estados. Cada novo estado é gerado por um dos três procedimentos que são denominados previsor, varredor e concludor. O procedimento apropriado é determinado pela regra pontuada no estado  $S$ , especificamente pelo símbolo de gramática (se houver) após o ponto na regra.

Em nosso exemplo, o primeiro estado a ser examinado contém a regra  $(\$ \rightarrow \cdot S)$ . Como o ponto é seguido pelo símbolo  $S$ , esse estado “espera” ver uma ocorrência de  $S$  em seguida na entrada. Como  $S$  é um símbolo não terminal da gramática, o procedimento previsor gera todos os estados correspondentes a uma possível análise de  $S$ . Nesse caso, como há somente uma alternativa para  $S$ , a saber, que  $S \rightarrow \text{Substantivo Verbo}$ , somente um estado,  $(S \rightarrow \cdot \text{Substantivo Verbo})$ , é acrescentado a  $S_0$ . Como esse estado está esperando uma parte da fala, indicada pelo símbolo não terminal Substantivo após o ponto, o algoritmo examina a próxima palavra da entrada para verificar essa previsão. Isso é feito pelo procedimento varredor, e, como a próxima palavra combina com a previsão (*mary* realmente é um Substantivo), o varredor gera um novo estado registrando o casamento:  $(\text{Substantivo} \rightarrow \text{mary} \bullet)$ . Como esse estado depende da palavra de entrada  $p_1$ , ele se torna o primeiro estado na lista de estados  $S_1$ , em vez de ser acrescentado a  $S_0$ . Nesse ponto, a tabela, contendo duas listas de estados, se parece com o seguinte, onde, após cada estado, citamos o procedimento que o gerou:

$S_0: [(\$ \rightarrow \cdot S),$	estado inicial fictício
$(S \rightarrow \cdot \text{Substantivo Verbo})]$	previsor
$S_1: [(\text{Substantivo} \rightarrow \text{mary} \bullet)]$	varredor

Cada estado na lista de estados  $S_0$  foi processado, de modo que o algoritmo se move para  $S_1$  e considera o estado  $(\text{Substantivo} \rightarrow \text{mary} \bullet)$ . Como este é um estado concluído, o procedimento concludor é aplicado. Para cada estado que espera um Substantivo, ou seja, que tem o padrão  $\bullet \text{ Substantivo}$ , o concludor gera um novo estado que registra a descoberta de um Substantivo avançando o ponto sobre o símbolo Substantivo. Nesse caso, o concludor produz o estado  $(S \rightarrow \cdot \text{Substantivo Verbo})$  em  $S_0$  e gera o novo estado  $(S \rightarrow \text{Substantivo} \bullet \text{ Verbo})$  na lista  $S_1$ . Esse estado espera uma parte do discurso, que faz com que o varredor examine a próxima palavra de entrada  $p_2$ . Como  $p_2$  é um Verbo, o varredor gera o estado  $(\text{Verbo} \rightarrow \text{corre} \bullet)$  e o acrescenta a  $S_2$ , o que resulta na tabela a seguir:

$S_0: [(\$ \rightarrow \cdot S),$	inicio
$(S \rightarrow \cdot \text{Substantivo Verbo})]$	previsor
$S_1: [(\text{Substantivo} \rightarrow \text{mary} \bullet),$	varredor
$(S \rightarrow \text{Substantivo} \bullet \text{ Verbo})]$	concludor
$S_2: [(\text{Verbo} \rightarrow \text{corre} \bullet)]$	varredor

Processando o novo estado  $S_2$ , o concludor avança o ponto em  $(S \rightarrow \text{Substantivo} \bullet \text{ Verbo})$  para produzir  $(S \rightarrow \text{Substantivo Verbo} \bullet)$ , do qual o concludor gera o estado  $(\$ \rightarrow \$ \bullet)$  significando uma análise bem-sucedida de uma sentença. A tabela final para *mary corre*, com três listas de estado, é:

$S_0: [(\$ \rightarrow \cdot S),$	inicio
$(S \rightarrow \cdot \text{Substantivo Verbo})]$	previsor
$S_1: [(\text{Substantivo} \rightarrow \text{mary} \bullet),$	varredor
$(S \rightarrow \text{Substantivo} \bullet \text{ Verbo})]$	concludor
$S_2: [(\text{Verbo} \rightarrow \text{corre} \bullet),$	varredor
$(S \rightarrow \text{Substantivo Verbo} \bullet),$	concludor
$(\$ \rightarrow \$ \bullet)]$	concludor

### O algoritmo de Earley: pseudocódigo

Para representar computacionalmente as listas de estado produzidas pelas regras de pares pontuados anteriormente, criamos índices para mostrar o quanto do lado direito de uma regra gramatical foi analisado. Primeiro, descrevemos essa representação e depois oferecemos o pseudocódigo para implementá-la dentro do algoritmo de Earley.

Cada estado na lista de estados é estendido com um índice indicando até que ponto o fluxo de entrada foi processado. Assim, estendemos cada descrição de estado para uma representação (*regra pontuada [i, j]*), onde o par  $[i, j]$  indica o quanto do lado direito, RHS, da regra gramatical foi visto ou analisado até o momento presente. Para o lado direito de uma regra analisada que inclui zero ou mais componentes vistos e não vistos indicados pelo  $\bullet$ , temos  $(A \rightarrow \text{Visto} \bullet \text{Não_visto}, [i, j])$ , onde  $i$  é o inicio de Visto e  $j$  é a posição do  $\bullet$  na sequência de palavras.

Agora, acrescentamos índices aos estados de análise discutidos anteriormente para a sentença mary corre:

$(\$ \rightarrow \bullet S, [0, 0])$	produzido por previsor, $i = j = 0$ , nada analisado
$(\text{Substantivo} \rightarrow \text{mary} \bullet, [0, 1])$	varredor vê $p_1$ entre índices de palavra 0 e 1
$(S \rightarrow \text{Substantivo} \bullet \text{Verbo}, [0, 1])$	concluidor viu Substantivo (mary) entre 0 e 1
$(S \rightarrow \text{Substantivo Verbo} \bullet, [0, 2])$	concluidor viu sentença S entre 0 e 2

Assim, o padrão de indexação de estado mostra os resultados produzidos por cada um dos três geradores de estado usando as regras pontuadas com o índice de palavras  $p_i$ .

Resumindo, os três procedimentos para gerar os estados da lista de estados são: previsor gerando estados com índice  $[j, j]$  entrando na tabela $[j]$ , varredor considerando a palavra  $p_{j+1}$  para gerar estados indexados por  $[j, j+1]$  em tabela $[j+1]$ , e concluidor operando sobre regras com índice  $[i, j]$ ,  $i < j$ , acrescentando uma entrada de estado em tabela $[j]$ . Note que um estado da regra pontuada,  $[i, j]$ , sempre entra na lista de estados tabela $[j]$ . Assim, as listas de estados incluem tabela $[0]$ , ..., tabela $[n]$  para uma sentença de  $n$  palavras. Agora que apresentamos o esquema de indexação para representar a tabela, mostramos o pseudocódigo para o analisador de Earley.

```

função ANÁLISE-EARLEY(palavras, gramática) retorna tabela
    início
        tabela := vazia
        INCLUINATABELA((\$ → ∙ S, [0, 0]), tabela[0]) % estado inicial fictício
        para i de 0 a TAMANHO(palavras) faça
            para cada estado em tabela[i] faça
                se regra_rhs(estado) = ... ∙ A ... e A não faz parte do discurso
                    então PREVISOR(estado)
                senão se regra_rhs(estado) = ... ∙ L ...
                    então VARREDOR(estado) % L faz parte do discurso
                senão CONCLUIDOR(estado) % regra_rhs = RHS ∙
            fim
            procedimento PREVISOR((A → ... ∙ B ..., [i, j]))
                início
                    para cada (B → RHS) em gramática faça
                        INCLUINATABELA((B → ∙ RHS, [j, j]), tabela[j])
                fim
            procedimento VARREDOR((A → ... ∙ L ..., [i, j]))
                início
                    se (L → palavra[j]) está_em gramática
                        então INCLUINATABELA((L → palavra[j] ∙, [j, j + 1]), tabela[j + 1])
                fim
            procedimento CONCLUIDOR((B → ... ∙, [j, k]))
                início
                    para cada (A → ... ∙ B ..., [i, j]) na tabela[j] faça

```

```

INCLUINATABELA((A → ... B * ..., [i, k]), tabela[k])
fim

procedimento INCLUINATABELA(estado, lista-estado)
íncio
    se estado não está em lista-estado
        então INCLUINOFINAL(estado, lista-estado)
    fim

```

Nosso primeiro exemplo, a análise de Earley da sentença Mary corre, serviu como uma apresentação simples, porém ilustrativa, das listas de estados e seus índices. Uma sentença mais complexa (e ambígua), John ligou para Mary de Denver, é apresentada na Sala Virtual com o código para implementar uma análise usando o algoritmo de Earley nas linguagens Prolog e Java. A ambiguidade dessa sentença é refletida na produção de duas análises diferentes que refletem essa ambiguidade. A seleção de uma dessas análises é realizada pela execução do componente de trás para a frente do analisador de Earley baseado em programação dinâmica.

## 15.3 Analisadores e semântica da rede de transição

Até este ponto, oferecemos representações e algoritmos que dão suporte à análise sintática baseada em símbolos. A análise de relacionamentos sintáticos, mesmo quando restrita à análise sensível ao contexto (por exemplo, concordância substantivo-verbo), não considera os relacionamentos semânticos. Nesta seção, apresentamos as *redes de transição* para resolver essa questão.

### 15.3.1 Analisadores por rede de transição

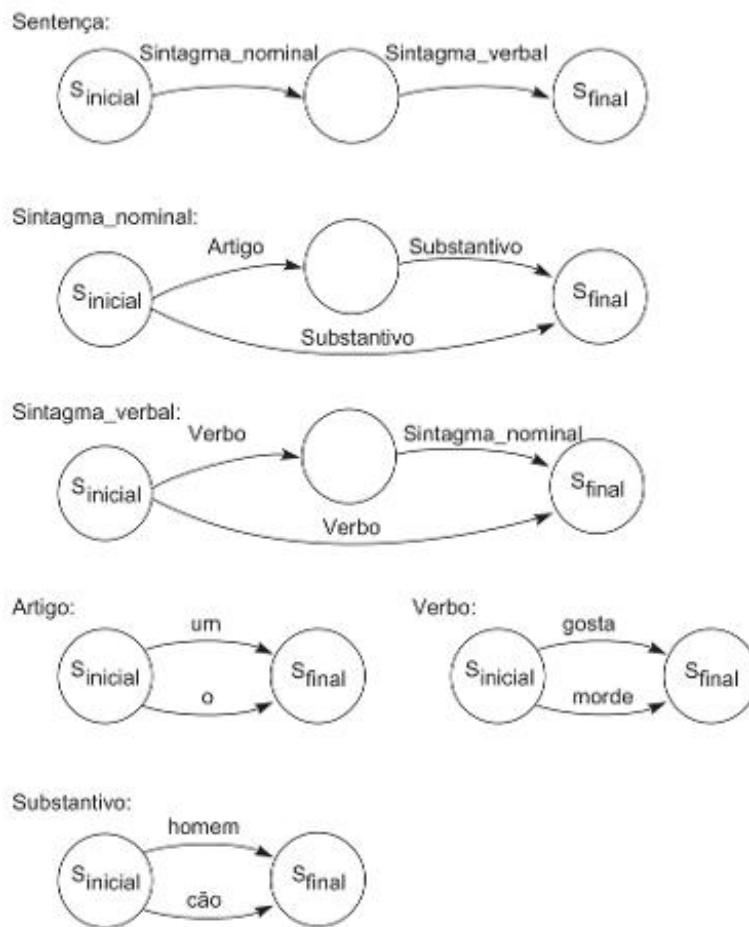
Um analisador por rede de transição representa uma gramática como um conjunto de máquinas de estados finitos ou *redes de transição*. Cada rede corresponde a um único não terminal na gramática. Os arcos nas redes são símbolos terminais ou não terminais. Cada caminho na rede, do estado inicial até o estado final, corresponde a uma regra para aquele não terminal; a sequência de rótulos de arco no caminho é a sequência de símbolos no lado direito da regra. A gramática da Seção 15.2.1 é representada pelas redes de transição da Figura 15.5. Quando houver mais que uma regra para um não terminal, a rede correspondente terá múltiplos caminhos, indo do início até o objetivo. Por exemplo, as regras *sintagma\_nominal*  $\leftrightarrow$  *substantivo* e *sintagma\_nominal*  $\leftrightarrow$  *artigo substantivo* são capturadas por caminhos alternativos por meio da rede *sintagma\_nominal*.

Encontrar uma transição bem-sucedida a partir da rede para um não terminal corresponde a substituir aquele não terminal pelo lado direito de uma regra da gramática. Por exemplo, para analisar uma sentença, um analisador por rede de transição deve encontrar uma transição a partir da rede da sentença. Ele começa no estado inicial ( $S_{inicial}$ ) e realiza a transição *sintagma\_nominal* e, então, a transição *sintagma\_verbal* para alcançar o estado final ( $S_{final}$ ). Isso equivale a substituir o símbolo sentença original pelo par de símbolos *sintagma\_nominal* *sintagma\_verbal*.

Para atravessar um arco, o analisador examina o seu rótulo. Se o rótulo for um símbolo terminal, o analisador verifica a sequência de entrada para ver se a próxima palavra casa com o rótulo do arco. Se ela não casar, a transição não poderá ser feita. Se o arco for rotulado com um símbolo não terminal, o analisador evoca a rede para esse não terminal e, recursivamente, tenta encontrar um caminho para atravessá-lo. Se o analisador não conseguir encontrar um caminho por meio dessa rede, o arco de alto nível não poderá ser atravessado. Isso faz com que o analisador retroceda e tente outro caminho a partir da rede. Dessa forma, o analisador tenta encontrar um caminho por meio da rede de sentença; se ele for bem-sucedido, a cadeia de entrada será uma sentença válida da gramática.

Considere a sentença simples “Cão morde”, conforme analisada e ilustrada na Figura 15.6:

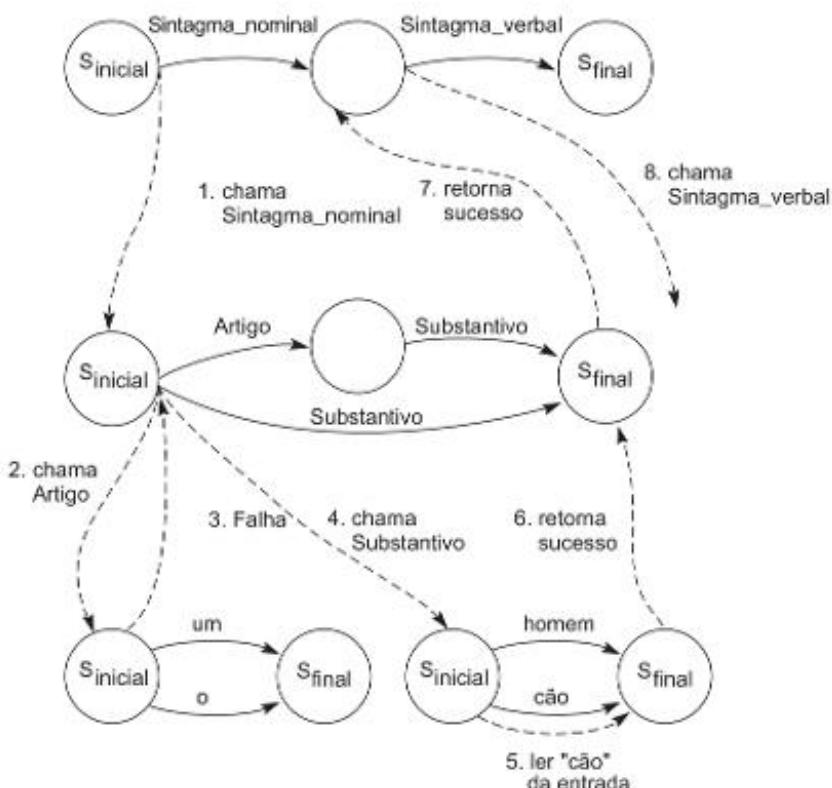
Figura 15.5 Definição por rede de transição de uma gramática simples.



1. O analisador começa com a rede de sentença e tenta se mover através do arco rotulado **sintagma\_nominal**. Para isso, ele evoca a rede para **sintagma\_nominal**.
2. Na rede de **sintagma\_nominal**, o analisador tenta primeiro a transição marcada como **artigo**. Isso o faz desviar para a rede de **artigo**.
3. Ele não consegue encontrar um caminho para o nó final da rede de **artigo**, porque a primeira palavra da sentença, “Cão”, não casa com nenhum dos rótulos de arco. O analisador detecta a falha e retrocede para a rede **sintagma\_nominal**.
4. O analisador tenta seguir o arco rotulado **substantivo** na rede de **sintagma\_nominal** e desvia para a rede de **substantivo**.
5. O analisador consegue atravessar o arco rotulado “cão”, porque ele corresponde à primeira palavra da sequência de entrada.
6. A rede **substantivo** retorna sucesso. Isso permite que o arco rotulado **substantivo** na rede de **sintagma\_nominal** seja cruzado para o estado final.
7. A rede de **sintagma\_nominal** retorna o sucesso à rede de alto nível, permitindo a transição do arco rotulado **sintagma\_nominal**.
8. Uma sequência de passos similares é seguida para a análise da parte **sintagma\_verbal** da sentença.

O pseudocódigo para um analisador de rede de transição aparece a seguir. Ele é definido usando duas funções mutuamente recursivas, análise e transição. Análise toma um símbolo da gramática como argumento: se o símbo-

Figura 15.6 Rastreamento de uma análise por rede de transição da sentença "Cão morde".



Se o símbolo for um terminal, análise tenta casá-lo com a próxima palavra na sequência de entrada. Se o símbolo for não terminal, análise evoca a rede de transição associada ao símbolo e chama transição para encontrar um caminho através da rede. Para analisar uma sentença, chame análise(sentença).

```

função análise(símbolo_gramatical);
  início
    salvar ponteiro para posição atual na sequência de entrada;
    caso
      símbolo_gramatical é um terminal:
        se símbolo_gramatical casa com a próxima palavra na sequência de entrada
          então retornar (sucesso)
        senão início
          reiniciar sequência de entrada;
          retornar (falha)
        fim;
      símbolo_gramatical é um não terminal:
        início
          evocar a rede de transição rotulada pelo símbolo grammatical;
          estado := estado inicial da rede;
          se transição(estado) retorna sucesso
            então retornar (sucesso)
          senão início
    
```

```

    reiniciar sequência de entrada;
    retornar (falha)
fim
fim
fim
fim.

função transição(estado_atual);
íncio
caso
    estado_atual é um estado final:
        retornar (sucesso)
    estado_atual não é um estado final:
        enquanto houver transições não examinadas além do estado_atual
            fazer íncio
                símbolo_gramatical := o rótulo da próxima transição não examinada;
                se análise(símbolo_gramatical) retorna (sucesso)
                    então íncio
                        próximo_estado := estado no fim da transição;
                        se transição(próximo_estado) retorna sucesso;
                            então retornar (sucesso)
                fim
            fim
        retornar (falha)
fim
fim.

```

Transição toma um estado em uma rede de transição como seu argumento e tenta achar um caminho, pela estratégia em profundidade, por essa rede. Como o analisador pode cometer um erro e ser obrigado a retroceder, análise retém um ponteiro para a posição atual na sequência de entrada. Isso permite que a sequência de entrada seja reiniciada a partir dessa posição, no caso de o analisador retroceder.

Esse analisador por rede de transição determina se uma sentença é gramaticamente correta, mas ele não constrói uma árvore sintática. Isso pode ser feito se as funções retomarem uma subárvore da árvore sintática, em vez do símbolo sucesso. As modificações que realizariam isso seriam as seguintes:

1. Cada vez que a função análise for chamada com um símbolo terminal como argumento e que o terminal casar com o símbolo seguinte de entrada, ela retorna uma árvore consistindo em um único nó folha com esse símbolo.
2. Quando análise é chamada com um símbolo\_gramatical não terminal, ela chama transição. Se transição obter sucesso, ela retorna um conjunto ordenado de subárvores (descrito a seguir). Análise combina essas árvores em uma árvore cuja raiz é símbolo\_gramatical e cujos filhos são as subárvores retornadas por transição.
3. Na procura por um caminho através de uma rede, transição chama análise com o rótulo de cada arco. Caso obtenha sucesso, análise retorna uma árvore que representa uma análise daquele símbolo. Transição salva essas subárvores em um conjunto ordenado e, encontrando um caminho através da rede, retorna o conjunto ordenado de árvores de análise que corresponde à sequência dos rótulos dos arcos ao longo do caminho.

### 15.3.2 A hierarquia de Chomsky e as gramáticas sensíveis ao contexto

Na Seção 15.2.1, definimos um pequeno subconjunto do português usando uma *gramática livre de contexto*. Ela permite que as regras tenham apenas um único símbolo não terminal no seu lado esquerdo. Consequen-

tamente, a regra pode ser aplicada a qualquer ocorrência daquele símbolo, independentemente do seu contexto. Embora as gramáticas livres de contexto tenham se mostrado uma ferramenta poderosa para definir linguagens de programação e outros formalismos da ciência da computação, há razões para se acreditar que elas não são suficientemente poderosas, por si só, para representar as regras de sintaxe da linguagem natural. Por exemplo, considere o que acontece se acrescentarmos substantivos e verbos, tanto no singular como no plural, à gramática da Seção 15.2.1:

```
substantivo ↔ homens  
substantivo ↔ cães  
verbo ↔ mordem  
verbo ↔ gostam
```

Embora a gramática resultante analise corretamente sentenças como “Os cães mordem os homens”, ela também aceitará sentenças como “Um homens acaricia um cães”. O analisador aceitará essas sentenças porque as regras atuais não podem usar contexto para determinar quando as formas singular e plural precisam ser coordenadas. Por exemplo, a regra que define uma sentença como um *síntagma\_nominal* seguido de um *síntagma\_verbal* não requer que o sujeito e o verbo concordem em número; o mesmo problema ocorre ao se forçar a concordância de artigos com substantivos.

As linguagens livres de contexto podem ser estendidas para tratar dessas situações, mas uma técnica mais natural é que a gramática seja sensível ao contexto, onde os componentes da árvore de análise são criados para restringir uns aos outros. Chomsky (1965) propôs inicialmente um mundo de gramáticas hierárquicas e ainda mais poderosas (Hopcroft e Ullman, 1979). Na base da hierarquia está a classe das *linguagens regulares*, cujas gramáticas podem ser definidas usando uma máquina de estados finitos (Seção 3.1). Embora as linguagens regulares tenham muitos usos em ciência da computação, elas não são suficientemente poderosas para representar a sintaxe da maioria das linguagens de programação.

As *linguagens livres de contexto* estão acima das linguagens regulares na hierarquia de Chomsky, e são definidas usando regras de reescrita como aquelas da Seção 15.2.1; as regras livres de contexto podem ter apenas um símbolo não terminal no seu lado esquerdo. Os analisadores por rede de transição são capazes de analisar a classe de linguagens livres de contexto. É interessante notar que, se não permitirmos recursão em um analisador por rede de transição (isto é, os arcos podem ser rotulados apenas com símbolos terminais, uma transição não pode “chamar” outra rede), então a classe de linguagens que pode ser assim definida corresponde às expressões regulares. Assim, as linguagens regulares são um subconjunto próprio das linguagens livres de contexto.

As *linguagens sensíveis ao contexto* formam um superconjunto próprio das linguagens livres de contexto. Elas são definidas usando *gramáticas sensíveis ao contexto*, que permitem mais de um símbolo no lado esquerdo de uma regra e tornam possível definir um contexto no qual aquela regra pode ser aplicada. Isso assegura a satisfação de restrições globais como concordância em número e outras verificações semânticas. A única restrição das regras sensíveis ao contexto é que o lado direito seja, no mínimo, tão longo quanto o lado esquerdo (Hopcroft e Ullman, 1979).

Uma quarta classe, formando um superconjunto das linguagens sensíveis ao contexto, é a classe das linguagens *recursivamente enumeráveis*, que podem ser definidas usando regras de produção irrestritas; como essas linguagens são menos restritas que as regras sensíveis ao contexto, as linguagens recursivamente enumeráveis são um superconjunto próprio das linguagens sensíveis ao contexto. Essa classe não é de interesse ao se definir a sintaxe da linguagem natural, embora ela seja importante na teoria da ciência da computação. O restante desta seção foca no português como uma linguagem sensível ao contexto.

Uma gramática livre de contexto simples para sentenças da forma artigo substantivo verbo que força concordância de número entre artigo e substantivo e entre sujeito e verbo é dada por:

```
sentença → síntagma_nominal síntagma_verbal  
síntagma_nominal ↔ artigo número substantivo  
síntagma_nominal ↔ número substantivo  
número ↔ singular
```

```
número ↔ plural  
artigo singular ↔ um singular  
artigo singular ↔ o singular  
artigo plural ↔ uns plural  
artigo plural ↔ os plural  
singular substantivo ↔ cão singular  
singular substantivo ↔ homem singular  
plural substantivo ↔ homens plural  
plural substantivo ↔ cães plural  
sintagma_verbal singular ↔ verbo singular  
sintagma_verbal plural ↔ verbo plural  
verbo singular ↔ morde  
verbo singular ↔ gosta  
verbo plural ↔ mordem  
verbo plural ↔ gostam
```

Nessa gramática, os símbolos singular e plural oferecem restrições para determinar quando podem ser aplicadas diferentes regras para artigo, substantivo e sintagma\_verbal, assegurando a concordância em número. Uma derivação da sentença “Os cães mordem” usando essa gramática é dada por:

```
sentença.  
sintagma_nominal sintagma_verbal.  
artigo plural substantivo sintagma_verbal.  
O plural substantivo sintagma_verbal.  
Os cães plural sintagma_verbal.  
Os cães plural verbo.  
Os cães mordem.
```

De modo semelhante, podemos usar gramáticas sensíveis ao contexto para realizar verificações de concordância semântica. Por exemplo, poderíamos desabilitar sentenças como “Homem morde cão” acrescentando um não terminal, *ato\_de\_morder*, à gramática. Esse não terminal poderia ser verificado na regra para evitar que qualquer sentença envolvendo “morde” tenha “homem” como sujeito.

Embora gramáticas sensíveis ao contexto possam definir estruturas de linguagem que não podem ser capturadas usando gramáticas livres de contexto, elas têm uma série de desvantagens para o projeto de analisadores práticos:

1. Gramáticas sensíveis ao contexto aumentam dramaticamente o número de regras e de não terminais da gramática. Imagine a complexidade de uma gramática sensível ao contexto que incluisse concordâncias em número, gênero e todas as outras formas de concordância requeridas em português.
2. Elas obscurecem a estrutura frasal da linguagem que é tão claramente representada nas regras livres de contexto.
3. Ao tentar lidar com verificações de concordância mais complicadas e com consistência semântica na própria gramática, elas perdem muito do benefício da separação dos componentes sintático e semântico da linguagem.
4. Gramáticas sensíveis ao contexto não tratam do problema da construção de uma representação semântica do significado do texto. Um analisador que simplesmente aceita ou rejeita sentenças não é suficiente; ele precisa retornar uma representação útil do significado semântico da sentença.

Na Sala Virtual deste livro, apresentamos os analisadores livres de contexto e sensíveis ao contexto e geradores de sentença. O regime de controle para esses programas em Prolog é descendente recursivo por profundidade. Em seguida, examinamos as *redes de transição estendidas* (ATN, do inglês *Augmented Transition Networks*), uma extensão das redes de transição que pode definir linguagens sensíveis ao contexto, mas que tem várias vantagens sobre as gramáticas sensíveis ao contexto em relação ao projeto de analisadores.

### 15.3.3 Semântica: analisadores ATN

Uma alternativa às gramáticas sensíveis ao contexto é manter a estrutura mais simples das gramáticas livres de contexto e estender essas regras com procedimentos associados que realizam os testes contextuais necessários. Esses procedimentos são executados quando uma regra é invocada durante a análise. Em vez de usar a gramática para descrever noções como número, tempo e gênero, representamos isso como *características* associadas aos terminais e não terminais da gramática. Os procedimentos associados às regras da gramática acessam essas características para atribuir valores e realizar os testes necessários. As gramáticas que usam extensões de gramáticas livres de contexto para implementar sensibilidade a contexto incluem as *gramáticas de estrutura de frase estendidas* (Heidorn, 1975; Sowa, 1984), *gramáticas de lógica estendidas* (Allen, 1987) e a *rede de transição estendida* (ATN).

Nesta seção, apresentamos a análise ATN e esboçamos o projeto de um analisador ATN simples para sentenças sobre o “mundo dos cães” introduzido na Seção 15.2.1. Abordamos os dois primeiros passos da Figura 15.2: a criação de uma árvore sintática e seu uso para construir uma representação do significado da sentença. Nesse exemplo, usamos grafos conceituais, embora os analisadores ATN possam ser usados também com redes semânticas, roteiros, quadros ou representações lógicas.

As redes de transição estendidas expandem as redes de transição permitindo que procedimentos sejam associados aos arcos das redes. Um analisador ATN executa esses procedimentos associados quando ele atravessa os arcos. Os procedimentos podem atribuir valores a características gramaticais e realizar testes, causando a falha de uma transição se certas condições (como concordância em número) não forem satisfeitas. Esses procedimentos também constroem uma árvore sintática, que é usada para gerar uma representação semântica interna do significado da sentença.

Representamos terminais e não terminais como identificadores (por exemplo, verbo, sintagma\_nominal) com características associadas. Por exemplo, uma palavra é descrita usando sua raiz morfológica com características correspondentes à sua parte no discurso, número, gênero etc. Não terminais na gramática são descritos de forma similar. Um sintagma nominal é descrito por seus artigos, substantivo, número e gênero. Tanto terminais como não terminais podem ser representados usando estruturas do tipo quadro, com *slots* e valores determinados. Os valores desses *slots* especificam características gramaticais ou ponteiros para outras estruturas. Por exemplo, o primeiro *slot* de um quadro de sentença contém um ponteiro para uma definição de sintagma nominal. A Figura 15.7 mostra os quadros para os não terminais sentença, sintagma\_nominal e sintagma\_verbal na nossa gramática simples.

As palavras individuais são representadas usando estruturas similares. Cada palavra no dicionário é definida por um quadro que especifica sua parte do discurso (artigo, substantivo etc.), sua raiz morfológica e suas características gramaticais significativas. No exemplo dado, apenas verificamos a concordância em número e registramos apenas essa característica. Gramáticas mais sofisticadas indicam a pessoa e outras características. Esses registros do dicionário podem indicar, também, a definição por grafo conceitual do significado da palavra na interpretação semântica. O dicionário completo para a nossa gramática aparece na Figura 15.8.

**Figura 15.7** Estruturas representando os não terminais sentença, sintagma\_nominal e sintagma\_verbal da gramática.

Sentença	Sintagma_nominal	Sintagma_verbal
Sintagma_nominal:	Determinante:	Verbo:
Sintagma_verbal:	Substantivo:	Número:
	Número:	Objeto:

A Figura 15.9 apresenta uma rede ATN para a nossa gramática, com descrições em pseudocódigo dos testes realizados em cada arco. Os arcos são rotulados tanto com não terminais da gramática (como na Figura 15.5) quanto com números; esses números são usados para indicar a função associada a cada arco. Essas funções devem ser computadas com sucesso para que o arco seja atravessado.

**Figura 15.8** Entradas no dicionário para uma rede ATN simples.

Palavra	Definição	Palavra	Definição
um	PARTE_DO_DISCURSO: artigo RAIZ: um NÚMERO: singular	gosta PARTE_DO_DISCURSO: verbo RAIZ: gosta NÚMERO: singular	
uns	PARTE_DO_DISCURSO: artigo RAIZ: um NÚMERO: plural	gostam PARTE_DO_DISCURSO: verbo RAIZ: gosta NÚMERO: plural	
morde	PARTE_DO_DISCURSO: verbo RAIZ: morde NÚMERO: singular	homem PARTE_DO_DISCURSO: substantivo RAIZ: homem NÚMERO: singular	
mordem	PARTE_DO_DISCURSO: verbo RAIZ: morde NÚMERO: plural	homens PARTE_DO_DISCURSO: substantivo RAIZ: homem NÚMERO: plural	
cão	PARTE_DO_DISCURSO: substantivo RAIZ: cão NÚMERO: singular	o PARTE_DO_DISCURSO: artigo RAIZ: o NÚMERO: singular	
cães	PARTE_DO_DISCURSO: substantivo RAIZ: cão NÚMERO: plural	os PARTE_DO_DISCURSO: artigo RAIZ: o NÚMERO: plural	

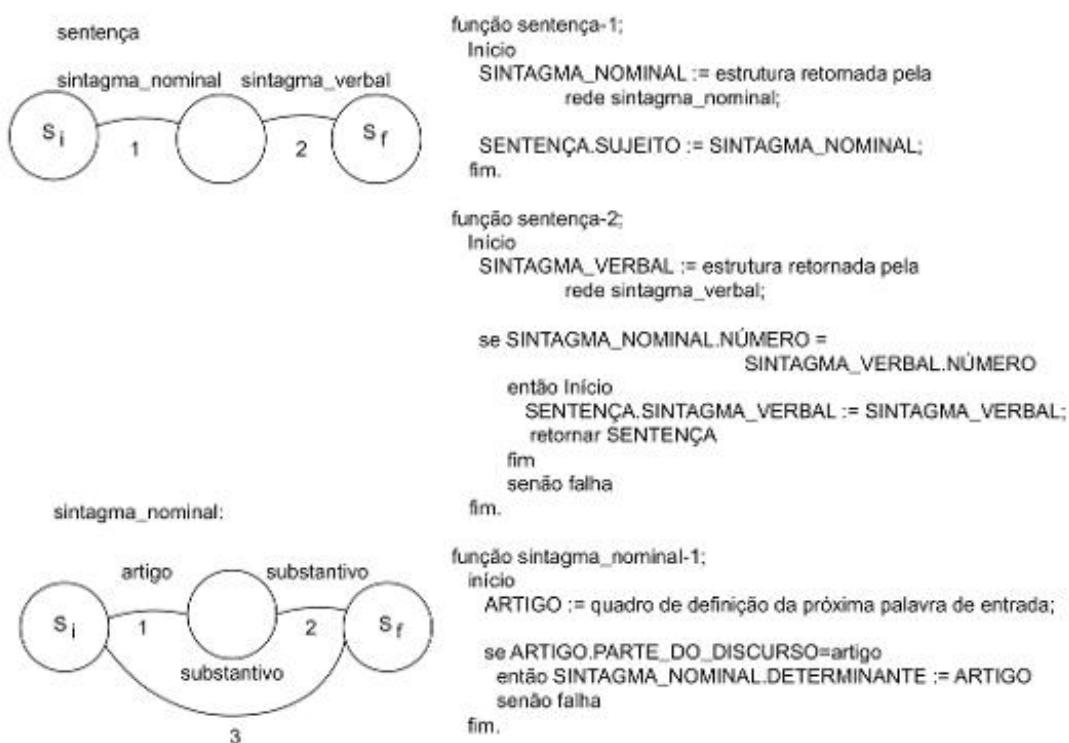
Quando o analisador chama uma rede para um não terminal, ele cria um novo quadro para aquele não terminal. Por exemplo, iniciando a rede `sintagma_nominal`, ele cria um novo quadro `sintagma_nominal`. Os *slots* do quadro são preenchidos pelas funções para aquela rede. A esses *slots* podem ser atribuídos valores das características gramaticais ou ponteiros para componentes da estrutura sintática (por exemplo, um `sintagma_verbal` pode consistir em um verbo e um `sintagma_nominal`). Quando o estado final é alcançado, a rede retorna essa estrutura.

Quando a rede atravessa arcos rotulados como substantivo, artigo e verbo, ela lê a próxima palavra da sequência de entrada e recupera a definição da palavra do dicionário. Se a palavra não for a parte do discurso esperada, a regra falha; caso contrário, o quadro de definição é retomado.

Na Figura 15.9 são indicados quadros e *slots* usando uma notação Quadro.Slot; por exemplo, o *slot* de número do quadro verbo é indicado por `VERBO.NÚMERO`. Conforme a análise prossegue, cada função constrói e retorna um quadro que descreve a estrutura sintática associada. Essa estrutura inclui ponteiros para estruturas retornadas pelas redes de nível mais baixo. A função de alto nível da sentença retorna uma estrutura sentença representando a árvore sintática para a entrada. Essa estrutura é passada para o interpretador semântico. A Figura 15.10 mostra a árvore sintática que é retornada para a sentença “O cão gosta de um homem”.

A próxima fase do processamento de linguagem natural toma a árvore sintática, tal qual a da Figura 15.10, e constrói uma representação semântica do conhecimento do domínio e do significado da sentença.

**Figura 15.9** Uma gramática ATN que verifica a concordância em número e constrói uma árvore sintática (continua).



**Figura 15.9 (cont.)** Uma gramática ATN que verifica a concordância em número e constrói uma árvore sintática.

```

função sintagma_nominal-2;
início
  SUBSTANTIVO := quadro de definição da próxima palavra
  de entrada;
  se SUBSTANTIVO.PARTE_DO_DISCURSO=substantivo e
    SUBSTANTIVO.NÚMERO concorda com
      SINTAGMA_NOMINAL.DETERMINANTE.NÚMERO
  então início
    SINTAGMA_NOMINAL.SUSTANTIVO := SUBSTANTIVO;
    SINTAGMA_NOMINAL.NÚMERO := SUBSTANTIVO.NÚMERO
    retornar SINTAGMA_NOMINAL
  fim
  senão falha
fim.

função sintagma_nominal-3;
início
  SUBSTANTIVO := quadro de definição da próxima palavra
  de entrada;
  se SUBSTANTIVO.PARTE_DO_DISCURSO=substantivo
  então início
    SINTAGMA_NOMINAL.DETERMINANTE := não especificado;
    SINTAGMA_NOMINAL.SUSTANTIVO := SUBSTANTIVO
    SINTAGMA_NOMINAL.NÚMERO := SUBSTANTIVO.NÚMERO
  fim
  senão falha
fim.

sintagma_verbal:

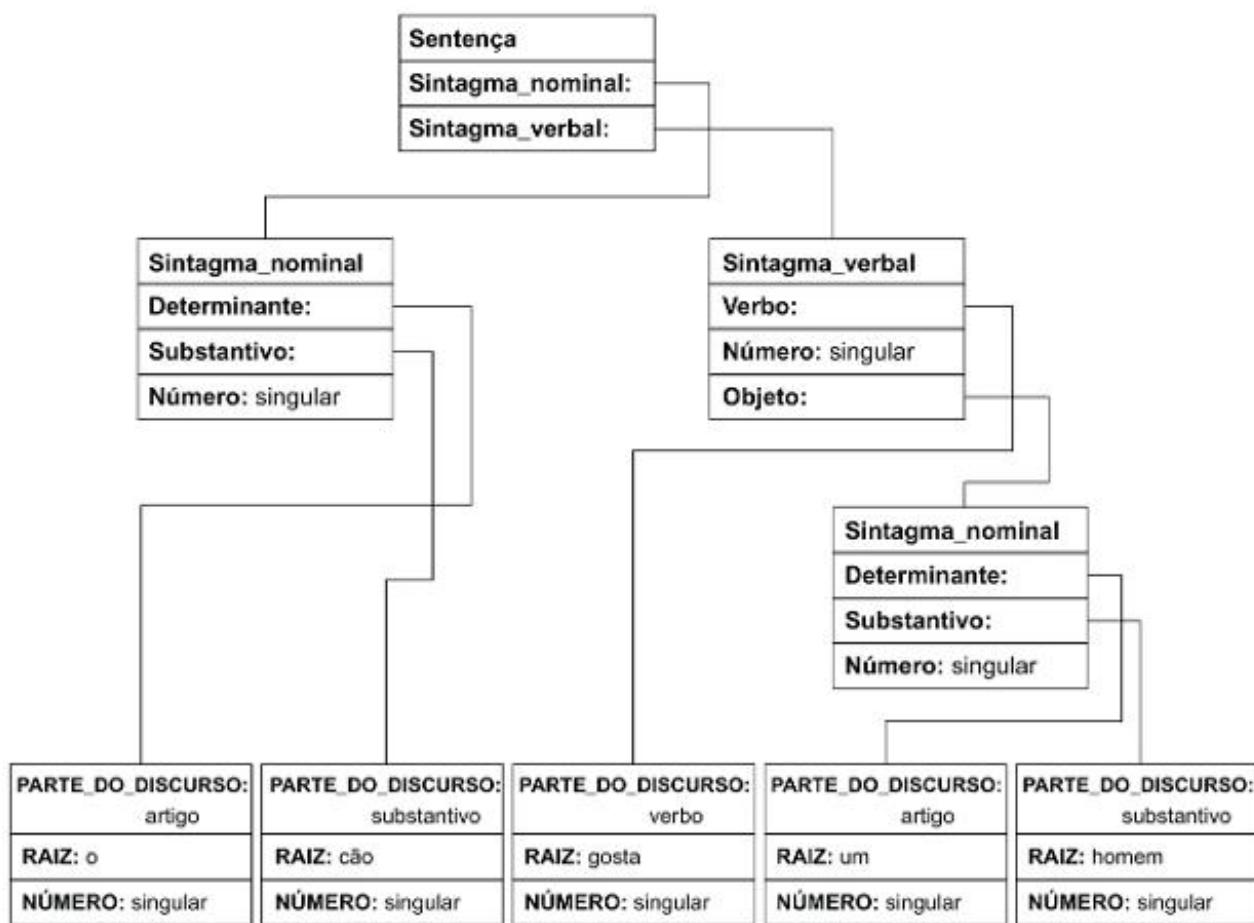
  função sintagma_verbal-1;
início
  VERBO := quadro de definição da próxima palavra de entrada;
  se VERBO.PARTE_DO_DISCURSO = verbo
  então início
    SINTAGMA_VERBAL.VERBO := VERBO;
    SINTAGMA_VERBAL.NÚMERO := VERBO.NÚMERO;
  fim;
fim.

função sintagma_verbal-2;
início
  SINTAGMA_NOMINAL := estrutura retomada pela rede
    sintagma_nominal;
  SINTAGMA_VERBAL.OBJETO := SINTAGMA_NOMINAL;
  retornar SINTAGMA_VERBAL
fim.

função sintagma_verbal-3;
início
  VERBO := quadro de definição da próxima palavra de entrada;
  se VERBO.PARTE_DO_DISCURSO=verbo
  então inicio
    SINTAGMA_VERBAL.VERBO := VERBO;
    SINTAGMA_VERBAL.NÚMERO := VERBO.NÚMERO;
    SINTAGMA_VERBAL.OBJETO := não especificado;
    retornar SINTAGMA_VERBAL;
  fim;
fim.

```

**Figura 15.10** Árvore sintática para a sentença “O cão gosta de um homem” retornada pelo analisador ATN.



### 15.3.4 Combinando conhecimento sintático e semântico com ATNs

O interpretador semântico constrói uma representação do significado da cadeia de entrada começando pela raiz, ou nó sentença, e se deslocando através da árvore de análise. Em cada nó, ele interpreta recursivamente os filhos daquele nó e combina os resultados em um único grafo conceitual; esse grafo é passado para cima na árvore. Por exemplo, o interpretador semântico constrói uma representação de sintagma\_verbal construindo recursivamente representações dos filhos do nó, verbo e sintagma\_nominal, combinando-os para formar uma interpretação do sintagma verbal. Isso é passado para o nó sentença e combinado com a representação do sujeito.

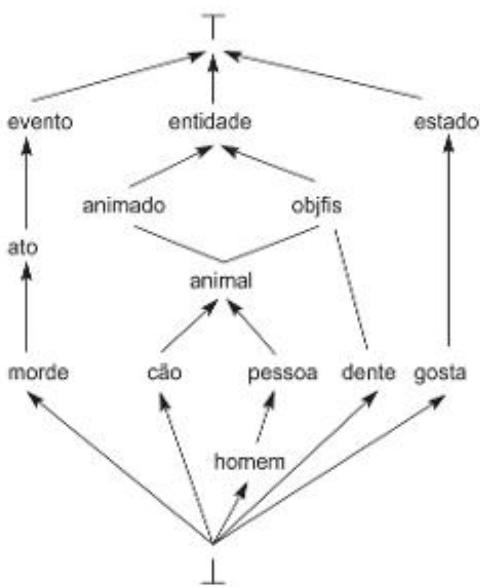
A recursão para nos terminais da árvore sintática. Alguns deles, como os substantivos, os verbos e os adjetivos, fazem com que conceitos sejam recuperados da base de conhecimento. Outros, como os artigos, não correspondem diretamente a conceitos da base de conhecimento, mas qualificam outros conceitos do grafo.

O interpretador semântico no nosso exemplo usa uma base de conhecimento para o “mundo dos cães”. Os conceitos na base de conhecimento incluem os objetos cão e homem e as ações gosta e morde. Esses conceitos são descritos pela hierarquia de tipos da Figura 15.11.

Além dos conceitos, precisamos definir as relações que serão usadas nos nossos grafos conceituais. Para esse exemplo, usamos os seguintes conceitos:

agente liga um ato a um conceito do tipo animado e define a relação entre ação e o objeto animado que causa a ação.

**Figura 15.11** Hierarquia de tipo usada no exemplo do “mundo dos cães”.



experimentador liga um estado a um conceito do tipo animado. Ele define a relação entre um estado mental e o seu experimentador.

instrumento liga um ato a uma entidade e define o instrumento usado em uma ação.

objeto liga um evento ou estado a uma entidade e representa a relação verbo-objeto.

parte liga conceitos do tipo objfis e define a relação entre o todo e a parte.

O verbo desempenha um papel particularmente importante na construção de uma interpretação, pois ele define os relacionamentos entre sujeito, objeto e outros componentes da sentença. Representamos cada verbo usando um *quadro de caso* que especifica:

1. Os relacionamentos linguísticos (agente, objeto, instrumento, e assim por diante) apropriados para aquele verbo em particular. Verbos transitivos, por exemplo, têm um objeto; verbos intransitivos não.
2. Restrições sobre os valores que podem ser atribuídos a um componente do quadro de caso. Por exemplo, no quadro de caso para o verbo “morde”, declaramos que o agente deve ser do tipo cão. Isso faz com que “Homem morde cão” seja rejeitado como semanticamente incorreto.
3. Valores padrão dos componentes do quadro de caso. No quadro “morde”, temos um valor padrão dentes para o conceito ligado à relação instrumento.

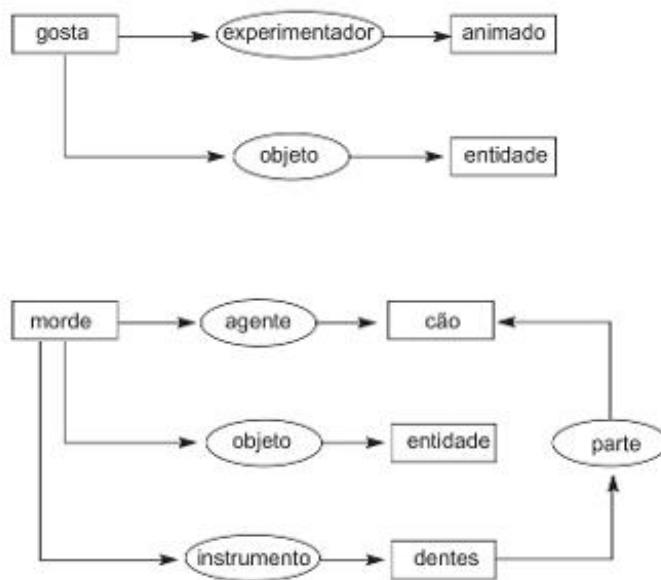
Os quadros de caso para os verbos gosta e morde aparecem na Figura 15.12.

Definimos as ações que constroem uma representação semântica com regras e procedimentos para cada nó potencial na árvore sintática. Para o nosso exemplo, regras são descritas como procedimentos em pseudocódigo. Em cada procedimento, se uma união especificada ou outro teste falhar, aquela interpretação é rejeitada como semanticamente incorreta:

```

procedimento sentença;
  inicio
    chama sintagma_nominal para obter uma representação do sujeito;
    chama sintagma_verbal para obter uma representação do sintagma_verbal;
    usando união e restrição, ligar o conceito de substantivo retornado para o
      sujeito ao agente do grafo para o sintagma_verbal
  fim.
  
```

**Figura 15.12** Quadros de caso para os verbos gosta e morde.



```

procedimento sintagma_nominal;
início
    chama substantivo para obter uma representação do substantivo;
    caso
        o artigo é indefinido e o número é singular: o conceito do substantivo é genérico;
        o artigo é definido e o número é singular: associe marcador ao conceito do substantivo;
        número é plural: indica que o conceito do substantivo é plural
    fim caso
fim.

procedimento sintagma_verbal;
início
    chama verbo para obter uma representação do verbo;
    se o verbo tem um objeto
        então início
            chama sintagma_nominal para obter uma representação do objeto;
            usando união e restrição, associe conceito para o objeto ao objeto do verbo
        fim
    fim.

procedimento verbo;
início
    recupere o quadro de caso do verbo
fim.

procedimento substantivo;
início
    recupere o conceito para o substantivo
fim.

```

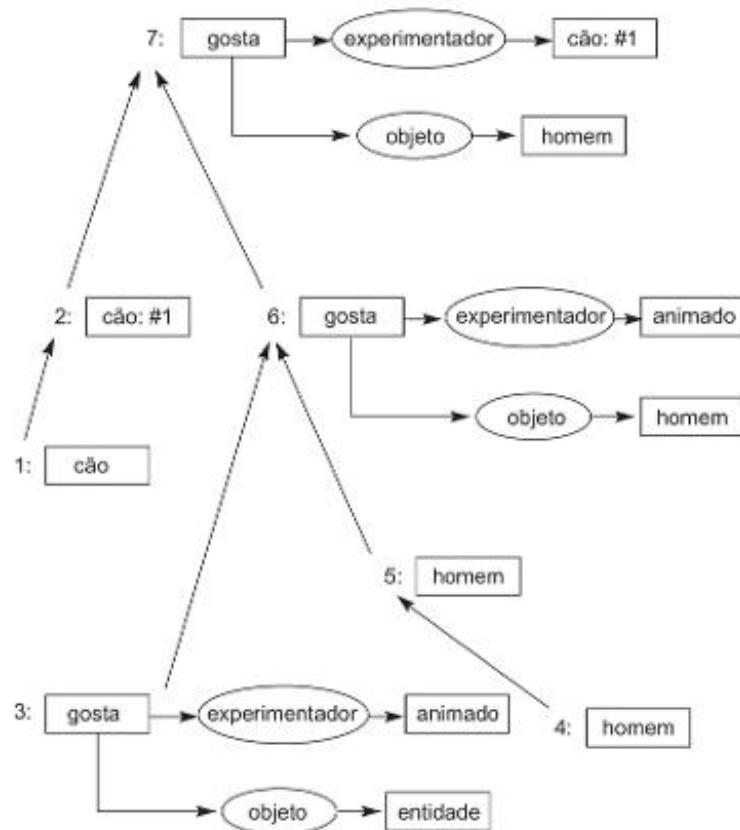
Artigos não correspondem a conceitos na base de conhecimento, eles apenas determinam se o conceito do substantivo é genérico ou específico. Não discutimos a representação de conceitos plurais; veja em Sowa (1984) o seu tratamento como grafos conceituais.

Usando esses procedimentos com a hierarquia de conceitos da Figura 15.11 e os quadros de caso da Figura 15.12, rastreamos as ações do interpretador semântico ao construir uma representação semântica da sentença “O cão gosta de um homem”, a partir da árvore sintática da Figura 15.10. Esse rastreamento aparece na Figura 15.13.

As ações tomadas nesse exemplo são (os números se referem à Figura 15.13):

1. Iniciando no nó da sentença, chame sentença.
2. sentença chama sintagma\_nominal.
3. sintagma\_nominal chama substantivo.
4. substantivo retorna um conceito para o substantivo cão (1 na Figura 15.13).
5. Como o artigo é definido, sintagma\_nominal associa um marcador individual ao conceito (2) e retorna esse conceito para sentença.
6. sentença chama sintagma\_verbal.
7. sintagma\_verbal chama verbo, que recupera o quadro de caso para gosta (3).
8. sintagma\_verbal chama sintagma\_nominal, que, por sua vez, chama substantivo para recuperar o conceito para homem (4).
9. Como o artigo é indefinido, sintagma\_nominal deixa esse conceito genérico (5).
10. O procedimento sintagma\_verbal restringe o conceito de entidade no quadro de caso e o une ao conceito de homem (6). Essa estrutura é retornada para sentença.
11. sentença une o conceito cão: #1 ao nó experimentador do quadro de caso (7).

**Figura 15.13** Construção de uma representação semântica da árvore sintática da Figura 15.10.



Esse grafo conceitual representa o significado da sentença.

A geração da linguagem é um problema correlacionado abordado pelos programas de compreensão de linguagem natural. A geração de sentenças em português requer a construção de uma saída semanticamente correta a partir de uma representação interna de um significado. Por exemplo, a relação *agente* indica um relacionamento sujeito-verbo entre dois conceitos. Algumas abordagens simples permitem que as palavras apropriadas sejam conectadas em *moldes* de sentenças armazenados. Esses moldes são padrões para sentenças e fragmentos, como sintagmas nominais e preposicionados. A saída é construída percorrendo-se o grafo conceitual e combinando-se esses fragmentos. Abordagens mais sofisticadas para a geração de linguagem usam gramáticas transformacionais para mapear o significado para um conjunto de sentenças possíveis (Winograd, 1972; Allen, 1987).

Na Sala Virtual deste livro, montamos um analisador de rede semântica descendente recursivo completo em Prolog. Esse analisador usa operadores de grafo, como *join* e *restrict*, para restringir as representações de rede semântica ligadas aos nós folha da árvore de análise.

Na Seção 15.5, mostraremos como um programa pode construir representações internas de fenômenos de linguagem. Essas representações são usadas pelos programas de várias formas, dependendo da aplicação em particular, sendo que apresentamos várias delas. Mas, primeiro, na Seção 15.4, apresentamos abordagens estocásticas para capturar os padrões e regularidades da linguagem.

## 15.4 Ferramentas estocásticas para análise de linguagem

Na Seção 15.1, discutimos uma decomposição natural de estruturas de sentença que dava suporte à compreensão de linguagem. Nas seções 15.2 e 15.3, observamos que os aspectos semânticos e de conhecimento intensivo da linguagem poderiam ser abordados por meio de estruturas representacionais ao nível de palavra e sentença. Nesta seção, introduzimos ferramentas estocásticas que dão suporte, nesses mesmos níveis, à análise baseada em padrões de estruturas para a compreensão de linguagem.

### 15.4.1 Introdução: técnicas estatísticas para análise de linguagem

As técnicas estatísticas de linguagem são métodos adequados quando consideramos a linguagem natural como um processo aleatório. No linguajar diário, a aleatoriedade sugere falta de estrutura, de definição ou de entendimento. Entretanto, ver a linguagem natural como um processo aleatório *generaliza* o ponto de vista determinista. Isto é, as técnicas estatísticas (ou estocásticas) podem modelar precisamente tanto aquelas partes da linguagem que são bem definidas como também aquelas partes que realmente têm algum grau de aleatoriedade.

Ver a linguagem como um processo aleatório nos permite redefinir muitos dos problemas básicos da compreensão de linguagem natural, de uma maneira rigorosa e matemática. Por exemplo, um exercício interessante é tomar várias sentenças, digamos o parágrafo anterior, incluindo pontos e parênteses, e imprimir essas mesmas palavras ordenadas por um gerador de números aleatórios. O resultado fará muito pouco sentido. É interessante notar (Jurafsky e Martin, 2008) que essas mesmas restrições baseadas em padrões operam em muitos níveis da análise linguística, incluindo padrões acústicos, combinações fonêmicas, a análise da estrutura gramatical, e assim por diante. Como exemplo do uso de ferramentas estocásticas, consideramos o problema da rotulação de parte do discurso.

A maioria das pessoas está familiarizada com esse problema de classificação gramatical. Queremos rotular cada palavra em uma sentença como substantivo, verbo, preposição, adjetivo, e assim por diante. Além disso, se a palavra é um verbo, desejamos saber se ele é ativo, passivo, transitivo ou intransitivo. Se a palavra é um substantivo, se ele é singular ou plural, e assim por diante. As dificuldades aparecem com palavras como “balança”. Se dissermos “balança de farmácia”, balança é um substantivo, mas se dissermos “a cadeira balança”, então balança é um verbo. A seguir, apresentamos uma citação de Picasso com a rotulação correta das partes do discurso:

A arte é uma mentira que nos faz ver a verdade.

Artigo Substantivo Verbo Artigo Substantivo Pronome Pronome Verbo Verbo Artigo Substantivo

Para começar nossa análise, definimos inicialmente o problema formalmente. Temos um conjunto de palavras da nossa linguagem  $S_p = \{p_1, \dots, p_n\}$ , por exemplo {a, aba, ..., zorro}, e um conjunto de partes de discurso ou rótulos  $S_r = \{r_1, \dots, r_m\}$ . Uma sentença com  $n$  palavras é uma sequência de  $n$  variáveis aleatórias,  $P_1, P_2, \dots, P_n$ . Essas variáveis são denominadas *aleatórias* porque elas podem assumir um dos valores de  $S_p$  com certa probabilidade. Os rótulos  $R_1, R_2, \dots, R_n$ , formam, também, uma sequência de variáveis aleatórias. O valor que  $R_i$  assume será denotado  $t_i$ , e o valor de  $P_i$  é  $p_i$ . Queremos encontrar a sequência de valores para esses rótulos que seja a mais provável, dadas as palavras na sentença. Formalmente, queremos escolher  $r_1, \dots, r_n$  para maximizar:

$$P(R_1 = r_1, \dots, R_n = r_n | P_1 = p_1, \dots, P_n = p_n)$$

Na Seção 5.2, definimos que  $p(X | Y)$  significa *a probabilidade de X dado que Y tenha ocorrido*. Normalmente, omitimos a referência às variáveis aleatórias e escrevemos apenas:

$$P(r_1, \dots, r_n | p_1, \dots, p_n)$$

equação 1

Note que, se conhecêssemos exatamente essa distribuição de probabilidade e se tivéssemos tempo suficiente para maximizar em todos os conjuntos possíveis de rótulos, sempre obteríamos o melhor conjunto possível de rótulos para as palavras consideradas. Além disso, se houvesse realmente apenas uma sequência correta de rótulos para cada sentença, uma ideia que o seu professor de gramática teria apoiado, essa técnica probabilística encontraria sempre a sequência correta! Assim, a probabilidade para a sequência correta seria 1, e a probabilidade para todas as outras sequências seria 0. Isso é o que queremos dizer quando afirmamos que o ponto de vista estatístico pode generalizar o ponto de vista determinista.

Na realidade, por causa das limitações de espaço de armazenamento, de dados e de tempo, não podemos usar essa técnica e devemos encontrar algum tipo de aproximação. O resto desta seção trata das maneiras cada vez melhores de aproximar a equação 1.

Observe, inicialmente, que podemos reescrever a equação 1 de uma forma mais usual:

$$P(r_1, \dots, r_n | p_1, \dots, p_n) = P(r_1, \dots, r_n, p_1, \dots, p_n) / P(p_1, \dots, p_n)$$

e como maximizamos essa equação escolhendo  $r_1, \dots, r_n$ , podemos simplificar a equação 1 escrevendo:

$$P(r_1, \dots, r_n, p_1, \dots, p_n) =$$

$$P(r_1)P(p_1 | r_1)P(r_2 | r_1, p_1) \dots P(r_n | p_1, \dots, p_{n-1}, r_1, \dots, r_{n-1}) =$$

$$\prod_{i=1}^n P(r_i | r_1, \dots, r_{i-1}, p_1, \dots, p_{i-1}) P(p_i | r_1, \dots, r_i, p_1, \dots, p_{i-1})$$

equação 2

Note que a equação 2 é equivalente à equação 1.

### 15.4.2 Uma abordagem por modelo de Markov

Na prática, e como vimos anteriormente em nossa discussão dos capítulos 9.3 e 13, a maximização de equações com probabilidades condicionadas a muitas variáveis aleatórias, como a que encontramos na equação 2, é uma tarefa complexa. Há três razões para isso: primeiro, é difícil armazenar a probabilidade de uma variável aleatória condicionada a muitas outras, porque o número de probabilidades possíveis cresce exponencialmente com o número de variáveis condicionais. Segundo, mesmo se pudéssemos armazenar todos os valores de probabilidade, provavelmente seria difícil estimar os seus valores. A estimativa normalmente é feita empiricamente, contando-se o número de ocorrências de um evento em um conjunto de treinamento previamente rotulado; por isso, se um evento ocorrer apenas poucas vezes no conjunto de treinamento, não obteremos uma boa estimativa da sua probabilidade. Ou seja, é mais fácil estimar  $P(\text{gato} | o)$  que  $P(\text{gato} | \text{O cão caçou } o)$ , pois haverá menos oco-

rências da última probabilidade no conjunto de treinamento. Finalmente, encontrar a cadeia de rótulos que maximiza estruturas como a equação 2, como será mostrado a seguir, é um processo muito demorado.

Primeiro, precisamos fazer algumas aproximações úteis para a equação 2. A primeira aproximação é:

$P(r_i | r_1, \dots, r_{i-1}, p_1, \dots, p_{i-1})$  é, aproximadamente,  $p(r_i | r_{i-1})$

e

$P(p_i | r_1, \dots, r_i, p_1, \dots, p_{i-1})$  é, aproximadamente,  $P(p_i | r_i)$ .

Essas são as chamadas *suposições de primeira ordem de Markov*, como vimos na Seção 9.3, porque elas supõem que o que está sob consideração depende somente do que o precedeu imediatamente, ou seja, é independente do que ocorreu em um passado distante.

Inserindo essas aproximações na equação 2, obtemos:

$$\prod_{i=1}^n P(r_i | r_{i-1}) P(p_i | r_i) \quad \text{equação 3}$$

A equação 3 pode ser calculada imediatamente porque as suas probabilidades podem ser facilmente estimadas e armazenadas. Lembre-se de que a equação 3 é apenas uma estimativa para  $P(r_1, \dots, r_n | p_1, \dots, p_n)$ , e ainda precisamos maximizá-la escolhendo os rótulos, isto é,  $r_1, \dots, r_n$ . Felizmente, há um algoritmo de programação dinâmica, denominado algoritmo de Viterbi (Viterbi, 1967; Forney, 1973; para PD ver seções 4.1 e 15.2; para Viterbi, Seção 10.3) que nos permite fazer isso. O algoritmo de Viterbi calcula a probabilidade de  $r^2$  sequências de rótulos para cada palavra da sentença, onde  $r$  é o número de rótulos possíveis. Para um determinado passo, as sequências de rótulos em consideração são da seguinte forma:

artigo artigo	{melhor sequência restante}
artigo verbo	{melhor sequência restante}
...	
artigo substantivo	{melhor sequência restante}
...	
...	
substantivo artigo	{melhor sequência restante}
...	
substantivo substantivo	{melhor sequência restante}

onde {melhor sequência restante} é a sequência mais provável de rótulos, encontrada dinamicamente para as últimas  $n - 2$  palavras para o rótulo  $n - 1$  dado.

Há uma posição na tabela para cada valor possível para o rótulo de número  $n - 1$  e o de número  $n$  (assim, temos  $r^2$  sequências de rótulos). A cada passo, o algoritmo encontra as probabilidades maiores e acrescenta um rótulo a cada melhor sequência restante. Pode-se garantir que esse algoritmo encontra a sequência de rótulos que maximiza a equação 3, com uma complexidade de  $O(r^2 s)$ , onde  $r$  é o número de rótulos e  $s$  é o número de palavras na sentença. Se  $P(r_i)$  for condicionada pelos últimos  $n$  rótulos, em vez dos últimos dois, o Algoritmo de Viterbi terá uma complexidade de  $O(r^n s)$ . Assim, vemos por que o condicionamento a muitas variáveis passadas aumenta o tempo necessário para encontrar um valor de maximização.

Felizmente, as aproximações usadas na equação 3 funcionam bem. Com cerca de 200 rótulos possíveis e um conjunto de treinamento grande para estimar as probabilidades, um classificador usando esses métodos alcança uma precisão de 97%, o que o aproxima à precisão humana. A surpreendente precisão da aproximação de Markov, junto à sua simplicidade, a torna útil em muitas aplicações. Por exemplo, a maioria dos sistemas de reconhecimento de fala usa o chamado modelo *trígrama* para fornecer “conhecimento gramatical” ao sistema para predizer palavras que o locutor proferiu. O modelo trígrama é um modelo de Markov simples que estima a probabilidade da palavra atual condicionada às duas palavras anteriores. Ele usa o algoritmo de Viterbi e outras técnicas que já descrevemos. Para obter mais detalhes sobre essa técnica e outras relacionadas, veja Jurafsky e Martin (2008).

### 15.4.3 Uma abordagem por árvore de decisão

Um problema óbvio com a abordagem de Markov é que ela considera apenas o contexto local. Se, em vez de rotular palavras com partes simples do discurso desejássemos fazer algo como identificar um agente, identificar um objeto ou decidir se verbos são ativos ou passivos, então necessitariam de um contexto mais rico. A seguinte sentença ilustra esse problema:

A política expressa em dezembro pelo presidente garante taxas reduzidas.

Na realidade, o *presidente* é o agente (da passiva), mas um programa usando um modelo de Markov identificaria a *política* como o agente e *expressa* como um verbo ativo. Podemos imaginar que um programa teria melhor resultado ao fazer uma escolha probabilística do agente nesse tipo de sentença se ele pudesse fazer perguntas como “O substantivo considerado é inanimado?”, ou então “A palavra *pelo* aparece um pouco antes do substantivo considerado?”

Lembre-se de que o problema da rotulação é equivalente a maximizar a equação 2, isto é,

$$\prod_{i=1}^n P(r_i | r_1, \dots, r_{i-1}, p_1, \dots, p_{i-1}) P(p_i | r_1, \dots, r_i, p_1, \dots, p_{i-1}).$$

Teoricamente, considerar um contexto mais amplo envolve simplesmente encontrar melhores estimativas para essas probabilidades. Isso sugere que poderíamos usar respostas às questões gramaticais colocadas anteriormente para refinar as probabilidades.

Há várias formas de se lidar com essa questão. Primeiro, podemos combinar a abordagem de Markov com as técnicas de análise sintática que apresentamos nas primeiras seções deste capítulo. Um segundo método nos permite encontrar probabilidades condicionadas a questões do tipo *sim* ou *não* com o algoritmo ID3 (apresentado em detalhes na Seção 10.3 e construído em Lisp na Sala Virtual) ou outro algoritmo equivalente. As árvores de decisão do ID3 têm a vantagem adicional de, em um conjunto muito grande de questões possíveis, escolherem aquelas que são as melhores para refinar as estimativas de probabilidade. Para tarefas mais complicadas do processamento de linguagem natural, como a análise sintática, as árvores baseadas no ID3 são preferíveis em detrimento dos modelos de Markov. A seguir, descrevemos como empregar o algoritmo ID3 para construir uma árvore de decisão para ser usada em análise de sentenças.

Lembre-se de que, na seção anterior, fizemos a pergunta “O substantivo considerado é inanimado?” Podemos formular esse tipo de questão apenas se conhecermos quais palavras correspondem a entidades animadas e quais são inanimadas. Na realidade, há uma técnica automática que pode atribuir palavras a esses tipos de classes. A técnica é denominada *agrupamento por informação mútua*. A informação mútua compartilhada entre duas variáveis aleatórias X e Y é definida como segue:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x, y) \log_2 \frac{P(x, y)}{P(x)P(y)}$$

Para realizar agrupamento por informação mútua sobre um vocabulário de palavras, começamos colocando cada palavra do vocabulário em um conjunto distinto. Em cada passo, calculamos a informação mútua média entre conjuntos usando um bigrama, que é um modelo da próxima palavra, e escolhemos uma combinação de dois conjuntos de palavras que minimiza a perda de informação mútua para todas as classes.

Por exemplo, se tivermos inicialmente as palavras gato, bichano, corre e verde, no primeiro passo do algoritmo, temos os conjuntos:

{gato} {bichano} {corre} {verde}.

É provável que a probabilidade da próxima palavra, dado que a palavra anterior era gato, seja aproximadamente igual à probabilidade da palavra seguinte, dado que a palavra anterior era bichano. Em outras palavras:

$P(\text{come} | \text{gato})$  é aproximadamente igual a  $P(\text{come} | \text{bichano})$

$P(\text{mia} | \text{gato})$  é aproximadamente igual a  $P(\text{mia} | \text{bichano})$

Assim, se X1, X2, Y1 e Y2 são variáveis aleatórias tais que:

$$X1 = \{\text{gato}, \text{bichano}, \text{corre}, \text{verde}\}$$

Y1 = palavra que segue X1

$$X2 = \{\text{gato, bichano}, \text{corre}, \text{verde}\}$$

Y2 = palavra que segue X2,

então a informação mútua entre X2 e Y2 não é muito menor que a informação mútua entre X1 e Y1; assim, é provável que gato e bichano possam ser combinados. Se continuarmos esse procedimento até termos combinado todas as classes possíveis, obtemos uma árvore binária. Então, podemos atribuir *códigos binários* a palavras com base nos ramos da árvore que alcançam o nó folha que contém aquela palavra. Isso reflete a significação semântica da palavra. Por exemplo:

gato = 01100011

bichano = 01100010

Além disso, poderíamos ter encontrado que palavras “prováveis substantivos” seriam todas aquelas palavras que têm um 1 no bit mais à esquerda e que palavras que representam com maior probabilidade objetos inanimados seriam aquelas cujo terceiro bit é 1.

Essa nova codificação de palavras do dicionário permite que o analisador formule questões mais efetivamente. Note que o agrupamento não leva em consideração o contexto, de forma que, muito embora “livro” possa ser agrupado como uma palavra “provável substantivo”, podemos querer que o nosso modelo rotule-a como um verbo quando ela for encontrada na frase “eu me livro do problema”.

#### 15.4.4 Técnicas probabilísticas para análise

As técnicas estocásticas já foram usadas em muitos domínios da linguística computacional e há, ainda, muitas oportunidades de aplicá-las a áreas que têm resistido a abordagens simbólicas tradicionais.

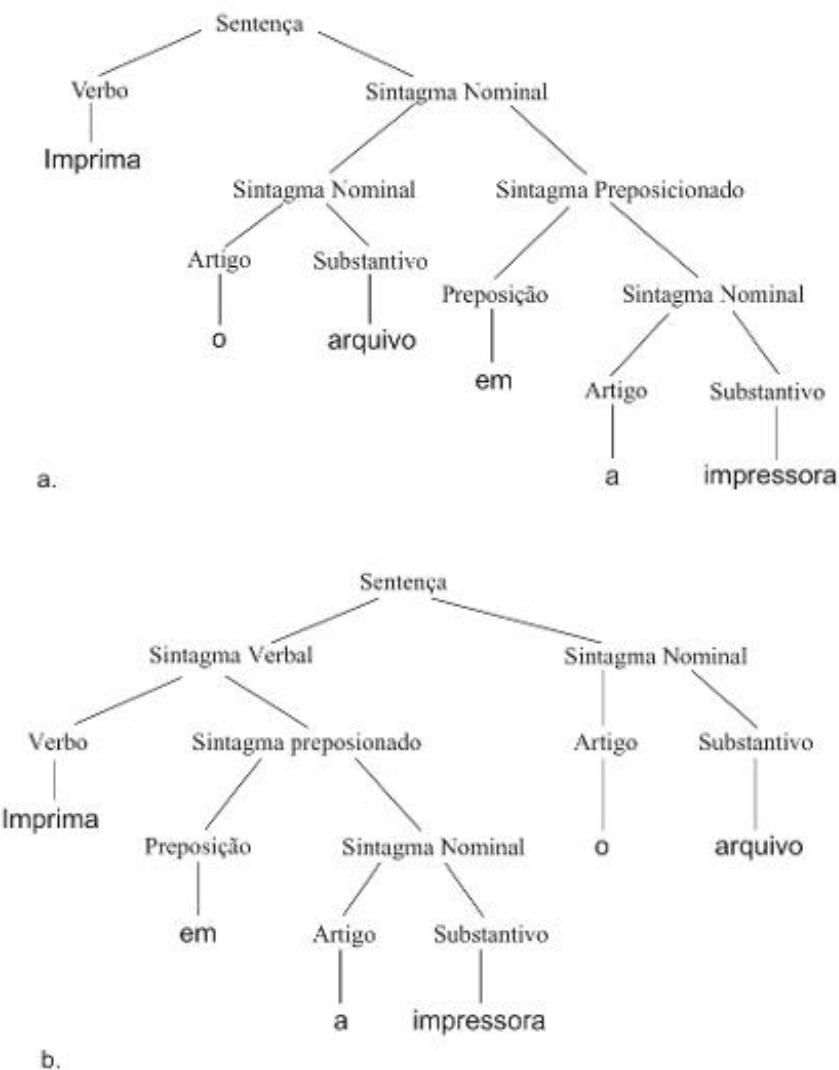
O uso de métodos estatísticos em análise gramatical foi motivado, primeiro, pelo problema da ambiguidade. A ambiguidade surge do fato de que, frequentemente, há várias possibilidades de análise gramatical para uma dada sentença, e precisamos escolher qual delas é a melhor. Por exemplo, a sentença *Imprima o arquivo na impressora* pode ser analisada usando as duas árvores apresentadas na Figura 15.14.

Em situações como essa, regras gramaticais, apenas, não são suficientes para escolher a análise correta. No caso de *Imprima o arquivo na impressora*, precisamos considerar alguma informação sobre o contexto e a semântica. Na realidade, o principal uso de técnicas estocásticas no domínio da análise gramatical é ajudar a resolver ambiguidades. Nesse exemplo, podemos usar a mesma ferramenta usada na rotulação de parte do discurso, o algoritmo ID3. O ID3 nos auxilia a prever a probabilidade de uma análise gramatical ser correta com base em questões semânticas sobre a sentença. No caso, quando há alguma ambiguidade sintática na sentença, podemos escolher aquela análise com a probabilidade mais alta de ser correta. Como já mencionado anteriormente, essa técnica requer um grande *corpus de treinamento* com as suas análises corretas.

Recentemente, pesquisadores da comunidade de modelagem estatística de linguagem natural se tornaram mais ambiciosos e tentaram usar técnicas estatísticas sem uma gramática para realizar a análise. Embora os detalhes da análise sem gramática estejam fora do escopo deste livro, é suficiente dizer que isso está mais relacionado com reconhecimento de padrões que com as técnicas de análise tradicionais discutidas anteriormente neste capítulo.

A análise sem gramática tem obtido bastante sucesso. Em experimentos que compararam um analisador baseado em gramática tradicional com um analisador sem gramática para a tarefa de análise gramatical do mesmo conjunto de sentenças, o analisador baseado em gramática alcançou um escore, usando uma métrica popular, a medida de *parênteses cruzados* (*crossing-brackets*), de 69%, enquanto o analisador sem gramática alcançou 78% (Magerman, 1994). Esses resultados são bons, embora não sejam excepcionais. O mais importante é que a gramática no analisador tradicional foi desenvolvidameticulosamente por um linguista treinado ao longo

Figura 15.14 Duas análises diferentes de Imprima o arquivo na impressora.



de cerca de dez anos, enquanto o analisador sem gramática essencialmente não usou nenhuma informação incorporada, apenas modelos matemáticos sofisticados capazes de inferir a informação necessária a partir dos dados de treinamento. Para obter mais detalhes sobre análise gramatical sem gramática e questões associadas, veja Manning e Schütze (1999).

A compreensão da fala, a conversão de fala em texto e o reconhecimento de textos manuscritos são outras três áreas que têm uma longa história de uso de métodos estocásticos para modelar a linguagem. O método estatístico mais comumente usado nessas áreas é o modelo de trigramas para a predição da palavra seguinte. A força desse modelo está na sua simplicidade: ele prevê a palavra seguinte com base nas duas últimas palavras. Recentemente, houve esforços na comunidade de linguagem estatística para manter a simplicidade e a facilidade de uso desse modelo, ao mesmo tempo, incorporando restrições gramaticais e dependências de distâncias mais longas. Essa nova abordagem usa o que é chamado de *trigramas gramaticais*, que incorporam informação por associações entre pares de palavras (isto é, sujeito-verbo, artigo-substantivo e verbo-objeto). Coletivamente, essas associações são chamadas de *gramática de ligações*. A gramática de ligações é bem mais simples e mais fácil de ser construída que as gramáticas tradicionais usadas pelos linguistas e funciona bem com métodos probabilísticos.

Berger et al. (1994) descrevem um programa estatístico, Candide, que traduz textos do francês para o inglês. Candide usa estatística e teoria da informação para desenvolver um modelo probabilístico do processo de tradução. Ele é treinado apenas com uma grande coleção de pares de sentenças, em inglês e em francês, obtendo resultados comparáveis e, em alguns casos, até melhores, aos do Systran (Berger et al., 1994), um programa comercial de tradução. Particularmente interessante é o fato de que, no processo de tradução, o sistema Candide não realiza análise gramatical tradicional. Em vez disso, usa trigramas gramaticais e gramática de ligações, mencionadas no parágrafo anterior.

### 15.4.5 Analisadores probabilísticos livres de contexto

Nesta seção, apresentamos dois analisadores probabilísticos livres de contexto, baseados em estrutura e lexicalizados. O analisador baseado em estrutura usa medidas de probabilidade para cada regra de análise gramatical que ocorre. A probabilidade de cada regra de análise é uma função de essa regra acontecer em combinação com as probabilidades de seus constituintes.

Demonstramos o *analisador probabilístico baseado em estrutura* estendendo o conjunto de regras de gramática livres de contexto da Seção 15.2 com medidas de probabilidade:

1. sentença  $\leftrightarrow$  sintagma\_nominal sintagma\_verbal  $P(s) = P(r1) P(sn) P(sv)$
2. sintagma\_nominal  $\leftrightarrow$  substantivo  $P(sn) = P(r2) P(substantivo)$
3. sintagma\_nominal  $\leftrightarrow$  artigo substantivo  $P(sn) = P(r3) P(artigo) P(substantivo)$
4. sintagma\_verbal  $\leftrightarrow$  verbo  $P(sv) = P(r4) P(verbo)$
5. sintagma\_verbal  $\leftrightarrow$  verbo sintagma\_nominal  $P(sv) = P(r5) P(verbo) P(sn)$
6. artigo  $\leftrightarrow$  um  $P(artigo) = P(um)$
7. artigo  $\leftrightarrow$  o  $P(artigo) = P(o)$
8. substantivo  $\leftrightarrow$  homem  $P(substantivo) = P(homem)$
9. substantivo  $\leftrightarrow$  cão  $P(substantivo) = P(cão)$
10. verbo  $\leftrightarrow$  gosta  $P(verbo) = P(gosta)$
11. verbo  $\leftrightarrow$  morde  $P(verbo) = P(morde)$

As medidas de probabilidade para as palavras individuais podem ser tomadas de sua probabilidade de ocorrência dentro de um *corpus* em particular de sentenças em português. A medida de probabilidade para cada regra de gramática, o  $P(r_i)$ , pode ser determinada pela frequência com que essa regra gramatical ocorre dentro das sentenças de um *corpus* da linguagem. Naturalmente, o exemplo aqui é simples — ele serve para caracterizar esse tipo de analisador probabilístico livre de contexto.

O segundo exemplo, o *analisador probabilístico livre de contexto lexicalizado*, também será demonstrado estendendo as regras de gramática da Seção 15.2. Nessa situação, queremos levar em conta vários aspectos da sentença. Primeiro, podemos ter probabilidades de bigrama ou trígrama do uso da linguagem. Novamente, seremos dependentes de um *corpus* de linguagem apropriado para obter medidas de bigrama ou trígrama. Assim como em nosso exemplo anterior, também usaremos medidas de probabilidade, obtidas do *corpus* da linguagem, para a ocorrência de cada uma das regras de análise. Por fim, teremos medidas de probabilidade de nomes e verbos particulares correndo juntas. Por exemplo, se o substantivo do sujeito for cão, a probabilidade é a de que o verbo seja morde. Note que essa medida, embora tomada de um *corpus* da linguagem, também pode impor a concordância substantivo-verbo.

Chamamos nosso analisador de *lexicalizado*, pois podemos focar nas probabilidades de padrões específicos de palavras ocorrerem juntas. Para simplificar, apresentamos nosso analisador apenas para bigramas:

1. sentença  $\leftrightarrow$  sintagma\_nominal(substantivo) sintagma\_verbal(verbo)  
 $P(\text{sentença}) = P(r1) P(sn) p(sv) P(\text{verbo} | \text{substantivo})$
2. sintagma\_nominal  $\leftrightarrow$  substantivo  
 $P(sn) = P(r2) p(\text{substantivo})$

3. sintagma\_nominal ↔ artigo substantivo  
 $P(sn) = P(r3) \cdot P(\text{artigo}) \cdot P(\text{substantivo}) \cdot P(\text{substantivo} | \text{artigo})$
4. sintagma\_verbal ↔ verbo  
 $P(sv) = P(r4) \cdot P(\text{verbo})$
5. sintagma\_verbal ↔ verbo sintagma\_nominal  
 $P(sv) = P(r5) \cdot P(\text{verbo}) \cdot P(\text{sintagma\_nominal}(\text{substantivo})) \cdot P(\text{substantivo} | \text{verbo})$
6. artigo ↔ um  $P(\text{artigo}) = P(\text{um})$
7. artigo ↔ o  $P(\text{artigo}) = P(\text{o})$
8. substantivo ↔ homem  $P(\text{substantivo}) = P(\text{homem})$   
 $P(\text{homem} | \text{um})$   
 $P(\text{homem} | \text{o})$   
etc.  
 $P(\text{gosta} | \text{homem})$   
etc.

Cada um dos dois analisadores probabilísticos desta seção foi criado em Prolog na Sala Virtual.

Existem muitas outras áreas em que as técnicas estocásticas rigorosas de modelagem de linguagem ainda não foram experimentadas, mas em que podem produzir resultados úteis. A extração da informação, ou o problema de obter certa quantidade de informação concreta a partir de um texto escrito, é uma aplicação em potencial. Uma segunda aplicação ocorre nos padrões de casamento de tom na voz falada, para unir os conceitos de uma base de conhecimento específica. Por fim, é claro, existe a busca na web semântica. Outros detalhes sobre métodos estocásticos para o processamento de linguagem natural podem ser encontrados em Jurafsky e Martin (2008) e Manning e Schütze (1999).

Encerramos o Capítulo 15 com vários exemplos de uso de linguagem de computador.

## 15.5 Aplicações da linguagem natural

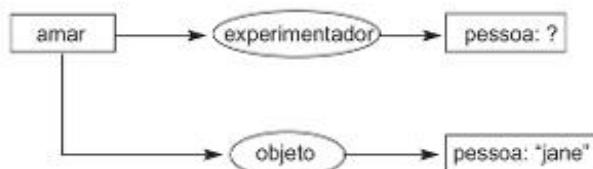
### 15.5.1 Compreensão de histórias e resposta automática a questões

Um teste interessante para a tecnologia de compreensão de linguagem natural é escrever um programa que possa ler uma história ou outro trecho de texto em linguagem natural e responder questões sobre ele. No Capítulo 7, discutimos algumas questões representacionais envolvidas na compreensão de histórias, incluindo a importância de se combinar conhecimento de fundo com o conteúdo explícito do texto. Como ilustrado na Figura 15.2, um programa pode realizar isso executando junções de redes entre a interpretação semântica da entrada e as estruturas de grafo conceitual em uma base de conhecimento. Representações mais sofisticadas, como roteiros (Seção 7.1.4), podem modelar situações mais complexas envolvendo eventos que ocorrem ao longo do tempo.

Uma vez que o programa tenha construído uma representação expandida do texto, ele pode responder inteligentemente a questões sobre o que leu. O programa analisa a questão transformando-a em uma representação interna, casando, então, a consulta com a representação expandida da história. Considere o exemplo da Figura 15.2. O programa leu a sentença “Tarzan beijou Jane” e construiu uma representação expandida.

Suponha que perguntemos ao programa “Quem ama Jane?”. Ao analisar essa consulta, o interrogativo, quem, o quê, por quê etc., indica a intenção da questão. Consultas *quem* questionam sobre o agente da ação; consultas *o quê* questionam sobre o objeto da ação; consultas *como* questionam sobre os meios pelos quais a ação foi realizada, e assim por diante. A pergunta “Quem ama Jane?” produz o grafo da Figura 15.15. O nó agente do grafo é marcado com um ? para indicar que ele é o objetivo da questão. Essa estrutura é, então, unida à representação expandida do texto original. O conceito que se torna ligado ao conceito pessoa: ? no grafo da consulta é a resposta à questão: “Tarzan ama Jane”.

**Figura 15.15** Grafo conceitual para a consulta Quem ama Jane?



### 15.5.2 Uma interface para banco de dados

O principal gargalo no projeto de programas para compreensão de linguagem natural é a aquisição de conhecimento suficiente sobre o domínio do discurso. A tecnologia atual é limitada a domínios restritos, com semânticas bem definidas. Uma área de aplicação que satisfaz esses critérios é a de desenvolvimento de interfaces com linguagem natural para bancos de dados. Embora os bancos de dados armazenem enormes quantidades de informação, essa informação é altamente regular e limitada em escopo; além disso, as semânticas dos bancos de dados são bem definidas. Essas características, com a utilidade de um banco de dados que pode aceitar consultas em linguagem natural, torna as interfaces de bancos de dados uma aplicação importante da tecnologia para a compreensão de linguagem natural.

A tarefa de uma interface de banco de dados é traduzir uma pergunta em linguagem natural em uma consulta bem formada na linguagem do banco de dados. Por exemplo, usando a linguagem de banco de dados SQL como alvo (Ullman, 1982), a interface em linguagem natural traduziria a pergunta “Quem contratou John Smith?” na consulta:

```

SELECT GERENTE
FROM GERENTE_DE_PESSOAL
WHERE EMPREGADO = 'John Smith'
  
```

Ao realizar essa tradução, o programa deve fazer mais do que traduzir a consulta original; ele precisa, também, decidir onde procurar no banco de dados (a relação GERENTE\_DE\_PESSOAL), o nome do campo a acessar (GERENTE) e as restrições da consulta (EMPREGADO = ‘John Smith’). Nenhuma dessas informações estava na pergunta original; elas foram encontradas em uma base de conhecimento que conhecia a organização do banco de dados e o significado de potenciais perguntas.

Um *banco de dados relacional* organiza os dados em relações envolvendo os domínios das entidades. Por exemplo, suponha que estejamos construindo um banco de dados de empregados e gostaríamos de acessar o salário de cada empregado e o gerente que o contratou. Esse banco de dados consistiria em três *domínios*, ou conjuntos de entidades: o conjunto de gerentes, o conjunto de empregados e o conjunto de salários. Poderíamos organizar esses dados em duas relações, *salário\_empregado*, que relaciona um empregado com o seu salário, e *gerente\_de\_pessoal*, que relaciona um empregado ao seu gerente. Em um banco de dados relacional, as relações são normalmente exibidas como tabelas que enumeram as ocorrências da relação. As colunas das tabelas normalmente são nomeadas; esses nomes são chamados de *atributos* da relação. A Figura 15.16 mostra as tabelas para as relações *salário\_empregado* e *gerente\_de\_pessoal*. *gerente\_de\_pessoal* tem dois atributos, o *empregado* e o *gerente*. Os valores da relação são os pares empregados e gerentes.

Se supusermos que cada empregado tem um único nome, um único gerente e um único salário, então o nome do empregado pode ser usado como *chave* tanto para o atributo salário como para o atributo gerente. Um atributo é uma chave para outro atributo se ele determinar unicamente o valor dos elementos para o outro atributo. Uma consulta válida indica um atributo-alvo e especifica um valor ou um conjunto de restrições; o banco de dados retorna os valores especificados do atributo-alvo. Podemos indicar o relacionamento entre chaves e outros atributos graficamente de várias formas, incluindo os *diagramas entidade-relacionamento* (Ullman, 1982) e os *diagramas de fluxo de dados* (Sowa, 1984). Essas técnicas exibem o mapeamento de chaves para atributos usando grafos direcionados.

**Figura 15.16** Duas relações em um banco de dados de empregados.

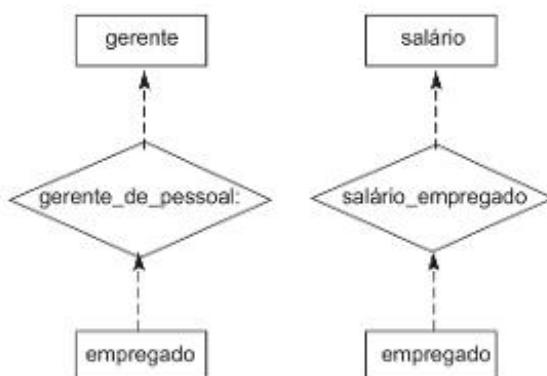
gerente_de_pessoal:		salário_empregado:	
empregado	gerente	empregado	Salário
John Smith	Jane Martinez	John Smith	\$ 35.000,00
Alex Barrero	Ed Angel	Alex Barrero	\$ 42.000,00
Don Morrison	Jane Martinez	Don Morrison	\$ 50.000,00
Jan Claus	Ed Angel	Jan Claus	\$ 40.000,00
Anne Cable	Bob Veroff	Anne Cable	\$ 45.000,00

Podemos estender os grafos conceituais de modo a incluírem diagramas desses relacionamentos (Sowa, 1984). A relação do banco de dados que define o mapeamento é indicada por um losango, que é rotulado com o nome da relação. Os atributos da relação são expressos como conceitos em um grafo conceitual, e a direção das setas indica o mapeamento de chaves para outros atributos. Os grafos entidade-relacionamento para as relações `salário_empregado` e `gerente_de_pessoal` podem ser vistos na Figura 15.17.

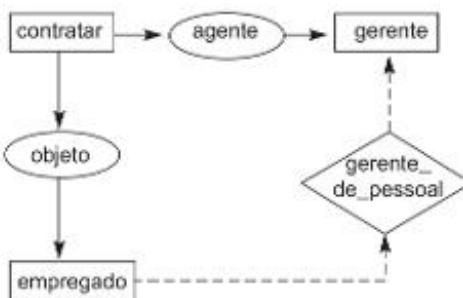
Ao traduzir do português para uma consulta formal, devemos determinar o registro que contém a resposta, o campo desse registro que deve ser retomado e os valores das chaves que determinam esse campo. Em vez de traduzir diretamente do português para a linguagem de banco de dados, traduzimos primeiro para uma linguagem mais expressiva, como os grafos conceituais. Isso é necessário porque muitas consultas em português são ambíguas ou requerem interpretação adicional para produzir uma consulta a um banco de dados bem formado. O uso de uma linguagem de representação mais expressiva ajuda nesse processo.

A interface por linguagem natural analisa e interpreta a consulta por um grafo conceitual, como descrito anteriormente neste capítulo. Ela combina esse grafo com a informação contida na base de conhecimento usando operações de junção e restrição. Nesse exemplo, queremos tratar perguntas como “Quem contratou John Smith?”, ou “Quanto ganha John Smith?”. Para cada consulta potencial, armazenamos um grafo que define o seu verbo, os papéis para esse verbo e todos os diagramas entidade-relacionamento relevantes para a pergunta. A Figura 15.18 mostra o conteúdo da base de conhecimento para o verbo “contratar”.

O interpretador semântico produz um grafo da pergunta do usuário e o junta ao conteúdo apropriado da base de conhecimento. Se houver um grafo de relação para entidade associado que mapeie chaves para o objetivo da pergunta, o programa pode usá-lo para formar uma consulta ao banco de dados. A Figura 15.19 mostra o grafo de

**Figura 15.17** Diagramas entidade-relacionamento das relações `gerente_de_pessoal` e `salário_empregado`.

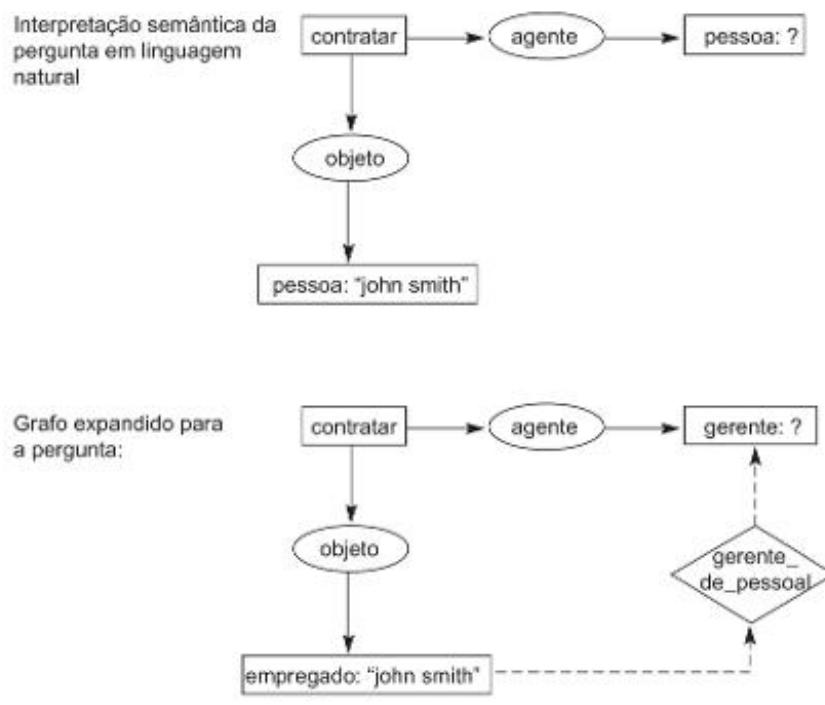
**Figura 15.18** Conteúdo da base de conhecimento para consultas sobre “contratação”.



consulta para a pergunta “Quem contratou John Smith?” e o resultado da junção desse grafo com o conteúdo da base de conhecimento da Figura 15.18. Ela mostra, também, a consulta SQL que é formada a partir desse grafo. Note que o nome do registro apropriado, o campo-alvo e a chave para a consulta não haviam sido especificados na pergunta em linguagem natural. Eles foram inferidos pela base de conhecimento.

Na Figura 15.18, sabia-se apenas que o agente e o objeto da consulta original eram do tipo pessoa. Para juntá-los com o conteúdo da base de conhecimento para contratar, eles foram inicialmente restritos aos tipos gerente e

**Figura 15.19** Desenvolvimento de uma consulta a um banco de dados a partir do grafo de uma entrada em linguagem natural.



Consulta na linguagem de banco de dados SQL:

```
SELECT GERENTE  
FROM GERENTE_DE_PESSOAL  
WHERE EMPREGADO = "john smith"
```

empregado, respectivamente. A hierarquia de tipos poderia, assim, ser usada para realizar verificações de tipo sobre a consulta original. Se john smith não fosse do tipo empregado, a pergunta seria inválida e o programa poderia detectar isso.

Uma vez que o grafo expandido da pergunta tenha sido construído, o programa examina o conceito-alvo, marcado com um ?, e determina que a relação gerente\_de\_pessoal mapeia uma chave para esse conceito. Como a chave está ligada a um valor de john smith, a pergunta é válida e o programa pode formar a consulta apropriada ao banco de dados. A tradução do grafo de entidade-relacionamento para SQL, ou alguma outra linguagem, é imediata.

Embora esse exemplo seja simples, ele ilustra o uso de uma abordagem baseada em conhecimento para construir uma interface para bancos de dados por linguagem natural. As ideias no nosso exemplo são expressas em grafos conceituais, mas elas poderiam ser mapeadas para outras representações, como *quadros* ou linguagens baseadas em lógica de predicados.

### 15.5.3 Um sistema para extração de informações para a web

A rede mundial de computadores (*World Wide Web*) oferece muitos desafios interessantes, bem como oportunidades para software de inteligência artificial e compreensão de linguagem natural. Um dos maiores desafios é a necessidade de software inteligente para resumir conteúdo “interessante” baseado na *web*. Na verdade, esse problema tem duas partes: localizar informações particularmente interessantes na *web* e depois extrair (ou minerar) componentes críticos dessa informação. Embora o primeiro problema de localizar informações potencialmente interessantes seja crítico (e muitas das técnicas dos capítulos 10, 13 e 15 sejam importantes), agora consideraremos a técnica de casamento e preenchimento para tratar do segundo problema.

Após localizar a informação, talvez por casamento de palavras-chave ou usando uma máquina de busca mais sofisticada, um *sistema de extração de informações* recebe como entrada esse texto sem restrições e, então, o resume em relação a um domínio pré-especificado ou tópico de interesse. Ele encontra informação útil sobre o domínio e codifica essa informação em uma forma adequada para relatar ao usuário ou para construir um banco de dados estruturado.

Diferentemente de um sistema para compreensão em profundidade de linguagem natural, os sistemas para extração de informações leem superficialmente o texto, procurando encontrar seções relevantes, focando apenas no processamento dessas seções. Propomos uma técnica baseada em modelo semelhante às técnicas de extração propostas por vários pesquisadores (Cardie, 1997; Cardie e Mooney, 1999). Uma visão geral de nosso sistema de extração de informações é apresentada nas figuras 15.20 e 15.21. Suponha que queiramos localizar e resumir a partir das informações da WWW relacionadas a cargos de professor universitário para ciência da computação. A Figura 15.20 apresenta um anúncio de emprego alvo e também apresenta um modelo da informação que queremos que o nosso software extraia desse e de outros anúncios semelhantes.

Nas primeiras tentativas de extração de informação, os sistemas de processamento de linguagem natural (PLN) eram bastante variados quanto à sua abordagem. Em um dos extremos estavam os sistemas que processavam texto usando ferramentas tradicionais: uma quebra sintática completa de cada sentença, acompanhada por uma análise semântica detalhada. Frequentemente, seguia-se ainda um processamento no nível do discurso. No outro extremo, havia os sistemas que usavam técnicas de casamento de palavras-chave, com pouco ou nenhum conhecimento, nem análise no nível linguístico. Entretanto, conforme foram sendo construídos e avaliados mais e mais sistemas, as limitações dessas duas abordagens extremas se tornaram óbvias. Uma arquitetura mais moderna para extração de informação, adaptada de Cardie (1997), é apresentada nas figuras 15.20 e 15.21. Embora os detalhes dessa arquitetura possam diferir em aplicações distintas, as figuras indicam as principais funções realizadas na extração.

Primeiro, cada sentença do sítio *web* “interessante” é desmembrada e rotulada. O rotulador estocástico apresentado na Seção 15.4 poderia ser usado para isso. O estágio de análise gramatical da sentença, na sequência, realiza a análise gramatical que produz grupos de substantivos, verbos, sintagmas e outras estruturas gramaticais. A seguir, a fase de extração encontra e rotula entidades semânticas relevantes para o tópico da extração. No nosso exemplo, isso irá identificar o nome do empregador, o local de trabalho, os requisitos do cargo (escolaridade, habilidades em computação, e assim por diante), a hora em que a entrevista começa etc.

**Figura 15.20** Amostra de texto, modelo de resumo e extração de informação para anúncio em ciência da computação.

Amostra de um anúncio de emprego em Ciência da Computação (trecho):

O Departamento de Ciência da Computação da University of New Mexico... está realizando uma seleção para preencher duas vagas para estágio probatório para cargos permanentes. Estamos interessados em contratar pessoas com interesse em:

Software, incluindo ferramentas de análise, de projeto e de desenvolvimento...

Sistemas, incluindo arquitetura, compiladores, redes...

...

Os candidatos devem ter completado um curso de doutorado em...

O departamento tem programas de pesquisa reconhecidos internacionalmente em computação adaptativa, inteligência artificial,... e mantém intensas colaborações científicas com o Santa Fe Institute e vários laboratórios nacionais...

Modelo parcialmente preenchido para a amostra:

**Empregador:** Departamento de Ciência da Computação, University of New Mexico

**Cidade do emprego:** Albuquerque

**Estado do emprego:** NM 87131

**Descrição do cargo:** Estágio probatório para cargo de professor universitário

**Qualificações para o cargo:** Doutorado em...

**Conhecimentos requeridos:** software, sistemas, ...

**Experiência em plataformas:** ...

**Informações sobre o empregador:** (texto anexo)

**Figura 15.21** Uma arquitetura para extração de informações, adaptada de Cardie (1997).

1. Texto:	O Departamento de Ciência da Computação da University of New Mexico está realizando uma seleção para preencher duas vagas para estágio probatório em cargos permanentes. Estamos interessados em contratar pessoas ...
2. Desmembramento e rotulação:	O/det Departamento/substantivo de/prep ...
3. Análise gramatical:	Departamento/sujeito está realizando/verbo busca/objeto Computação/substantivo
4. Extração:	Empregador: Departamento de Ciência da Computação Descrição do cargo: Estágio probatório para cargo de professor universitário ...
5. Correspondência:	estágio probatório para cargo permanente = professor universitário New Mexico = NM ...
6. Geração de modelo:	Como mostra a Figura 15.19

A fase de extração é o primeiro estágio do processo inteiramente específico para o domínio. Durante a extração, o sistema identifica relações específicas entre componentes relevantes do texto. No nosso exemplo, o Departamento de Ciência da Computação é visto como o empregador e o local do emprego é visto como University of New Mexico. A fase de combinação aborda questões como sinônima e resolução de anáforas. Exemplos de sinônimos são o estágio probatório para cargo permanente e o cargo de *professor universitário*, bem como *New Mexico* e *NM*. A resolução de anáforas liga *Departamento de Ciência da Computação* na primeira sentença com *nós* na segunda sentença.

As inferências no nível do discurso feitas durante a fase de combinação ajudam a fase de geração do modelo, que determina o número de relacionamentos distintos no texto, mapeia essas porções de informação extraídas para cada campo do modelo e produz o modelo de saída final.

Apesar do progresso recente, os sistemas de extração de informação atuais ainda têm problemas. Primeiro, a precisão e a robustez desses sistemas podem ser enormemente melhoradas, já que os erros de extração parecem se originar numa compreensão muito superficial do texto de entrada. Segundo, pode ser difícil e demorado construir um sistema de extração de informação em um novo domínio (Cardie, 1997). Esses dois problemas estão relacionados à natureza específica do domínio da tarefa de extração. O processo de extração de informação pode melhorar se as suas fontes de conhecimento forem ajustadas ao domínio particular, mas a modificação manual do conhecimento linguístico específico do domínio é difícil e suscetível a erros.

Apesar disso, existe atualmente uma série de aplicações interessantes. Glasgow et al. (1997) construíram um sistema para apoiar agentes de seguro na análise de aplicações de seguro de vida. Soderland et al. (1995) têm um sistema para extrair sintomas, achados físicos, resultados de testes e diagnósticos, a partir de registros de pacientes para processamento de seguros. Há programas para analisar jornais a fim de encontrar e resumir negócios como *joint ventures* (MUC-5, 1994), sistemas que classificam automaticamente documentos legais (Holowczak e Adam, 1997) e programas que extraem informação de listagens de emprego em computação (Nahm e Mooney, 2000).

#### 15.5.4 Usando algoritmos de aprendizado para generalizar a informação extraída

Outra aplicação combina várias ideias apresentadas neste capítulo, bem como algoritmos de aprendizado de máquina (Seção 10.3). Cardie e Mooney (1999) e Nahm e Mooney (2000) sugeriram que a informação extraída de um texto pode ser generalizada por algoritmos de aprendizado de máquina e que o resultado pode ser reusado na tarefa de extração de informação.

A abordagem é simples. Modelos de resumo de texto, preenchidos completa ou parcialmente, como visto, por exemplo, na Figura 15.20, são coletados de sítios apropriados da web. A informação resultante desses modelos é, então, armazenada em um banco de dados relacional, onde algoritmos de aprendizado, como ID3 ou C4.5, são usados para extrair árvores de decisão, que, como visto na Seção 9.3, podem refletir relacionamentos de regras implícitos nos conjuntos de dados. (Essa técnica é denominada *mineração de dados* — do inglês *data mining*.)

Mooney e seus colegas propõem que esses relacionamentos descobertos sejam usados para refinar os modelos e as estruturas de conhecimento originais usados na extração de informação. Entre os exemplos desse tipo de informação que poderiam ser descobertos da aplicação de análise de empregos em Ciência da Computação, mostrada na Seção 15.5.3, poderiam se incluir: se o cargo é o de professor em Ciência da Computação, então não é necessária experiência em uma plataforma computacional em particular; se as universidades estão contratando professores, então é necessária experiência em pesquisa etc. Outros detalhes podem ser encontrados em Nahm e Mooney (2000).

Há um grande número de questões atuais interessantes em linguística computacional que estão fora do nosso objetivo a tratar. Concluimos com três questões abertas. Primeira, qual é a unidade fundamental para se compreender a linguagem humana? Várias tentativas foram feitas para responder a essa pergunta, desde a análise completa da sentença na linguagem natural, tentativas em reconhecimento de palavras isoladas, até o foco em fonemas individuais. Será que o foco na silaba é a chave para se compreender a linguagem humana? Quanto da palavra precisamos ouvir antes que ela seja reconhecida? Qual é a granularidade ideal para a análise do discurso falado (Greenberg, 1999)?

A segunda é: o acesso aos conceitos em uma base de conhecimento por meio do reconhecimento de sons pode levar a mais condicionamento para a interpretação subsequente da linguagem? Suponha que a sílaba seja uma unidade útil para a análise de linguagem. Suponha que os bigramas de sílaba sejam úteis para sugerir as palavras de um interlocutor. Essas palavras podem ser mapeadas em conceitos de uma base de conhecimento e o conhecimento relacionado pode ser usado para aprimorar a interpretação das sílabas subsequentes? Alguém que liga para um agente de viagens computadorizado poderia reconhecer as sílabas sugerindo um nome de cidade, e o computador poderia então antecipar questões relacionadas à visita a essa cidade.

Por fim, a última, que é um objetivo a ser atingido a longo prazo em processamento da linguagem natural: a criação da web é semântica; como isso acontecerá? Ou ela acontecerá? Existe alguma metodologia sob a qual os computadores podem “compreender” alguma coisa? Ou sempre haverá “truques” controlados por padrões, baseados em busca, que são bons o suficiente para “convencer” as pessoas de que o computador as comprehende (o teste de Turing)? Será que esse método “bom o suficiente” não é o que nós humanos fazemos um com o outro de qualquer forma? O capítulo final continua esta e outras discussões relacionadas.

## 15.6 Epílogo e referências

Como sugere este capítulo, há várias abordagens para se definir gramáticas e analisadores de sentenças em linguagem natural. Apresentamos os analisadores ATN e os modelos de Markov como exemplos típicos dessas abordagens. O leitor interessado deve ficar atento a outras possibilidades. Entre elas estão as *gramáticas transformacionais, gramáticas semânticas, gramáticas de casos e gramáticas funcionais e de características* (Winograd, 1983; Allen, 1995; Jurafsky e Martin, 2008).

As gramáticas transformacionais usam regras livres de contexto para representar a *estrutura profunda*, ou significação, da sentença. Essa estrutura profunda pode ser representada como uma árvore de análise que não apenas consiste de terminais e não terminais, mas inclui, também, um conjunto de símbolos denominados *marcadores gramaticais*, que representam características, como número, tempo e outros aspectos sensíveis ao contexto da estrutura linguística. A seguir, um conjunto de regras do mais alto nível, chamadas de regras transformacionais, transformam essa estrutura profunda em uma *estrutura superficial*, que está mais próxima da forma real que a sentença terá. Por exemplo, “Tom gosta de Jane” e “Jane é querida por Tom” têm a mesma estrutura profunda, mas têm estruturas superficiais diferentes.

As regras transformacionais agem sobre as próprias árvores de análise, realizando as verificações que requerem contextualização global, e produzem uma estrutura superficial adequada. Por exemplo, uma regra transformacional pode verificar que a característica de número do nó que representa o sujeito de uma sentença é a mesma característica de número do nó do verbo. Regras transformacionais podem, também, mapear uma estrutura profunda única em estruturas superficiais alternativas, por exemplo, modificando a voz de ativa para passiva, ou transformando uma forma declarativa em interrogativa. Embora as gramáticas transformacionais não sejam discutidas neste livro, elas são uma alternativa importante às gramáticas de estrutura frasal expandida.

Terry Winograd oferece um tratamento abrangente de gramáticas e análise de linguagem natural em *Language as a Cognitive Process* (1983). Esse livro oferece um tratamento particularmente completo das gramáticas transformacionais. *Natural Language Understanding*, de James Allen (1987, 1995), fornece uma visão geral do projeto e da implementação de programas para a compreensão de linguagem natural. *Introduction to Natural Language Processing*, de Mary Dee Harris (1985), é outro texto geral sobre linguagem natural, expandindo as questões levantadas neste capítulo. Recomendamos, também, *Natural Language Processing in PROLOG*, de Gerald Gazdar e Chris Mellish (1989). Charniak (1993) e Charniak et al. (1993) tratam de questões sobre métodos estocásticos para rotulação de parte do discurso.

A análise semântica de linguagem natural envolve uma série de questões difíceis que são abordadas em representação de conhecimento (Capítulo 7). Em *Computational Semantics*, Charniak e Wilks (1976) apresentam artigos que tratam dessas questões. Por causa da dificuldade em modelar o conhecimento e o contexto social requerido,

dos para interação em linguagem natural, muitos autores questionaram a possibilidade de expandir essa tecnologia para além de domínios restritos. A pesquisa inicial sobre análise de discurso pode ser encontrada em Linde (1974), Grosz (1977) e Grosz e Sidner (1990). *Understanding Computers and Cognition*, de Winograd e Flores (1986); *Minds, Brains, and Programs*, de John Searle (1980); e *On the Origin of Objects* (Smith, 1996) tratam dessas questões.

*Inside Computer Understanding*, de Schank e Riesbeck (1981), discute a compreensão de linguagem natural usando a tecnologia da dependência conceitual. *Scripts, Plans, Goals and Understanding*, de Schank e Abelson (1977), discute o papel de estruturas de alto nível para organização de conhecimento em programas de linguagem natural.

*Speech Acts*, de John Searle (1969), discute o papel da pragmática e do conhecimento contextual na modelagem do discurso. Fass e Wilks (1983) propuseram a *teoria da preferência semântica* como um veículo para modelar a semântica em linguagem natural. A preferência semântica é uma generalização das gramáticas de caso que permite transformações sobre quadros de casos. Isso oferece maior flexibilidade para representar a semântica e permite a representação de conceitos como metáfora e analogia. Para uma discussão completa da hierarquia de Chomsky, veja Hopcroft e Ullman (1979). Somos gratos a John Sowa (1984) em função do nosso tratamento dos grafos conceituais.

Apresentações recentes de técnicas de processamento de linguagem podem ser encontradas em *Speech and Language Processing*, de Jurafsky e Martin (2008); *Foundations of Statistical Natural Language Processing*, de Manning e Schütze (1999); e *Survey of the State of the Art in Human Language Technology*, Cole (1997) e na edição de inverno de 1997 da *AI Magazine*.

Existem agora diversos produtos comerciais muito interessantes para diversos aspectos da compreensão e da geração de linguagem baseadas em computador, bem como a mineração de dados baseada na linguagem. É muito bom experimentar esses produtos, mas, como eles normalmente não são de fonte aberta, não os discutimos aqui.

Recomendamos como referências excelentes para se manter atualizado nas tendências de pesquisa em compreensão de linguagem natural, dos pontos de vista tanto tradicional quanto estocástico, os anais das conferências anuais em IA: *AAAI* e *IJCAI*, publicados pela AAAI Press por meio da MIT Press. A *Association of Computational Linguistics* também possui suas configurações anuais, nacionais e internacionais, além do *Journal of the Association for Computational Linguistics*. Essas são importantes fontes de tecnologia atual.

## 15.7 Exercícios

1. Classifique cada uma das sentenças a seguir como sendo sintaticamente incorreta; sintaticamente correta, mas sem sentido; com sentido, mas falsa; ou verdadeira. Onde é que cada um desses problemas é encontrado no processo de compreensão?

Ideias verdes desbotadas dormem furiosamente.

Moscas da fruta gostam de uma banana.

Cães os mordem homem um.

Cristóvão Colombo descobriu o Brasil.

Este exercício é fácil.

Eu quero estar no fundo do mar à sombra em um jardim de polvos.

2. Discuta as estruturas representacionais e o conhecimento necessário para compreender as seguintes sentenças.

O cão marrom comeu o osso.

Acople a roda grande ao eixo com o parafuso hexagonal.

Mary regou as plantas.

O espírito é forte, mas a carne é fraca.

Meu reino por um cavalo!

3. Analise cada uma das sentenças a seguir usando a gramática do “mundo dos cães” da Seção 15.2.1. Quais delas são inválidas? Por quê?
- O cão morde o cão.  
O cão grande morde o homem.  
Emma gosta do menino.  
O homem gosta.  
Morda o homem.
4. Estenda a gramática do mundo dos cães de modo que ela inclua as sentenças inválidas do Exercício 3.
5. Analise cada uma das sentenças a seguir usando a gramática sensível ao contexto da Seção 15.2.3.
- O homem gosta do cão.  
O cão morde o homem.
6. Produza uma árvore de análise para cada uma das sentenças a seguir. Você precisará estender a nossa gramática simples com estruturas linguísticas mais complexas, como advérbios, adjetivos e sintagmas preposicionados. Se uma sentença tiver mais que uma análise possível, desenhe os diagramas de todas elas e explique a informação semântica que seria usada para escolher uma análise específica.
- O tempo passa rápido como uma flecha, mas eu como uma banana.  
Tom deu o grande livro vermelho para Mary na terça-feira.  
O raciocínio é uma arte, e não uma ciência.  
Errar é humano, mas perdoar é divino.
7. Estenda a gramática do mundo dos cães para incluir adjetivos em sintagmas nominais. Certifique-se de que seja permitido um número indeterminado de adjetivos. Dica: use uma regra recursiva, `lista_adjetivo`, que esteja vazia ou contenha um adjetivo seguido de uma lista de adjetivos. Mapeie essa gramática para redes de transição.
8. Acrescente as seguintes regras de gramática livre de contexto à gramática do mundo dos cães da Seção 15.2.1. Mapeie a gramática resultante para redes de transição.
- `sentença ↔ sintagma_nominal sintagma_verbal sintagma_preposicionado`  
`sintagma_preposicionado ↔ preposição sintagma_nominal`  
`preposição ↔ com`  
`preposição ↔ para`  
`preposição ↔ sobre`
9. Defina um analisador ATN para a gramática do mundo dos cães com adjetivos (Exercício 7) e sintagmas preposicionais (Exercício 8).
10. Defina conceitos e relações em grafos conceituais necessários para representar o significado da gramática do Exercício 9. Defina os procedimentos para construir uma representação semântica a partir da árvore de análise.
11. Estenda a gramática sensível ao contexto da Seção 15.2.3 para testar a concordância semântica entre o sujeito e o verbo. Especificamente, os homens não devem morder os cães, embora cães possam gostar ou morder homens. Realize uma modificação similar para a gramática ATN.
12. Expanda a gramática ATN da Seção 15.2.4 para incluir perguntas com `quem` e `o que`.
13. Descreva como os modelos de Markov da Seção 15.4 poderiam ser combinados com as abordagens mais simbólicas para a compreensão de linguagem das seções 15.1-15.3. Descreva como os modelos estocásticos poderiam ser combinados com os sistemas tradicionais baseados em conhecimento do Capítulo 8.
14. Estenda o exemplo de interface de banco de dados da Seção 15.5.2 de modo que ela possa responder perguntas da forma “Quanto ganha Don Morrison?”. Você deverá estender a gramática, a linguagem de representação e a base de conhecimento.
15. Tome o problema anterior e coloque as palavras, incluindo a pontuação, em uma ordem aleatória.
16. Suponha que os gerentes estejam listados na relação `salário_empregado` com outros empregados no exemplo da Seção 15.5.2. Estenda o exemplo de modo que ele trate consultas como “Encontre qualquer empregado que ganhe mais que seu gerente”.

17. Como poderiam ser combinadas as abordagens estocásticas da Seção 15.4 com as técnicas para análise de bancos de dados discutidas na Seção 15.5.2?
18. O uso da abordagem estocástica para descobrir padrões em um banco de dados relacional é uma área importante de pesquisa atual, algumas vezes denominada *mineração de dados* (ver Seção 10.3). Como esse trabalho poderia ser usado para responder perguntas, como aquelas apresentadas na Seção 15.5.2 sobre bancos de dados relacionais?
19. Como projeto, construa um sistema de extração de informação para um domínio de conhecimento a ser usado na *web*. Para sugestões, veja a Seção 15.5.3.
20. Pela Sala Virtual deste livro, tome a estrutura dos analisadores livres de contexto e sensíveis ao contexto em Prolog e acrescente adjetivos, advérbios e sintagmas proposicionais à gramática.
21. Pela Sala Virtual deste livro, tome a estrutura do analisador probabilístico livre de contexto em Prolog e acrescente adjetivos, advérbios e sintagmas proposicionais à gramática.

## Epílogo

*O potencial da ciência da computação, se totalmente explorado e desenvolvido, nos levará a um plano mais alto de conhecimento do mundo. A ciência da computação nos ajudará a ganhar uma maior compreensão dos processos intelectuais. Ela melhorará nosso conhecimento do processo de aprendizado, do processo do pensamento e do processo do raciocínio. A ciência da computação nos fornecerá modelos e ferramentas conceituais para as ciências cognitivas. Assim como as ciências físicas dominaram os empreendimentos intelectuais da humanidade durante este século quando os pesquisadores exploravam a natureza da matéria e a origem do universo, hoje em dia nós estamos iniciando a exploração do universo intelectual das ideias, estruturas do conhecimento e da linguagem. Eu prevejo que continuarão sendo feitos avanços significativos que alterarão as nossas vidas enormemente... Eu posso prever um entendimento de como organizar e manipular o conhecimento...*

—J. HOPCROFT, ACM Turing Award Lecture, 1987

*What is mind? No matter.  
What is matter? Never mind...*

—HOMER SIMPSON

*O que importa é o que aprendemos depois de conhecermos tudo.*

—EARL WEAVER, Baltimore Orioles

### Reflexões sobre a natureza da inteligência

Embora este livro tenha flirtado com as implicações filosóficas mais elevadas da inteligência artificial, enfatizamos fundamentalmente as técnicas de engenharia usadas para construir artefatos inteligentes baseados em computação. Nestas páginas finais, gostaríamos de retornar àquelas questões mais profundas, para considerarmos os fundamentos filosóficos da inteligência artificial, para reavaliarmos a possibilidade de uma ciência da inteligência usando técnicas de IA e para especularmos sobre o progresso futuro da disciplina.

Como notamos em todo este livro, a pesquisa sobre a cognição humana e a resolução de problemas têm dado contribuições importantes à teoria da inteligência artificial e ao projeto de programas de IA. E, reciprocamente, o trabalho em IA serviu de inspiração tanto para a construção de modelos como para o teste empírico em muitas disciplinas como a biologia, a linguística e a psicologia cognitiva. Ao concluirmos nossa apresentação, gostaria-

mos de reviver esse espírito eclético e considerar questões como os limites das representações, a importância da materialização física para os processos mentais e o papel da cultura no crescimento e na interpretação do conhecimento. Essas questões nos levam a outras questões científicas e filosóficas, como aquelas que tratam da falibilidade de modelos e da natureza e capacidades do próprio método científico. Nossas observações nos levam a defender uma abordagem interdisciplinar que combine os trabalhos em IA com as descobertas em psicologia, linguística, biologia, antropologia, epistemologia e em outros campos que exploram o espectro completo do pensamento humano e seus produtos. Acreditamos que explorando as interseções dessas disciplinas, bem como as tensões entre elas, poderemos compreender melhor os processos subjacentes à inteligência, quer sejam eles baseados na biologia ou na mecânica.

Tradicionalmente, o trabalho em inteligência artificial tem se baseado na hipótese do sistema simbólico físico (Newell e Simon, 1976). A pesquisa a partir dessa perspectiva produziu estruturas de dados e estratégias de busca cada vez mais sofisticadas, que, por sua vez, levaram a muitos sucessos importantes, tanto ao criar ferramentas que podem alcançar aspectos do comportamento inteligente, como também ao iluminar os diversos componentes que constituem a inteligência humana. Entretanto, é importante notar que muito da prática de IA que emergiu da sua abordagem inicial reposava em suposições que derivam do racionalismo filosófico. Como definido pela tradição racionalista, a própria inteligência é amplamente vista como um processo de raciocínio lógico, de solução científica de problemas e uma abordagem empírica, imediatista, para a compreensão do universo. Somos da opinião que o racionalismo filosófico restringiu excessivamente tanto os métodos atuais de inteligência artificial como o escopo de seus questionamentos.

Em todo este livro, apresentamos muitos desenvolvimentos recentes, incluindo modelos alternativos (probabilísticos) de aprendizado, resolução de problemas baseada em agentes e distribuída, abordagens para a inteligência corporificada e situada, bem como noções de computação evolucionária e vida artificial. Essas abordagens ao entendimento da inteligência oferecem alternativas muito desejadas ao reducionismo racionalista. Os modelos biológicos e sociais da inteligência mostraram que a inteligência humana é, em grande parte, produto de nossos corpos e sentidos, de nossas instituições culturais e sociais, da arte que criamos e apreciamos, das histórias que ouvimos e passamos adiante. Estabelecendo métodos para construir simulações computacionais de muitos desses processos complexos, como a evolução ou a adaptação de padrões neurais no cérebro humano, essas abordagens recentes têm dado à IA um novo e poderoso conjunto de ferramentas para complementar as suas técnicas mais tradicionais.

A inteligência artificial, como a maior parte da ciência da computação, é um campo jovem. Enquanto a física e a biologia podem medir o seu progresso em séculos, a computação moderna ainda conta sua idade em décadas. No Capítulo 16, tentamos integrar as descobertas das diferentes abordagens de IA em uma ciência unificada de sistemas inteligentes. Propomos que ciência e engenharia, filosofia e nossos julgamentos estéticos precisam nos guiar na nossa criação contínua de novos artefatos e experimentos, os quais, quando usados apropriadamente, podem oferecer uma compreensão da ciência mais geral dos sistemas inteligentes. Esse capítulo continua a longa tradição introduzida no Capítulo 1 de estabelecer uma base epistemológica para a IA, não tanto para responder aos seus críticos (de fato, muitos de seus desafios ainda aguardam por respostas), mas no sentido positivo de tentar explorar e iluminar o caminho à frente.

# *In inteligência artificial como investigação empírica*

*A ciência da computação é uma disciplina empírica. Nós a teríamos chamado de ciência experimental, mas assim como a astronomia, a economia e a geologia, algumas de suas formas únicas de observação e experimentação não se enquadram em um estereótipo estreito do método experimental. Apesar disso, elas lidam com experimentos. Cada nova máquina que é construída é um experimento. A construção real de uma máquina coloca uma questão à natureza; e recebemos a resposta observando a máquina em operação e analisando-a por todos os meios analíticos e de medida disponíveis. Cada novo programa que é construído é um experimento. Ele coloca uma questão à natureza e seu comportamento oferece indícios de uma resposta. Nem máquinas nem programas são caixas-pretas; eles são artefatos que foram projetados, tanto em hardware como em software, e podemos abri-los e examinar o seu interior. Nós podemos relacionar sua estrutura ao seu comportamento e tirar muitas lições de um único experimento.*

—A. NEWELL e H. A. SIMON, ACM Turing Award Lecture, 1976

*O estudo de máquinas que pensam nos ensina mais sobre o cérebro do que poderíamos aprender por métodos introspectivos. O homem ocidental está se exteriorizando na forma de dispositivos eletrônicos.*

—WILLIAM S. BURROUGHS, *Naked Lunch*

*Onde está o conhecimento que perdemos com a informação?*

—T. S. ELIOT, *Choruses from the Rock*

## 16.0 Introdução

Para muitos, os aspectos mais surpreendentes do trabalho em inteligência artificial estão relacionados com a extensão com que a IA, e mesmo muito da ciência da computação, se revela como uma disciplina empírica. Isso é surpreendente, porque a maioria das pessoas inicialmente vê esses campos em termos de seus fundamentos matemáticos, ou alternativamente, de engenharia. Do ponto de vista matemático, algumas vezes denominado perspectiva “limpa”, existe o desejo racionalista de levar padrões de prova e análise ao projeto de dispositivos computacionais inteligentes. Da perspectiva de engenharia, ou “suja”, a tarefa é muitas vezes vista como simplesmente construir artefatos bem-sucedidos que a sociedade deseja chamar de “inteligentes”. Infelizmente, ou felizmente, dependendo da filosofia do leitor, a complexidade do software inteligente e as ambiguidades inerentes às suas interações com os mundos da natureza e da atividade humana frustram a análise tanto da perspectiva puramente matemática como da puramente de engenharia.

Além disso, se o objetivo da inteligência artificial for alcançar o nível de uma ciência e se tornar um componente crítico da *ciência dos sistemas inteligentes*, deve-se incluir uma mistura de métodos analíticos e empíricos no projeto, execução e análise de seus artefatos. Desse ponto de vista, cada programa de IA pode ser visto como um experimento: ele propõe uma questão ao mundo natural e os resultados de executar esse programa são a resposta da

natureza. A resposta da natureza aos nossos comprometimentos de projeto e programação molda nosso entendimento do formalismo, do mecanismo e, finalmente, da natureza da própria inteligência (Newell e Simon, 1976).

Diferentemente de vários estudos mais tradicionais da cognição humana, os projetistas de artefatos computacionais inteligentes podem inspecionar o funcionamento interno de nossos “objetos”. Pode-se interromper a execução de um programa, examinar o estado interno e modificar sua estrutura conforme desejar. Como observaram Newell e Simon, a estrutura dos computadores e de seus programas indica seu comportamento potencial: eles podem ser examinados e suas representações e algoritmos de busca podem ser entendidos. O poder dos computadores como ferramenta para compreender a inteligência é um produto dessa dualidade. Computadores programados adequadamente são capazes de alcançar níveis de complexidade semântica e comportamental que requerem que sejam caracterizados em termos psicológicos, assim como de oferecer uma oportunidade para uma inspeção de seus estados internos que é, em grande parte, negada aos cientistas que estudam outras formas de vida intelectuais.

Felizmente para a continuidade do trabalho em IA, bem como para estabelecer uma ciência de sistemas inteligentes, técnicas psicológicas mais modernas, especialmente aquelas relacionadas com a neuropsicologia, lançaram nova luz sobre os diversos modos de inteligência humana. Sabemos agora, por exemplo, que a função inteligente humana não é monolítica e uniforme. Em vez disso, ela é modular e distribuída. O seu poder é visto nos órgãos dos sentidos, como a retina humana, que podem coletar e pré-processar informação visual. De modo semelhante, aprendizado não é uma faculdade uniforme e homogênea. Ele é uma função de múltiplos ambientes e sistemas diferentes, cada um adaptado para alcançar objetivos especializados. A análise MRI, junto a varreduras PET, EEG e aliadas a procedimentos de aquisição de imagens físicas neurais, suportam uma imagem diversa e cooperativa das funcionalidades internas dos sistemas inteligentes reais.

Se o trabalho em IA deve alcançar o nível de uma ciência, devemos tratar também questões filosóficas importantes, especialmente aquelas relacionadas com a epistemologia, ou as questões de como um sistema inteligente “conhece” o seu mundo. Essas questões abordam desde o que é o objeto de estudo da inteligência artificial, até questões mais profundas, como o questionamento da validade e da utilidade da hipótese do sistema físico simbólico. Outras questões incluem o que é um “símbolo” na abordagem simbólica de IA e como os símbolos se relacionariam a conjuntos de nós ponderados em um modelo conexionista. Questionamos, também, o papel do racionalismo expresso no viés indutivo visto na maioria dos programas de aprendizado e como isso pode ser comparado com a falta irrestrita de estrutura frequentemente vista nas abordagens de aprendizado não supervisionado, por reforço e emergente. Finalmente, devemos questionar o papel da personificação, da localização e do viés social na solução de problemas. Concluímos nossa discussão das questões filosóficas propondo uma *epistemologia construtivista* que se enquadra confortavelmente tanto no nosso comprometimento à IA como ciência bem como à IA como investigação empírica.

E assim, neste capítulo final, retornamos às questões colocadas no Capítulo 1: O que é inteligência? Ela pode ser formalizada? Como podemos construir mecanismos que exibem inteligência? Como a inteligência humana e a artificial se enquadram em um contexto mais amplo de uma ciência dos sistemas inteligentes? Na Seção 16.1, iniciamos com uma definição de inteligência artificial revisada, que mostra como o trabalho atual em IA, embora enraizado na hipótese do sistema físico simbólico de Newell e Simon, estendeu suas ferramentas, técnicas e investigações para um contexto muito mais amplo. Exploramos essas abordagens alternativas à questão da inteligência e consideramos seu potencial para o projeto de máquinas inteligentes e para ser um componente da ciência dos sistemas inteligentes. Na Seção 16.2, esclarecemos quanto das técnicas de psicologia cognitiva moderna, da neuropsicologia, bem como da epistemologia, pode ser usado para melhor compreender o empreendimento da inteligência artificial.

Finalmente, na Seção 16.3, discutimos alguns dos desafios que restam para os praticantes da moderna IA, bem como para os epistemólogos. Muito embora as abordagens tradicionais de IA tenham sido culpadas por um reducionismo racionalista, novas visões e ferramentas interdisciplinares também apresentam deficiências. Por exemplo, os criadores dos algoritmos genéticos e os pesquisadores em vida artificial definem o mundo da inteligência de um ponto de vista darwiniano: “O que existe, é o que sobrevive”. Conhecimento é visto também como “saber como”, em vez de “saber o que” em um mundo complexo situado. Para um cientista, perguntas requerem explicações, e “sucesso” e “sobrevivência” não são, por si só, suficientes.

Neste capítulo final, discutiremos o futuro da IA explorando as questões filosóficas que devem ser tratadas para criar uma ciência computacional da inteligência. Concluímos que a metodologia empírica da IA é uma ferramenta importante, e talvez a melhor, para explorar a natureza da inteligência.

## 16.1 Inteligência artificial: uma definição revisada

### 16.1.1 A inteligência e a hipótese do sistema físico simbólico

Com base em nossa experiência nos últimos 15 capítulos, oferecemos uma definição revisada de inteligência artificial:

IA é o estudo dos mecanismos subjacentes ao comportamento inteligente por meio da construção e da avaliação de artefatos que tentam representar esses mecanismos.

Por essa definição, inteligência artificial é menos uma teoria sobre os mecanismos subjacentes à inteligência e mais uma metodologia empírica para construir e testar possíveis modelos para suportar tal teoria. Ela é um comprometimento com o método científico de projetar, executar e avaliar experimentos com o objetivo de refinar o modelo e continuar a experimentar. Entretanto, o mais importante é que essa definição, assim como o próprio campo da IA, ataca diretamente séculos de obscurantismo filosófico sobre a natureza da mente. Ela dá às pessoas que desejam compreender o que é a nossa característica provavelmente mais definitiva como seres humanos uma alternativa à religião, à superstição, ao dualismo cartesiano, aos placebos modernos ou à busca de inteligência em alguma idiossincrasia ainda não descoberta da mecânica quântica (Penrose, 1989). Se a ciência que fundamenta a inteligência artificial fez alguma contribuição ao conhecimento humano, esta se deu na confirmação de que inteligência não é um vapor místico que permeia os homens e os anjos, mas sim o efeito de um conjunto de princípios e mecanismos que podem ser entendidos e aplicados no projeto de máquinas inteligentes. Devemos notar que nossa definição revisada *não define* inteligência; em vez disso, ela propõe um papel coerente para a inteligência *artificial* para explorar a natureza e a expressão dos fenômenos inteligentes.

Historicamente, a abordagem dominante para a inteligência artificial tem envolvido a construção de formalismos representacionais e seus mecanismos de raciocínio associados baseados em busca. O princípio condutor da metodologia inicial da IA é a hipótese do *sistema físico simbólico*, articulada originalmente por Newell e Simon (1976). Essa hipótese afirma que:

A condição necessária e suficiente para um sistema físico exibir ação inteligente geral é que ele seja um sistema físico simbólico.

*Suficiente* significa que a inteligência pode ser alcançada por um sistema físico simbólico qualquer, organizado apropriadamente.

*Necessário* significa que qualquer agente que exiba inteligência geral deve ser uma ocorrência de um sistema físico simbólico. A necessidade da hipótese do sistema físico simbólico requer que um agente inteligente qualquer, seja ele humano, alienígena ou um computador, alcance a inteligência por meio da implementação física de operações sobre estruturas simbólicas.

A *ação inteligente geral* significa o mesmo escopo de ação observado na ação humana. Dentro de limites físicos, o sistema exibe um comportamento apropriado para os seus fins e um adaptativo para as demandas do seu ambiente.

Newell e Simon resumiram os argumentos de *necessidade* e *suficiência* dessa hipótese (Newell e Simon, 1976; Newell, 1981; Simon, 1981). Nos anos seguintes, tanto a IA como a ciência cognitiva exploraram o território delineado por essa hipótese.

A hipótese do sistema físico simbólico levou a quatro compromissos metodológicos significativos: (a) o uso de símbolos e sistemas de símbolos como meio para descrever o mundo; (b) o projeto de mecanismos de busca, especialmente busca heurística, para explorar o espaço de potenciais inferências que esses sistemas simbólicos poderiam suportar; (c) o abandono da personificação da arquitetura cognitiva, ou seja, supôs-se que um sistema sim-

bólico projetado apropriadamente poderia fornecer uma descrição causal completa da inteligência, independentemente de seu meio de implementação. Finalmente, (d) por esse ponto de vista, a IA se tornou empírica e construtivista: ela procurou compreender a inteligência construindo modelos funcionais dela.

Na visão do sistema de símbolos, os termos de uma linguagem, referidos como *símbolos*, foram usados para denotar ou referenciar algo diferente do que eles mesmos. Como termos verbais em uma linguagem natural, os símbolos representavam ou se referiam a coisas em um mundo de um agente inteligente. Tarski (1956, Seção 2.3) pôde oferecer a possibilidade de uma *ciência da significação* nesses relacionamentos entre objeto e referente.

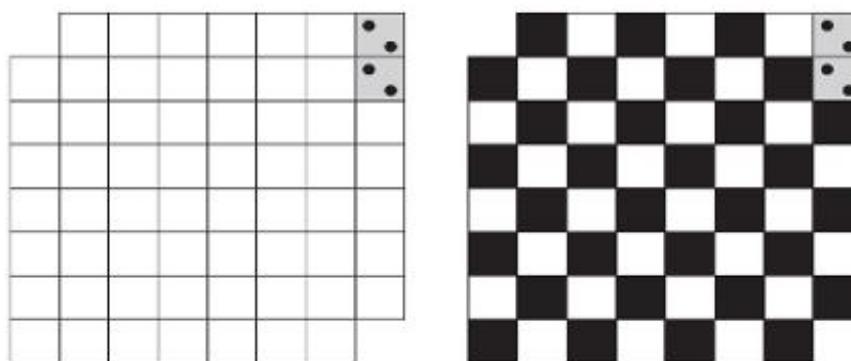
Mais do que isso, o uso de símbolos pela IA vai além das questões abordadas em uma semântica de Tarski, estendendo símbolos à representação de todas as formas de conhecimento, habilidades, intenção e causalidade. Todos esses esforços construtivos se baseiam no fato de que símbolos com sua semântica podem ser embutidos em um sistema formal. Eles definem uma *linguagem de representação*. Essa capacidade de formalizar modelos simbólicos é essencial para a modelagem da inteligência como a execução de um programa de computador. Em capítulos anteriores, estudamos várias representações em detalhe: o cálculo de predicados, as redes semânticas, os roteiros, os grafos conceituais, quadros e objetos. A matemática dos sistemas formais nos permite tratar questões como consistência, completude e complexidade, bem como discutir a organização do conhecimento. A evolução dos formalismos representacionais nos permitiu estabelecer relacionamentos semânticos mais complexos (ricos). Os sistemas de herança, por exemplo, constituem uma teoria semântica de conhecimento taxonômico. Definindo formalmente uma herança de classes, tais linguagens simplificam a construção de programas inteligentes e fornecem modelos testáveis da organização de categorias possíveis de inteligência em si.

Intimamente ligada aos esquemas representacionais e seu uso em raciocínio está a noção de busca. Busca é o exame passo a passo de estados de problema dentro do arcabouço de espaço de estados (e comprometido semanticamente *a priori*) procurando soluções, objetivos de subproblemas, simetrias do problema, ou qualquer outro aspecto do problema considerado. Representação e busca estão ligadas entre si, porque um compromisso com uma representação em particular determina um espaço de estados a ser buscado. De fato, alguns problemas podem se tornar mais difíceis ou mesmo impossíveis em decorrência da má escolha de uma linguagem de representação. A discussão sobre o viés indutivo, que veremos mais adiante neste capítulo, ilustra esse ponto.

Um exemplo dramático e frequentemente citado do inter-relacionamento entre busca e representação, bem como da dificuldade de escolher uma representação adequada (seria possível automatizar esse processo de otimização da seleção de representação?), é o problema da colocação de peças de dominó em um tabuleiro de xadrez truncado. Considere um tabuleiro de xadrez e um conjunto de dominós tal que cada dominó cubra exatamente dois quadrados do tabuleiro. Suponha, também, que faltam alguns quadrados no tabuleiro; na Figura 16.1, o canto superior esquerdo e o canto inferior direito foram removidos.

O problema do tabuleiro de xadrez truncado questiona se há uma forma de colocar peças de dominó sobre o tabuleiro de modo que cada quadrado do tabuleiro seja coberto e cada dominó cubra exatamente dois quadrados.

**Figura 16.1** Tabuleiro de xadrez truncado com dois quadrados cobertos por uma peça de dominó.



Podemos tentar resolver o problema tentando todas as posições de dominós sobre o tabuleiro; essa é a abordagem óbvia baseada em busca, e é uma consequência natural de se representar o tabuleiro por uma matriz simples, ignorando características aparentemente irrelevantes como a cor dos quadrados. A complexidade de tal busca é enorme e requereria heurísticas para uma solução eficiente. Por exemplo, poderíamos podar soluções parciais que deixassem quadrados isolados. Poderíamos, também, começar resolvendo o problema de um tabuleiro menor, como  $2 \times 2$  e  $3 \times 3$ , e tentar estender a solução para a situação  $8 \times 8$ .

Uma solução mais sofisticada, baseada em uma representação mais complexa, nota que cada colocação de um domínio precisa cobrir um quadrado branco e um quadrado preto. Esse tabuleiro truncado tem 32 quadrados pretos e 30 quadrados brancos; assim, a configuração desejada não será possível. Isso levanta uma séria questão para os mecanismos de raciocínio simbólicos: dispomos de representações que permitem que os resolvidores de problemas acessem o conhecimento com esse grau de flexibilidade e criatividade? Como uma representação em particular pode automaticamente mudar sua estrutura conforme mais coisas vão sendo aprendidas sobre um domínio de problema?

As heurísticas formam o terceiro componente da IA simbólica, junto à representação e à busca. Uma heurística é um mecanismo para organizar a busca ao longo das alternativas oferecidas por uma representação em particular. As heurísticas são concebidas para superar a complexidade da busca exaustiva, a barreira para soluções úteis para muitas classes de problemas interessantes. Para poder calcular, como ocorre com os seres humanos, a inteligência requer a escolha informada de “o que fazer a seguir”. Em toda a história da pesquisa em IA, as heurísticas assumiram muitas formas.

As primeiras técnicas de solução de problemas aplicadas em IA, como a *subida de encosta*, no programa de damas de Samuel (Seção 4.1), ou a *análise de meios e fins*, no resolvidor geral de problemas, de Newell, Shaw e Simon (Seção 13.1), vieram de outras disciplinas de IA, como a *pesquisa operacional*, e amadureceram gradualmente, tornando-se técnicas gerais para resolução de problemas em IA. As propriedades de busca, como *admissibilidade, monotonicidade e grau de informação*, são resultados importantes desses estudos iniciais. Essas técnicas são frequentemente referenciadas como *métodos fracos*. Métodos fracos são estratégias gerais de busca para serem aplicadas a classes inteiras de domínios de problemas (Newell e Simon, 1972; Ernst e Newell, 1969). Vimos esses métodos e suas propriedades nos capítulos 2, 3, 4, 6 e 14.

Nos capítulos 8, 9 e 10, introduzimos os *métodos fortes* para a solução de problemas em IA, com os sistemas especialistas baseados em regras, o raciocínio baseado em modelo e baseado em casos e o aprendizado simbólico. Contrastando com os resolvidores de problemas por método fraco, os métodos fortes focam na informação específica de cada área de problema, como a medicina interna ou o cálculo integral, em vez de se concentrar em conceber métodos heurísticos que possam generalizar para várias áreas de problemas. Os métodos fortes formam a base para os sistemas especialistas e outras abordagens de conhecimento intensivo para solucionar problemas. Os métodos fortes enfatizam questões como a quantidade de conhecimento necessário para resolver o problema, aprendizado e aquisição de conhecimento, a representação sintática do conhecimento, o tratamento de incertezas e questões relacionadas com a qualidade do conhecimento.

### **Por que não construímos sistemas simbólicos realmente inteligentes?**

Há muitas críticas que podem ser feitas à característica de inteligência do *sistema físico simbólico*. A maior parte delas se origina de questões de significação semântica e da *fundamentação* de símbolos e sistemas de símbolos. A natureza do “significado”, é claro, também tem impacto na ideia de inteligência como busca por meio de estruturas simbólicas pré-interpretadas e a “utilidade” implícita do uso de heurísticas. A noção de significado na IA tradicional, no mínimo, é muito fraca. Contudo, a tentação de avançar para uma semântica mais baseada na matemática, como a abordagem de mundos possíveis, de Tarski, parece ser equivocada. Ela reforça o projeto racionalista de substituir a inteligência flexível e evolutiva de um agente personificado por um mundo de ideias claras e distintas acessíveis consistentemente.

A *fundamentação* do significado é uma questão que tem sempre frustrado tanto os proponentes como os críticos da IA e dos empreendimentos da ciência cognitiva. A questão da fundamentação questiona como os símbolos podem ter significado. Searle (1980) discute esse ponto na chamada *sala chinesa*. Ele se coloca em uma sala com o objetivo de traduzir sentenças do chinês para o inglês; ali ele recebe um conjunto de símbolos chineses,

procura os símbolos em um grande sistema de catalogação de símbolos chineses e, então, produz os conjuntos apropriados de símbolos do inglês. Searle afirma que embora ele não saiba absolutamente nada de chinês, o seu “sistema” pode ser visto como uma máquina de tradução do chinês para o inglês.

Na verdade, *existe* um problema aqui. Embora qualquer um que trabalhe nas áreas de tradução de máquina ou de compreensão de linguagem natural (Capítulo 15) possa afirmar que a “máquina tradutora” de Searle, associando cegamente um conjunto de símbolos a outro, conseguiria produzir resultados de qualidade muito baixa, permanece o fato de que a geração atual de sistemas inteligentes tem uma capacidade muito limitada de interpretar conjuntos de símbolos de uma forma “significativa”. Esse problema de semântica de suporte muito fraca também permeia as abordagens atuais baseadas em computação para as áreas sensoriais, seja visual, cinestésica ou verbal.

Nas áreas da compreensão de linguagem humana, Lakoff e Johnson (1999) questionam que a capacidade de criar, usar, trocar e interpretar símbolos com significado advém de uma personificação humana dentro de um contexto social evolutivo. Esse contexto é físico, social e ocorre nesse momento; ele dá suporte e é responsável pela habilidade humana de sobreviver, evoluir e reproduzir. Isso torna possível um mundo de raciocínio por analogia, o uso e a apreciação do humor e as experiências de música e arte. Nossa geração atual de ferramentas e técnicas de IA está muito longe de ser capaz de codificar e utilizar um sistema com “significado” equivalente.

Como resultado direto dessa fraca codificação semântica, a metodologia de busca tradicional de IA explora estados e contextos de estados que são pré-interpretados. Isso significa que um criador de programa de IA “imputa” ou “impõe” aos símbolos do programa um contexto de significações semânticas. Um resultado direto dessa codificação pré-interpretada é que tarefas ricas em inteligência, como o aprendizado e a linguagem, podem produzir apenas uma função calculada dessa interpretação. Assim, muitos sistemas de IA têm capacidades muito limitadas para evoluir novas associações de significados conforme eles exploram seus ambientes (Luger et al., 2002).

Por fim, como um resultado direto de nossa atual capacidade limitada de modelar a semântica, os nossos empreendimentos mais bem-sucedidos são aquelas aplicações em que somos capazes de abstrair a partir de um rico contexto personificado e social e, ao mesmo tempo, capturar os componentes essenciais para a solução de um problema com símbolos pré-interpretados. Muitos deles foram tratados ao longo deste livro. Entretanto, mesmo essas áreas permanecem frágeis, sem múltiplas interpretações e apenas com limitada capacidade de se recuperar automaticamente de falhas.

Ao longo de sua breve história, a comunidade de pesquisadores em inteligência artificial tem explorado as ramificações da hipótese do sistema físico simbólico e desenvolvido os seus próprios desafios para aquela visão anteriormente dominante. Como ilustrado nos últimos capítulos deste livro, o sistema de símbolos explícitos e de busca não é o único meio representacional para capturar a inteligência. Modelos de computação baseados na arquitetura do cérebro animal, bem como nos processos de evolução biológica, também fornecem estruturas possíveis para a compreensão da inteligência em termos de processos científicamente exploráveis e empiricamente reproduzíveis. Nas próximas seções deste capítulo final, exploramos as ramificações dessas abordagens.

### 16.1.2 Computação conexionista ou “neural”

Uma alternativa significativa para a hipótese do sistema físico simbólico é a pesquisa em redes neurais e outros modelos de computação biologicamente inspirados. As redes neurais, por exemplo, são modelos computacionais de cognição, fisicamente implementados, não dependentes por completo de símbolos explicitamente referenciados e pré-interpretados que caracterizam um mundo. Como o “conhecimento” em uma rede neural está distribuído através de estruturas da rede, normalmente é difícil, e até mesmo impossível, isolar conceitos individuais em nós e pesos específicos da rede. Na verdade, qualquer parte da rede pode participar na representação de conceitos diferentes. Em consequência, as redes neurais são um contraexemplo importante no mínimo para a cláusula de *necessidade* da hipótese do sistema físico simbólico.

As redes neurais e as arquiteturas genéticas deslocam a ênfase da inteligência artificial dos problemas de representação simbólica e estratégias de inferência consistentes para problemas de aprendizado e adaptação. As redes neurais, assim como os seres humanos e outros animais, são mecanismos para a adaptação ao mundo: a estrutura de uma rede neural treinada é modelada pelo aprendizado tanto quanto pelo projeto. A inteligência de uma

rede neural não requer que o mundo seja reformulado como um modelo simbólico explícito. Em vez disso, a rede é moldada por suas interações com o mundo, refletidas em traços implícitos de experiência. Essa abordagem tem realizado uma série de contribuições para a compreensão da inteligência, dando um modelo plausível do mecanismo subjacente da personificação física de processos mentais, uma descrição mais viável dos processos de aprendizado e desenvolvimento, uma demonstração da capacidade de adaptações locais, simples para moldar um sistema complexo em resposta a fenômenos reais. Finalmente, elas nos oferecem uma ferramenta poderosa para a neurociência cognitiva.

Exatamente por serem tão diferentes, as redes neurais podem responder a uma série de questões que podem estar fora das capacidades expressivas da IA simbólica. Uma classe importante dessas questões diz respeito à percepção. A natureza não é tão generosa assim para nos fornecer as nossas percepções como conjuntos de expressões do cálculo de predicados. As redes neurais oferecem um modelo de como poderíamos reconhecer padrões “significativos” no caos de estímulos sensoriais.

Por causa da sua representação distribuída, as redes neurais com frequência são mais robustas que os seus contrapontos explicitamente simbólicos. Uma rede neural treinada de forma adequada pode efetivamente categorizar ocorrências novas, exibindo uma percepção parecida com a humana baseada em similaridade, em vez da necessidade estritamente lógica. De modo semelhante, a perda de alguns neurônios não compromete necessariamente o desempenho de uma rede neural grande. Isso resulta da redundância frequentemente excessiva, inerente dos modelos de redes neurais.

Talvez o aspecto mais atraente das redes conexionistas seja a sua capacidade de aprender. Em vez de tentar construir um modelo simbólico detalhado do mundo, as redes neurais se baseiam na plasticidade de sua estrutura para se adaptar diretamente à experiência. Mais do que construir um modelo de mundo, elas são moldadas pela sua experiência com o mundo. O aprendizado é um dos aspectos mais importantes da inteligência. O problema do aprendizado é também aquele que levanta algumas das questões mais difíceis para a pesquisa em computação neural.

### **Por que não construímos um cérebro?**

Na verdade, a geração atual de sistemas conexionistas é muito pouco parecida com o sistema nervoso humano! Visto que o tópico de *plausibilidade neural* é uma questão de pesquisa crítica, começamos com essa questão e depois consideramos o desenvolvimento e o aprendizado. As pesquisas em neurociência cognitiva (Squire e Kosslyn, 1998; Gazzaniga, 2000; Hugdahl e Davidson, 2003) fornecem novas perspectivas para a compreensão da arquitetura cognitiva humana. Nesta seção, apresentamos brevemente algumas descobertas e comentamos como elas podem se relacionar com o empreendimento da IA. Consideramos questões em três níveis: primeiro, no nível do neurônio, segundo, no nível da arquitetura neural e, finalmente, comentamos a representação cognitiva ou o problema da *codificação*.

Primeiro, no nível do neurônio individual, Shephard (1998) e Carlson (1994) identificaram múltiplos tipos diferentes de arquiteturas neurais para células, cada uma das quais especializada conforme a sua função e o seu papel dentro do sistema maior. Entre elas estão *células receptoras sensoriais* geralmente encontradas na pele e passando informação de entrada para outras estruturas celulares, *interneurônios* cuja tarefa básica é a comunicação dentro de agrupamentos de células, *neurônios principais* cuja tarefa é a comunicação entre os agrupamentos de células e *neurônios motores* cuja tarefa é a saída do sistema.

A atividade neural é elétrica. Padrões de fluxos iônicos entrando e saindo do neurônio determinam se um neurônio está ativo ou em repouso. O neurônio típico tem um potencial de repouso de  $-70\text{ mV}$ . Quando uma célula está ativa, certas substâncias químicas são liberadas do terminal axônico. Essas substâncias, denominadas *neurotransmissores*, influenciam a membrana pós-sináptica, tipicamente se ligando a sitios receptores específicos, como uma chave em uma fechadura, desencadeando outros fluxos iônicos. Quando os fluxos iônicos atingem um nível crítico, cerca de  $-50\text{ mV}$ , eles produzem um *potencial de ação*, um mecanismo de disparo tudo ou nada indicando que a célula disparou. Assim, os neurônios se comunicam por meio de sequências de códigos binários.

As alterações pós-sinápticas do potencial de ação são de dois tipos, *inibitórias*, encontradas principalmente em estruturas celulares de interneurônios, ou *excitatórias*. Esses potenciais positivos e negativos são constantemente gerados por todas as sinapses no sistema dendrítico. Sempre que o efeito resultante de todos esses eventos

for o de alterar os potenciais da membrana dos neurônios correspondentes de  $-70$  mV para  $-50$  mV, o limiar é ultrapassado, e fluxos iônicos intensos são novamente iniciados nos axônios daquelas células.

Em segundo lugar, no nível da arquitetura neural, há um total de cerca de  $10^{10}$  neurônios no córtex cerebral, uma folha fina retorcida que cobre todo o hemisfério cerebral. Grande parte do córtex é dobrada sobre si mesma, aumentando a área superficial total. Da perspectiva computacional precisamos saber não apenas o número total de sinapses, mas também os parâmetros relativos ao número de entradas (*fan-in*) e ao número de saídas (*fan-out*). Shephard (1998) estima que esses números sejam da ordem de  $10^5$ .

Finalmente, afora as diferenças nas células e arquiteturas dos sistemas neural e computacional, existe um profundo problema de representação cognitiva. Por exemplo, ignoramos até mesmo como memórias simples são codificadas no córtex. Ou como um rosto é reconhecido e como o reconhecimento de um rosto pode ligar um agente a sentimentos de alegria ou de tristeza. Sabemos muito sobre os aspectos físicos e químicos do cérebro, mas relativamente pouco sobre como o sistema neural codifica e usa “padrões” dentro de seu contexto.

Uma das questões mais difíceis para os pesquisadores, tanto da comunidade neural como da computacional, diz respeito ao papel do conhecimento inato no aprendizado. Pode ocorrer aprendizado efetivo em uma *tábula rasa*, ou *lousa vazia*, a partir de nenhum conhecimento inicial e aprendendo totalmente da experiência? Ou ele precisa iniciar com algum viés indutivo prévio? A experiência no projeto de programas de aprendizado de máquina sugere que algum tipo de conhecimento *a priori*, normalmente expresso como um viés indutivo, é necessário para o aprendizado em ambientes complexos.

A capacidade das redes conexionistas em convergir para uma generalização com significação, a partir de um conjunto de dados de treinamento, tem se mostrado sensível ao número de neurônios artificiais, à topologia da rede e aos algoritmos de treinamento específicos utilizados. Juntos, esses fatores constituem um viés indutivo tão forte como o que pode ser encontrado em qualquer representação simbólica. A pesquisa no desenvolvimento humano dá suporte a essa conclusão. Há, cada vez mais, evidência, por exemplo, de que as crianças herdam uma série de vieses cognitivos implementados por conexões físicas (“hard-wired”), o que permite o aprendizado de domínios de conceitos como a linguagem e a física de senso comum. A caracterização de vieses inatos em redes neurais é uma área ativa de pesquisa (Elman et al., 1996).

A questão do viés inato se torna ainda mais problemática se considerarmos problemas de aprendizado mais complexo. Por exemplo, suponha que estejamos desenvolvendo um modelo computacional de descoberta científica e que desejemos modelar a mudança proposta por Copérnico de uma visão geocêntrica do universo para uma visão heliocêntrica. Isso requer que representemos em um computador tanto a visão de Copérnico como a de Ptolomeu. Embora possamos representar essas visões como padrões de ativação em uma rede neural, nossa rede não nos diria nada sobre o seu comportamento como *teorias*. Em vez disso, preferimos explicações como “Copérnico não estava satisfeito com a complexidade do sistema ptolomaico e preferia o modelo mais simples da translação dos planetas em torno do Sol”. Explicações como essa requerem símbolos. Claramente, as redes neurais precisam ser capazes de permitir raciocínio simbólico; apesar de tudo, os seres humanos usam redes neurais e eles parecem manipular símbolos muito bem. A fundamentação neural do raciocínio simbólico é uma importante área de pesquisa em aberto.

Outro problema é o papel do desenvolvimento no aprendizado. As crianças não conseguem aprender simplesmente com base em dados disponíveis. Sua capacidade de aprendizado em domínios específicos aparece em estágios de desenvolvimento bem definidos (Karmiloff-Smith, 1992). Uma questão interessante é se essa progressão do desenvolvimento é puramente um resultado da biologia e personificação humana, ou se ela reflete limites logicamente necessários sobre a capacidade de uma inteligência aprender invariâncias no mundo. Será que os estágios de desenvolvimento poderiam funcionar como um mecanismo para decompor o problema de aprendizado sobre o mundo em subproblemas mais manipuláveis? Será que uma série de restrições de desenvolvimento, artificialmente impostas, poderia fornecer às redes neurais o arcabouço necessário para o aprendizado em um mundo complexo?

A aplicação de redes neurais a problemas práticos levanta uma série de problemas adicionais. As propriedades fundamentais das redes neurais que as tornam tão atrativas, como a adaptabilidade e a robustez em relação a dados faltantes ou ambíguos, também criam problemas para a sua aplicação prática. Pelo fato de as redes serem treinadas, em vez de programadas, o seu comportamento é difícil de prever. Existem algumas diretrizes básicas

para projetar redes que convirjam apropriadamente em um dado domínio de problema. Por fim, é difícil construir explicações de *por que* uma rede chegou a uma conclusão em particular e, frequentemente, elas tomam a forma de uma argumentação estatística. Todas essas questões são temas atuais de pesquisa.

Poderíamos questionar se as redes conexionistas e a IA mais simbólica são modelos de inteligência tão diferentes assim. Ambas compartilham uma série de pontos comuns importantes, especialmente que a inteligência é, fundamentalmente, codificada como computação e tem limites fundamentais e formais como a hipótese de Church e Turing (Luger, 1994; Capítulo 2). Ambas as abordagens também oferecem modelos de mente moldados pela aplicação a problemas práticos. Ainda mais importante, entretanto, é o fato de que ambas as abordagens negam o dualismo filosófico e colocam os fundamentos da inteligência sobre a estrutura e a função de dispositivos fisicamente realizados.

Acreditamos que uma total reconciliação dessas duas abordagens tão diferentes para capturar inteligência será inevitável. Quando ela for realizada, uma teoria de como os símbolos podem ser reduzidos a padrões em uma rede e, por sua vez, como eles podem influenciar a futura adaptação dessa rede, será uma contribuição extraordinária. Isso embasará uma série de desenvolvimentos, como a integração de facilidades perceptivas, baseadas em redes, e de raciocínio, baseadas em conhecimento, em uma única inteligência. Enquanto isso, contudo, as duas comunidades científicas têm um trabalho considerável a fazer, e não vemos razão para que elas não possam continuar a coexistir. Para aqueles que não se sentem confortáveis com dois modelos de inteligência aparentemente incomensuráveis, mesmo a física funciona bem com a noção intuitivamente contraditória de que a luz algumas vezes é mais bem compreendida como uma onda e, outras vezes, como partícula, embora os dois pontos de vista possam ser agrupados pela teoria da sequência (Norton, 1999).

### 16.1.3 Agentes, emergência e inteligência

A computação baseada em agentes e as teorias modulares da cognição levantam outro conjunto de questões interessantes para os pesquisadores da inteligência artificial. Uma importante escola de pensamento da ciência cognitiva afirma que a mente é organizada em conjuntos de unidades funcionais especializadas (Minsky, 1985; Fodor, 1983). Esses módulos são especialistas e empregam uma série de estruturas e funções inatas, desde a solução do problema fisicamente conectada até o viés indutivo, para darem conta da diversidade de problemas com que eles, como agentes práticos, precisam lidar. Isso faz sentido: como uma única rede neural, ou outro sistema, pode ser treinada para tratar funções tão diversas como a percepção, o controle motor, a memória e o raciocínio de alto nível? As teorias modulares da inteligência fornecem um arcabouço para responder a essas questões e, também, uma direção para continuar investigando questões como a natureza dos vieses inatos em módulos individuais e mecanismos de interação entre módulos.

Os modelos de computação genéticos e emergentes fornecem uma das abordagens mais novas e excitantes para a compreensão tanto da inteligência humana como da artificial. Demonstrando que o comportamento inteligente global pode surgir da cooperação de um grande número de agentes individuais, restritos e independentes, as teorias genéticas e emergentes tratam resultados complexos expressos por inter-relacionamentos de estruturas relativamente simples.

Em um exemplo de Holland (1995), os mecanismos que mantêm uma grande cidade como Nova York abastecida com pão demonstram os processos fundamentais subjacentes à emergência da inteligência em um sistema baseado em agentes. É improvável que pudéssemos escrever um planejador centralizado que abastecesse com sucesso os habitantes de Nova York com a rica variedade de pães a que eles estão acostumados. De fato, o experimento malsucedido do mundo comunista com o planejamento centralizado revelou as limitações de tais abordagens! Entretanto, apesar das dificuldades práticas para escrever um algoritmo de planejamento centralizado, capaz de manter Nova York abastecida com pães, os esforços fracamente coordenados dos diversos padeiros da cidade, dos entregadores, fornecedores de matéria-prima, bem como dos varejistas, resolvem o problema muito bem. Como em todos os sistemas emergentes baseados em agentes, não existe um plano central. Cada um tem apenas um conhecimento limitado das necessidades de pão da cidade; cada padeiro simplesmente tenta otimizar

as suas próprias oportunidades de negócio. A solução para o problema global emerge das atividades coletivas desses agentes locais e independentes.

Pela demonstração de como comportamentos quase ótimos, robustos, altamente guiados por objetivo podem surgir das interações de agentes individuais locais, esses modelos fornecem ainda outra resposta a antigas questões filosóficas das origens da mente. A lição central das abordagens emergentes é que a inteligência completa pode surgir, e realmente surge, das interações de muitas inteligências de agentes simples, individuais, locais e personificados.

A segunda principal característica dos modelos emergentes é a sua fundamentação na seleção darwiniana como mecanismo básico que molda o comportamento dos agentes individuais. No exemplo da padaria, cada padeiro aparentemente não se comporta de uma maneira que seja, em algum sentido, globalmente ótima. A fonte da otimização não está em um projeto central; ela está no simples fato de que padeiros que não conseguem satisfazer as necessidades dos consumidores fracassarão. É por meio das operações persistentes dessas pressões seletivas que os padeiros individuais obtêm os comportamentos que levam à sua sobrevivência individual, bem como a um comportamento coletivo útil emergente.

A combinação de uma arquitetura distribuída, baseada em agentes com as pressões adaptativas da seleção natural, constitui um modelo poderoso das origens e operações da mente. Psicólogos evolucionários (Cosmides e Tooby, 1992, 1994; Barkow et al., 1992) forneceram um modelo da maneira como a seleção natural moldou o desenvolvimento da estrutura e de vieses inatos na mente humana. A base da psicologia evolucionária é uma visão da mente como altamente modular, como um sistema de agentes interativos altamente especializados. De fato, as discussões da psicologia evolucionária frequentemente compararam a mente a um canivete suíço, uma coleção de ferramentas especializadas que podem ser aplicadas para resolver problemas diferentes.

Existe uma crescente evidência de que a mente humana é, de fato, altamente modular. Fodor (1983) oferece um argumento filosófico para a estrutura modular da mente. Minsky (1985) explorou as ramificações de teorias modulares para a inteligência artificial. Essa arquitetura é importante para teorias da evolução da mente. Seria difícil imaginar como a evolução poderia moldar um único sistema tão complexo como a mente. Entretanto, é plausível que a evolução, trabalhando ao longo de milhões de anos, pudesse moldar sucessivamente habilidades cognitivas especializadas, individuais. Conforme a evolução do cérebro avançava, ela poderia também atuar sobre combinações de módulos, formando os mecanismos que capacitam os módulos a interagirem, compartilharem informação e cooperarem para realizar tarefas cognitivas cada vez mais complexas (Mithen, 1996).

Teorias da seleção neural (Edelman, 1992) mostram como esses mesmos processos podem explicar a adaptação do sistema neural individual. O darwinismo neural modela a adaptação de sistemas neurais em termos darwinianos: o reforço de circuitos particulares no cérebro e o enfraquecimento de outros é um processo de seleção em resposta ao mundo. Em contraste com métodos de aprendizado simbólicos, que procuram extrair informação de dados de treinamento e usar essa informação para construir modelos do mundo, teorias de seleção neural examinam o efeito de pressões seletivas sobre populações de neurônios e suas interações. Edelman (1992, página 81) afirma:

Ao considerar a ciência do cérebro como uma ciência de reconhecimento, considero implicitamente que o reconhecimento não é um processo instrutivo. Não ocorre transferência direta de informação, como também isso não ocorre em processos evolucionários ou imunológicos. O reconhecimento é seletivo.

As tecnologias de agentes oferecem também modelos de cooperação social. Usando abordagens baseadas em agentes, os economistas construíram modelos informativos (se não completamente preditivos) de mercados econômicos. Tecnologias de agentes têm exercido crescente influência no projeto de sistemas de computação distribuída, na construção de ferramentas de busca na Internet e na implementação de ambientes de trabalho cooperativos.

Por fim, modelos baseados em agentes têm exercido influência em teorias da consciência. Daniel Dennett (1991, 2006), por exemplo, baseou uma descrição da função e da estrutura da consciência em uma arquitetura de agente da mente. Ele começa argumentando que é incorreto questionar onde a consciência está localizada na mente/cérebro. Em vez disso, sua *teoria de múltiplos esquemas da consciência* foca no papel da consciência nas interações de agentes em uma arquitetura mental distribuída. No curso da percepção, controle motor, solução de problemas, aprendizado e outras atividades mentais, formamos coalizões de agentes interativos, que são altamen-

te dinâmicas, alterando-se em resposta às necessidades de diferentes situações. A consciência, segundo Dennett, serve como um mecanismo de ligação para essas coalizões, suportando a interação de agentes e elevando coalizões críticas de agentes interativos ao primeiro plano do processamento cognitivo.

### ***Quais questões limitam uma aproximação da inteligência baseada em agentes?***

As abordagens baseadas em agente e “emergentes” revelaram uma série de problemas que precisam ser resolvidos para que as suas promessas sejam realizadas. Por exemplo, ainda temos que preencher os passos que permitiram a evolução de habilidades cognitivas de alto nível como a linguagem. Como os esforços de paleontólogos para reconstruir a evolução de espécies, o rastreamento do desenvolvimento desses problemas de alto nível demandará um grande trabalho detalhado adicional. Precisamos tanto enumerar os agentes que formam a arquitetura da mente como também rastrear a sua evolução ao longo do tempo.

Outro problema importante para as teorias baseadas em agentes é a explicação das interações entre os módulos. Embora o modelo de mente como “canivete suíço” seja uma construção intuitiva interessante, os módulos que compõem a mente não são tão independentes como as lâminas de um canivete. As mentes exibem interações extensas, altamente fluidas entre domínios cognitivos: podemos falar sobre coisas que vemos, indicando uma interação entre módulos visuais e linguísticos. Podemos construir prédios que servem a determinados propósitos sociais específicos, indicando uma interação entre inteligência técnica e social. Os poetas podem construir metáforas táteis para cenas visuais, indicando uma interação fluida entre módulos visuais e táteis. A definição das representações e processos que permitem essas interações entre módulos é uma área ativa de pesquisa (Karmiloff-Smith, 1992; Mithen, 1996; Lakoff e Johnson, 1999).

As aplicações práticas de tecnologias baseadas em agentes estão se tornando também cada vez mais importantes. Por meio do uso de simulações computacionais baseadas em agentes, é possível modelar sistemas complexos que não podem ser descritos por uma formulação matemática fechada, sendo assim impossível de serem estudados em detalhe. Técnicas baseadas em simulação têm sido aplicadas a uma série de fenômenos, como a adaptação do sistema imunológico humano e o controle de processos complexos, incluindo aceleradores de partículas, o comportamento de mercados de câmbio globais e o estudo de sistemas meteorológicos. As questões representacionais e computacionais que precisam ser resolvidas para implementar tais simulações continuam a guiar as pesquisas em representações do conhecimento, algoritmos e até o projeto de hardware de computadores.

Outros problemas práticos que as arquiteturas de agentes precisam tratar incluem protocolos de comunicação entre agentes, especialmente quando agentes locais têm conhecimento limitado sobre o problema global ou mesmo sobre qual conhecimento os outros agentes já possuem. Além disso, existem poucos algoritmos para a decomposição de grandes problemas em subproblemas coerentes, orientados a agentes, ou mesmo como recursos limitados podem ser distribuídos entre agentes. Essas e outras questões relacionadas a agentes foram apresentadas na Seção 7.4.

Talvez o aspecto mais excitante das teorias emergentes da mente seja o seu potencial para colocar as atividades mentais dentro de um modelo unificado de emergência de ordem a partir do caos. Mesmo a breve visão geral fornecida nesta seção citou trabalhos usando teorias emergentes para modelar uma série de processos, desde a evolução do cérebro ao longo do tempo, passando pelas forças que permitem o aprendizado em indivíduos, até a construção de modelos econômicos e sociais do comportamento. Há algo de extraordinariamente atraente na noção de que os mesmos processos emergentes de ordem, como moldados pelos processos darwinianos, sejam capazes de explicar o comportamento inteligente em uma variedade de resoluções, desde as interações de neurônios individuais e a conformação da estrutura modular do cérebro, até o funcionamento de mercados econômicos e sistemas sociais. É como se a inteligência tivesse uma geometria fractal, onde os mesmos processos emergentes aparecem em qualquer nível de resolução que consideremos o sistema.

#### **16.1.4 Modelos probabilísticos e a tecnologia estocástica**

No inicio da década de 1950, técnicas estocásticas foram usadas para tratar da compreensão e a geração de expressões em linguagem natural. Claude Shannon (1948) aplicou modelos probabilísticos, incluindo cadeias discretas de Markov, à tarefa de processamento de linguagem. Shannon (1951) também pegou emprestada a noção de entropia da

termodinâmica como um modo de medir a capacidade de informação de uma mensagem. Nessa mesma época, a Bell Labs também criou o primeiro sistema estatístico capaz de reconhecer os dez dígitos, 0, ..., 9, conforme falados por um único indivíduo. Ele funcionou com precisão de 97 a 99% (Davis et al., 1952; Jurasky e Martin, 2008).

Durante as décadas de 1960 e 1970, métodos Bayesianos de raciocínio continuaram em grande parte no segundo plano da atividade de pesquisa em IA. A tecnologia de linguagem natural explorou muitas das abordagens simbólicas descritas na Seção 7.1. Embora muitos sistemas especialistas, por exemplo, o MYCIN, criasse suas próprias “álgebras do fator de certeza”, como visto na Seção 9.2, vários outros, incluindo o PROSPECTOR, tomaram a abordagem Bayesiana (Duda et al., 1979a). No entanto, a complexidade desses sistemas se tornou rapidamente intratável. Conforme indicamos na Seção 9.3, o uso completo da regra de Bayes em um programa de diagnóstico médico de tamanho realista de 200 doenças e 2000 sintomas exigiria a coleta e a integração de 800 milhões de partes de informação.

No final da década de 1980, Judea Pearl (1988) ofereceu um modelo computacionalmente tratável de raciocínio diagnóstico no contexto de relacionamentos causais dentro de um domínio de problema: redes Bayesianas de crença (RBC). As RBCs relaxam duas restrições do modelo Bayesiano completo. Primeiro, supõe-se uma “causalidade” implícita na inferência estocástica; ou seja, o raciocínio vai da causa ao efeito e não é circular, isto é, um efeito, não pode retornar à própria causa. Isso dá suporte à representação de RBCs como um grafo aciclico direcionado (seções 3.1, 9.3). Segundo, as RBCs consideram que os genitores diretos de um nó suportam influência causal completa nesse nó. Todos os outros nós são considerados condicionalmente independentes ou influenciaram muito pouco e são ignorados.

A pesquisa de Pearl (1988, 2000) renovou o interesse nas abordagens estocásticas para a modelagem do mundo. Como vimos na Seção 9.3, as RBCs ofereceram uma ferramenta de representação muito poderosa para a inferência diagnóstica (abolutiva). Isso é especialmente verdade no caso da natureza dinâmica de sistemas humanos e estocásticos: à medida que o mundo muda com o tempo, nosso conhecimento é enriquecido: algumas causas explicam mais o que vemos, enquanto outras causas potenciais “são eliminadas”. A pesquisa no projeto de sistemas estocásticos, bem como seus esquemas de inferência de suporte, está realmente apenas começando.

O final dos anos 1980 também viu uma nova energia de pesquisa aplicada a questões de processamento de linguagem. Como vimos na Seção 15.4, essas abordagens estocásticas incluíram novas técnicas de análise, marcação e muitas outras para retirar a ambiguidade das expressões da linguagem. Uma faixa completa dessas técnicas poderá ser encontrada em livros sobre reconhecimento de voz e tarefas em processamento de linguagem (Manning e Schütz, 1999; Jurasky e Martin, 2008).

Com o interesse renovado e os sucessos nas abordagens estocásticas para caracterizar o comportamento inteligente, uma pessoa pode questionar naturalmente quais poderiam ser suas limitações.

### ***A inteligência é fundamentalmente estocástica?***

Há uma tremenda atração em direção a um ponto de vista estocástico para interações de agente em um mundo mutável. Muitos poderiam argumentar que o “sistema de representação” de um ser humano é fundamentalmente estocástico, ou seja, condicionado pelo mundo de percepções e causas em que ele está imerso. Certamente, o ponto de vista comportamentalista/empiricista acharia essa conjectura atraente. Teoristas de ação situada e personificada poderiam ir mais além e criar a hipótese de que os relacionamentos condicionados que um agente tem com seu ambiente físico e social oferecem uma explicação suficiente da acomodação do agente bem-sucedido com esse mundo. Alguns pesquisadores modernos ainda afirmam que os aspectos da função neural são fundamentalmente estocásticos (Hawkins, 2004; Doya et al., 2007).

Consideremos a linguagem. Um dos pontos fortes da expressão oral e escrita, conforme apontaram Chomsky e outros, é a sua natureza *generativa*. Isso significa que, dentro do conjunto de vocabulário e formas de linguagem disponível, expressões novas e previamente não experimentadas ocorrem naturalmente. Isso acontece tanto no nível de criação de sentenças novas quanto com palavras individuais, gerando, por exemplo: “*Google it!*” Um relato estocástico da linguagem deverá demonstrar esse poder de criação. Atualmente, as limitações da informação da linguagem coletada, sejam *treebanks* ou outros corpos de dados, podem restringir radicalmente o uso da tecnologia estocástica. Isso porque a informação coletada precisa oferecer um cenário apropriado (ou prévio) para interpretar a nova situação atual.

Modelos estocásticos de domínios de aplicação, para um motor de avião ou sistema de transmissão, podem ter limitações semelhantes. Necessariamente, sempre haverá suposições de *mundo fechado* ou *modelo mínimo*, Seção 9.1, para um sistema realisticamente complexo. Porém, as redes Bayesianas dinâmicas mostram agora promessas para interpretar o estado do sistema com o passar do tempo. Contudo, ainda haverá capacidades limitadas para prever situações novas com o tempo.

Em um nível de explicação mais alto, pode ser difícil para um modelo probabilístico considerar um deslocamento de seu próprio sistema explicativo, ou paradigma. Em que sentido, conforme observado na Seção 16.1.2, um modelo estocástico poderia explicar teorias ou rearrumações de nível mais alto das visões conceituais, ou seja, reavaliar questões relacionadas à adequação do próprio modelo com, talvez, a necessidade de deslocar para diferentes pontos de vista? Esses tópicos continuam sendo questões de pesquisa importantes, e restringem as técnicas estocásticas para compreender a incerteza.

Mas ainda há o fato de que, normalmente sem instrução explícita, os agentes “criam” modelos muito bem-sucedidos. Como vemos por um ponto de vista construtivista, na Seção 16.2, *um* modelo é uma condição *sine qua non* para um agente compreender seu mundo, ou seja, se não houver um comprometimento do que “significa” o mundo, os fenômenos não são percebidos nem compreendidos!

Além dos argumentos filosóficos que apresentamos, existem várias limitações práticas ao uso de sistemas estocásticos. A geração atual de RBCs é proposicional por natureza. Apenas recentemente foi possível criar leis gerais ou relacionamentos como  $\forall X \text{ homem}(X) \rightarrow \text{inteligente}(X)$  com uma distribuição anexada. Além disso, deseja-se que as RBCs incluam relacionamentos recursivos, especialmente para a análise de série temporal. A pesquisa para desenvolver sistemas de representação estocásticos de primeira ordem, tratando dessas questões em geral, é importante para a pesquisa contínua. Conforme observamos na Seção 13.3, agora existem ferramentas de modelagem estocástica de primeira ordem e Turing completa (Pless et al., 2006). Também é importante explorar a aplicação desses modelos estocásticos gerais para dados neuropsicológicos, onde foram recentemente aplicados (Burge et al., 2007; Doya et al., 2007).

A seguir, discutimos aspectos psicológicos e filosóficos da inteligência humana que têm impacto na criação, utilização e avaliação de uma inteligência artificial.

## 16.2 A ciência dos sistemas inteligentes

Não é por coincidência que um grande subgrupo da comunidade de inteligência artificial tenha focado a sua pesquisa no entendimento da inteligência *humana*. O ser humano fornece o exemplo prototípico de atividade inteligente, e os engenheiros de IA, muito embora *não* estejam normalmente comprometidos em “fazer programas que agem como pessoas”, raramente ignoram a solução humana. Algumas aplicações, como o raciocínio de diagnóstico, são, com frequência, modeladas de forma deliberada a partir dos processos de solução de especialistas humanos que trabalham na área. Ainda mais importante é o fato de que a compreensão da inteligência humana é, por si só, um desafio científico fascinante e em aberto.

A *ciência cognitiva* moderna, ou *ciência dos sistemas inteligentes* (Luger, 1994), começou com o advento do computador digital, muito embora, como vimos no Capítulo 1, tenha havido muitos ancestrais dessa disciplina, desde Aristóteles, passando por Descartes e Boole, até os teóricos mais modernos, como Turing, McCulloch e Pitts — os fundadores do modelo de *redes neurais* —, e John von Neumann — um dos primeiros proponentes da vida artificial (*a-life*). Entretanto, o estudo se tornou ciência com a capacidade de projetar e executar experimentos baseados nessas noções teóricas e, em grande parte, isso se tornou possível com o computador. Por fim, poderíamos perguntar: “Existe uma ciência de inteligência propriamente dita?”. Podemos também perguntar: “Uma ciência dos sistemas inteligentes pode dar suporte à construção de inteligências artificiais?”

Nas próximas seções, discutimos brevemente como as ciências psicológicas, epistemológicas e sociológicas dão suporte às pesquisas e desenvolvimentos em IA.

### 16.2.1 Restrições psicológicas

As pesquisas iniciais em ciência cognitiva examinaram as soluções humanas para problemas lógicos, jogos simples, planejamento e aprendizado de conceitos (Feigenbaum e Feldman, 1963; Newell e Simon, 1972; Simon, 1981). Em paralelo com o seu trabalho no *Logic Theorist*, na Seção 14.1, Newell e Simon começaram a comparar suas abordagens computacionais com as estratégias de busca usadas por seres humanos. Os seus dados consistiam em *protocolos verbais*, descrições feitas pela pessoa humana de seus pensamentos durante o processo de encontrar a solução de um problema como uma prova lógica. Newell e Simon compararam, então, esses protocolos com o comportamento do programa computacional na solução do mesmo problema. Os pesquisadores encontraram similaridades notáveis e diferenças interessantes entre os problemas e os sujeitos.

Esses projetos iniciais estabeleceram a metodologia que a *ciência cognitiva* empregaria durante as décadas seguintes:

1. Com base em dados de seres humanos resolvendo classes particulares de problemas, projete um esquema representacional e uma estratégia de busca associada para resolver o problema.
2. Execute o modelo computacional para produzir um rastreamento do seu comportamento durante a solução.
3. Observe pessoas trabalhando nesses problemas e rastreie parâmetros mensuráveis de seu processo de solução, como aqueles encontrados em protocolos verbais, movimentos de olhos e anotações de resultados parciais.
4. Analise e compare as soluções humana e por computador.
5. Revise o modelo por computador para a próxima rodada de testes e comparações com os humanos.

Essa metodologia empírica é descrita na palestra de Newell e Simon do Turing Award Lecture citada no início deste capítulo. Um aspecto importante da ciência cognitiva é o uso de experimentos para validar uma arquitetura para a solução do problema, seja ela um sistema de produção, conexionista, emergente ou uma arquitetura baseada na interação de agentes distribuídos.

Nos últimos anos, uma dimensão completamente nova foi adicionada a esse paradigma. Agora, não apenas os programas podem ser desconstruídos e observados durante o ato de solução do problema, mas os seres humanos e outras formas de vida também podem sê-lo. Uma série de novas técnicas de aquisição de imagens foi recentemente incluída nas ferramentas disponíveis para observar a atividade cortical. Aqui se inclui a *magnetoencefalografia* (MGE), que detecta os campos magnéticos gerados por populações de neurônios. Diferentemente dos potenciais elétricos gerados por essas populações, o campo magnético não é distorcido pelo crânio e pelo couro cabeludo, possibilitando uma maior resolução.

Uma segunda tecnologia de aquisição de imagens é a *tomografia de emissão de pósitrons*, ou PET (do inglês *position emission tomography*). Uma substância radiativa, geralmente O<sup>15</sup>, é injetada na corrente sanguínea. Quando uma região particular do cérebro está ativa, mais quantidade desse agente passa por detectores sensíveis do que quando a região está em repouso. A comparação das imagens em repouso e em atividade pode potencialmente revelar a localização das funções cerebrais com uma resolução de cerca de 1 cm (ver Stytz e Frieder, 1990).

Outra técnica de análise neural é a aquisição de *imagem de ressonância magnética funcional*, ou F-MRI (do inglês *functional magnetic resonance imaging*). Essa abordagem emergiu do processo de aquisição de imagens padrão, baseado na ressonância magnética nuclear (NMR, do inglês *nuclear magnetic resonance*). Assim como PET, essa abordagem compara estados neurais em repouso e em atividade para revelar a localização das funções cerebrais.

Outra contribuição para a localização de funções cerebrais, com uma ligação importante com as técnicas de aquisição de imagens mencionadas anteriormente, são os algoritmos desenvolvidos por Barak Pearlmuter e seus colegas (Pearlmuter e Parra, 1997; Tang et al., 1999, 2000a, 2000b). Esses pesquisadores são capazes de processar padrões complexos ruidosos, encontrados frequentemente como resultado das várias técnicas de aquisição de imagens neurais, e quebrá-los em seus componentes. Esse é um passo essencial no processo de análise, uma vez que os padrões existenciais normais, como os movimentos dos olhos, o ritmo de respiração e os batimentos cardíacos, estão entrelaçados com padrões de disparo de neurônios, que desejamos compreender.

O resultado das pesquisas recentes na neurociência cognitiva (Squire e Kosslyn, 1998; Shephard, 1998; Gazzaniga, 2000) contribuiu enormemente para o nosso entendimento dos componentes neurais envolvidos na

atividade inteligente. Muito embora uma análise crítica desses resultados esteja fora do escopo deste livro, listamos e referenciamos a seguir várias questões importantes:

Na área da percepção e da atenção há o *problema da associação*. Pesquisadores, como Anne Treisman (1993, 1998) e Jeff Hawkins (2004), verificaram que as representações perceptivas dependem de códigos neurais distribuídos para relacionar as partes e as propriedades de objetos entre si e questionam que mecanismo é necessário para “ligar” as informações relativas a um objeto e que o distinguem de outros objetos.

Na área de busca visual, que mecanismos neurais são responsáveis pela percepção de objetos inseridos em grandes cenas complexas? Alguns experimentos mostram que a supressão de informação de objetos irrelevantes desempenha um papel importante na seleção de objetivos de busca (Luck, 1998). Além disso, como “aprendemos” a ver (Sagi e Tanen, 1998)?

Na área de plasticidade em percepção, Gilbert (1992, 1998) sustenta que o que vemos não é estritamente um reflexo das características de uma cena, mas é altamente dependente dos processos pelos quais nosso cérebro tenta interpretar a cena.

Como o sistema cortical representa e indexa temporariamente informação correlata, incluindo a interpretação de percepções e a produção de atividade motora (Ivry, 1998)?

Em estudos sobre memória, hormônios de estresse liberados durante situações emocionalmente estimulantes modulam processos de memória (Cahill e McGaugh, 1998). Isso se relaciona com o problema da *fundamentação*: como pensamentos, palavras e percepções adquirem significado para um agente? Em que sentido pode haver uma “tristeza (ou lágrimas) em coisas”, a *lacremae rerum* de Virgílio?

Os aspectos acústico-fonéticos da fala fornecem princípios organizacionais importantes para ligar a pesquisa em neurociência a teorias cognitivas e linguísticas (Miller et al., 1998). Como estão integrados os componentes sintáticos e semânticos do córtex (Gazzaniga, 2000)?

Como um indivíduo adquire uma linguagem específica? Que estágios neurofisiológicos são responsáveis por esse desenvolvimento (Kuhl, 1993, 1998)?

Como podemos compreender o desenvolvimento? O que é a *plasticidade de período crítico* e as reorganizações em adultos constatadas em sistemas sensoriais da epiderme de mamíferos (O’Leary et al., 1999)? Os estágios de desenvolvimento são críticos para a “construção” da inteligência? Veja Karmiloff-Smith (1992) e Gazzaniga (2000) para obter mais detalhes.

A prática da inteligência artificial certamente não requer conhecimento extenso desses e de outros domínios neuropsicológicos correlacionados. Entretanto, esse tipo de conhecimento pode apoiar a engenharia de artefatos inteligentes, bem como ajudar a colocar a pesquisa e o desenvolvimento em IA dentro do contexto da ciência mais ampla dos sistemas inteligentes. Por fim, a criação de uma síntese psicológica, neuropsicológica e computacional é verdadeiramente excitante, mas ela requer uma epistemologia madura, a qual discutiremos a seguir.

### 16.2.2 Questões epistemológicas

*Se você não souber para onde está indo, acabará em algum outro lugar...*

— Atribuído a YOGI BERRA

O desenvolvimento da inteligência artificial tem sido moldado por uma série de desafios e questões importantes. Compreensão de linguagem natural, planejamento, raciocínio em situações de incerteza e aprendizado de máquina são exemplos típicos de tipos de problemas que capturam um aspecto essencial de comportamento inteligente. É mais importante ainda o fato de que sistemas inteligentes operando em cada um desses domínios requerem conhecimento de propósito, prática e desempenho em contextos situados e inseridos socialmente. Para realizar isso, devemos examinar o *comprometimento epistemológico* de um programa que se propõe a ser “inteligente”.

Esse comprometimento epistemológico reflete tanto a semântica, que suporta o uso de símbolos, como as estruturas de símbolos empregadas. A tarefa nessas situações é descobrir e explorar as *invariâncias* existentes em um domínio de problema. “Invariância” é um termo usado para descrever regularidades ou aspectos significativos

manipuláveis de ambientes complexos. Na atual discussão, os termos *símbolos* e *sistemas simbólicos* são usados genericamente, incluindo desde a tradição dos símbolos explícitos de Newell e Simon (1976), até os nós e as arquiteturas de rede de um sistema conexionista e os termos emergentes da genética e da vida artificial. Embora as observações que faremos a seguir sejam gerais para a maior parte da IA, focaremos nossa análise em questões de aprendizado de máquina, já que, neste livro, criamos vários exemplos e algoritmos para aprendizado.

Apesar desse progresso, o aprendizado de máquina permanece sendo um dos problemas mais difíceis a ser enfrentado em inteligência artificial. Há três questões que limitam nosso entendimento e o progresso atual nas pesquisas: primeiro, o problema da *generalização e do aprendizado em excesso*; segundo, o papel do *viés induutivo* em aprendizado; e terceiro, o *dilema empiricista*, ou a compreensão da evolução sem restrições. Os dois últimos problemas são correlacionados: o viés induutivo implícito de muitos algoritmos de aprendizado é uma expressão do problema do racionalista de ser influenciado por expectativas, isto é, o que aprendemos frequentemente parece ser uma função direta do que esperamos aprender. Do ponto de vista oposto, vimos a discussão sobre vida artificial, em que há muito poucas expectativas *a priori* do que se deve aprender. É realmente suficiente dizer: “construa isto e isto acontecerá”? De acordo com a afirmação atribuída a Yogi Berra no início desta seção, a resposta é, provavelmente: não! Nas próximas seções, abordamos brevemente essas questões.

### O problema da generalização

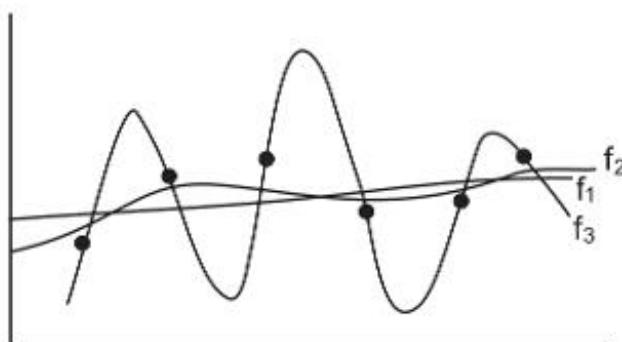
Os exemplos usados para introduzir os vários modelos de aprendizado — simbólico, conexionista e emergente — eram, na maioria das vezes, muito restritos. Por exemplo, as arquiteturas conexionistas frequentemente continham apenas alguns nós, ou uma camada oculta parcial. Para os nossos objetivos, isso é apropriado à medida que as principais leis de aprendizado podem ser adequadamente explicadas no contexto de poucos neurônios ou camadas parciais. Entretanto, isso pode ser muito enganoso, pois as aplicações de redes neurais em geral são consideravelmente maiores, e o problema de escala é importante. Por exemplo, para o aprendizado retropropagação, normalmente é necessário um grande número de exemplos de treinamento com redes grandes para resolver problemas de interesse prático significativo. Muitos pesquisadores comentam extensamente o problema de selecionar números apropriados de valores de entrada, a relação entre parâmetros de entrada e nós ocultos e o número de ensaios de treinamento necessários para a convergência (Hecht-Nielsen, 1990; Zurada, 1992; Freeman e Skapura, 1991). De fato, a qualidade e a quantidade de dados de treinamento é uma questão importante em qualquer algoritmo de aprendizado. Sem um viés apropriado ou um extenso conhecimento embutido do domínio, um algoritmo de aprendizado pode fracassar completamente na tentativa de encontrar padrões em dados ruidosos, esparsos ou mesmo ruins. Além de reconhecer que essas são questões difíceis, importantes e abertas, esse aspecto de “engenharia” do aprendizado não é tratado em nosso livro.

Um problema correlacionado é a questão da “suficiência” em aprendizado. Quando podemos dizer que nossos algoritmos capturaram suficientemente as restrições ou as invariâncias importantes do domínio de um problema? Reservamos uma porção de nossos dados originais para testar nossos algoritmos de aprendizado? A quantidade de dados que temos está relacionada com a qualidade do aprendizado? Talvez o julgamento da suficiência seja heurístico ou estético: nós, seres humanos, muitas vezes vemos nossos algoritmos como “suficientemente bons”.

Ilustremos esse problema de generalização com um exemplo, usando retropropagação para induzir uma função geral de um conjunto de pontos de dados. A Figura 16.2 poderia representar pontos de dados a partir dos quais nosso algoritmo deve generalizar. As linhas nesse conjunto de pontos representam funções induzidas por um algoritmo de aprendizado. Lembre-se de que, uma vez que o algoritmo esteja treinado, desejamos apresentar a ele novos pontos de dados, e o algoritmo deve produzir uma boa generalização para esses dados também.

A função induzida  $f_1$  poderia representar um ajuste acurado de mínimos quadrados médios. Com mais treinamento, o sistema produziria  $f_2$ , que parece ser um “bom” ajuste ao conjunto de pontos de dados; mas  $f_2$  não captura exatamente os pontos de dados. Com mais treinamento ainda, podemos produzir funções que se ajustam exatamente aos dados, mas que oferecem generalizações muito ruins para outros pontos de dados. Esse fenômeno é referido como *treinamento excessivo* de uma rede. Um dos pontos fortes do aprendizado retropropagação é que, em muitos domínios de aplicação, ele realmente produz generalizações efetivas, isto é, aproximações funcionais que se ajustam bem aos dados de treinamento e também tratam novos dados de forma adequada. Entretanto, não

**Figura 16.2** Um conjunto de dados e três aproximações de funções.



é um problema trivial identificar o ponto em que uma rede passa de um estado de treinamento escasso para um estado de treinamento excessivo. É ingênuo esperar que seja possível apresentar dados brutos a uma rede neural, ou a qualquer outra ferramenta de aprendizado, e então simplesmente ficar observando, enquanto ela produz as generalizações mais efetivas e mais úteis para lidar com novos problemas similares.

Concluímos colocando de volta essa questão da generalização no seu contexto epistemológico. Quando resolvedores de problemas fazem e utilizam generalizações (sejam símbolos, nós de redes ou outros quaisquer) durante o processo de solução, eles criam invariantes e, muito provavelmente, sistemas de invariantes, para explorar o domínio do problema/solução e produzir generalizações relacionadas. Esse mesmo ponto de vista trouxe aos vieses do processo de solução do problema o sucesso eventual do empreendimento. Na próxima subseção, tratamos melhor dessa questão.

### **Viés indutivo, o conhecimento a priori racionalista**

As técnicas de aprendizado automático dos capítulos 10 a 13, e também a maioria das técnicas de IA, refletem o viés *a priori* de seus criadores. O problema do viés indutivo é que as representações e estratégias de busca resultantes nos dão um meio para codificar um mundo já interpretado. Raramente elas nos oferecem mecanismos para questionar as nossas interpretações, gerando novos pontos de vista, ou para retroceder e mudar perspectivas quando elas são improdutivas. Esse viés implícito leva à armadilha epistemológica racionalista de ver no mundo exatamente aquilo e apenas aquilo que esperamos ou que estamos treinados para ver.

O papel do viés indutivo precisa se tornar explícito em cada paradigma de aprendizado. No entanto, só porque nenhum viés indutivo é reconhecido, não significa que ele não exista e que ele não afete de forma crítica os parâmetros de aprendizado. No aprendizado simbólico, o viés indutivo é normalmente óbvio, por exemplo, usando uma rede semântica para o aprendizado de conceitos. Nos algoritmos de aprendizado de Winston (1975a, Seção 10.1), os vieses incluem a representação do relacionamento conjuntivo e a importância de usar “erros pequenos” para refinar restrições. Vimos vieses similares no uso de predicados na busca em espaço de versões (Seção 10.2), em árvores de decisão do ID3 (Seção 10.3) ou mesmo nas regras para o Meta-DENDRAL (Seção 10.5).

Entretanto, como avisamos nos capítulos 10 a 13, muitos aspectos do comportamento conexionista e do aprendizado genético também assumem um viés indutivo. Por exemplo, as limitações de redes de perceptrons levaram à introdução de nós ocultos. Poderíamos perguntar qual é a contribuição dos nós ocultos para a generalização da solução. Uma forma de entender o papel dos nós ocultos é que eles acrescentam dimensões no espaço de representação. Como um exemplo simples, vimos na Seção 11.3.3 que os pontos de dados para o problema do *ou-exclusivo* não eram linearmente separáveis em duas dimensões. O peso aprendido no nó oculto, entretanto, fornece outra dimensão à representação. Em três dimensões, os pontos são separáveis usando um plano bidimensional. Dadas as duas dimensões do espaço de entrada e o nó oculto, a camada de saída dessa rede pode ser vista como um perceptron ordinário, que está à procura de um plano que separa os pontos em um espaço tridimensional.

Outra perspectiva complementar é a de que muitos dos “diferentes” paradigmas de aprendizado compartilham vieses indutivos comuns (algumas vezes não óbvios). Apontamos muitos deles: a relação entre a descoberta de agrupamentos com CLUSTER/2 (Seção 10.5), o perceptron (Seção 11.2) e as redes de protótipos (Seção 11.3). Salientamos que a contrapropagação, a rede acoplada que usa aprendizado competitivo não supervisionado sobre uma camada de Kohonen, junto ao aprendizado hebbiano supervisionado em uma camada de Grossberg, é de várias maneiras semelhante ao aprendizado retropropagação. Na rede contrapropagação, os dados agrupados na camada de Kohonen têm um papel semelhante às generalizações aprendidas pelos nós ocultos da rede retropropagação.

As ferramentas que apresentamos são similares em vários aspectos importantes. Na verdade, mesmo a descoberta de protótipos representativos de agrupamentos de dados oferece o caso complementar à aproximação de funções. Na primeira situação, procuramos classificar conjuntos de dados; na segunda, geramos funções que separam explicitamente agrupamentos de dados entre si. Vimos isso quando o algoritmo de classificação por distância mínima usado pelo perceptron também forneceu parâmetros que definem a separação linear dos dados.

Mesmo as generalizações que produzem funções podem ser vistas de vários pontos de vista diferentes. As técnicas estatísticas, por exemplo, têm sido capazes, já há um longo período de tempo, de descobrir correlações entre dados. A expansão iterativa de séries de Taylor pode ser usada para aproximar a maior parte das funções. Algoritmos de aproximação polinomial têm sido usados há mais de um século para aproximar funções por ajuste a pontos de dados.

Em resumo, os comprometimentos assumidos em relação a um esquema de aprendizado (seja ele simbólico, conexionista, emergente ou estocástico) influenciam, em grande parte, os resultados que podemos esperar do esforço para solucionar o problema. Quando apreciamos esse efeito sinergético ao longo de todo o processo de projeto de resolvidores de problema computacionais, podemos frequentemente melhorar as nossas chances de sucesso, bem como interpretar nossos resultados com mais reflexão.

### O dilema empiricista

Se as abordagens atuais de aprendizado de máquina, especialmente o aprendizado supervisionado, possuem um viés indutivo dominante, o aprendizado não supervisionado, incluindo muitas das abordagens genéticas e evolucionárias, tem que enfrentar o problema oposto, algumas vezes denominado *dilema empiricista*. Um tema dessas áreas de pesquisa é o modo pelo qual soluções emergem, alternativas evoluem ou uma população reflete a sobrevivência do mais adaptado. Isso é muito poderoso, especialmente por se situar no contexto da busca paralela e distribuída. Mas há um problema: como podemos saber que estamos em algum lugar se não estamos certos para onde estamos indo?

Há mais de dois mil anos, Platão colocou esse problema nas palavras do escravo Meno:

E como você pode investigar, Sócrates, aquilo que você ainda não sabe? O que você proporá como objeto de investigação? E se você descobrir o que queria, como você saberá se isso é o que você não sabia (Platão, 1961)?

Muitos pesquisadores demonstraram que Meno estava certo; ver Mitchell (1997) e os teoremas denominados *No Free Lunch* (Não há almoço grátis), de Wolpert e Macready (1995). O empiricista, na verdade, requer as sobras de um conhecimento racionalista *a priori* para salvar a ciência!

Apesar de tudo, permanece uma grande excitação acerca dos modelos de aprendizado não supervisionados e evolucionários, por exemplo, a criação de redes baseadas em exemplares, ou de minimização de energia, que podem ser vistas como atratores de ponto fixo ou bacias de atração para invariâncias relacionais complexas. Observamos como os pontos de dados “se acomodam” em torno de atratores e somos tentados a ver essas novas arquiteturas como ferramentas para modelar fenômenos dinâmicos. Quais são, poderíamos perguntar, os limites de computação desses paradigmas?

Na verdade, pesquisadores demonstraram (Siegelman e Sontag, 1991) que as redes recorrentes são computacionalmente completas, isto é, equivalentes à classe das máquinas de Turing. Essa equivalência com as máquinas de Turing estende resultados anteriores: Kolmogorov (1957) mostrou que, para qualquer função contínua, existe uma rede neural que calcula essa função. Foi mostrado, também, que uma rede retropropagação com uma camada oculta pode aproximar qualquer função de uma classe mais restrita de funções contínuas (Hecht-Nielsen, 1989).

Analogamente, vimos na Seção 11.3 que von Neumann criou os autômatos de estados finitos, que eram Turing completos. Assim, as redes conexionistas e os autômatos de estados finitos parecem ser duas classes de algoritmos capazes de computar praticamente qualquer função computável. Além disso, os vieses indutivos se aplicam SIM a modelos não supervisionados, tanto a modelos genéticos de aprendizado como a modelos emergentes; vieses representacionais se aplicam ao projeto de nós, redes e aos genomas; e vieses algorítmicos se aplicam à busca, à recompensa e aos operadores de seleção.

O que é, então, que os aprendizes não supervisionados, sejam eles conexionistas, genéticos ou máquinas de estados finitos evolutivas nas suas várias formas, podem nos oferecer?

1. Uma das características mais atrativas do aprendizado conexionista é que a maioria dos modelos é guiada pelos dados ou exemplos. Isto é, embora a sua arquitetura seja projetada explicitamente, eles aprendem por exemplos, generalizando a partir de dados em um domínio particular de problema. Mas ainda surge a questão de se os dados são suficientes ou suficientemente limpos para não perturbar o processo de solução. E como o projetista pode saber isso?
2. Os algoritmos genéticos também permitem uma busca poderosa e flexível de um espaço de problema. A busca genética é guiada tanto pela diversidade forçada por mutações quanto por operadores, como a recombinação (*crossover*) e a inversão, que preservam aspectos importantes da informação paterna em gerações sucessivas. Como o projetista do programa pode preservar e estimular esse compromisso entre diversidade e preservação?
3. Os algoritmos genéticos e as arquiteturas conexionistas podem ser vistos como ocorrências do processamento paralelo e assíncrono. Eles fornecem resultados realmente por meio do esforço assíncrono paralelo, que não seria possível com a programação sequencial explícita?
4. Embora a inspiração neural e sociológica não seja importante para muitos praticantes modernos do aprendizado conexionista e genética, essas técnicas refletem muitos aspectos importantes da evolução e seleção natural. Vimos modelos para aprendizado por redução de erro com os modelos do perceptron, da rede retropropagação e o modelo hebbiano. Vimos, também, as redes autoassociativas de Hopfield na Seção 11.3.4. Vários modelos de evolução foram refletidos nos paradigmas do Capítulo 12.
5. Por fim, todos os paradigmas de aprendizado são ferramentas para a investigação empírica. Conforme capturamos as invariantes do nosso mundo, será que as nossas ferramentas serão suficientemente poderosas e expressivas para fazer outras perguntas relacionadas à natureza da percepção, do aprendizado e da compreensão?

Na próxima seção, propomos que uma epistemologia construtivista, com os métodos experimentais da inteligência artificial moderna, ofereça as ferramentas e técnicas para continuar a construção e a exploração de uma ciência dos sistemas inteligentes.

### A reconciliação construtivista

*Teorias são como redes: aquele que as lança é quem captura...*

—L. WITTGENSTEIN

Os construtivistas defendem a hipótese de que toda a compreensão é resultado de uma interação entre padrões de energia no mundo e categorias mentais impostas no mundo pelo agente cognitivo (Piaget, 1954, 1970; von Glaserfeld, 1978). Usando descrições de Piaget, *assimilamos* fenômenos externos de acordo com nossa compreensão atual e *acomodamos* nossa compreensão às “demandas” do fenômeno.

Os construtivistas frequentemente usam o termo *esquema* para descrever uma estrutura *a priori* usada para organizar a experiência do mundo externo. Esse termo é originário do psicólogo britânico Bartlett (1932) e suas raízes filosóficas remontam a Kant (1781/1964). Por esse ponto de vista, a observação não é passiva e neutra, mas sim ativa e interpretativa.

A informação percebida, o conhecimento *a posteriori* de Kant, nunca se ajusta precisamente aos nossos esquemas pré-concebidos, *a priori*. Dessa tensão, os vieses baseados em esquemas que o sujeito utiliza para organizar a experiência são modificados ou substituídos. A necessidade de acomodação em face das interações

malsucedidas com o ambiente provoca um processo de equilíbrio cognitivo. Assim, a epistemologia cognitiva é fundamentalmente baseada em evolução e refinamento. Uma consequência importante do construtivismo é que a interpretação de qualquer situação envolve a imposição dos conceitos e categorias do observador na realidade (um viés indutivo).

Quando Piaget (1954, 1970) propôs uma abordagem construtivista à compreensão, ele a chamou de *epistemologia genética*. A falta de um ajuste confortável dos esquemas atuais ao mundo “como ele é” cria uma tensão cognitiva. Essa tensão provoca um processo de revisão de esquemas. A revisão de esquemas, a *acomodação* de Piaget, explica a evolução continuada da compreensão do agente em direção ao *equilíbrio*.

A revisão de esquemas e o movimento continuado em direção ao equilíbrio é uma predisposição genética de um agente para uma acomodação às estruturas da sociedade e ao mundo externo. Ela combina essas duas forças e representa uma predisposição personificada para a sobrevivência. A modificação de esquemas é uma função tanto *a priori* de nossa genética como *a posteriori* da sociedade e do mundo. Ela reflete a personificação de um agente controlado por sobrevivência, de uma existência no espaço e no tempo.

Existe uma mistura aqui das tradições empíricista e racionalista, mediada pelos objetivos de sobrevivência do agente. Como seres personificados, os agentes não podem compreender nada além do que passa primeiro por seus sentidos. Pela acomodação, agentes sobrevivem por meio do aprendizado dos padrões gerais de um mundo externo. O que é percebido é mediado pelo que é esperado; o que é esperado é influenciado pelo que é percebido: isto é, essas duas funções podem ser entendidas apenas em termos uma da outra. Nesse sentido, os modelos estocásticos — tanto Bayesianos quanto Markovianos — são apropriados, pois a experiência prévia condiciona as interpretações atuais (Doya et al., 2007).

Por fim, nós, como agentes, raramente nos damos conta dos esquemas que permitem nossa interação com o mundo. Apesar de sermos as fontes de viés e preconceito, tanto na ciência quanto na sociedade, frequentemente estamos inconscientes dos esquemas *a priori*. Eles são os construtores de nosso equilíbrio com o mundo e (normalmente) não um elemento perceptivo de uma vida mental consciente.

Por que uma epistemologia construtivista é particularmente útil para tratar dos problemas de inteligência artificial? Como um agente, dentro de um ambiente, pode compreender seu próprio conhecimento dessa situação? Acreditamos que o construtivismo ajuda a tratar do problema do *acesso epistemológico* da filosofia e da psicologia. Por mais de um século há uma discussão em ambas as áreas entre duas facções, uma facção positivista, que propõe inferir fenômenos mentais a partir do comportamento físico observável, e uma abordagem mais fenomenológica, que permite o uso da primeira pessoa observadora para acessar os fenômenos cognitivos. Esse partidarismo existe porque os dois modos de acesso a fenômenos psicológicos requerem alguma forma de construção de modelo e inferência.

Em comparação com objetos físicos, como cadeiras e portas que, ingenuamente, parecem ser diretamente acessíveis, os estados mentais e disposições de um sujeito parecem particularmente difíceis de caracterizar. Acreditamos que a dicotomia entre o acesso direto aos fenômenos físicos e o acesso indireto aos fenômenos mentais é ilusória. A análise construtivista mostra que nenhuma experiência é possível sem o uso de algum modelo ou esquema para organizar essa experiência. Na investigação científica, bem como nas nossas experiências usuais, isso implica que *todos* os acessos a fenômenos do mundo se dão por meio da exploração, da aproximação e do refinamento contínuo de modelos.

### **Logo, qual é o projeto do praticante de IA?**

Como praticantes de IA, somos construtivistas. Montamos, testamos e refinamos modelos. Mas o que aproximamos em nossa criação de modelo? Discutimos essa questão nos próximos parágrafos, mas primeiro fazemos uma observação epistemológica: em vez de tentar capturar a essência das “coisas fora” de nós, o resolvedor de problemas de IA é mais bem servido tentando simular as heurísticas de criação, refinamento e equilíbrio do modelo do próprio agente inteligente.

Apenas o solipsista extremo (ou mentalmente desafiado) negaria a “realidade” do mundo extramatéria. Mas o que é esse chamado “mundo real”? Além de ser uma combinação complexa de “coisas materiais” e “coisas abstratas”, ele também é um sistema de átomos, moléculas, quarks, gravidade, relatividade, células, DNA e (talvez ainda) supersequências. Para todos esses conceitos existem apenas modelos exploratórios controlados pelos requisiti-

tos exploratórios de agentes controlados por equilíbrio. Novamente, esses modelos exploratórios não são referentes a um mundo externo. Em vez disso, eles capturam as tensões de equilíbrio dinâmico do agente inteligente e social, de uma inteligência material evoluindo e continuamente calibrando-se dentro do espaço e do tempo.

Mas o acesso e a criação do “real” também é alcançado por meio do comprometimento do agente. Um agente personificado *cria o real* por meio de uma afirmação existencial de que um modelo percebido de suas expectativas é bom o bastante para tratar de algumas de suas necessidades e propósitos práticos. Esse ato de compromisso serve de base para símbolos e sistemas de símbolos que o agente usa em seus contextos material e social. Essas construções têm fundamentação porque são afirmadas como boas o bastante para alcançar os aspectos de sua finalidade. Essa fundamentação também é vista no uso da linguagem do agente. Searle (1969) está correto em sua noção de fenômenos de fala como *atos*. Essa questão de *fundamentação* faz parte do motivo pelo qual os computadores possuem problemas fundamentais com expressões de inteligência, incluindo suas demonstrações de linguagem e aprendizado. Que aptidão poderia ser dada a um computador que lhe conceda propósitos e objetivos apropriados? Embora Dennett (1987) atribua fundamentação a um computador resolvendo problemas que exigem e usam “inteligência”, a falta de fundamentação *suficiente* é vista com facilidade nas simplificações, fragilidade e apreciação do contexto quase sempre limitado.

O uso e a fundamentação de símbolos por agentes animados implica ainda mais. As particularidades da personificação do agente humano e os contextos sociais mediam suas interações com seu mundo. Alguns componentes críticos que dão suporte a metáforas de compreensão, aprendizado e linguagem, que mediam nossa compreensão da arte, vida e amor, são: sistemas auditivos e visuais sensíveis a determinada largura de banda; exibição do mundo como um bípede ereto, com braços, pernas, mãos; estar em um mundo com clima, estações, sol e escuridão; parte de uma sociedade com objetivos e propósitos em evolução; um indivíduo que nasce, reproduz e morre.

Devo igualar-te a um dia de verão?  
Mais afável e belo é o teu semblante:  
O vento esfolha maioinda em botão,  
Dura o termo estival um breve instante.

*Shakespeare, Soneto XVIII*

Concluímos com um resumo final de questões críticas que dão suporte e delimitam nossos esforços na criação de uma ciência de sistemas inteligentes.

### 16.3 IA: Questões atuais e direções futuras

*como o geômetra que procura a  
quadratura do círculo e não consegue encontrá-la  
pensando sobre o princípio necessário para isso,  
assim eu me encontrava ante aquela nova visão...*

— DANTE, *Paradiso*

Embora o emprego de técnicas de IA para solucionar problemas práticos tenha demonstrado a sua utilidade, o uso dessas técnicas para fundar uma ciência geral da inteligência é um problema difícil e permanente. Nesta seção final, retornamos às questões que nos levaram a entrar no campo da inteligência artificial e a escrever este livro: é possível especificar uma descrição computacional formal dos processos que habilitam a inteligência?

A caracterização computacional da inteligência começa com a especificação abstrata de dispositivos computacionais. As pesquisas realizadas nos anos 1930, 1940 e 1950 iniciaram essa tarefa, com Turing, Post, Markov e Church, que contribuíram com formalismos que descrevem a computação. O objetivo dessas pesquisas não era apenas especificar o que se pretendia calcular, mas especificar os limites para o que podia ser calculado. A máquina universal de Turing (Turing, 1950) é a especificação mais comumente estudada, embora as regras de reescrita

de Post, a base para a computação por sistema de produção (Post, 1943), sejam também uma contribuição importante. O modelo de Church (1941), baseado em funções parcialmente recursivas, é uma fundamentação importante para as linguagens funcionais modernas, como Scheme, Ocaml e Standard ML.

Os teóricos provaram que todos esses formalismos têm poder computacional equivalente, à medida que qualquer função computável por um deles é computável pelos outros. Na verdade, é possível mostrar que a máquina universal de Turing é equivalente a qualquer dispositivo computacional moderno. Com base nesses resultados, a tese de Church-Turing formula o argumento ainda mais forte de que não se pode definir qualquer modelo de computação que seja mais poderoso do que esses modelos conhecidos. Uma vez que estabelecemos equivalência de especificações computacionais, libertamo-nos do meio de mecanizar essas especificações: podemos implementar nossos algoritmos com válvulas, silício, protoplasma ou brinquedos de armar. O projeto automatizado em um meio pode ser visto como equivalente a mecanismos em outro meio. Isso torna o método de investigação empírico ainda mais crítico, uma vez que experimentamos em um meio para testar nosso entendimento de mecanismos implementados em outro meio.

Uma das possibilidades é que a máquina universal de Turing e Post seja genérica demais. Paradoxalmente, a inteligência pode exigir um mecanismo computacional menos poderoso, com um controle mais focado. Levesque e Brachman (1985) sugeriram que a inteligência humana pode requerer representações mais eficientes computacionalmente (embora menos expressivas), como cláusulas de Horn para o raciocínio, a restrição de conhecimento factual para literais fundamentais e o uso de sistemas de manutenção da verdade computacionalmente tratáveis. Os modelos de inteligência emergentes e os baseados em agentes parecem também aderir a essa filosofia.

Outro ponto abordado pela equivalência formal dos nossos modelos de mecanismos é a questão da dualidade e o problema da mente-corpo. Pelo menos desde o tempo de Descartes (ver Seção 1.1), filósofos têm levantado a questão da interação e da integração entre mente, consciência e corpo físico. Eles ofereceram todo tipo de resposta possível, desde o materialismo total até a negação da existência material, e até mesmo a intervenção de um deus benigno! A IA e a ciência cognitiva rejeitam o dualismo cartesiano, preferindo um modelo material de mente, baseado na implementação física ou na instanciação de símbolos, a especificação formal de mecanismos computacionais para manipular esses símbolos, a equivalência de paradigmas representacionais e a mecanização do conhecimento e da destreza em modelos personificados. O sucesso dessas pesquisas é uma indicação da validade desse modelo (Johnson-Laird, 1988; Dennett, 1987; Luger et al., 2002).

Entretanto, ainda restam muitas questões dentro da fundamentação epistemológica da inteligência em um sistema físico. A seguir, resumimos novamente várias dessas questões críticas.

- 1. O problema da representação.** Newell e Simon levantaram a hipótese de que o sistema simbólico físico e a busca são as caracterizações necessárias e suficientes de inteligência (ver Seção 16.1). Os sucessos dos modelos neurais ou subsimbólicos e das abordagens genética e emergente para a inteligência são refutações à hipótese simbólica física? Ou eles são simplesmente outras ocorrências dela?

Mesmo uma interpretação fraca dessa hipótese — que o sistema físico simbólico é um modelo *suficiente* de inteligência — produziu muitos resultados poderosos e úteis no campo moderno da ciência cognitiva. Isso significa que podemos implementar sistemas simbólicos físicos que demonstrarão comportamento inteligente. A suficiência permite a criação e o teste de modelos simbólicos para muitos aspectos da capacidade humana (Pylyshyn, 1984; Posner, 1989). Mas a interpretação forte — que o sistema simbólico físico e a busca são *necessários* para a atividade inteligente — permanece com uma questão em aberto (Searle, 1980; Weizenbaum, 1976; Winograd e Flores, 1986; Dreyfus e Dreyfus, 1985; Penrose, 1989).

- 2. O papel da personificação em cognição.** Uma das principais suposições da hipótese do sistema simbólico físico é que a implementação particular desse sistema é irrelevante para o seu desempenho; tudo o que importa é a sua estrutura formal. Isso foi desafiado por vários pensadores (Searle, 1980; Johnson, 1987; Agre e Chapman, 1987; Brooks, 1989; Varela et al., 1993), que afirmam, essencialmente, que os requisitos da ação inteligente no mundo exigem uma personificação física que permita ao agente estar totalmente integrado naquele mundo. A arquitetura dos computadores modernos não permite esse grau de situabilidade, requerendo que uma inteligência artificial interaja com seu mundo por meio da janela extremamente limitada de dispositivos de entrada/saída atuais. Se esse desafio estiver correto, então, embora a intel-

gência de máquina seja possível, ela requererá uma interface muito diferente daquela disponível nos computadores atuais. (Para ver mais comentários sobre esse assunto, consulte a Seção 15.0, sobre aspectos da compreensão de linguagem natural, e a Seção 16.2.2, sobre questões epistemológicas.)

3. **Cultura e inteligência.** Tradicionalmente, a inteligência artificial tem focado na mente individual como a única fonte de inteligência; agimos como se uma explicação do modo como o cérebro codifica e manipula conhecimento fosse uma explicação completa das origens da inteligência. Entretanto, poderíamos argumentar que a melhor forma de considerar o conhecimento é como uma construção social, e não individual. Em uma teoria da inteligência *baseada em meme* (Edelman, 1992), a própria sociedade carrega componentes essenciais de inteligência. É possível que uma compreensão do contexto social do conhecimento e do comportamento humano seja tão importante para uma teoria da inteligência quanto uma compreensão da dinâmica da mente ou do cérebro individual.
4. **Caracterizando a natureza da interpretação.** A maioria dos modelos computacionais na tradição representacional trabalha com um domínio já interpretado, isto é, existe um comprometimento implícito e *a priori* do projetista do sistema com um contexto interpretativo. Por esse comprometimento há pouca capacidade de modificar contextos, objetivos ou representações, conforme a solução do problema evoluí. Atualmente, existe pouco esforço para desvelar o processo pelo qual o ser humano constrói interpretações. A visão de Tarskian de um compromisso semântico com um mapeamento entre símbolos e objetos é certamente fraca demais, e não explica, por exemplo, o fato de que um domínio possa ter diferentes interpretações dependendo de diferentes objetivos práticos. Os linguistas geralmente tentam remediar as limitações da semântica de Tarskian adicionando uma teoria de pragmática (Austin, 1962). As pesquisas em análise de discurso, com a sua dependência fundamental do uso de símbolos em um contexto, lidaram extensamente com essas questões nos últimos anos, embora o problema seja muito mais amplo, pois ele lida com o uso e a falha de ferramentas referenciais em geral (Lave, 1988; Grosz e Sidner, 1990). A tradição semiótica iniciada por C. S. Peirce (1958) e continuada por Eco, Sebeok e outros (Eco, 1976; Grice, 1975; Sebeok, 1985) adota uma abordagem mais radical para a linguagem. Ela coloca as expressões simbólicas dentro do contexto mais amplo dos sinais e interpretação de sinais. Isso sugere que a significação de um símbolo pode ser entendida apenas no contexto de seu papel como intérprete, isto é, no contexto de uma interpretação e de uma interação com o ambiente (ver Seção 16.2.2).
5. **Indeterminação representacional.** A conjectura da indeterminação representacional de Anderson (Anderson, 1978) sugere que, em princípio, seria impossível determinar qual esquema representacional aproxima, de forma mais apropriada, a solução humana de um problema no contexto de uma ação particular de desempenho habilidoso. Essa conjectura se baseia no fato de que todo esquema representacional está indissociavelmente ligado a uma arquitetura computacional maior, bem como a estratégias de busca. Na análise detalhada da perícia humana, pode ser impossível controlar suficientemente o processo, de modo que possamos determinar a representação; ou estabelecer uma representação a ponto de um processo poder ser determinado unicamente. Assim como o princípio da incerteza da física, em que um fenômeno pode ser alterado pelo mesmo processo que o mede, isso é uma preocupação importante para construir modelos de inteligência, mas não precisa limitar a sua utilidade.  
Mais importante ainda é o fato de que as mesmas críticas podem ser dirigidas ao próprio modelo computacional, em que os vieses indutivos de símbolos e de busca no contexto da hipótese de Church/Turing ainda não restringem suficientemente um sistema. A necessidade percebida de um esquema representacional ótimo pode muito bem ser o resquício de um sonho racionalista, enquanto o cientista simplesmente requer modelos robustos o suficiente para restringir questões empíricas. A prova de qualidade de um modelo está na sua capacidade de oferecer uma interpretação, de predizer e de ser revisado.
6. **A necessidade de conceber modelos computacionais que sejam refutáveis.** Popper (1959) e outros afirmaram que teorias científicas precisam ser refutáveis. Isso significa que devem existir circunstâncias sob as quais o modelo *não* é uma aproximação bem-sucedida do fenômeno. A razão óbvia para isso é que *qualquer* número de ocorrências experimentais positivas não é suficiente para a confirmação de um modelo. Além disso, muitas pesquisas novas são realizadas em resposta à falha de teorias existentes.

A natureza geral da hipótese do sistema simbólico físico, bem como dos modelos de inteligência situados e emergentes, pode tornar impossível que eles sejam refutáveis e, portanto, de uso limitado como modelos. A mesma crítica pode ser dirigida às conjecturas da tradição fenomenológica (ver item 7). Algumas estruturas de dados da IA, como as redes semânticas, são também tão gerais que elas podem modelar quase qualquer coisa descritível ou, como a máquina universal de Turing, podem calcular qualquer função computável. Assim, quando perguntamos a um pesquisador de IA ou da ciência cognitiva em que condições o seu modelo para inteligência *não* funciona, normalmente é muito difícil para ele encontrar a resposta.

7. **As limitações do método científico.** Vários pesquisadores (Winograd e Flores, 1986; Weizenbaum, 1976) afirmam que os aspectos mais importantes da inteligência não são modelados e, em princípio, não podem ser modelados e, em particular, não com uma representação simbólica, seja ela qual for. Nessas áreas estão incluídos o aprendizado, a compreensão de linguagem natural e a produção de atos da fala. Essas questões têm raízes profundas em nossa tradição filosófica. As críticas de Winograd e Flores, por exemplo, são baseadas em questões levantadas em fenomenologia (Husserl, 1970; Heidegger, 1962).

Podemos traçar as origens da maior parte das suposições da IA moderna retrocedendo desde Carnap, Frege e Leibniz, passando por Hobbes, Locke e Hume, até Aristóteles. Essa tradição afirma que os processos inteligentes seguem leis universais e são, em princípio, comprehensíveis.

Heidegger e seus seguidores representam uma abordagem alternativa para compreender a inteligência. Para Heidegger, a consciência reflexiva está fundamentada em um mundo de experiências personificadas (um mundo vivo). Essa posição, compartilhada por Winograd e Flores, Dreyfus e outros, afirma que a compreensão que uma pessoa tem das coisas está baseada na atividade prática de “usá-las” ao lidar diariamente com o mundo. Esse mundo é sobretudo um contexto socialmente organizado de papéis e propósitos. Esse contexto, e a atividade humana dentro dele, não é algo explicável por proposições e entendido por teoremas. Ele é um processo que molda e que é continuamente criado. Em um sentido fundamental, a pericia humana não se traduz em saber *o que*, mas sim, em um mundo de normas sociais evolutivas e propósitos implícitos, em saber *como*. Somos inherentemente incapazes de formular nosso conhecimento e a maior parte de nosso comportamento inteligente em uma linguagem, seja ela formal ou natural.

Consideremos esse ponto de vista. Em primeiro lugar, como uma crítica à tradição racionalista *pura*, ele está correto. O racionalismo afirma que a atividade, a inteligência e a responsabilidade humanas podem, ao menos em princípio, ser representadas, formalizadas e compreendidas. As pessoas mais reflexivas não acreditam que esse seja o caso, reservando papéis importantes para a emoção, a autoafirmação e o comprometimento responsável (no mínimo!). O próprio Aristóteles disse, em seu *Essay on Rational Action*, “Por que é que eu não me sinto compelido a fazer o que me foi imposto?”. Há várias atividades humanas fora do domínio do método científico que desempenham um papel essencial na interação humana responsável; elas não podem ser reproduzidas por máquinas ou abolidas por elas.

Entretanto, a tradição científica de examinar dados, construir modelos, executar experimentos e examinar resultados, refinando os modelos para novos experimentos, trouxe para a humanidade um nível importante de compreensão, explanação e capacidade de predição. O método científico é uma poderosa ferramenta para aumentar a compreensão humana. Apesar disso, continua existindo uma série de advertências a essa abordagem que os cientistas precisam compreender.

Primeiro, eles não devem confundir o modelo com o fenômeno que está sendo modelado. O modelo nos permite aproximar de forma progressiva o fenômeno: haverá sempre, necessariamente, um “resíduo” que não é explicável do ponto de vista empírico. Nesse sentido, a indeterminação representacional também *não* é a questão. Um modelo é usado para explorar, explicar e prever; se ele permite aos cientistas fazer isso, ele é bem-sucedido (Kuhn, 1962). Na verdade, modelos diferentes podem explicar adequadamente diferentes aspectos de um único fenômeno, por exemplo, as teorias da luz como onda e partícula.

Além disso, quando pesquisadores afirmam que certos aspectos de fenômenos inteligentes estão fora do escopo e dos métodos da tradição científica, essa mesma afirmação pode ser verificada somente utilizando essa mesma tradição. O método científico é a única ferramenta que temos para explicar em que sentido as questões podem ainda estar fora de nosso entendimento atual. Todos os pontos de vista,

mesmo aqueles da tradição fenomenológica, para terem algum significado, precisam ter relação com as noções de explicação atuais — até mesmo para ser coerente sobre a extensão em que um fenômeno não pode ser explicado.

O aspecto mais excitante do trabalho em inteligência artificial é que, para ser coerente e contribuir para esse empreendimento, precisamos abordar essas questões. Para entender a solução de problemas, o aprendizado e a linguagem, precisamos compreender o nível filosófico de representações e conhecimento. De uma forma modesta, somos solicitados a resolver a tensão de Aristóteles entre a *teoria* e a *prática*, a realizar uma união da compreensão com a prática, da teoria com a prática, a viver entre a ciência e a arte. Como praticantes de IA, somos construtores de ferramentas. Nossas representações, algoritmos e linguagens são ferramentas para o projeto e a construção de mecanismos que exibem comportamento inteligente. Por meio de experimentos, testamos a sua adequação computacional para resolver problemas, mas também nossa própria compreensão dos fenômenos inteligentes.

Na realidade, temos uma tradição nesse sentido: Descartes, Leibniz, Bacon, Pascal, Hobbes, Babbage, Turing e outros, cujas contribuições apresentamos no Capítulo 1. A engenharia, a ciência e a filosofia; a natureza de ideias, conhecimento e perícia; o poder e as limitações de formalismos e mecanismos; estas são as limitações e tensões com as quais precisamos conviver e a partir das quais continuamos as nossas explorações.

## 16.4 Epílogo e referências

Recomendamos ao leitor as referências do final do Capítulo 1 e acrescentamos *Computation and Cognition* (Pylyshyn, 1984) e *Understanding Computers and Cognition* (Winograd e Flores, 1986). Para questões da ciência cognitiva, veja Newell e Simon (1972), Pylyshyn (1973, 1980), Norman (1981), Churchland (1986), Posner (1989), Luger (1994), Franklin (1995), Baillard (1997), Elman et al. (1996) e Jeannerod (1997).

Haugeland (1981, 1997), Dennett (1978, 2006) e Smith (1996) descrevem os fundamentos filosóficos de uma ciência dos sistemas inteligentes. O livro de Anderson (1990) sobre psicologia cognitiva oferece exemplos valiosos de modelos de processamento de informação. Pylyshyn (1984) e Anderson (1978, 1982, 1983a) fornecem descrições detalhadas de muitas questões críticas em ciência cognitiva, incluindo uma discussão da indeterminação representacional. Dennett (1991) aplica a metodologia da ciência cognitiva para uma exploração da estrutura da própria consciência. Recomendamos, também, livros sobre filosofia da ciência (Popper, 1959; Kuhn, 1962; Bechtel, 1988; Hempel, 1965; Lakatos, 1976; Quine, 1963).

Por fim, em *Philosophy in the Flesh*, Lakoff e Johnson (1999) sugerem possíveis respostas ao problema da fundamentação. *The Embodied Mind* (Varela et al., 1993), Suchman (1987) e *Being There* (Clark, 1997) descrevem aspectos importantes da personificação como suporte à inteligência.

Fornecemos ao leitor os endereços de dois importantes grupos:

The Association for the Advancement of Artificial Intelligence  
445 Burgess Drive, Menlo Park, CA 94025

Computer Professionals for Social Responsibility  
P.O. Box 717, Palo Alto, CA 94301



# *Referências*

- ABELSON, H.; SUSSMAN, G. J. **Structure and Interpretation of Computer Programs**. Cambridge, MA: MIT Press, 1985.
- ACKLEY, D. H.; HINTON, G. E.; SEJNOWSKI, T.J. A learning algorithm for Boltzmann machines. **Cognitive Science**, v. 9, 1985.
- ACKLEY, D. H.; LITTMAN, M. Interactions between learning and evolution. In: LANGTON et al. (eds). **Artificial Life II**. Reading, MA: Addison-Wesley, 1992.
- ADLER, M. R. et al. Conflict resolution strategies for nonhierarchical distributed agents. **Distributed Artificial Intelligence**, Vol. 112. San Francisco: Morgan Kaufmann, 1989.
- AGRE, P.; CHAPMAN, D. Pengi: an implementation of a theory of activity. **Proceedings of the Sixth National Conference on Artificial Intelligence**, p. 268-272. CA: Morgan Kaufmann, 1987.
- AHO, A. V.; ULLMAN, J. D. **Principles of Compiler Design**. Reading, MA: Addison-Wesley, 1977.
- AJI, S. M.; MCELIECE, R. J. The generalized distributive law. **IEEE Transactions on Information Theory**, v. 46, n. 2, p. 498-519, 2000
- ALLEN, J. **Natural Language Understanding**. Menlo Park, CA: Benjamin/Cummings, 1987.
- \_\_\_\_\_. **Natural Language Understanding**. 2. ed. Menlo Park, CA: Benjamin/Cummings, 1995.
- ALLEN, J.; HENDLER, J.; TATE, A. **Readings in Planning**. Los Altos, CA: Morgan Kaufmann, 1990.
- ALTY, J. L.; COOMBS, M. J. **Expert Systems: Concepts and Examples**. Manchester: NCC Publications, 1984.
- ANDERSON, J. A. et al. Distinctive features, categorical perception and probability learning: Some applications of a neural model. **Psychological Review**, v. 84, p. 413-451, 1977.
- ANDERSON, J. R. Arguments concerning representations for mental imagery. **Psychological Review**, v. 85, p. 249-277, 1978.
- \_\_\_\_\_. Acquisition of cognitive skill. **Psychological Review**, v. 89, p. 369-406, 1982.
- \_\_\_\_\_. Acquisition of proof skills in geometry. In: Michalski et al. **Machine Learning: An Artificial Intelligence Approach**, p. 191-219. Palo Alto, CA : Tioga, 1983.
- \_\_\_\_\_. **The Architecture of Cognition**. Cambridge, MA: Harvard University Press, 1983b.
- \_\_\_\_\_. **Cognitive Psychology and its Implications**. Nova York: W. H. Freeman, 1990.
- ANDERSON, J. R.; BOWER, G.H. **Human Associative Memory**. Hillsdale, NJ: Erlbaum, 1973.
- ANDREWS, P. **An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof**. Nova York: Academic Press, 1986.
- APPELT, D. **Planning English Sentences**. Londres: Cambridge University Press, 1985.
- ARBIB, M. Simple self-reproducing universal automata. **Information and Control**, v. 9, p. 177-189, 1966.
- ASPRAY, W.; BURKS, A. W. (Ed.). **Papers of John Von Neumann on Computing and Computer Theory**. Cambridge, MA: MIT Press, 1987.
- AUER, P.; HOLTE, R. C.; MAASS, W. Theory and application of agnostic pac-learning with small decision trees. **Proceedings of the Twelfth International Conference on Machine Learning**, p. 21-29. San Francisco: Morgan Kaufmann, 1995.
- AUSTIN, J. L. **How to Do Things with Words**. Cambridge, MA: Harvard University Press, 1962.
- BACH, E.; HARMS, R. (Eds.). **Universals of Linguistic Theory**. Nova York: Holt, Rinehart e Winston, 1968.
- BACON, F. **Novum Organum**. Londini: Apud (Bonham Norton e) Joannem Billium, 1620.
- BALLARD, D. **An Introduction to Natural Computation**. Cambridge, MA: MIT Press, 1997.
- BALZER, R. et al. HEARSAY III: A domain independent framework for expert systems. **Proceedings AAAI 1980**. Menlo Park CA: AAAI Press, 1980.
- BAREISS, E. R.; PORTER, B.W.; WEIR, C.C. Protos: An exemplar-based learning apprentice. **International Journal of Man-Machine Studies**, v. 29, p. 549-561, 1988.

- BARKER, V. E.; O'CONNOR, D.E. 1989. Expert Systems for configuration at DIGITAL: XCON and Beyond. **Communications of the ACM**, v. 32, n. 3, p. 298-318.
- BARKOW, J. H.; COSMIDES, L.; TOOBY, J. **The Adapted Mind**. Nova York: Oxford Univ. Press, 1992.
- BARR, A.; FEIGENBAUM, E. (Eds.). **Handbook of Artificial Intelligence**. Los Altos, CA: William Kaufman, 1989.
- BARTLETT, F. **Remembering**. Londres: Cambridge University Press, 1932.
- BATESON, G. **Mind and Nature: A Necessary Unity**. Nova York: Dutton, 1979.
- BAYES, T. Essay towards solving a problem in the doctrine of chances. **Philosophical Transactions of the Royal Society of London**, p. 370-418, Londres: The Royal Society, 1763.
- BECHTEL, W. **Philosophy of Mind**. Hillsdale, NJ: Lawrence Erlbaum, 1988.
- BELLMAN, R. E. **Dynamic Programming**. Princeton, NJ: Princeton University Press, 1956.
- BENSON, S. Action Model Learning and Action Execution in a Reactive Agent. **Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)**, 1995.
- BENSON, S.; NILSSON, N. Reacting, Planning and Learning in an Autonomous Agent. In: Furukawa, K.; Michie, D.; Muggleton, S. (Eds.). **Machine Intelligence**, v. 14. Oxford: Clarendon Press, 1995.
- BERGER, A. et al. The Candide System for Machine Translation. **Human Language Technology: Proceedings of the ARPA Workshop on Speech and Natural Language**. San Mateo, CA: Morgan Kaufmann, 1994.
- BERNARD, D. E. et al. Design of the remote agent experiment for spacecraft autonomy. **Proceedings of the IEEE Aerospace Conference**, 1998.
- BERTSEKAS, D. P.; TSITSIKLIS, J.N. **Neuro-Dynamic Programming**. Belmont, MA: Athena, 2005.
- BHAGAT, P. M. **Pattern Recognition in Industry**. Amsterdam: Elsevier, 1996.
- BISHOP, C. H. **Neural Networks for Pattern Recognition**. Oxford: Oxford University Press, 1995.
- BLEDSOE, W. W. Splitting and reduction heuristics in automatic theorem proving. **Artificial Intelligence**, v. 2, p. 55-77, 1971.  
\_\_\_\_\_. Non-resolution theorem proving. **Artificial Intelligence**, v. 9, 1977.
- BOBROW, D. G. Dimensions of representation. In: BOBROW, D. G.; COLLINS, A. (Eds.). **Representation and Understanding: Studies in cognitive science**. Nova York: Academic Press, 1975.
- BOBROW, D. G.; COLLINS A. (Eds.). **Representation and Understanding**. Nova York: Academic Press, 1975.
- BOBROW, D. G.; WINOGRAD, T. An overview of KRL, a knowledge representation language. **Cognitive Science**, v. 1, n. 1, 1977.
- BOND, A. H.; GASSER, L. (Eds.). **Readings in Distributed Artificial Intelligence**. San Francisco: Morgan Kaufmann, 1988.
- BOOLE, G. **The Mathematical Analysis of Logic**. Cambridge: MacMillan, Barclay & MacMillan, 1847.  
\_\_\_\_\_. **An Investigation of the Laws of Thought**. Londres: Walton & Maberly, 1854.
- BOYER, R. S.; MOORE, J.S. **A Computational Logic**. Nova York: Academic Press, 1979.
- BRACHMAN, R. J. I lied about the trees. **AI Magazine**, v. 6, n. 3, 1985.
- BRACHMAN, R. J.; LEVESQUE, H. J. **Readings in Knowledge Representation**. Los Altos, CA: Morgan Kaufmann, 1985.
- BRACHMAN, R. J.; LEVESQUE, H. J.; REITER, R., (Eds.). **Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning**. Los Altos, CA: Morgan Kaufmann, 1990.
- BROOKS, R. A. A robust layered control system for a mobile robot. **IEEE Journal of Robotics and Automation**, v. 4, p. 14-23, 1986.  
\_\_\_\_\_. A hardware retargetable distributed layered architecture for mobile robot control. **Proceedings IEEE Robotics and Automation**, p. 106-110. Raliegh, NC, 1987.  
\_\_\_\_\_. A robot that walks: Emergent behaviors from a carefully evolved network. **Neural Computation**, v. 1, n. 2, p. 253-262, 1989.  
\_\_\_\_\_. Intelligence without representation. **Proceedings of IJCAI-91**, p. 569-595. San Mateo, CA: Morgan Kaufmann, 1991a.  
\_\_\_\_\_. Challenges for complete creature architectures. In: MEYER, J. A.; WILSON, S. W. (Eds.). **From animals to animats. Proceedings of the First International Conference on Simulation of Adaptive Behavior**. Cambridge, MA: MIT Press/Bradford Books, 1991.  
\_\_\_\_\_. The cog project, In: MATSUI, T. (Ed.). **Journal of the Robotics Society of Japan, Special Issue (Mini) on Humanoid**, v. 15, n. 7, 1997.
- BROOKS, R.; STEIN, L. Building brains for bodies. **Autonomous Robots**, v. 1, p. 7-25, 1994.
- BROWN, J. S.; BURTON, R. R. Diagnostic models for procedural bugs in basic mathematical skills. **Cognitive Science**, v. 2, 1978.
- BROWN, J. S.; VANLEHN, K. Repair theory: A generative theory of bugs in procedural skills. **Cognitive Science**, v. 4, p. 379-426, 1980.
- BROWN, J. S.; BURTON, R. R.; DE KLEER, J. Pedagogical, natural language and knowledge engineering techniques in SOPHIE. In: SLEEMAN, D.; BROWN, J. S. **Intelligent Tutoring Systems**. Nova York: Academic Press, 1982.
- BROWNSTON, L. et al. **Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming**. Reading, MA: Addison-Wesley, 1985.

- BUCHANAN, B. G.; MITCHELL, T. M. Model-directed learning of production rules. In: WATERMAN, D.; HAYES-ROTH, F. **Pattern Directed Inference Systems**. Nova York: Academic Press, 1978.
- BUCHANAN, B. G.; SHORTLIFFE, E. H. (Eds.). **Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project**. Reading, MA: Addison-Wesley, 1984.
- BUNDY, A. **Computer Modelling of Mathematical Reasoning**. Nova York: Academic Press, 1983.
- \_\_\_\_\_. The use of explicit plans to guide inductive proofs. In: LUSK, R.; OVERBEEK, R. (Eds.). **Proceedings of CADE**, v. 9, p. 111-120, Nova York: Springer Verlag, 1988.
- BUNDY, A. et al. Solving mechanics problems using meta-level inference. **Proceedings of IJCAI-1979**, p. 1017-1027, 1979.
- BUNDY, A.; WELHAM, R. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. **Artificial Intelligence**, v. 16, p. 189-212, 1981.
- BURGES, C. J. C. A tutorial on support vector machines for pattern recognition. **Data Mining and Knowledge Discovery**, v. 2, n. 2, p. 161-167, 1998.
- BURHANS, D. T.; SHAPIRO, S. C. Defining answer classes using resolution refutation. **Journal of Applied Logic**, v. 5, n. 1, p. 70-91, 2007.
- BURKS, A. W. **Theory of Self Reproducing Automata**. University of Illinois Press, 1966.
- \_\_\_\_\_. **Essays on Cellular Automata**. University of Illinois Press, 1970.
- \_\_\_\_\_. Von Neumann's self-reproducing automata. In: ASPRAY, W.; BURKS, A. W. (Ed.). **Papers of John Von Neumann on Computing and Computer Theory**. Cambridge, MA: MIT Press, 1987.
- BURMEISTER, B.; HADDADI, A.; MATYLIS, G. Applications of multi-agent systems in traffic and transportation. **IEEE Transactions in Software Engineering**, v. 144, n. 1, p. 51-60, 1997.
- BURSTALL, R. M.; DARLINGTON, J. A. A transformational system for developing recursive programs. **JACM**, v. 24, jan. 1977.
- BUSUOIC, M.; GRIFFITHS, D. Cooperating intelligent agents for service management in communications networks. **Proceedings of 1993 Workshop on Cooperating Knowledge Based Systems**, p. 213-226. University of Keele, UK, 1994.
- BUTLER, M. (Ed.). **Frankenstein, or The Modern Prometheus**: the 1818 text by Mary Shelly. Nova York: Oxford University Press, 1998.
- CAHILL, L.; MCGAUGH, J. L. Modulation of memory storage. In: Squire, L. R.; Kosslyn, S. M. (Eds.). **Findings and Current Opinion in Cognitive Neuroscience**. Cambridge: MIT Press, 1998.
- CARBONELL, J. G. Learning by analogy: Formulating and generalizing plans from past experience. MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning: An Artificial Intelligence Approach**. v. 1. Palo Alto, CA: Tioga, 1983.
- \_\_\_\_\_. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M., (Eds.). **Machine Learning: An Artificial Intelligence Approach**. v. 2. Los Altos, CA: Morgan Kaufmann, 1986.
- CARBONELL, J.G.; MICHALSKI, R. S.; MITCHELL, T. M. An overview of machine learning. MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning: An Artificial Intelligence Approach**. v. 1. Palo Alto, CA: Tioga, 1983.
- CARDIE, C. Empirical methods in information extraction. **AI Magazine**, p. 65-79, inverno, 1997.
- CARDIE, C.; MOONEY, R. J. Machine learning and natural language. **Machine Learning**, v. 34, p. 5-9, 1999.
- CARLSON, N. R. **Physiology of Behavior**. 5. ed. Needham Heights MA: Allyn Bacon, 1994.
- CARNAP, R. On the application of inductive logic. **Philosophy and Phenomenological Research**, v. 8, p. 133-148, 1948.
- CECCATO, S. **Linguistic Analysis and Programming for Mechanical Translation**. Nova York: Gordon & Breach, 1961.
- CHANG, C. L.; LEE, R. C. T. **Symbolic Logic and Mechanical Theorem Proving**. Nova York: Academic Press, 1973.
- CHARNIAK, E. Toward a model of children's story comprehension. **Report No. TR-266**, AI Laboratory, MIT, 1972.
- \_\_\_\_\_. Bayesian networks without tears. **AAAI Magazine**, v. 12, n. 4, p. 50-63, 1991.
- \_\_\_\_\_. **Statistical Language Learning**. Cambridge, MA: MIT Press, 1993.
- CHARNIAK, E. et al. Equations for part-of-speech tagging. **Proceedings of the Eleventh National Conference on Artificial Intelligence**. Menlo Park, CA: AAAI/MIT Press, 1993.
- CHARNIAK, E.; McDERMOTT, D. **Introduction to Artificial Intelligence**. Reading, MA: Addison-Wesley, 1985.
- CHARNIAK, E. et al. **Artificial Intelligence Programming**. 2. ed. Hillsdale, NJ: Erlbaum, 1987.
- CHARNIAK, E.; SHIMONY, S. Probabilistic semantics for cost based abduction. **Proceedings of the Eighth National Conference on Artificial Intelligence**, p. 106-111. Menlo Park, CA: AAAI Press/ MIT Press, 1990.
- CHARNIAK, E.; WILKS, Y. **Computational Semantics**. Amsterdam: North-Holland, 1976.
- CHEN, L.; SYCARA, K. Webmate: A personal agent for browsing and searching. **Proceedings of Second International Conference on Autonomous Agents (Agents 98)**, 1998.
- CHICKERING, D. M.; GEIGER, D.; HECKERMAN, D. Learning Bayesian networks is NP-hard. **Microsoft Research Technical Report**, MSR-TR-94-17, 1994.

- CHOMSKY, N. **Aspects of the Theory of Syntax**. Cambridge, MA: MIT Press, 1965.
- CHORAFAS, D. N. **Knowledge Engineering**. Nova York: Van Nostrand Reinhold, 1990.
- CHUNG, K. T.; WU, C. H. Dynamic scheduling with intelligent agents. **Metra Application Note 105**. Palo Alto: Metra, 1997.
- CHURCH, A. The calculi of lambda-conversion. **Annals of Mathematical Studies**, v. 6. Princeton: Princeton University Press, 1941.
- CHURCHLAND, P. S., **Neurophilosophy**: Toward a Unified Science of the Mind/Brain. Cambridge, MA: MIT Press, 1986.
- CIRAVEGNA, F.; WILKS, Y. Designing adaptive information extraction for the semantic web in Amilcare. In: SHADBOLT; O'HARA (Eds.). **Advanced Knowledge Technologies: Selected Papers - 2004**, p. 251-266, UK: AKT Press, 2004.
- CLANCY, W. J. The advantages of abstract control knowledge in expert system design. **AAAI-3**, 1983.
- \_\_\_\_\_. Heuristic Classification. **Artificial Intelligence**, v. 27, p. 289-350, 1985.
- CLANCY, W. J.; SHORTLIFFE, E. H. (Eds.). **Readings in Medical Artificial Intelligence**: the First Decade. Reading, MA: Addison Wesley, 1984b.
- CLARK, A. **Being There**: Putting Brain, Body, and World Together Again. Cambridge, MA: MIT Press, 1997.
- CODD, E. F. **Cellular Automata**. Nova York: Academic Press, 1968.
- \_\_\_\_\_. Private communication to J. R. Koza. In: KOZA, J. R. **Genetic Programming**: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: MIT Press, 1992.
- COHEN, P.R.; FEIGENBAUM, E.A. **The Handbook of Artificial Intelligence**. v. 3. Los Altos, CA: William Kaufmann, 1982.
- COLE, R. A. (Ed.). **Survey of the State of the Art in Human Language Technology**. Nova York: Cambridge University Press, 1997.
- COLE, P.; MORGAN, J. L. (Ed.). **Studies in Syntax**. v. 3. Nova York: Academic Press, 1975.
- COLLINS A.; QUILLIAN, M.R. Retrieval time from semantic memory. **Journal of Verbal Learning & Verbal Behavior**, v. 8, p. 240-247, 1969.
- COLMERAUER, A. **Les Grammaires de Metamorphose**, Groupe Intelligence Artificielle, Université Aix-Marseille II, 1975.
- COLMERAUER, A. et al. **Un Systeme de Communication Homme-machine en Français**. Research Report, Groupe Intelligence Artificielle, Université Aix-Marseille II, França, 1973.
- COOMBS, M. J. (Ed.). **Developments in Expert Systems**. Nova York: Academic Press, 1984.
- CORERA, J. M.; LARESGOITI, I.; JENNINGS, N. R. Using arcon, part 2: Electricity transportation management. **IEEE Expert**, v. 11, n. 6, p. 71-79, 1996.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. J. **Introduction to Algorithms**. Cambridge, MA: MIT Press, 1990.
- COSMIDES, L.; TOOBY, J. Cognitive adaptations for social exchange. In: BARKOW, J. H.; COSMIDES, L.; TOOBY, J. **The Adapted Mind**. Nova York: Oxford Univ. Press, 1992.
- \_\_\_\_\_. Origins of domain specificity: the evolution of functional organization. In: HIRSCHFELD, L. A.; GELMAN, S. A., (Ed.) **Mapping the Mind**: Domain Specificity in Cognition and Culture. Cambridge: Cambridge University Press, 1994.
- COWELL, R. G. et al. **Probabilistic Networks and Expert Systems**. Nova York: Springer Verlag, 1999.
- COTTON, S.; BUNDY, A.; WALSH, T. Automatic invention of integer sequences. **Proceedings of the AAAI-2000**. Cambridge: MIT Press, 2000.
- CRICK, F. H.; ASANUMA, C. Certain aspects of the anatomy and physiology of the cerebral cortex. In: McCLELLAND, J. L.; RUMELHART, D. E.; THE PDP Research Group. **Parallel Distributed Processing**. Cambridge, MA: MIT Press, 1986.
- CRISTIANINI, N.; SHAWE-TAYLOR, J. **An Introduction to Support Vector Machines and other Kernel-Based Learning Methods**. Cambridge UK: Cambridge University Press, 2000.
- CRUTCHFIELD, J. P.; MITCHELL, M. The Evolution of Emergent Computation. **Working Paper 94-03-012**. Santa Fe Institute, 1995.
- CUSSENS, J. Parameter estimation in stochastic logic programs. **Machine Learning**, v. 44, p. 245-271, 2001.
- CUTOSKY, M. R. et al. PACT: An experiment in integrating concurrent engineering systems. **IEEE Transactions on Computers**, v. 26, n. 1, p. 28-37, 1993.
- DAHL, V. **Un Système Dédutif d'Interrogation de Banques de Données en Espagnol**, PhD thesis, Université Aix-Marseille, 1977.
- DAHL, V.; McCORD, M. C. Treating Coordination in Logic Grammars. **American Journal of Computational Linguistics**, v. 9, p. 69-91, 1983.
- DALLIER, J. H. **Logic for Computer Science: Foundations of Automatic Theorem Proving**. Nova York: Harper & Row, 1986.
- DARR, T. P.; BIRMINGHAM, W. P. An attribute-space representation and algorithm for concurrent engineering. **AIEDAM**, v. 10, n. 1, p. 21-35, 1996.
- DAVIS, E. **Representations of Commonsense Knowledge**. Los Altos, CA: Morgan Kaufmann, 1990.
- DAVIS, K. H.; BIDDULPH, R.; BALASHEK, S. Automatic recognition of spoken digits. **Journal of the Acoustical Society of America**, v. 24, n. 6, p. 637-642, 1952.
- DAVIS, L. Applying Adaptive Algorithms to Epistatic Domains. **Proceedings of the International Joint Conference on Artificial Intelligence**, p. 162-164, 1985.

- DAVIS, M.; MATIJASEVIC, Y.; ROBINSON, J. Hilbert's tenth problem: Diophantine equations; Positive aspects of a negative solution. *Proceedings of Symposia in Pure Mathematics*, v. 28, p. 323-378, 1976.
- DAVIS, R.; LENAT, D.B. **Knowledge-based Systems in Artificial Intelligence**. Nova York: McGraw-Hill, 1982.
- DAVIS, R.; HAMSCHER, W. Model-based reasoning: Trouble shooting. In: SHROBE, H. E. (Ed.). **Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence**. San Francisco: Morgan Kaufmann, 1988. Reprinted in HAMSCHER et al. (Eds.). **Readings in Model Based Diagnosis**. San Francisco: Morgan Kaufmann, 1992.
- DAVIS, R. et al. Diagnosis Based on Description of Structure and Function. *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, California: AAAI Press, 1982.
- DECHTER, R. Bucket elimination: A unifying framework for probabilistic inference. *Proceedings of Conference on Uncertainty in AI (UAI-96)*, 1996.
- DE DOMBAL, F.T.; STANILAND, J.R.; CLAMP, S.E. Human and computer-aided diagnosis of abdominal pain: Further report with emphasis on performance of clinicians. *British Medical Journal*, v. 1, p. 376-380, 1974.
- DeGROOT, M.H. **Probability and Statistics**, 2. ed. Reading Massachusetts: Addison-Wesley, 1989.
- DEJONG, G.; MOONEY, R. Explanation-based learning: An alternative view. *Machine Learning*, v. 1, n. 2, p. 145-176, 1986.
- DE KLEER, J. Qualitative and quantitative knowledge of classical mechanics. *Technical Report AI-TR-352*, AI Laboratory, MIT, 1975.
- \_\_\_\_\_. **Local methods for localizing faults in electronic circuits** (MIT AI Memo 394). Cambridge, MA: MIT, 1976.
- \_\_\_\_\_. Choices without backtracking. *Proceedings of the Fourth National Conference on Artificial Intelligence*, Austin, TX, p. 79-84. Menlo Park, CA: AAAI Press, 1984.
- \_\_\_\_\_. An assumption based truth maintenance system, *Artificial Intelligence*, v. 28, 1986.
- DE KLEER, J.; WILLIAMS, B. C. Diagnosing multiple faults. *Artificial Intelligence*, v. 32, n. 1, p. 92-130, 1987.
- \_\_\_\_\_. Diagnosis with behavioral modes. *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 1324-1330. Cambridge MA: MIT Press, 1989.
- DEMPSTER, A. P. A generalization of Bayesian inference. *Journal of the Royal Statistical Society*, v. 30 (Series B), p. 1-38, 1968.
- DEMPSTER, A. P.; LAIRD, N. M.; RUBIN, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B*, v. 39, p. 1-38, 1977.
- DENNITT, D. C. **Brainstorms: Philosophical Essays on Mind and Psychology**. Montgomery, AL: Bradford, 1978.
- \_\_\_\_\_. **Elbow Room: The Varieties of Free Will Worth Wanting**. Londres: Cambridge University Press, 1984.
- \_\_\_\_\_. **The Intentional Stance**. Cambridge MA: MIT Press, 1987.
- \_\_\_\_\_. **Consciousness Explained**. Boston: Little, Brown, 1991.
- \_\_\_\_\_. **Darwin's Dangerous Idea: Evolution and the Meanings of Life**. Nova York: Simon & Schuster, 1995.
- \_\_\_\_\_. **Sweet Dreams: Philosophical Obstacles to a Science of Consciousness**. Cambridge: MIT Press, 2006.
- DESCARTES, R. **Six Metaphysical Meditations, Wherein it is Proved That there is a God and that Man's Mind is really Distinct from his Body**. W. Moltneux, translator. Londres: Printed for B. Tooke, 1680.
- DIETTERICH, T. G. Learning at the knowledge level. *Machine Learning*, v. 1, n. 3, p. 287-316, 1986.
- DIETTERICH, T. G.; MICHALSKI, R. S. Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods. *Proceedings IJCAI*, v. 6, 1981.
- \_\_\_\_\_. Learning to predict sequences. In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning: An Artificial Intelligence Approach**. v. 2. Los Altos, CA: Morgan Kaufmann, 1986.
- DOMINGOS, P.; PAZZANI, M. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning*, v. 29, p. 103-130, 1997.
- DOORENBOS, R.; ETZIONI, O.; WELD, D. A scalable comparison shopping agent for the world wide web. *Proceedings of the First International Conference on Autonomous Agents (Agents 97)*, p. 39-48, 1997.
- DOYA, K. et al. (Eds.). **Bayesian Brain**. Cambridge: MIT Press, 2007.
- DOYLE, J. A truth maintenance system. *Artificial Intelligence*, v. 12, 1979.
- \_\_\_\_\_. Some theories of reasoned assumptions: An essay in rational psychology. *Tech. Report CS-83-125*. Pittsburgh: Carnegie Mellon University, 1983.
- DRESSLER, O. An extended basic ATMS. *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, p. 143-163. Editado por Reinfrank, de Kleer, Ginsberg e Sandewall. Notas de aula em Artificial Intelligence 346. Heidelberg: Springer-Verlag, 1988.
- DREYFUS, H. L.; DREYFUS, S. E. **Mind Over Machine**. Nova York: Macmillan/The Free Press, 1985.
- DRUZDEL, M. J. Qualitative Verbal Explanations in Bayesian Belief Networks. *AISB Quar*, v. 1, n. 96, 1996.
- DRUZDEL, M. J.; HENRION, M. Efficient reasoning in qualitative probabilistic networks. *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, p. 548-553, 1993.
- DUDA, R. O.; HART, P. E. **Pattern Classification and Scene Analysis**. Nova York: John Wiley, 1973.

- DUDA, R. O.; GASCHNIG, J.; HART, P. E. Model design in the PROSPECTOR consultant system for mineral exploration. In: MICHIE, D. (Ed.). **Expert Systems in the Micro-electronic Age**. Edinburgh: Edinburgh Univ. Press, 1979.
- DUDA, R. O. et al. **A computer-based consultant for mineral exploration**. SRI International, 1979.
- DURFEE, E. H.; LESSER, V. Negotiating task decomposition and allocation using partial global planning. **Distributed Artificial Intelligence: Vol II**, GASSER, L; HUHNS, M. (Eds.). San Francisco: Morgan Kaufmann, p. 229-244, 1989.
- DURKIN, J. **Expert Systems: Design and Development**. Nova York: Macmillan, 1994.
- ECO, U. **A Theory of Semiotics**. Bloomington, Indiana: University of Indiana Press, 1976.
- EDELMAN, G. M. **Bright Air, Brilliant Fire: On the Matter of the Mind**. Nova York: Basic Books, 1992.
- EDWARDS, D. **Introduction to Graphical Modeling**. Nova York: Springer Verlag, 2000.
- ELMAN, J.L. et al. **Rethinking Innateness: A Connectionist Perspective on Development**. Cambridge, MA: MIT Press, 1996.
- ENGELMORE, R.; MORGAN, T. (Ed.). **Blackboard Systems**, Londres: Addison-Wesley, 1988.
- ERMAN, L.D. et al. The HEARSAY II speech understanding system: Integrating knowledge to resolve uncertainty. **Computing Surveys**, v. 12, n. 2, p. 213-253, 1980.
- ERMAN, L. D.; LONDON, P. E.; FICKAS, S. F. The design and an example use of HEARSAY III. **Proceedings IJCAI 7**, 1981.
- ERNST, G. W.; NEWELL, A. **GPS: A Case Study in Generality and Problem Solving**. Nova York: Academic Press, 1969.
- EULER, L. The seven bridges of Konigsberg, 1735. In: Newman, J. R. (Ed.). **The World of Mathematics**. Nova York: Simon and Schuster, 1956.
- EVANS, T. G. A heuristic program to solve geometric analogy problems. In: MINSKY, M. (Ed.). **Semantic Information Processing**. Cambridge, MA: MIT Press, 1968.
- FALKENHAINER, B. 1990. Explanation and theory formation. In: Shrager, J.; Langley, P. (Eds.). **Computational Models of Discovery and Theory Formation**. San Mateo, CA: Morgan Kaufman, 1990.
- FALKENHAINER, B.; FORBUS, K. D.; GENTNER, D. The structure mapping engine: Algorithm and examples. **Artificial Intelligence**, v. 41, n. 1, p. 1-64, 1989.
- FASS, D.; WILKS, Y. Preference semantics with ill-formedness and metaphor. **American Journal of Computational Linguistics IX**, p. 178-187, 1983.
- FATIMA, S. S.; WOOLDRIDGE, M.; JENNINGS, N. R. Bargaining with incomplete information. **Annals of Mathematics and Artificial Intelligence**, v. 44, n. 3, p. 207-232, 2005.
- \_\_\_\_\_. Multi-issue negotiation with deadlines. **Journal of Artificial Intelligence Research (JAIR)**, v. 27, p. 381-417, 2006.
- FEIGENBAUM, E.A.; FELDMAN, J. (Eds.). **Computers and Thought**. Nova York: McGraw-Hill, 1963.
- FEIGENBAUM, E. A.; MCCORDUCK, P. **The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World**. Reading, MA: Addison-Wesley, 1983.
- FIKES, R. E.; NILSSON, N.J. STRIPS: A new approach to the application of theorem proving to artificial intelligence. **Artificial Intelligence**, v. 1, n. 2, 1971.
- FIKES, R.E.; HART, P. E.; NILSSON, N. J. Learning and executing generalized robot plans. **Artificial Intelligence**, v. 3, n. 4, p. 251-88, 1972.
- FILLMORE, C. J. The case for case. In: BACH, E.; HARMS, R. (Eds.). **Universals of Linguistic Theory**. Nova York: Holt, Rinehart e Winston, 1968.
- FISCHER, K.; MULLER, J.P.; PISCHEL, M. Cooperative transportation scheduling: An application domain for DAI. **Applied Artificial Intelligence**, v. 10, n. , p. 1-34, 1996.
- FISHER, D. H. Knowledge acquisition via incremental conceptual clustering. **Machine Learning**, v. 2, p. 139-172, 1987.
- FISHER, D. H.; PAZZANI, M. J.; LANGLEY, P. **Concept Formation: Knowledge and Experience in Unsupervised Learning**. San Mateo, CA: Morgan Kaufmann Publishing, 1991.
- FISHER, R. A. On the mathematical foundation of theoretical statistics. **Philosophical Transactions of the Royal Society of London**, Series A, v. 222, p. 309-368, 1922.
- FODOR, J. A. **The Modularity of Mind**. Cambridge, MA: MIT Press, 1983.
- FORBUS, K. D.; DE KLEER, J. Focusing the ATMS. **Proceedings of the Seventh National Conference on Artificial Intelligence**, p. 193-198. Menlo Park: CA, 1988.
- \_\_\_\_\_. **Building Problem Solvers**. Cambridge, MA: MIT Press, 1993.
- FORD, K. M.; HAYES, P. J. Turing Test Considered Harmful. **Proceedings of International Joint Conference on Artificial Intelligence**, Montreal, 1995.
- FORD, K. M., GLYMOEUR, C.; HAYES, P. J. **Android Epistemology**. Cambridge: MIT Press, 1995.
- FORGY, C. L. 1982. RETE: a fast algorithm for the many pattern/many object pattern match problem. **Artificial Intelligence**, v. 19, n. 1, p. 17-37.
- FORNEY JR, G. D. The Viterbi Algorithm. **Proceedings of the IEEE**, p. 268-278, mar. 1973.

- FORREST, S. Emergent Computation: Self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. *Physica D*, v. 42, p. 1-11, 1990.
- FORREST, S.; MITCHELL, M. What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation. *Machine Learning*, v. 13, p. 285-319, 1993a.
- \_\_\_\_\_. Relative building block fitness and the building block hypothesis. In: Whitley, L.D., (Ed.). **Foundations of Genetic Algorithms**, v. 2. Los Altos, CA: Morgan Kaufmann, 1993.
- FRANCIS, W. N. A taged corpus – problems and prospects. In: GREENBAUM, S.; LEECH, G.; SVARTVIK, J. (Eds.). **Studies in English Linguistics for Randolph Quirk**, p. 192-209. Nova York: Longman, 1979.
- FRANKLIN, S. **Artificial Minds**. Cambridge, MA: MIT Press, 1995.
- FREEMAN, J. A.; SKAPURA, D. M. **Neural Networks: Algorithms, Applications and Programming Techniques**. Nova York: Addison-Wesley, 1991.
- FREGE, G. **Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens**. Halle: L. Niebert, 1879.
- \_\_\_\_\_. **Die Grundlagen der Arithmetic**. Breslau: W. Koeber, 1884.
- FREY, B. J. **Graphical Models for Machine Learning and Digital Communication**. Cambridge: MIT Press, 1998.
- FRIEDMAN, N. The Bayesian structural EM algorithm, *Proceedings of the Fourteenth Conference of Uncertainty in Artificial Intelligence*, p. 252-262. San Francisco: Morgan Kaufmann, 1998.
- FRIEDMAN, N. et al. Learning probabilistic relational models. *Proc. Int. Joint Conf. on Artificial Intelligence*, p. 1300-1307, 2001.
- GADAMER, H. G. *Philosophical Hermeneutics*. Traduzido por D.E. Linge. Berkeley: University of California Press, 1976.
- GALILEO GALILEI. *Discorsi E Dimonstrazioni Matematiche Intorno a Due Nuove Scienze*. Leiden, 1638.
- GALLIER, J. H. **Logic for Computer Science: Foundations of Automatic Theorem Proving**. Nova York: Harper and Row, 1986.
- GANZINGER, H.; MEYER, C.; VEANES, M. The two-variable guarded fragment with transitive relations. *Proceedings of the Annual Symposium on Logic in Computer Science*, p. 24-34, 1999.
- GARDNER, M. Mathematical Games. *Scientific American*, out. 1970.
- \_\_\_\_\_. Mathematical Games. *Scientific American*, fev. 1971.
- GAREY, M.; JOHNSON, D. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. San Francisco: Freeman, 1979.
- GAZDAR, G.; MELLISH, C. **Natural Language Processing in PROLOG: An Introduction to Computational Linguistics**. Reading, MA: Addison-Wesley, 1989.
- GAZZANIGA, M. S. (Ed.). **The New Cognitive Neurosciences**, 2. ed. Cambridge: MIT Press, 2000.
- GENSERETH, M.; NILSSON, N. **Logical Foundations of Artificial Intelligence**. Los Altos, CA: Morgan Kaufmann, 1987.
- GENSERETH, M. R. The role of plans in intelligent teaching systems. In: Sleeman, D. e Brown, J. S. **Intelligent Tutoring Systems**. Nova York: Academic Press, 1982.
- GENNARI, J. H.; LANGLEY, P.; FISHER, D. Models of incremental concept formation. *Artificial Intelligence*, v. 40, n. 1-3, p. 11-62, 1989.
- GENTNER, D. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, v. 7, p. 155-170, 1983.
- GETTOOR, L. et al. Learning probabilistic relational models. *Journal of Relational Data Mining*, p. 307-335, 2001.
- GHAHRAMANI, Z.; JORDAN, M. I. Factorial hidden Markov models. *Machine Learning*, v. 29, p. 245-274, 1997.
- GIARRATANO, J.; RILEY, G. **Expert Systems: Principles and Programming**. PWS-Kent Publishing Co, 1989.
- \_\_\_\_\_. **Expert Systems: Principles and Programming**. Boston MA: Thomson - Course Technology, 2005.
- GILBERT, C. D. Plasticity in visula perception and physiology. In: SQUIRE, L.R.; KOSSLYN, S.M. (Eds.). **Findings and Current Opinion in Cognitive Neuroscience**. Cambridge: MIT Press, 1998.
- \_\_\_\_\_. Horizontal integration and cortical dynamics. *Neuron*, v. 9, p. 1-13, 1992.
- GINSBURG, M. (Ed.). *Readings in Non-monotonic Reasoning*. San Francisco: Morgan Kaufmann, 1987.
- GLASGOW, B. et al. MITA: An information extraction approach to analysis of free-form text in life insurance applications. *Proceedings of the Ninth Conference of Innovative Applications of Artificial Intelligence*, p. 992-999, 1997.
- GLUCK, M.; CORTER, J. Information, uncertainty and the utility of categories. *Seventh Annual Conference of the Cognitive Science Society in Irvine, Calif.*, 1985.
- GLYMOEUR, C., FORD, K. e HAYES, P. **Android Epistemology**. Menlo Park, CA: AAAI Press, 1995.
- GLYMOEUR, C.; COOPER, G. F. (Eds.). **Computation, Causation and Discovery**. Cambridge: MIT Press, 1999.
- GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. Reading, MA: Addison-Wesley, 1989.
- GOODWIN, J. An Improved Algorithm for Non-Monotonic Dependency Net Update. *Technical Report LITH-MAT-R-82-23*. Department of Computer Science and Information Science, Linköping University, Linköping, Sweden, 1982.
- GOULD, S. J. **Ontogeny and Phylogeny**. Cambridge MA: Belknap Press, 1977.
- \_\_\_\_\_. **Full House: The Spread of Excellence from Plato to Darwin**. Nova York: Harmony Books, 1996.
- GRAHAM, P. **On LISP: Advanced Techniques for Common LISP**. Englewood Cliffs, NJ: Prentice Hall, 1993.

- \_\_\_\_\_. **ANSI Common Lisp**. Englewood Cliffs, NJ: Prentice Hall, 1995.
- GREEN, C. Applications of theorem proving to problem solving. In: WALKER, D. E.; NORTON, L. M. (Eds.). **Proceedings of the International Joint Conference on Artificial Intelligence**, p. 219-239, 1969a.
- \_\_\_\_\_. Theorem-proving by resolution as a basis for question-answering systems. In: MICHIE, D.; MELTZER, B. (Eds.). **Machine Intelligence**, v. 4, p. 183-205. Edinburgh: University Press, 1969b.
- GREENBERG, S. Speaking in shorthand - A syllable-centric perspective for understanding pronunciation variation. **Speech Communication**, v. 29, p. 159-176, 1999.
- GREENE, B. **The Elegant Universe: Superstrings, Hidden Dimensions, and the Quest for the Ultimate Theory**. Nova York: Norton, 1999.
- GRICE, H. P. Logic and conversation. In: COLE, P.; MORGAN, J. L. (Ed.). **Studies in Syntax**. v. 3. Nova York: Academic Press, 1975.
- GROSSBERG, S. Adaptive pattern classification and universal recoding. I. Parallel development and coding of neural feature detectors. **Biological Cybernetics**, v. 23, p. 121-134, 1976.
- \_\_\_\_\_. **Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition and Motor Control**. Boston: Reidel Press, 1982.
- GROSSBERG, S. (Ed.). **Neural Networks and Natural Intelligence**. Cambridge, MA: MIT Press, 1988.
- GROSZ, B. **The representation and use of focus in dialogue understanding**. PhD thesis, University of California, Berkeley, 1977.
- GROSZ, B.; SIDNER, C. L. Plans for discourse. In: Cohen, P.R.; Pollack, M. E. (Eds.). **Intentions in Communications**, p. 417-444. Cambridge, MA: MIT Press, 1990.
- HADDAWY, P. Generating Bayesian networks from probability logic knowledge bases. **Proceedings of the Uncertainty in Artificial Intelligence Conference**, p. 262-269. Menlo Park CA: AAAI Press, 1994.
- HALL, R. P. **Computational approaches to analogical reasoning**: A comparative analysis, v. 39, n. 1, p. 39-120, 1989.
- HAMMOND, K. (Ed.). **Case Based Reasoning Workshop**. San Mateo, CA: Morgan Kaufmann, 1989.
- HAMSCHER, W.; CONSOLE, L.; DE KLEER, J. **Readings in Model-based Diagnosis**. San Mateo: Morgan Kaufmann, 1992.
- HANSON, J. E.; CRUTCHFIELD, J. P. The Attractor-basin portrait of a cellular automaton. **Journal of Statistical Physics**, v. 66, n. 5/6, p. 1415-1462, 1992.
- HARMON, P.; KING, D. **Expert Systems**: Artificial Intelligence in Business. Nova York: Wiley, 1985.
- HARMON, P.; MAUS, R.; MORRISSEY, W. **Expert Systems**: Tools and Applications. Nova York: Wiley, 1988.
- HARRIS, M. D. **Introduction to Natural Language Processing**. Englewood Cliffs NJ: Prentice Hall, 1985.
- HARRISON, C. D.; LUGER, G. F. Data mining using web spiders, (submitted). **UNM Computer Science Technical Report TR-CS-2001-34**, 2002.
- HAUGELAND, J. (Ed.). **Mind Design**: Philosophy, Psychology, Artificial Intelligence. Cambridge, MA: MIT Press, 1981.
- \_\_\_\_\_. **Artificial Intelligence**: the Very Idea. Cambridge/Bradford, MA: MIT Press, 1985.
- \_\_\_\_\_. **Mind Design**: Philosophy, Psychology, Artificial Intelligence, 2. ed. Cambridge, MA: MIT Press, 1997.
- HAUSSLER, D. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. **Artificial Intelligence**, v. 36, p. 177-222, 1988.
- HAYES, P. J. In defense of logic. **Proceedings IJCAI-77**, p. 559-564, Cambridge, MA, 1977.
- \_\_\_\_\_. The logic of frames. In: METZING, D. (Ed.). **Frame Conceptions and Text Understanding**, p. 46-61. Berlin: deGruyter 1979.
- HAYES-ROTH, B. Agents on stage: Advancing the state of the art in AI. **Proceedings of the Four-teenth IJCAI**, p. 967-971. Cambridge, MA: MIT Press, 1995.
- \_\_\_\_\_. Plans and Behavior in Intelligent Agents. **Technical Report, Knowledge Systems Laboratory (KSL-95-35)**, 1993.
- HAYES-ROTH, F.; WATERMAN, D.; LENAT, D. **Building Expert Systems**. Reading, MA: Addison-Wesley, 1984.
- HEBB, D. O. **The Organization of Behavior**. Nova York: Wiley, 1949.
- HECHT-NIELSEN, R. Neural analog processing. **Proc. SPIE**, v. 360, p. 180-189. Bellingham, WA, 1982.
- \_\_\_\_\_. Counterpropagation networks. **Applied Optics**, v. 26, p. 4979-4984, dez. 1987.
- \_\_\_\_\_. Theory of the backpropagation neural network. **Proceedings of the International Joint Conference on Neural Networks**, v. 1, p. 593-611. Nova York: IEEE Press, 1989.
- \_\_\_\_\_. **Neurocomputing**. Nova York: Addison-Wesley, 1990.
- HEIDEGGER, M. **Being and Time**. Traduzido por J. Masquarrie e E. Robinson. Nova York: Harper & Row, 1962.
- HEIDORN, G. E. Augmented phrase structure grammar. In: SCHANK, R. C.; NASH-WEBBER, B. L. (Ed.). **Theoretical Issues in Natural Language Processing**. Association for Computational Linguistics, 1975.
- HELMAN, D. H. (Ed.). **Analogical Reasoning**. Londres: Kluwer Academic, 1988.
- HELMAN, P.; VEROFF, R. **Intermediate Problem Solving and Data Structures**: Walls and Mirrors, Menlo Park, CA: Benjamin/Cummings, 1986.

- HEMPEL, C. G. **Aspects of Scientific Explanation**. Nova York: The Free Press, 1965.
- HIGHTOWER, R. **The Devore universal computer constructor**. Presentation at the Third Workshop on Artificial Life, Santa Fe, NM, 1992.
- HILLIS, D. W. **The Connection Machine**. Cambridge, MA: MIT Press, 1985.
- HINTON, G. E.; SEJNOWSKI, T. J. Learning and relearning in Boltzmann machines. In: McCLELLAND, J. L.; RUMELHART, D. E.; THE PDP Research Group. **Parallel Distributed Processing**, 2 v. Cambridge, MA: MIT Press, 1986.
- \_\_\_\_\_. Neural network architectures for AI. Tutorial, AAAI Conference, 1987.
- HIRSCHFELD, L. A.; GELMAN, S. A. (Ed.). **Mapping the Mind: Domain Specificity in Cognition and Culture**. Cambridge: Cambridge University Press, 1994.
- HOBES, T. **Leviathan**. Londres: Impresso para A. Crooke, 1651.
- HOBBS, J. R.; MOORE, R. C. **Formal Theories of the Commonsense World**. Norwood, NJ: Ablex, 1985.
- HODGES, A. **Alan Turing: The Enigma**. Nova York: Simon and Schuster, 1983.
- HOFSTADTER, D. **Fluid Concepts and Creative Analogies**. Nova York: Basic Books, 1995.
- HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. University of Michigan Press, 1975.
- \_\_\_\_\_. Studies of the spontaneous emergence of self-replicating systems using cellular automata and formal grammars. In: LINDENMAYER, A.; ROSENBERG, G. (Ed.). **Automata, Languages, Development**. Nova York: North-Holland, 1976.
- \_\_\_\_\_. Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning: An Artificial Intelligence Approach**, v. 2. Los Altos, CA: Morgan Kaufmann, 1986.
- \_\_\_\_\_. **Hidden Order: How Adaptation Builds Complexity**. Reading, MA: Addison-Wesley, 1995.
- HOLLAND, J. H. et al. **Induction: Processes of Inference, Learning and Discovery**. Cambridge, MA: MIT Press, 1986.
- HOLOWCZAK, R. D.; ADAM, N. R. Information extraction-based multiple-category document classification for the global legal information network. **Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence**, p. 1013-1018, 1997.
- HOLYOAK, K. J. The pragmatics of analogical transfer. **The Psychology of Learning and Motivation**, v. 19, p. 59-87, 1985.
- HOPCROFT, J. E.; ULLMAN, J. D. **Introduction to Automata Theory, Languages and Computation**. Reading, MA: Addison-Wesley, 1979.
- HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities. **Proceedings of the National Academy of Sciences**, v. 79, 1982.
- \_\_\_\_\_. Neural networks and physical systems with emergent collective computational abilities. **Proceedings of the National Academy of Sciences**, v. 79, p. 2554-2558, 1984.
- HOROWITZ, E.; SAHNI, S. **Fundamentals of Computer Algorithms**. Rockville, MD: Computer Science Press, 1978.
- HSU, F. **Behind Deep Blue: Building the Computer that Defeated the World Chess Champion**. Princeton, NJ: Princeton University Press, 2002.
- HUANG, C.; DARWICHE, A. Inference in belief networks: A procedural guide. **International Journal of Approximate Reasoning**, v. 15, n. 3, p. 225-263, 1996.
- HUGDAHL, K.; DAVIDSON, R. J. (Eds.). **The Asymmetrical Brain**. Cambridge: MIT Press, 2003.
- HUME, D. **An Inquiry Concerning Human Understanding**. Nova York: Bobbs-Merrill, 1748.
- HUSSERL, E. **The Crisis of European Sciences and Transcendental Phenomenology**. Traduzido por D. Carr. Evanston, IL: Northwestern University Press, 1970.
- \_\_\_\_\_. **Ideas: General Introduction to Pure Phenomenology**. Nova York: Collier, 1972.
- HUYGENS, C. Ratiociniis in ludo aleae. In: van Schoot, F. (Ed.). **Exercitionum Mathematicorum**. Amsterdam: Elsevier, 1657.
- IGNIZIO, J. P. **Introduction to Expert Systems: The Development and Implementation of Rule-Based Expert Systems**. Nova York: McGraw-Hill, 1991.
- IVRY, I. B. The representation of temporal information in perception and motor control. In: SQUIRE, L. R.; KOSSLYN, S. M. (Eds.). **Findings and Current Opinion in Cognitive Neuroscience**. Cambridge: MIT Press, 1998.
- JAAKKOLA, T.; JORDAN, M. I. Variational probabilistic inference and the QMR-DT database. **Journal of Artificial Intelligence Research (JAIR)**, v. 10, p. 291-322, 1998.
- JACKSON, P. **Introduction to Expert Systems**. Reading, MA: Addison-Wesley, 1986.
- JEANNEROD, M. **The Cognitive Neuroscience of Action**. Oxford: Blackwell, 1997.
- JENNINGS, N. R. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. **Artificial Intelligence**, v. 75, p. 195-240, 1995.
- JENNINGS, N. R.; WOOLDRIDGE, M. (Eds.). Applying agent technology. **Agent Technology: Foundations, Applications, and Markets**. Berlin: Springer-Verlag, 1998.

- JENNINGS, N. R.; SYCARA, K. P.; WOOLDRIDGE, M. A roadmap for agent research and development. **Journal of Autonomous Agents and MultiAgent Systems**, v. 1, n. 1, p. 7-36, 1998.
- JENSEN, F. V. **An Introduction to Bayesian Networks**. Nova York: Springer Verlag, 1996.
- \_\_\_\_\_. **Bayesian Networks and Decision Graphs**. Nova York: Springer Verlag, 2001.
- JOHNSON, L.; KERAVNOU, E. T. **Expert Systems Technology**: A Guide. Cambridge, MA: Abacus Press, 1985.
- JOHNSON, M. **The Body in the Mind**: The Bodily Basis of Meaning, Imagination and Reason. Chicago: University of Chicago Press, 1987.
- JOHNSON-LAIRD, P. N. **Mental Models**. Cambridge, MA: Harvard University Press, 1983.
- \_\_\_\_\_. **The Computer and the Mind**. Cambridge, MA: Harvard University Press, 1988.
- JORDAN, M. I. et al. An introduction to variational methods for graphical models. **Machine Learning**, v. 37, n. 2, p. 183-233, 1999.
- JORDAN, M. I. (Ed.). **Learning in Graphical Models**. Boston: Kluwer Academic, 1999.
- JOSEPHSON, J. R.; JOSEPHSON, S. G. (Ed.). **Abductive Inference**: Computation, Philosophy and Technology. Nova York: Cambridge University Press, 1994.
- JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**, 2. ed. Upper Saddle River, NJ: Prentice Hall, 2008.
- KANT, I. **Immanuel Kant's Critique of Pure Reason**, traduzido por SMITH, N.K. Nova York: St Martin's Press, 1781/1964.
- KARMILOFF-SMITH, A. **Beyond Modularity**: A Developmental Perspective on Cognitive Science. Cambridge, MA: MIT Press, 1992.
- KAUFMANN, M.; MANOLIOS, P.; MOORE, J. S. (Eds.). **Computer-Aided Reasoning**: ACL2 Case Studies. Boston: Kluwer Academic, 2000.
- KAUTZ, H.; SELMAN, B.; SHAH, M. The hidden web. **AI Magazine**, v. 18, n. 2, p. 27-35, 1997.
- KEDAR-CABELLI, S. T. Analogy - From a unified perspective. In: HELMAN, D. H. (Ed.). **Analogical Reasoning**. Londres: Kluwer Academic, 1988.
- KEDAR-CABELLI, S. T.; McCARTY, L. T. Explanation based generalization as resolution theorem proving. **Proceedings of the Fourth International Workshop on Machine Learning**, 1987.
- KERAVNOU, E. T.; JOHNSON, L. **Competent Expert Systems**: A Case Study in Fault Diagnosis. Londres: Kegan Paul, 1986.
- KERSTING, K.; DE RAEDT, L. Bayesian logic programs. **Proc. of the Work-in-Progress Track of the Tenth Int. Conf. on Inductive Logic Programming**, p. 138-155. Menlo Park CA: AAAI Press, 2000.
- KING, S. H. **Knowledge Systems Through PROLOG**. Oxford: Oxford University Press, 1991.
- KLAHR, D.; LANGLEY, P.; NECHES, R. (Ed.). **Production System Models of Learning and Development**. Cambridge, MA: MIT Press, 1987.
- KLAHR, D.; WATERMAN, D. A. **Expert Systems**: Techniques, Tools and Applications. Reading, MA: Addison-Wesley, 1986.
- KLEIN, W. B.; WESTERVELT, R. T.; LUGER, G. F. A general purpose intelligent control system for particle accelerators. **Journal of Intelligent & Fuzzy Systems**. Nova York: John Wiley, 1999.
- KLEIN, W. B. et al. Teleo-reactive control for accelerator beamline tuning. **Artificial Intelligence and Soft Computing**: Proceedings of the IASTED International Conference. Anaheim: IASTED/ACTA Press, 2000.
- KODRATOFF, Y.; MICHALSKI, R. S. (Ed.). **Machine Learning**: An Artificial Intelligence Approach. v. 3. Los Altos, CA: Morgan Kaufmann, 1990.
- KOHONEN, T. Correlation matrix memories. **IEEE Transactions Computers**, v. 4, p. 353-359, 1972.
- \_\_\_\_\_. **Self-Organization and Associative Memory**. Berlin: Springer-Verlag, 1984.
- KOLLER, D.; PFEFFER, A. Object-oriented Bayesian networks. **Proceeding of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence**. San Francisco: Morgan Kaufmann, 1997.
- \_\_\_\_\_. Probabalistic frame-based systems. **Proceeding of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence**. San Francisco: Morgan Kaufmann, 1998.
- KOLMOGOROV, A. N. **Foundations of the Theory of Probability**. Nova York: Chelsea, 1950.
- \_\_\_\_\_. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition (em russo). **Dokl. Akad Nauk USSR**, v. 114, p. 953-956, 1957.
- \_\_\_\_\_. Three approaches to the quantitative definition of information. **Problems in Information Transmission**, v. 1, n. 1, p. 1-7, 1965.
- KOLODNER, J. L. Extending problem solver capabilities through case based inference. **Proceedings of the Fourth International Workshop on Machine Learning**. CA: Morgan Kaufmann, 1987.
- \_\_\_\_\_. Retrieving events from a case memory: A parallel implementation. **Proceedings of the Case Based Reasoning Workshop**. Los Altos, CA: Morgan Kaufmann, 1988a.
- \_\_\_\_\_. **Case Based Reasoning Workshop**. San Mateo, CA: Morgan Kaufmann, 1988b.
- \_\_\_\_\_. Improving human decision making through case-based decision aiding. **AI Magazine**, v. 12, n. 2, p. 52-68, 1991.
- \_\_\_\_\_. **Case-based Reasoning**. San Mateo, CA: Morgan Kaufmann, 1993.

- KORF, R. E. Search. In: SHAPIRO, E. **Concurrent Prolog: Collected Papers**, 2 v. Cambridge, MA: MIT Press, 1987.
- \_\_\_\_\_. Artificial intelligence search algorithms. In: ATALLAH, M. J. (Ed.). **CRC Handbook of Algorithms and Theory of Computation**. Boca Raton, FL: CRC Press, 1998.
- \_\_\_\_\_. Sliding-tile puzzles and Rubic's Cube in AI research. **IEEE Intelligent Systems**, p. 8-12, nov. 1999.
- \_\_\_\_\_. Best-first frontier search with delayed duplicate detection. **Proceedings of the National Conference on Artificial Intelligence**, p. 1380-1385, Cambridge, MA: MIT Press, 2004.
- KOSKO, B. Bidirectional associative memories. **IEEE Transactions Systems, Man & Cybernetics**, v. 18, p. 49-60, 1988.
- KOSORESOW, A. P. A fast-first cut protocol for agent coordination. **Proceedings of the Eleventh National Conference on Artificial Intelligence**, p. 237-242, Cambridge, MA: MIT Press, 1993.
- KOTON, P. Reasoning about evidence in causal explanation. **Proceedings of AAAI-88**. Cambridge, MA: AAAI Press/MIT Press, 1988a.
- \_\_\_\_\_. Integrating case-based and causal reasoning. **Proceedings of the Tenth Annual Conference of the Cognitive Science Society**. Northdale, NJ: Erlbaum, 1988b.
- KOWALSKI, R. Algorithm = Logic + Control. **Communications of the ACM**, v. 22, p. 424-436, 1979a.
- \_\_\_\_\_. **Logic for Problem Solving**. Amsterdam: North-Holland, 1979b.
- KOZA, J. R. Genetic evolution and co-evolution of computer programs. In: LANGTON, C. G. et al. (Eds.). **Artificial Life II, SFI Studies in the Sciences of Complexity**, v. X. Redwood City, CA: Addison-Wesley 1991.
- \_\_\_\_\_. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Cambridge, MA: MIT Press, 1992.
- \_\_\_\_\_. **Genetic Programming II: Automatic Discovery of Reusable Programs**. Cambridge, MA: MIT Press, 1994.
- \_\_\_\_\_. **Genetic Programming IV: Routine Human-Competitive Machine Intelligence**. Nova York: Springer-Verlag, 2003.
- KRULWICH, B. The BargainFinder agent: Comparison price shopping on the internet. In: WILLIAMS, J. (Ed.). **Bots, and other Internet Beasties**, p. 257-263, Indianapolis: Macmillan, 1996.
- KSCHISCHANG, F. R.; FREY, B. J.; LOELIGAR, H. A. Factor graphs and the sum product algorithm. **IEEE Transactions on Information Theory**, v. 46, n. 2, 2000.
- KUCERA, H.; FRANCIS, W. N. **Computational Analysis of Present-Day American English**. Providence, RI: Brown University Press, 1967.
- KUHL, P. K. Innate predispositions and the effects of experience in speech perception: The native language magnet theory. In: Boysson-Bardies, B. et al. (Eds.). **Developmental Neurocognition: Speech and Face Processing in the First Year of Life**, p. 259-274. Netherlands: Kluwer Academic, 1993.
- \_\_\_\_\_. Learning and representation in speech and language. In: SQUIRE, L. R.; KOSSLYN, S. M. (Eds.). **Findings and Current Opinion in Cognitive Neuroscience**. Cambridge: MIT Press, 1998.
- KUHN, T. S. **The Structure of Scientific Revolutions**. Chicago: University of Chicago Press, 1962.
- LAIRD, J.; ROSENBLOOM, P.; NEWELL, A. Chunking in SOAR: The Anatomy of a General Learning Mechanism. **Machine Learning**, v. 1, n. 1, 1986a.
- \_\_\_\_\_. **Universal Subgoaling and Chunking: The automatic Generation and Learning of Goal Hierarchies**. Dordrecht: Kluwer, 1986b.
- LAKOFF, G. **Women, Fire and Dangerous Things**. Chicago: University of Chicago Press, 1987.
- LAKOFF, G.; JOHNSON, M. **Philosophy in the Flesh**. Nova York: Basic Books, 1999.
- LAKATOS, I. **Proofs and Refutations: The Logic of Mathematical Discovery**. Cambridge: The University Press, 1976.
- LANGLEY, P. **Elements of Machine Learning**. San Francisco: Morgan Kaufmann, 1995.
- LANGLEY, P.; BRADSHAW, G. L.; SIMON, H. A. Bacon 5: The discovery of conservation laws. **Proceedings of the Seventh International Joint Conference on Artificial Intelligence**, 1981.
- LANGLEY, P.; et al. The search for regularity: Four aspects of scientific discovery. In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M., (Eds.). **Machine Learning: An Artificial Intelligence Approach**. v. 2. Los Altos, CA: Morgan Kaufmann, 1986.
- LANGTON, C. G. Studying artificial life with cellular automata. **Physica D**, p. 120-149, 1986.
- LANGLEY, P. **Scientific Discovery: Computational Explorations of the Creative Processes**. Cambridge, MA: MIT Press, 1987.
- LANGTON, C. G. **Artificial Life: Santa Fe Institute Studies in the Sciences of Complexity**, VI. Reading, MA: Addison-Wesley, 1989.
- \_\_\_\_\_. Computation at the edge of chaos: Phase transitions and emergent computation. **Physica D**, v. 42, p. 12-37, 1990.
- \_\_\_\_\_. **Artificial Life: An Overview**. Cambridge, MA: MIT Press, 1995.
- \_\_\_\_\_. **Artificial Life II, SFI Studies in the Sciences of Complexity**. v. 10. Reading, MA: Addison-Wesley, 1992.
- LAPLACE, P. **Essai Philosophique sur la Probabilitates**, 3. ed. Paris: Courcier Imprimeur, 1816.
- LARKIN, J. H. et al. Models of competence in solving physics problems. **Cognitive Science**, v. 4, 1980.

- LASKEY, K.; MAHONEY, S. Network fragments: Representing knowledge for constructing probabilistic models. **Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence**. San Francisco: Morgan Kaufmann, 1997.
- LAURITZEN, S. L. **Graphical Models**. Oxford UK: Oxford University Press, 1996.
- LAURITZEN, S. L.; SPIEGELHALTER, D. J. Local computations with probabilities on graphical structures and their application to expert systems. **Journal of the Royal Statistical Society, B**, v. 50, n. 2, p. 157-224, 1988.
- LAVE, J. **Cognition in Practice**. Cambridge: Cambridge University Press, 1988.
- LEAKE, D. B. **Constructing Explanations: A Content Theory**. Northdale, NJ: Erlbaum, 1992.
- \_\_\_\_\_. **Case-based Reasoning: Experiences, Lessons and Future Directions**. Menlo Park: AAAI Press, 1996.
- LEIBNIZ, G. W. **Philosophische Schriften**. Berlin, 1887.
- LENAT, D. B. On automated scientific theory formation: a case study using the AM program. **Machine Intelligence**, v. 9, p. 251-256, 1977.
- \_\_\_\_\_. AM: an artificial intelligence approach to discovery in mathematics as heuristic search. In: DAVIS, R.; LENAT, D.B. **Knowledge-based Systems in Artificial Intelligence**. Nova York: McGraw-Hill, 1982.
- LENAT, D. B. EURISKO: A program that learns new heuristics. **Artificial Intelligence**, v. 21, n. 1, 2, p. 61-98, 1983.
- LENAT, D. B.; BROWN, J. S. Why AM and Eurisko appear to work. **Artificial Intelligence**, v. 23, n. 3, 1984.
- LENNON, J.; McCARTNEY, P. **Rocky Raccoon. The White Album**. Apple Records, 1968.
- LESSER, V. R.; CORKILL, D. D. The distributed vehicle monitoring testbed. **AI Magazine**, v. 4, n. 3, 1983.
- LEVESQUE, H. Foundations of a functional approach to knowledge representation. **Artificial Intelligence**, v. 23, n. 2, 1984.
- LEVESQUE, H. J. A knowledge level account of abduction. **Proceedings of the Eleventh International Joint Conference on Artificial Intelligence**, p. 1051-1067. San Mateo, CA: Morgan Kaufman, 1989.
- LEVESQUE, H. J.; BRACHMAN, R. J. A fundamental tradeoff in knowledge representation and reasoning (revised version). In: BRACHMAN, R. J.; LEVESQUE, H. J. **Readings in Knowledge Representation**. Los Altos, CA: Morgan Kaufmann, 1985.
- LEWIS, J. A.; LUGER, G. F. A constructivist model of robot perception and performance. In: **Proceedings of the Twenty Second Annual Conference of the Cognitive Science Society**. Hillsdale, NJ: Erlbaum, 2000.
- LIEBERMAN, H. Letizia: An agent that assists web browsing. **Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence**, p. 924-929. Cambridge, MA: MIT Press, 1995.
- LIFSCHITZ, V. Some Results on Circumscription. **AAAI 1984**, 1984.
- \_\_\_\_\_. Pointwise Circumscription: Preliminary Report. **Proceedings of the Fifth National Conference on Artificial Intelligence**, p. 406-410. Menlo Park, CA: AAAI Press, 1986.
- LINDE, C. Information structures in discourse. **PhD thesis**, Columbia University, 1974.
- LINDENMAYER, A.; ROSENBERG, G. (Ed.). **Automata, Languages, Development**. Nova York: North-Holland, 1976.
- LINDSAY, R.K. et al. **Applications of artificial intelligence for organic chemistry**: the DENDRAL project. Nova York: McGraw-Hill, 1980.
- LJUNBERG, M.; LUCAS, A. The OASIS air traffic management system. **Proceedings of the Second Pacific Rim International Conference on AI (PRICAI-92)**, 1992.
- LLOYD, J. W. **Foundations of Logic Programming**. Nova York: Springer-Verlag, 1984.
- LOVELACE, A. Notes upon L.F. Menabrea's sketch of the Analytical Engine invented by Charles Babbage. In: MORRISON, P.; MORRISON, E., (Eds.). **Charles Babbage and His Calculating Machines**. Nova York: Dover, 1961.
- LOVELAND, D. W. **Automated Theorem Proving: a Logical Basis**. Nova York: North-Holland, 1978.
- LUCAS, R. **Mastering PROLOG**. Londres: UCL Press, 1996.
- LUCK, S. J. 1998. Cognitive and neural mechanisms in visual search. In: SQUIRE, L. R.; KOSSLYN, S. M. (Eds.). **Findings and Current Opinion in Cognitive Neuroscience**, Cambridge: MIT Press, 1998.
- LUGER, G. F. Formal analysis of problem solving behavior. In: **Cognitive Psychology: Learning and Problem Solving**. Milton Keynes: Open University Press, 1978.
- \_\_\_\_\_. **Cognitive Science: The Science of Intelligent Systems**. San Diego e Nova York: Academic Press, 1994.
- \_\_\_\_\_. **Computation & Intelligence: Collected Readings**. CA: AAAI Press/MIT Press, 1995.
- LUGER, G. F.; LEWIS, J. A.; STERN, C. Problem-solving as model refinement: Towards a constructivist epistemology. **Brain, Behavior, and Evolution**. Basil: Karger; 59:87-100, 2002.
- MAES, P. The dynamics of action selection. **Proceedings of the 11th International Joint Conference on Artificial Intelligence**, p. 991-997. Los Altos, CA: Morgan Kaufmann, 1989.
- \_\_\_\_\_. **Designing Autonomous Agents**. Cambridge, MA: MIT Press, 1990.
- \_\_\_\_\_. Agents that reduce work and information overload. **Communications of the ACM**, v. 37, n. 7, p. 31-40, 1994.
- MAGERMAN, D. Natural Language as Statistical Pattern Recognition. **PhD dissertation**, Stanford University, Department of Computer Science, 1994.

- MANNA, Z.; WALDINGER, R. **The Logical Basis for Computer Programming**. Reading, MA: Addison-Wesley, 1985.
- MANNA, Z.; PNUELI, A. **The Temporal Logic of Reactive and Concurrent Systems: Specification**. Berlin: Springer-Verlag, 1992.
- MANNING, C. D.; SCHUTZE, H. **Foundations of Statistical Natural Language Processing**. Cambridge, MA: MIT Press, 1999.
- MARCUS, M. **A Theory of Syntactic Recognition for Natural Language**. Cambridge, MA: MIT Press, 1980.
- MARKOV, A. A theory of algorithms, **National Academy of Sciences**, USSR, 1954.
- MARSHALL, J. B. **Metacat: A Self-Watching Cognitive Architecture for Analogy-Making and High-Level Perception**. PhD dissertation, Department of Computer Science, Indiana University, 1999.
- MARTIN, A. **Computational Learning Theory: An Introduction**. Cambridge: Cambridge University Press, 1997.
- MARTINS, J. The truth, the whole truth, and nothing but the truth: an indexed bibliography to the literature on truth maintenance systems. **AI Magazine**, v. 11, n. 5, p. 7-25, 1990.
- \_\_\_\_\_. A structure for epistemic states. In: Costa, (Ed.). **New Directions for Intelligent Tutoring Systems**. NATO ASI Series F. Heidelberg: Springer-Verlag, 1991.
- MARTINS, J.; SHAPIRO, S. C. Reasoning in multiple belief spaces. **Proceedings of the Eighth IJCAI**. San Mateo, CA: Morgan Kaufmann, 1983.
- MARTINS, J.; SHAPIRO, S. C. A Model for Belief Revision. **Artificial Intelligence**, v. 35, n. 1, p. 25-79, 1988.
- MASTERMAN, M. Semantic message detection for machine translation, using Interlingua. **Proceedings of the 1961 International Conference on Machine Translation**, 1961.
- MCALLESTER, D. A. **A three-valued truth maintenance system**. MIT AI Lab., Memo 473, 1978.
- MCALLESTER, D. A. PAC-Bayesian model averaging. **Proceedings of the Twelfth ACM Annual Conference on Computational Learning Theory COLT-99**, 1999.
- McCARTHY, J. Recursive functions of symbolic expressions and their computation by machine. **Communications of the ACM**, v. 3, n. 4, 1960.
- \_\_\_\_\_. Programs with common sense. In: MINSKY, M. (Ed.). **Semantic Information Processing**, p. 403-418. Cambridge, MA: MIT Press, 1968.
- \_\_\_\_\_. Epistemological problems in artificial intelligence. **Proceedings IJCAI-77**, p. 1038-1044, 1977.
- \_\_\_\_\_. Circumscription—A form of non-monotonic reasoning. **Artificial Intelligence**, v. 13, 1980.
- \_\_\_\_\_. Applications of Circumscription to Formalizing Common Sense Knowledge. **Artificial Intelligence**, v. 28, p. 89-116, 1986.
- McCARTHY, J.; HAYES, P. J. Some philosophical problems from the standpoint of artificial intelligence. In: MELTZER, B.; MICHIE, D. **Machine Intelligence 4**. Edinburgh: Edinburgh University Press, 1969.
- MCCLELLAND, J. L.; RUMELHART, D. E.; THE PDP Research Group. **Parallel Distributed Processing**, 2 v. Cambridge, MA: MIT Press, 1986.
- MCCORD, M. C. Using slots and modifiers in logic grammars for natural language. **Artificial Intelligence**, v. 18, p. 327-367, 1982.
- \_\_\_\_\_. Design of a PROLOG based machine translation system. **Proceedings of the Third International Logic Programming Conference**, London, 1986.
- MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, p. 115-133, 1943.
- McDERMOTT, D. Planning and acting. **Cognitive Science** 2, p. 71-109, 1978.
- McDERMOTT, D.; DOYLE, J. Non-monotonic logic I. **Artificial Intelligence**, v. 13, p. 14-72, 1980.
- McDERMOTT, J. RI, The formative years. **AI Magazine**, verão, 1981.
- \_\_\_\_\_. RI: A rule based configurer of computer systems. **Artificial Intelligence**, v. 19, 1982.
- McGONIGLE, B. Incrementing intelligent systems by design. **Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB-90)**, 1990.
- \_\_\_\_\_. Autonomy in the making: Getting robots to control themselves. **International Symposium on Autonomous Agents**. Oxford: Oxford University Press, 1998.
- McGRAW, K. L.; HARBISON-BRIGGS, K. **Knowledge Acquisition: Principles and Guidelines**, Englewood Cliffs, NJ: Prentice-Hall, 1989.
- MEAD, C. **Analog VLSI and Neural Systems**. Reading, MA: Addison-Wesley, 1989.
- MELTZER, B.; MICHIE, D. **Machine Intelligence 4**. Edinburgh: Edinburgh University Press, 1969.
- MERLEAU-PONTY, M. **Phenomenology of Perception**. Londres: Routledge & Kegan Paul, 1962.
- MEYER, J. A.; WILSON, S. W. (Eds.). **From animals to animats. Proceedings of the First International Conference on Simulation of Adaptive Behavior**. Cambridge, MA: MIT Press/Bradford Books, 1991.
- MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning: An Artificial Intelligence Approach**, v. 1. Palo Alto, CA: Tioga, 1983.

- \_\_\_\_\_. **Machine Learning: An Artificial Intelligence Approach.** v. 2. Los Altos, CA: Morgan Kaufmann, 1986.
- MICHALSKI, R. S.; STEPP, R. E. 1983. Learning from observation: conceptual clustering. In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning: An Artificial Intelligence Approach**, v. 1. Palo Alto, CA: Tioga, 1983.
- MICHIE, D. Trial and error. **Science Survey**, parte 2, BARNETT, S.A.; MCCLAREN, A. (Eds.), 129- 145. Harmondsworth UK: Penguin, 1961.
- \_\_\_\_\_. **Expert Systems in the Micro-electronic Age.** Edinburgh: Edinburgh Univ. Press, 1979.
- MIKA, P. et al. Foundations for service ontologies: Aligning OWL-S to DULCE. **Proceedings of the Thirteenth International World Wide Web Conference**. Nova York: ACM Press, 2004.
- MILLER, S. L.; DELANEY, T. V.; TALLAL, P. Speech and other central auditory processes: Insights from cognitive neuroscience. SQUIRE, L. R.; KOSSLYN, S. M. (Eds.). **Findings and Current Opinion in Cognitive Neuroscience**. Cambridge: MIT Press, 1998.
- MINSKY, M. (Ed.). **Semantic Information Processing**. Cambridge, MA: MIT Press, 1968.
- \_\_\_\_\_. A framework for representing knowledge, 1975. In: BRACHMAN, R. J.; LEVESQUE, H. J. **Readings in Knowledge Representation**. Los Altos, CA: Morgan Kaufmann, 1985
- \_\_\_\_\_. **The Society of Mind**, Nova York: Simon and Schuster, 1985.
- MINSKY, M.; PAPERT, S. **Perceptrons: An Introduction to Computational Geometry**. Cambridge, MA: MIT Press, 1969.
- MINTON, S. **Learning Search Control Knowledge**. Dordrecht: Kluwer Academic Publishers, 1988.
- MITCHELL, M. **Analogy-Making as Perception**. Cambridge, MA: MIT Press, 1993.
- \_\_\_\_\_. **An Introduction to Genetic Algorithms**. Cambridge, MA: The MIT Press, 1996.
- MITCHELL, M.; CRUTCHFIELD, J.; DAS, R. Evolving cellular automata with genetic algorithms: A review of recent work. **Proceedings of the First International Conference on Evolutionary Computation and Its Applications**. Moscow, Russia: Russian Academy of Sciences, 1996.
- MITCHELL, T. M. Version spaces: an approach to concept learning. **Report No. STAN-CS-78-711**, Computer Science Dept., Stanford University, 1978.
- \_\_\_\_\_. An analysis of generalization as a search problem. **Proceedings IJCAI**, v. 6, 1979.
- \_\_\_\_\_. The need for biases in learning generalizations. **Technical Report CBM-TR-177**. Department of Computer Science, Rutgers University, New Brunswick, NJ, 1980.
- \_\_\_\_\_. Generalization as search. **Artificial Intelligence**, v. 18, n. 2, p. 203-226, 1982.
- \_\_\_\_\_. **Machine Learning**. Nova York: McGraw Hill, 1997.
- MITCHELL, T. M.; UTGOFF, P. E.; BANARJI, R. Learning by experimentation: Acquiring and refining problem solving heuristics. In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning: An Artificial Intelligence Approach**, v. 1. Palo Alto, CA: Tioga, 1983.
- MITCHELL, T. M.; KELLER, R. M.; KEDAR-CABELLI, S. T. Explanation-based generalization: A unifying view. **Machine Learning**, v. 1, n. 1, p. 47-80, 1986.
- MITHEN, S. **The Prehistory of the Mind**. Londres: Thames & Hudson, 1996.
- MOCKLER, R. J.; DOLOGITE, D. G. **An Introduction to Expert Systems**. Nova York: Macmillan, 1992.
- MOONEY, R. J. Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning. In: **Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-96)**. Philadelphia PA. p. 82-91, 1996.
- MOORE, O. K.; ANDERSON, S. B. Modern logic and tasks for experiments on problem solving behavior. **Journal of Psychology**, v. 38, p. 151-160, 1954.
- MOORE, R. C. The role of logic in knowledge representation and commonsense reasoning. **Proceedings AAAI-82**, 1982.
- \_\_\_\_\_. Semantical considerations on nonmonotonic logic. **Artificial Intelligence**, v. 25, n. 1, p. 75-94, 1985.
- MORET, B. M. E.; SHAPIRO, H. D. **Algorithms from P to NP**, v. 1. Redwood City, CA: Benjamin/Cummings, 1991.
- MORRISON, P.; MORRISON, E. (Eds.). **Charles Babbage and His Calculating Machines**. NY: Dover, 1961.
- MUC-5, **Proceedings of the Fifth Message Understanding Conference**, San Francisco: Morgan Kaufmann, 1994.
- MYLOPOULOS, J.; LEVESQUE, H. J. An overview of knowledge representation. In: BRODIE, M. L.; MYLOPOULOS, J.; SCHMIDT, J. W. (Eds.). **Onconceptual modelling: Perspectives from artificial intelligence, databases, and programming languages**, p. 3-17. Nova York: Springer, 1984.
- NAGEL, E.; NEWMAN, J. R. **Gödel's Proof**. Nova York: New York University Press, 1958.
- NAHM, U. Y.; MOONEY R. J. A mutually beneficial integration of data mining and information extraction. **Proceedings of the American Assoc. for Artificial Intelligence Conf.**, p. 627-632, 2000.
- NECHES, R.; LANGLEY, P.; KLAHR, D. Learning, development and production systems. In: KLAHR, D.; LANGLEY, P.; NECHES, R. (Eds.). **Production System Models of Learning and Development**. Cambridge, MA: MIT Press, 1987.
- NEGOITA, C. **Expert Systems and Fuzzy Systems**. Menlo Park, CA: Benjamin/Cummings, 1985.

- NEVES, J. C. F. M.; LUGER, G. F.; CARVALHO, J. M. A formalism for views in a logic data base. In: **Proceedings of the ACM Computer Science Conference**, Cincinnati, OH, 1986.
- NEWELL, A. Physical symbol systems. In: NORMAN, D. A. **Perspectives on Cognitive Science**. Hillsdale, NJ: Erlbaum, 1981.
- \_\_\_\_\_. The knowledge level. **Artificial Intelligence**, v. 18, n. 1, p. 87-127, 1982.
- \_\_\_\_\_. **Unified Theories of Cognition**. Cambridge, MA: Harvard University Press, 1990.
- NEWELL, A.; ROSENBLOOM, P. S.; LAIRD, J. E. Symbolic architectures for cognition. In: Posner, M. I. **Foundations of Cognitive Science**. Cambridge, MA: MIT Press, 1989.
- NEWELL, A.; SHAW, J. C.; SIMON, H. A. Elements of a theory of human problem solving. **Psychological Review**, v. 65, p. 151-166, 1958.
- NEWELL, A.; SIMON, H. A. The logic theory machine. **IRE Transactions on Information Theory**, v. 2, p. 61-79, 1956.
- \_\_\_\_\_. GPS: a program that simulates human thought. In: BILLING, H. (Ed.). **Lernende Automaten**, p. 109-124. Munich: R. Oldenbourg KG, 1961.
- \_\_\_\_\_. Empirical explorations with the Logic Theory Machine: a case study in heuristics. In: FEIGENBAUM, E.A.; FELDMAN, J. (Eds.). **Computers and Thought**. Nova York: McGraw-Hill, 1963.
- \_\_\_\_\_. GPS: a program that simulates human thought. In: FEIGENBAUM, E.A.; FELDMAN, J. (Eds.). **Computers and Thought**. Nova York: McGraw-Hill, 1963b.
- \_\_\_\_\_. **Human Problem Solving**. Engelwood Cliffs, NJ: Prentice Hall, 1972.
- \_\_\_\_\_. Computer science as empirical inquiry: symbols and search. **Communications of the ACM**, v. 19, n. 3, p. 113-126, 1976.
- NG, H. T.; MOONEY, R. J. On the Role of Coherence in Abductive Explanation. **Proceedings of the Eighth National Conference on Artificial Intelligence**, p. 347-342. Menlo Park, CA: AAAI Press/ MIT Press, 1990.
- NG, R.; SUBRAHMANIAN, V. Probabilistic logic programming. **Information and Computation**, v. 101, n. 2, p. 150-201, 1992.
- NGO, L.; HADDAWY, P. Answering queries from context-sensitive knowledge bases. **Theoretical Computer Science**, v. 171, p. 147-177, 1997.
- NGO, L. et al. Efficient temporal probabilistic reasoning via context-sensitive model construction. **Computers in Biology and Medicine**, v. 27, p. 453-476, 1997.
- NII, H. P. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. **AI Magazine**, v. 7, n. 2, 1986a.
- \_\_\_\_\_. Blackboard systems: Blackboard application systems from a knowledge based perspective. **AI Magazine**, v. 7, n. 3, 1986b.
- NII, H. P.; AIELLO, N. AGE: A knowledge based program for building knowledge based programs. **Proceedings IJCAI 6**, 1979.
- NILSSON, N. J. **Learning Machines**. Nova York: McGraw-Hil, 1965.
- \_\_\_\_\_. **Problem-Solving Methods in Artificial Intelligence**. Nova York: McGraw-Hill, 1971.
- \_\_\_\_\_. **Principles of Artificial Intelligence**. Palo Alto, CA: Tioga, 1980.
- \_\_\_\_\_. Teleo-Reactive Programs for Agent Control. **Journal of Artificial Intelligence Research**, v. 1, p. 139-158, 1994.
- \_\_\_\_\_. **Artificial Intelligence: A New Synthesis**. San Francisco: Morgan Kaufmann, 1998.
- Nishibe, Y. et al. Distributed channel allocation in atm networks. **Proceedings of the IEEE Globecom Conference**, p. 12.2.1-12.2.7, 1993.
- NORDHAUSEN, B.; LANGLEY, P. 1990. An integrated approach to empirical discovery. In: SHRAGER, J.; LANGLEY, P. (Eds.). **Computational Models of Scientific Discovery and Theory Formation**. San Mateo, CA: Morgan Kaufmann, 1990.
- NORMAN, D. A. Memory, knowledge and the answering of questions. **CHIP Technical Report 25**, Center for Human Information Processing, University of California, San Diego, 1972.
- \_\_\_\_\_. **Perspectives on Cognitive Science**. Hillsdale, NJ: Erlbaum, 1981.
- NORMAN, D. A.; RUMELHART, D. E.; THE LNR Research Group. **Explorations in Cognition**. San Francisco: Freeman, 1975.
- O'KEEFE, R. **The Craft of PROLOG**. Cambridge, MA: MIT Press, 1990.
- O'LEARY, D. D. M.; YATES, P.; McLAUGHLIN, T. 1999. Mapping sights and smells in the brain: distinct chanisms to achieve a common goal. **Cell**, v. 96, p. 255-269.
- OLIVER, I. M.; SMITH, D.J.; HOLLAND, J. R. C. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. **Proceedings of the Second International Conference on Genetic Algorithms**, p. 224-230. Hillsdale, NJ: Erlbaum & Assoc., 1987.
- OLIVEIRA, E.; FONSECA, J. M.; STEIGER-GARCAO, A. MACIV: A DAI based resource management system. **Applied Artificial Intelligence**, v. 11, n. 6, p. 525-550, 1997.
- PAPERT, S. **Mindstorms**. Nova York: Basic Books, 1980.
- PASCAL, B. **Pensees de M. Pascal sur la Religion et sur quelques autres Sujets**. Paris: Chez Guillaume Desprez, 1670.
- PATIL, R.; SZOLOVITS, P.; SCHWARTZ, W. Causal understanding of patient illness in medical diagnosis. **Proceedings of the International Joint Conference on Artificial Intelligence**. Palo Alto: Morgan Kaufmann, 1981.

- PEARL, J. **Heuristics: Intelligent Strategies for Computer Problem Solving**. Reading, MA: Addison-Wesley, 1984.
- \_\_\_\_\_. **Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference**. Los Altos, CA: Morgan Kaufmann, 1988.
- \_\_\_\_\_. **Causality**. Nova York: Cambridge University Press, 2000.
- PEARLMUTTER, B. A.; PARRA, L. C. Maximum likelihood blind source separation: A context-sensitive generalization of ICA. **Advances in Neural Information Processing Systems**, v. 9, Cambridge, MA: MIT Press, 1997.
- PEIRCE, C. S. **Collected Papers 1931-1958**. Cambridge MA: Harvard University Press, 1958.
- PENROSE, R. **The Emperor's New Mind**. Oxford: Oxford University Press, 1989.
- PEREIRA, L. M.; WARREN, D. H. D. Definite clause grammars for language analysis – A survey of the formalism and a comparison with augmented transition networks. **Artificial Intelligence**, v. 13, p. 231-278, 1980.
- PERLIS, D. Autocircumscription. **Artificial Intelligence**, v. 36, p. 223-236, 1988.
- PERRIOLAT, F. et al. **Using archon**: Particle accelerator Control, v. 11, n. 6, p. 80-86, 1996.
- PFEFFER, A. IBAL: A probabilistic rational programming language. In: **Proceedings of International Joint Conference on Artificial Intelligence**, p. 733-740. Cambridge MA: MIT Press, 2001.
- PFEFFER, A. et al. SPOOK: A system for probabilistic object-oriented knowledge representation. **Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence**. San Francisco: Morgan Kaufmann, 1999.
- PIAGET, J. **The Construction of Reality in The Child**. Nova York: Basic Books, 1954.
- \_\_\_\_\_. **Structuralism**. Nova York: Basic Books, 1970.
- PINKER, S. **The Language Instinct**. Nova York: William Morrow and Company, 1994.
- PLATÃO (*Plato*). **The Collected Dialogues of Plato**, HAMILTON, E.; CAIRNS, H. (Eds.). Princeton: Princeton University Press, 1961.
- PLESS, D.; LUGER, G. F.; STERN, C. A new object-oriented stochastic modeling language. **Artificial Intelligence and Soft Computing: Proceedings of the IASTED International Conference**. Anaheim: IASTED/ACTA Press, 2000.
- PLESS, D.; LUGER, G. F. Towards general analysis of recursive probability models. In: **Proceedings of Uncertainty in Artificial Conference - 2001**. San Francisco: Morgan Kaufmann, 2001.
- \_\_\_\_\_. EM learning of product distributions in a first-order stochastic logic language. **Artificial Intelligence and Soft Computing: Proceedings of the IASTED International Conference**. Anaheim: IASTED/ACTA Press, 2003. Também disponível como: **University of New Mexico Computer Science Technical Report TR-CS-2003-01**, 2003.
- PLESS, D. et al. The design and testing of a first-order logic-based stochastic modeling language. **Int. Journal on Artificial Intelligence Tools**, v. 15, n. 6, p. 979-1005, 2006.
- POLYA, G. **How to Solve It**. Princeton, NJ: Princeton University Press, 1945.
- POPPER, K. R. 1959. **The Logic of Scientific Discovery**. Londres: Hutchinson.
- PORFÍRIO (*Porphyry*). Isagoge et in Aristotelis categorias commentarium. In: BUSSE, A. (Ed.). **Commentaria in Aristotelem Graeca**, v. 4, n. 1, 1887.
- POSNER, M. I. **Foundations of Cognitive Science**. Cambridge, MA: MIT Press, 1989.
- POST, E. Formal reductions of the general combinatorial problem. **American Journal of Mathematics**, v. 65, p. 197-268, 1943.
- POUNDSTONE, W. **The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge**. Nova York: William Morrow and Company, 1985.
- PRERAU, D. S. **Developing and Managing Expert Systems: Proven Techniques for Business and Industry**. Reading, MA: Addison-Wesley, 1990.
- PUTERMAN, M. L. Leverage and influence in autocorrelated regression. **Journal of Applied Statistics**, v. 37, n. 1, p. 76-86, 1988.
- PYLYSHYN, Z. W. What the mind's eye tells the mind's brain: a critique of mental imagery. **Psychological Bulletin**, v. 80, p. 1-24, 1973.
- \_\_\_\_\_. The causal power of machines. **Behavioral and Brain Sciences**, v. 3, p. 442-444, 1980.
- \_\_\_\_\_. **Computation and Cognition: Toward a Foundation for Cognitive Science**. Cambridge, MA: MIT Press, 1984.
- QUILLIAN, M. R. Word concepts: A theory and simulation of some basic semantic capabilities, 1967. In: BRACHMAN, R. J.; LEVESQUE, H. J. **Readings in Knowledge Representation**. Los Altos, CA: Morgan Kaufmann, 1985.
- QUINE, W. V. O. **From a Logical Point of View**. 2. ed. Nova York: Harper Torchbooks, 1963.
- QUINLAN, J. R. Learning efficient classification procedures and their application to chess endgames. In: MICHalski, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning: An Artificial Intelligence Approach**, v. 1. Palo Alto, CA: Tioga, 1983.
- \_\_\_\_\_. Induction of decision trees. **Machine Learning**, v. 1, n. 1, p. 81-106, 1986a.
- \_\_\_\_\_. The effect of noise on concept learning. In: MICHalski, R. S.; CARBONELL, J. G.; MITCHELL, T. M., (Eds.). **Machine Learning: An Artificial Intelligence Approach**. v. 2. Los Altos, CA: Morgan Kaufmann, 1986.
- \_\_\_\_\_. **C4.5: Programs for Machine Learning**. San Francisco: Morgan Kaufmann, 1993.
- \_\_\_\_\_. Bagging, Boosting and C4.5. **Proceedings AAAI 96**. Menlo Park: AAAI Press, 1996.

- QUINLAN, P. **Connectionism and Psychology**. Chicago: University of Chicago Press, 1991.
- RABINER, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, v. 77, n. 2, p. 257-286, 1989.
- RAPHAEL, B. SIR: A computer program for semantic information retrieval. In: MINSKY, M. (Ed.). **Semantic Information Processing**. Cambridge, MA: MIT Press, 1968.
- REDDY, D. R. Speech recognition by machine: A review. *Proceedings of the IEEE* 64, maio de 1976.
- REGGIA, J.; NAU, D. S.; WANG, P. Y. Diagnostic expert systems based on a set covering model. *International Journal of Man-Machine Studies*, v. 19, n. 5, p. 437-460, 1983.
- REINFRANK, M. Fundamental and Logical Foundations of Truth Maintenance. Linköping Studies in Science and Technology, Dissertation 221, Department of Computer and Information Science. Linköping University, Linköping, Suécia, 1989.
- REITER, R. On closed world databases, 1978. In: WEBBER, B. L.; NILSSON, N. J. **Readings in Artificial Intelligence**. Los Altos: Tioga Press, 1981.
- \_\_\_\_\_. A logic for default reasoning. *Artificial Intelligence*, v. 13, p. 81-132, 1980.
- \_\_\_\_\_. On reasoning by default. BRACHMAN, R. J.; LEVESQUE, H. J. **Readings in Knowledge Representation**. Los Altos, CA: Morgan Kaufmann, 1985.
- REITER, R.; CRISCUOLO, G. On interacting defaults. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981.
- REITMAN, W. R. **Cognition and Thought**. Nova York: Wiley, 1965.
- RICH, E.; KNIGHT, K. **Artificial Intelligence**, 2. ed. Nova York: McGraw-Hil, 1991.
- RICHARDSON, M.; DOMINGOS, P. Markov logic networks. *Journal of Machine Learning*, v. 62, p. 107-136, 2006.
- RISSLAND, E. L. Examples in legal reasoning: Legal hypotheticals. *Proceedings of IJCAI-83*. San Mateo, CA: Morgan Kaufmann, 1983.
- RISSLAND, E. L.; ASHLEY, K. HYPO: A case-based system for trade secrets law. *Proceedings, First International Conference on Artificial Intelligence and Law*, 1987.
- ROBERTS, D. D. **The Existential Graphs of Charles S. Pierce**. The Hague: Mouton, 1973.
- ROBINSON, J. A. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, v. 12, p. 23-41, 1965.
- ROCHESTER, N. et al. Test on a cell assembly theory of the actuation of the brain, using a large digital computer. In: ANDERSON, J. A.; ROSENFIELD, E. (Eds.). **Neurocomputing: Foundations of Research**. Cambridge, MA: MIT Press, 1988.
- ROSCH, E. e LLOYD, B.B. (Eds.). **Cognition and Categorization**. Hillsdale, NJ: Erlbaum, 1978.
- ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, p. 386-408, 1958.
- \_\_\_\_\_. **Principles of Neurodynamics**. Nova York: Spartan, 1962.
- ROSENBLOOM, P. S.; NEWELL, A. Learning by chunking, a production system model of practice. In: KLAHR, D.; LANGLEY, P.; NECHES, R. (Eds.). **Production System Models of Learning and Development**. Cambridge, MA: MIT Press, 1987.
- ROSENBLOOM, P. S.; LEHMAN, J. F.; LAIRD, J. E. Overview of Soar as a unified theory of cognition: Spring 1993. *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, 1993.
- ROSS, P. **Advanced PROLOG**. Reading, MA: Addison-Wesley, 1989.
- ROSS, S. M. **A First Course in Probability**, 3. ed. Londres: Macmillan, 1988.
- ROSS, T. **Fuzzy Logic with Engineering Applications**. Nova York: McGraw-Hill, 1995.
- RUMELHART, D. E.; LINDSAY, P. H.; NORMAN, D. A. 1972. A process model for long-term memory. In: TULVING, E.; DONALDSON, W. (Eds.). **Organization of memory**. Nova York: Academic Press, 1972.
- RUMELHART, D. E.; NORMAN, D. A. Active semantic networks as a model of human memory. *Proceedings IJCAI-3*, 1973.
- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning internal representations by error propagation. In: McCLELLAND, J. L.; RUMELHART, D. E.; THE PDP Research Group. **Parallel Distributed Processing**, 2 v. Cambridge, MA: MIT Press, 1986a.
- RUMELHART, D. E.; MCCLELLAND, J. L.; THE PDP Research Group. **Parallel Distributed Processing**, v. 1. Cambridge, MA: MIT Press, 1986b.
- RUSSELL, S. J. **The Use of Knowledge in Analogy and Induction**. San Mateo, CA: Morgan Kaufmann, 1989.
- RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. Englewood Cliffs, NJ: Prentice-Hall, 1995, 2003.
- \_\_\_\_\_. **Artificial Intelligence: A Modern Approach**, 2. ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.
- SACERDOTTI, E. D. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, v. 5, p. 115-135, 1974.
- \_\_\_\_\_. The non linear nature of plans. *Proceedings IJCAI 4*, 1975.
- \_\_\_\_\_. **A Structure of Plans and Behavior**. Nova York: Elsevier, 1977.
- SACKS, O. **The Man who Mistook his Wife for a Hat**. Nova York: Simon & Schuster, 1987.

- SAGI, D.; TANNE, D. Perceptual learning: Learning to see. In: SQUIRE, L. R.; KOSSLYN, S. M. (Eds.). **Findings and Current Opinion in Cognitive Neuroscience**. Cambridge: MIT Press, 1998.
- SAKHANENKO, N.; LUGER, G.F.; STERN, C. R. Managing dynamic contexts using failure-driven stochastic models. **Proceedings of FLAIRS Conference**. Menlo Park CA: AAAI Press, 2006.
- SAMUEL, A. L. 1959. Some studies in machine learning using the game of checkers. **IBM Journal of R & D**, v. 3, p. 211-229.
- SCHANK, R. C. **Dynamic Memory: A Theory of Reminding and Learning in Computers and People**. Londres: Cambridge University Press, 1982.
- SCHANK, R. C.; ABELSON, R. **Scripts, Plans, Goals and Understanding**. Hillsdale, NJ: Erlbaum, 1977.
- SCHANK, R. C.; COLBY, K. M. (Ed.). **Computer Models of Thought and Language**. San Francisco: Freeman, 1973.
- SCHANK, R. C.; NASH-WEBBER, B. L. (Ed.). **Theoretical Issues in Natural Language Processing**. Association for Computational Linguistics, 1975.
- SCHANK, R. C.; RIEGER, C. J. Inference and the computer understanding of natural language. **Artificial Intelligence**, v. 5, n. 4, p. 373-412, 1974.
- SCHANK, R. C.; RIESBECK, C. K. (Ed.). **Inside Computer Understanding: Five Programs Plus Miniatures**. Hillsdale, N. J.: Erlbaum, 1981.
- SCHOLKOPF, B. et al. New support vector algorithms. **Technical Report NC-TR-98-031**, NeuroCOLT Working Group, 1998.
- SCHOONDERWOLD, R.; HOLLAND, O.; BRUTEN, J. Ant-like agents for load balancing in telecommunications networks. **Proceedings of the First International Conference on Autonomous Agents (Agents-97)**, p. 209-216, 1997.
- SCHROOTEN, R.; VAN DE VELDE, W. Software agent foundation for dynamic interactive electronic catalogues. **Applied Artificial Intelligence**, v. 11, n. 5, p. 459-482, 1997.
- SCHWUTTKE, U. M.; QUAN, A. G. Enhancing performance of cooperating agents in real-time diagnostic systems. **Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence**, p. 332-337, 1993.
- SEARLE, J. **Speech Acts**. Londres: Cambridge University Press, 1969.
- SEARLE, J. R. Minds, brains and programs. **The Behavioral and Brain Sciences**, v. 3, p. 417-424, 1980.
- SEBEOOK, T. A. **Contributions to the Doctrine of Signs**. Lanham, MD: Univ. Press of America, 1985.
- SEDGEWICK, R. **Algorithms**. Reading, MA: Addison-Wesley, 1983.
- SEJNOWSKI, T. J.; ROSENBERG, C. R. Parallel networks that learn to pronounce English text. **Complex Systems**, v. 1, p. 145-168, 1987.
- SELFRIDGE, O. Pandemonium: A paradigm for learning. **Symposium on the Mechanization of Thought**. Londres: HMSO, 1959.
- SELMAN, B.; LEVESQUE, H. J. Abductive and Default Reasoning: A Computational Core. **Proceedings of the Eighth National Conference on Artificial Intelligence**, p. 343-348. Menlo Park, CA: AAAI Press/MIT Press, 1990.
- SELZ, O. **Über die Gesetze des Geordneten Denkverlaufs**. Stuttgart: Spemann, 1913.
- \_\_\_\_\_. **Zur Psychologie des Produktiven Denkens und des Irrtums**. Bonn: Friedrich Cohen, 1922.
- SHAFER, G.; PEARL, J. (Eds.). **Readings in Uncertain Reasoning**. Los Altos, CA: Morgan Kaufmann, 1990.
- SHANNON, C. A mathematical theory of communication. **Bell System Technical Journal**, 1948.
- \_\_\_\_\_. Prediction and entropy of printed English. **Bell System Technical Journal**, v. 30 p. 54-60, 1951.
- SHAPIRO, E. **Concurrent Prolog**: Collected Papers, 2 v. Cambridge, MA: MIT Press, 1987.
- SHAPIRO, S. C. A net structure for semantic information storage, deduction, and retrieval. **Proceedings of the Second International Joint Conference on Artificial Intelligence**, p. 512-523, 1971.
- \_\_\_\_\_. The SNePS semantic network processing system. In: FINDLER, N.V. (Ed.). **Associative Networks: Representation and Use of Knowledge by Computers**. Nova York: Academic Press, p. 179-203, 1979.
- \_\_\_\_\_. **Encyclopedia of Artificial Intelligence**. Nova York: Wiley-Interscience, 1987.
- \_\_\_\_\_. **Encyclopedia of Artificial Intelligence**. Nova York: Wiley1992.
- SHAPIRO, S. C. et al. Metacognition in SNePS. **AI Magazine**, v. 28, n. 1, p. 17-31, 2007.
- SHAVLIK, J. W.; DIETTERICH, T. G. (Ed.). **Readings in Machine Learning**. San Mateo, CA: Morgan Kaufmann, 1990.
- SHAVLIK, J. W.; MOONEY, R. J.; TOWELL, G. G. Symbolic and neural learning algorithms: An experimental comparison. **Machine Learning**, v. 6, n. 1, p. 111-143, 1991.
- SHEPHARD, G. M. **The Synaptic Organization of the Brain**. Nova York: Oxford University Press, 1998.
- SHRAGER, J.; LANGLEY, P. (Eds.). **Computational Models of Scientific Discovery and Theory Formation**. San Mateo, CA: Morgan Kaufmann, 1990.
- SIEGELMAN, H.; SONTAG, E. D. Neural networks are universal computing devices. **Technical Report SYCON 91-08**. New Jersey: Rutgers Center for Systems and Control, 1991.
- SIEKMANN, J. H.; WRIGHTSON, G. (Ed.). **The Automation of Reasoning: Collected Papers from 1957 to 1970**. v. I. Nova York: Springer-Verlag, 1983a.

- \_\_\_\_\_. **The Automation of Reasoning: Collected Papers from 1957 to 1970, v. II.** Nova York: Springer-Verlag, 1983b.
- SIMMONS, R. F. Storage and retrieval of aspects of meaning in directed graph structures. **Communications of the ACM**, v. 9, p. 211-216, 1966.
- \_\_\_\_\_. Semantic networks: Their computation and use for understanding English sentences. In: SCHANK, R. C.; COLBY, K. M. (Ed.). **Computer Models of Thought and Language**. San Francisco: Freeman, 1973.
- SIMON, D. P.; SIMON, H. A. Individual differences in solving physics problems. In: SIEGLER, R. (Ed.), **Children's thinking: What develops?** Hillsdale, NJ: Erlbaum, 1978.
- SIMON, H. A. **The Sciences of the Artificial**, 2. ed. Cambridge, MA: MIT Press, 1981.
- \_\_\_\_\_. Why should machines learn? In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. (Eds.). **Machine Learning: An Artificial Intelligence Approach**, v. 1. Palo Alto, CA: Tioga, 1983.
- SIMS, M. H. Empirical and analytic discovery in IL. **Proceedings of the Fourth International Workshop on Machine Learning**. Los Altos, CA: Morgan Kaufmann, 1987.
- SKINNER, J. M.; LUGER, G. F. A synergistic approach to reasoning for autonomous satellites. **Proceedings for NATO Conference of the Advisory Group for Aerospace Research and Development**, 1991.
- \_\_\_\_\_. An architecture for integrating reasoning paradigms. In: NOBEL, B.; RICH, C.; SWARTOUT, W., (Eds.). **Principles of Knowledge Representation and Reasoning**. San Mateo, CA: Morgan Kaufmann, 1992.
- \_\_\_\_\_. Contributions of a case-based reasoner to an integrated reasoning system. **Journal of Intelligent Systems**. Londres: Freund, 1995.
- SLEEMAN, D.; BROWN, J. S. **Intelligent Tutoring Systems**. Nova York: Academic Press, 1982.
- SMART, G.; LANGELAND-KNUDSEN, J. **The CRI Directory of Expert Systems**. Oxford: Learned Information (Europe) Ltd., 1986.
- SMITH, B.C. Prologue to reflection and semantics in a procedural language. In: BRACHMAN, R. J.; LEVESQUE, H. J. **Readings in Knowledge Representation**. Los Altos, CA: Morgan Kaufmann, 1985.
- \_\_\_\_\_. **On the Origin of Objects**. Cambridge, MA: MIT Press, 1996.
- SMITH, R. G. BAKER, J. D. The dipmeter advisor system: a case study in commercial expert system development. **Proc. 8th IJCAI**, p. 122-129, 1983.
- SMOLIAR, S. W. A View of Goal-oriented Programming, Schlumberger-Doll Research Note, 1985.
- SMYTH, P. Belief networks, hidden Markov models, and Markov random fields: A unifying view. **Pattern Recognition Letters**, v. 18, p. 1261-1268, Springer, 1997.
- SODERLAND, S. et al. CRYSTAL: Inducing a conceptual dictionary. **Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence**, p. 1314-1319, 1995.
- SOLOWAY, E.; BACHANT, J.; JENSEN, K. Assessing the maintainability of XCON-in-RIME: Coping with the problems of a very large rule base. **Proceedings AAAI-87**. Los Altos, CA: Morgan Kaufmann, 1987.
- SOWA, J. F. **Conceptual Structures: Information Processing in Mind and Machine**. Reading, MA: Addison-Wesley, 1984.
- SQUIRE, L. R.; KOSSLYN, S. M. (Eds.). **Findings and Current Opinion in Cognitive Neuroscience**. Cambridge: MIT Press, 1998.
- STERN, C. R. **An Architecture for Diagnosis Employing Schema-based Abduction**. PhD dissertation, Department of Computer Science, University of New Mexico, 1996.
- STERN, C. R.; LUGER, G. F. A model for abductive problem solving based on explanation templates and lazy evaluation. **International Journal of Expert Systems**, v. 5, n. 3, p. 249-265, 1992.
- \_\_\_\_\_. Abduction and abstraction in diagnosis: a schema-based account. In: FORD et al. (Eds.). **Situated Cognition: Expertise in Context**. Cambridge, MA: MIT Press, 1997.
- STUBBLEFIELD, W. A. **Source Retrieval in Analogical Reasoning: An Interactionist Approach**. PhD dissertation, Department of Computer Science, University of New Mexico, 1995.
- STUBBLEFIELD, W. A.; LUGER, G. F. Source Selection for Analogical Reasoning: An Empirical Approach. **Proceedings: Thirteenth National Conference on Artificial Intelligence**, 1996.
- STYTZ, M. P.; FRIEDER, O. Three dimensional medical imagery modalities: An overview. **Critical Review of Biomedical Engineering**, v. 18, p. 11-25, 1990.
- SUCHMAN, L. **Plans and Situated Actions: The Problem of Human Machine Communication**. Cambridge: Cambridge University Press, 1987.
- SUSSMAN, G. J. **A Computer Model of Skill Acquisition**. Cambridge, MA: MIT Press, 1975.
- SUTTON, R. S. Learning to predict by the method of temporal differences. **Machine Learning** 3, 9-44, 1988.
- \_\_\_\_\_. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. **Proceedings of the Seventh International Conference on Machine Learning**, p. 216-224. San Francisco: Morgan Kaufmann, 1990.
- \_\_\_\_\_. Dyna: An integrated architecture for learning, planning, and reacting. **SIGART Bulletin**, v. 2, p. 160-164, ACM Press, 1991.
- SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning**. Cambridge: MIT Press, 1998.

- SYCARA, K. et al. Distributed intelligent agents. **IEEE Expert**, v. 11, n. 6, p. 36-46, 1996.
- TANG, A. C.; PEARLMUTTER, B. A.; ZIBULEVSKY, M. Blind source separation of neuromagnetic responses. **Computational Neuroscience 1999**, Proceedings published in **Neurocomputing**, 1999.
- TANG, A. C. et al. A MEG study of response latency and variability in the human visual system during a visual-motor integration task. **Advances in Neural Info. Proc. Sys.** San Francisco: Morgan Kaufmann, 2000a.
- TANG, A. C.; PHUNG, D.; PEARLMUTTER, B. A. Direct measurement of interhemispherical transfer time (IHTT) for natural somatosensory stimulation during voluntary movement using MEG and blind source separation. **Society of Neuroscience Abstracts**, 2000b.
- TAKAHASHI, K. et al. Intelligent pp.: Collecting shop and service information with software agents. **Applied Artificial Intelligence**, v. 11, n. 6, p. 489-500, 1997.
- TARSKI, A. The semantic conception of truth and the foundations of semantics. **Philos. e Phenom. Res.**, v. 4, p. 341-376, 1944.  
\_\_\_\_\_. **Logic, Semantics, Metamathematics**. Londres: Oxford University Press, 1956.
- TESAURO, G. J. TD-Gammon, a self-teaching backgammon program achieves master-level play. **Neural Computation**, v. 6, n. 2, p. 215-219, 1994.  
\_\_\_\_\_. Temporal difference learning and TD-Gammon. **Communications of the ACM**, 38: 58-68, 1995.
- THAGARD, P. 1988. Dimensions of analogy. HELMAN, D. H. (Ed.). **Analogical Reasoning**. Londres: Kluwer Academic, 1988.
- THRUN, S.; BROOKS, R.; DURRANT-WHYTE, H. (Eds.). **Robotics Research: Results of the 12th International Symposium**. Heidelberg: Springer Tracts in Advanced Robotics, v. 28, 2007.
- TOURETZKY, D. S. **The Mathematics of Inheritance Systems**. Los Altos, CA: Morgan Kaufmann, 1986.  
\_\_\_\_\_. **Common LISP: A Gentle Introduction to Symbolic Computation**. Redwood City, CA: Benjamin/Cummings, 1990.
- TRAPPL, R.; PETTA, P. **Creating Personalities for Synthetic Actors**. Berlin: Springer-Verlag, 1997.
- TREISMAN, A. The perception of features and objects. In: BADDERLY, A.; WEISKRANTZ, L. (Eds.). **Attention: Selection, Awareness, and Control: A Tribute to Donald Broadbent**, p. 5-35. Oxford: Clarendon Pres, 1993.  
\_\_\_\_\_. The binding problem. In: SQUIRE, L. R.; KOSSLYN, S. M. (Eds.). **Findings and Current Opinion in Cognitive Neuroscience**. Cambridge: MIT Press, 1998.
- TURING, A. A. Computing machinery and intelligence. **Mind**, v. 59, p. 433-460, 1950.
- TURNER, R. **Logics for Artificial Intelligence**. Chichester: Ellis Horwood, 1984.
- ULLMAN, J. D. **Principles of Database Systems**. Rockville, MD: Computer Science Press, 1982.
- UREY, H. C. **The Planets: Their Origin and Development**. Yale University Press, 1952.
- UTGOFF, P. E. 1986. Shift of bias in inductive concept learning. In: MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M., (Eds.). **Machine Learning: An Artificial Intelligence Approach**. v. 2. Los Altos, CA: Morgan Kaufmann, 1986.
- VALIANT, L. G. A theory of the learnable. **CACM**, v. 27, p. 1134-1142, 1984.
- VAN DER GAAG, L. C. (Ed.). Special issue on Bayesian Belief Networks. **AISB Quarterly**, v. 94, 1996.
- VAN EMDEN, M.; KOWALSKI, R. The semantics of predicate logic and a programming language. **Journal of the ACM**, v. 23, p. 733-742, 1976.
- WARELA, F. J.; THOMPSON, E.; ROSCH, E. **The Embodied Mind: Cognitive Science and Human Experience**. Cambridge, MA: MIT Press, 1993.
- VELOSO, M. et al. CNITED-98 RoboCup-98 small robot world champion team. **AI Magazine**, v. 21, n. 1, p. 29-36, 2000.
- VERE, S. A. Induction of concepts in the predicate calculus. **Proceedings IJCAI 4**, 1975.  
\_\_\_\_\_. Inductive learning of relational productions. In: Waterman, D. e Hayes-Roth, F. **Pattern Directed Inference Systems**. Nova York: Academic Press, 1978.
- VEROFF, R. (Ed.). **Automated Reasoning and its Applications**. Cambridge, MA: MIT Press, 1997.
- VEROFF, R.; SPINKS, M. Axiomatizing the skew boolean propositional calculus. **Journal of Automated Reasoning**, v. 37, n. 1-2, p. 3-20, 2006.
- VIEIRA, R. et al. On the formal semantics of speech-act based communication in an agent-oriented programming language. **Journal of Artificial Intelligence Research (JAIR)**, v. 29, p. 221-267, 2007.
- VITERBI, A. J. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. **IEEE Transactions on Information Theory**, v. IT-13, n. 2, p. 260-269.
- VON GLASERFELD, E. An introduction to radical constructivism. **The Invented Reality**, Watzlawick, (Ed.), p. 17-40, Nova York: Norton, 1978.
- WALKER, A. et al. **Knowledge Systems and PROLOG: A Logical Approach to Expert Systems and Natural Language Processing**. Reading, MA: Addison-Wesley, 1987.
- WARREN, D. H. D. Generating conditional plans and programs. **Proc. AISB Summer Conference**, Edinburgh, p. 334-354, 1976.
- WARREN, D. H. D.; PEREIRA, F.; PEREIRA, L. M. **User's Guide to DEC-System 10 PROLOG**. Occasional Paper 15, Department of Artificial Intelligence, University of Edinburgh, 1979.

- WARREN, D. H. D.; PEREIRA, L. M.; PEREIRA, F. PROLOG—the language and its implementation compared with LISP. **Proceedings, Symposium on AI and Programming Languages**, SIGPLAN Notices, v. 12, p. 8, 1977.
- WATERMAN, D.; HAYES-ROTH, F. **Pattern Directed Inference Systems**. Nova York: Academic Press, 1978.
- WATERMAN, D. A. Machine Learning of Heuristics. Report No. STAN-CS-68-118, Computer Science Dept, Stanford University, 1968.
- \_\_\_\_\_. **A Guide to Expert Systems**. Reading, MA: Addison-Wesley, 1986.
- WATKINS, C. J. C. H. **Learning from Delayed Rewards**. PhD Thesis, Cambridge University, 1989.
- WAVISH, P.; GRAHAM, M. A situated action approach to implementing characters in computer games. **Applied Artificial Intelligence**, v. 10, n. 1, p. 53-74, 1996.
- WEBBER, B. L.; NILSSON, N. J. **Readings in Artificial Intelligence**. Los Altos: Tioga Press, 1981.
- WEISS, S. M.; KULIKOWSKI, C. A. **Computer Systems that Learn**. San Mateo, CA: Morgan Kaufmann, 1991.
- WEISS, S. M. et al. A model-based method for computer-aided medical decision-making. **Artificial Intelligence**, v. 11, n. 1-2, p. 145-172, 1977.
- WEIZENBAUM, J. **Computer Power and Human Reason**. San Francisco: W. H. Freeman, 1976.
- WELD, D. S. An introduction to least commitment planning. **AI Magazine**, v. 15, n. 4, p. 16-35, 1994.
- WELD, D. S.; DE KLEER, J. (Eds.). **Readings in Qualitative Reasoning about Physical Systems**. Los Altos, CA: Morgan Kaufmann, 1990.
- WELLMAN, M. P. Fundamental concepts of qualitative probabilistic networks. **Artificial Intelligence**, v. 44, n. 3, p. 257-303, 1990.
- WEYHRAUCH, R. W. Prolegomena to a theory of mechanized formal reasoning. **Artificial Intelligence**, v. 13, n. 1-2, p. 133-170, 1980.
- WHITEHEAD, A. N.; RUSSELL, B. **Principia Mathematica**, 2. ed. Londres: Cambridge University Press, 1950.
- WHITLEY, L. D. (Ed.). **Foundations of Genetic Algorithms 2**. Los Altos, CA: Morgan Kaufmann, 1993.
- WHITTAKER, J. **Graphical Models in Applied Multivariate Statistics**. Nova York: John Wiley, 1990.
- WIDROW, B.; HOFF, M. E. Adaptive switching circuits. **1960 IRE WESTCON Convention Record**, p. 96-104. Nova York, 1960.
- WILKS, Y. A. **Grammar, Meaning and the Machine Analysis of Language**. Londres: Routledge & Kegan Paul, 1972.
- \_\_\_\_\_. Are ontologies distinctive enough for computations over knowledge? **IEEE Intelligent Systems**, v. 19, n. 1, p. 74-76, 2004.
- WILLIAMS, B. C.; NAYAK, P. P. Immobile robots: AI in the new millennium. **AI Magazine**, v. 17, n. 3, p. 17-34, 1996a.
- \_\_\_\_\_. A model-based approach to reactive self-configuring systems. **Proceedings of the AAAI-96**, p. 971-978. Cambridge, MA: MIT Press, 1996b.
- \_\_\_\_\_. A reactive planner for a model-based executive. **Proceedings of the International Joint Conference on Artificial Intelligence**, Cambridge, MA: MIT Press, 1997.
- WINOGRAD, T. **Understanding Natural Language**. Nova York: Academic Press, 1972.
- \_\_\_\_\_. A procedural model of language understanding. In: SCHANK, R. C.; COLBY, K. M. (Ed.). **Computer Models of Thought and Language**. San Francisco: Freeman, 1973.
- \_\_\_\_\_. **Language as a Cognitive Process: Syntax**. Reading, MA: Addison-Wesley, 1983.
- WINOGRAD, T.; FLORES, F. **Understanding Computers and Cognition**. Norwood, N. J.: Ablex, 1986.
- WINSTON, P. H. (Ed.). **The Psychology of Computer Vision**. Nova York: McGraw-Hill, 1975b.
- \_\_\_\_\_. **Artificial Intelligence**, 3. ed. Reading, MA: Addison-Wesley, 1992.
- WINSTON, P. H. et al. Learning physical descriptions from functional definitions, examples and precedents. **National Conference on Artificial Intelligence**, p. 433-439. Washington, D. C.: Morgan Kaufmann, 1983.
- WINSTON, P. H.; HORN, B. K. P. **LISP**. Reading, MA: Addison-Wesley, 1984.
- WIRTH, N. **Algorithms + Data Structures = Programs**. Engelwood Cliffs, NJ: Prentice-Hall, 1976.
- WITTGENSTEIN, L. **Philosophical Investigations**. Nova York: Macmillan, 1953.
- WOLPERT, D. H.; MACREADY, W. G. **The No Free Lunch Theorems for Search**, Technical Report SFI-TR-95-02-010. Santa Fe, NM: The Santa Fe Institute, 1995.
- WOLSTENCROFT, J. Restructuring, reminding and repair: What's missing from models of analogy. **AICOM**, v. 2, n. 2, p. 58-71, 1989.
- WOOLDRIDGE, M. Agent-based computing. **Interoperable Communication Networks**. v. 1, n. 1, p. 71-97, 1998.
- \_\_\_\_\_. **Reasoning about Rational Agents**, Cambridge, MA: MIT Press, 2000.
- WOOLDRIDGE, M.; DUNNE, P. E. On the computational complexity of coalitional resource games. **Artificial Intelligence**, v. 170, n. 10, p. 835-871, 2006.
- WOOLDRIDGE, M. et al. Logic for automated mechanism design - A progress report. **Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)**, Menlo Park, CA: AAAI Press, 2007.
- WOODS, W. 1985. What's in a Link: Foundations for Semantic Networks. In: BRACHMAN, R. J.; LEVESQUE, H. J. **Readings in Knowledge Representation**. Los Altos, CA: Morgan Kaufmann, 1985.

- WOS, L.; ROBINSON, G. A. Paramodulation and set of support. **Proceedings of the IRIA Symposium on Automatic Demonstration**, Versailles. Nova York: Springer-Verlag, p. 367-410, 1968.
- WOS, L. **Automated Reasoning, 33 Basic Research Problems**. NJ: Prentice Hall, 1988.
- \_\_\_\_\_. The field of automated reasoning. **Computers and Mathematics with Applications**, v. 29, n. 2, p. xi-xiv, 1995.
- WOS, L. et al. **Automated Reasoning: Introduction and Applications**. Englewood Cliffs, NJ: Prentice-Hall, 1984.
- XIANG, Y.; POOLE, D.; BEDDOES, M. Multiply sectioned Bayesian networks and junction forrests for large knowledge based systems. **Computational Intelligence**, v. 9, n. 2, p. 171-220, 1993.
- XIANG, Y.; OLESEN, K. G.; JENSEN, F. V. Practical issues in modeling large diagnostic systems with multiply sectioned Bayesian networks. **International Journal of Pattern Recognition and Artificial Intelligence**, v. 14, n. 1, p. 59-71, 2000.
- YAGER, R. R.; ZADEH, L. A. **Fuzzy Sets, Neural Networks and Soft Computing**. Nova York: Van Nostrand Reinhold, 1994.
- YEDIDIA, J.; FREEMAN, W.; WEISS, Y. Generalized belief propagation. **Advances in Neural Information Processing Systems (NIPS)**, v. 13, p. 689-695, 2000.
- ZADEH, L. Commonsense knowledge representation based on fuzzy logic. **Computer**, v. 16, p. 61- 65, 1983.
- ZURADA, J. M. **Introduction to Artificial Neural Systems**. Nova York: West Publishing, 1992.

---

## Listagens de URL:

- IIa: <<http://www.aaai.org/ojs/index.php/aimagazine/article/view/1904>>. Acesso em: 31 out. 2013.
- 7.3.3: <<http://sopha.projects.semwebcentral.org>>. Acesso em: 29 out. 2013.
- 13.4: <[www.cs.ubc.ca/~murphyk/Bayes/bnintro.html#appl](http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html#appl)>. Acesso em: 29 out. 2013.
- <[www.cs.ubc.ca/~murphyk/Bayes/bayesrule.html](http://www.cs.ubc.ca/~murphyk/Bayes/bayesrule.html)>. Acesso em: 29 out. 2013.
- <[www.autonlab.org/tutorials/bayesnet.html](http://www.autonlab.org/tutorials/bayesnet.html)>. Acesso em: 31 out. 2013.

# Índice remissivo

## A

AAAI 552  
Abdução 64, 279, 289, 291-294  
baseada em custo 291  
baseada em lógica 289, 279  
cobertura de conjunto 289  
seleção baseada em coerência 291  
ABSTRIPS 275, 482  
Ação duradoura 270  
ACT\* 169  
Admissibilidade 106, 121-122, 135, 561  
Agentes 13-16, 185-188, 221-225, 443, 449, 565, 537-538 (*ver também* computação emergente, arquitetura de subordinação)  
Agrupamento aglomerante 359  
Agrupamento conceitual 323, 359-361  
Álgebra booleana 9  
Álgebra do fator de certeza de Stanford (*ver teoria da certeza de Stanford*)  
Algoritmo A (*ver Algoritmo A\**)  
Algoritmo A\* 122-125, 135  
Algoritmo *bucket brigade* 429-433  
Algoritmo *forward-backward* (*ver programação dinâmica*)  
Algoritmos genéticos 23-24, 416, 421-429, 444 (*ver também* computação emergente)  
avaliação de aptidão 515-519  
caixeiro-viajante 423-426  
codificação de Gray 426  
definição 421-423  
inversão 423-426, 575  
mutação 24, 420, 422-426, 428, 429, 436

operadores genéticos (*ver recombração, mutação, inversão, recombração ordenada, permutação*)  
parallelismo implícito 424, 428  
permutação 436  
Problema da Satisfação da FNC (Forma Normal Conjuntiva) 423-426  
programação genética 429-438  
recombração 422-429, 575  
recombração ordenada 425-426  
sistemas classificadores 420, 429-438, 447  
subida de encosta 426-427  
vida artificial 438-447  
AM 23, 358-359  
Analizador de Earley 519-523  
análise de tabela 519-523  
memorização 519-522  
pares pontuados 519-522  
previsor 520-523  
varredor 520-523  
Analizador por rede de transição 523-526  
Analizador probabilístico livre de contexto 542  
Análise de baixo para cima 518-519  
Análise de meios e fins 478-479, 481-482  
Análise estocástica 537-543  
estrutural 543  
lexicalizada 543  
Análise n-grama 157  
Análise sintática 515, 517-523  
Earley 519-523  
estocástica 537-544

programação dinâmica 519-522  
tabela 519-522  
Analogia 23, 324, 355-358  
inferência 356  
mapeamento estrutural 356  
recuperação 356  
sistematicidade 356  
Analogia transformacional 258  
Aprendizado “o vencedor-leva-tudo” 376, 392-398  
Aprendizado baseado em explicação 323, 351-355, 359  
aprendizado acelerado 355  
aprendizado em nível de conhecimento 355  
critérios de operacionalização 352  
estrutura de explicação 353  
generalização 351  
regressão ao objetivo 353  
Aprendizado baseado em similaridade 355  
Aprendizado competitivo 376, 392-400  
Aprendizado de estrutura e busca 461-462  
Monte Carlo da Cadeia de Markov (MCCM) 462-463  
navalha de Occam 462  
NP-difícil 462  
sobreajuste 462  
Aprendizado de máquina 23, 319-473  
ABE, aprendizado baseado em explicação 351-355  
agrupamento aglomerante 359-360  
agrupamento conceitual 323, 359-360  
algoritmo genético 118, 330, 416, 420-429

- AM 23, 358-359  
 aprendizado baseado em similaridade 323, 350, 355  
 aprendizado competitivo 376, 392-400  
 aprendizado contrapropagação 376  
 aprendizado de conceito 322  
 aprendizado de estrutura 461-464  
 aprendizado de Grossberg 396-398  
 aprendizado de parâmetros 459, 461, 463  
 aprendizado do perceptron 376, 379-386  
 aprendizado em nível de conhecimento 355-358  
 aprendizado evolucionário (*ver* algoritmo genético)  
 aprendizado hebbiano 376, 400-409  
 aprendizado indutivo 322  
 aprendizado não supervisionado 323, 358-366, 558, 574  
 aprendizado o-vencedor-leva-tudo 376, 392-398  
 aprendizado PAC, Probably Approximately Correct 349-350  
 aprendizado por coincidência 400-408  
 aprendizado por diferença temporal 367, 369-370  
 aprendizado por gradiente descendente 384-385  
 aprendizado por reforço 467-469, 472, 366-371  
 aprendizado Q 371  
 aprendizado supervisionado 323, 330, 358, 393, 401, 404-405  
 árvores de decisão 339-346, 540-541, 550-551  
 atribuição de crédito 337-338 (*ver* também algoritmo *buckets brigade*, aprendizado por reforço)  
 BACON 359  
 BAM, memória associativa bidirecional 409-413  
 busca de específica a geral 332-335  
 busca no espaço de versão 328-339  
 C4.5 345, 549  
 CLUSTER/2 323, 347, 361-362, 394, 574  
 COBWEB 323, 361-366, 394  
 computação emergente 419, 443-447  
 critérios de operacionalidade 352  
 descoberta 358-359  
 dilema do empiricista 572, 574-575  
 e busca heurística 325  
 e representação do conhecimento 324-325  
 eliminação de candidatos 329-335  
 empacotamento 345-346  
 espaço de conceito 325, 328-329  
 especialização 211-214, 328  
 estocástica 450-451, 467-471  
 estrutura de aprendizado baseada em símbolo 323-328  
 estrutura de explicação 353  
 EURISKO 359  
 falhas próximas 326  
 fechamento dedutivo 355  
 formação de categoria (*ver* agrupamento conceitual)  
 generalização 322, 323, 325-335  
 heurísticas 324-325, 335-338  
 ID3 322, 339-346, 350, 372  
 indução 322  
 indução de árvore de decisão de cima para baixo (*ver* ID3)  
 induzindo heurísticas de busca 335-338  
 LEX 335-338, 347  
 máquinas de vetor de suporte 399-400  
 memória associativa 405-409 (*ver* também redes semânticas, grafos conceituais)  
 memória autoassociativa (*ver* memória associativa, redes de Hopfield)  
 memória heteroassociativa (*ver* memória associativa)  
 meta-DENDRAL 351-352  
 ocorrências negativas e supergeneralização 330-331  
 programação dinâmica 106-112, 135, 370, 457, 460, 519-523  
 raciocínio analógico 355-358  
 redes Bayesianas dinâmicas 460-461  
 redes conexionistas, 218, 319, 322, 377-379, 409, 416, 417, 419, 429, 561, 562, 564, 565  
 redes de Hopfield 377, 413-416  
 redes de Kohonen 394-396  
 redes neurais (*ver* redes conexionistas)  
 redes *outstar* (*ver* aprendizado de Grossberg)  
 reforço 345-346  
 regressão ao objetivo 353  
 seleção teórica da informação 342-344  
 sobre aprendizado (*ver* generalização)  
 subida de encosta 104-105, 106-112, 135, 327, 351, 365, 386, 390, 426-427, 561  
 taxonomia numérica 359  
 teoria do aprendizado 348-350  
 vetores de recursos 348  
 viés conjuntivo 348  
 viés indutivo 346-348, 376, 417  
 Aprendizado do conceito 5, 327  
 Aprendizado evolucionário (*ver* algoritmo genético)  
 Aprendizado hebbiano 376, 400-409  
 Aprendizado indutivo 322-323  
 Aprendizado não supervisionado 323, 358-366, 558, 574  
 Aprendizado perceptron 376, 379-386  
 Aprendizado por coincidência 400 (*ver* também aprendizado hebbiano)  
 Aprendizado por contrapropagação 376, 396  
 Aprendizado por diferença temporal 367, 369  
 Aprendizado por gradiente descendente 384-385 (*ver* também regra delta)  
 Aprendizado por parâmetros 459, 461, 463  
 Aprendizado por reforço 366-371, 467-471  
 aprendizado Q 371  
 definição 366  
 diferença temporal 367, 369-370  
 jogo da velha 367-370  
 método de Monte Carlo 370  
 programação dinâmica 370  
 usando o processo de decisão de Markov 468-471  
 Aprendizado Q 371  
 Aprendizado supervisionado 323, 330, 358, 393, 401, 404-405  
 Aprofundamento iterativo 88-89  
 Aridade 43-46  
 Arquitetura de quadro-negro 162, 181-183

- Arquitetura de subsunção 187-188, 216-218, 443  
 Árvore de junção 308-309  
 Árvores de decisão 339-348, 362, 540-541, 550  
     na análise de linguagem natural 540-541  
 Atrator 376-377  
 Atribuição de crédito 337 (*ver também* algoritmo *bucket brigade*, aprendizado por reforço)  
 Autômatos celulares 439-447  
 Axônio 24-25
- B**
- Bacia de atração 376-377, 408  
 BACON 359  
 BAM, (*ver* memória associativa bidirecional)  
 Base de conhecimento 232-236  
 Baum-Welch 460  
 Burton 269  
 Busca baseada em recursão 162-166  
 Busca do específico ao geral 330-335  
 Busca em amplitude 83-89  
 Busca em espaço de estados 17, 34, 66, 69-101  
     admissibilidade 106, 121-122, 135, 561  
     Algoritmo A\* 122-125, 135  
     aprendizado de estrutura 461-464  
     busca guiada por objetivo 78-80, 176-178, 239-242  
     busca em amplitude 83-89  
     busca em feixe 134, 338  
     busca em profundidade 80, 82-88  
     busca em profundidade com aprofundamento iterativo 88-89  
     busca exaustiva 76  
     busca guiada por dados 78-80, 176-177, 244-247  
     busca guiada por padrões 164-166  
     busca não informada 89  
     busca oportunista 245  
     busca pela melhor escolha 112-121  
     busca recursiva 162-164  
     caminho mais curto 115  
     caminho-solução 85  
     e cálculo proposicional 89-100  
     e planejamento 262-274  
     encadeamento progressivo (*ver* busca guiada por dados)
- encadeamento regressivo (*ver* busca dirigida por objetivo)  
 estado 34  
 fator de ramificação 80, 132-133  
 grafos E/OU 90-93, 134, 240-242, 572  
 grau de informação 121-125, 134-135, 561  
 implementação 80-82  
 minimax 106-107, 125-131, 135  
 monotonicidade 106, 121-125, 561  
 poda alfa-beta 106, 131-132, 135  
 ramificar e limitar 76  
 resposta a perguntas 508-509  
 retrocesso 80-83, 92, 161, 167, 283  
 subida de encosta 104-105, 106-112, 135, 327, 351, 365, 386, 390, 426-427, 561  
 subobjetivo 78
- Busca em feixe 134, 338  
 Busca em grafo (*ver* busca em espaço de estados)  
 Busca em profundidade 80, 82-88  
 Busca guiada por dados 78-80, 176-177, 244-247  
 Busca guiada por objetivo 78-80, 176-178, 239-242  
 Busca no espaço de versões 328-338  
 Busca oportunista 245  
 Busca pela melhor escolha 89, 104, 105, 112-121, 135
- C**
- C4.5 345, 549  
 Caixearo-viajante 423-426  
 Cálculo de predicados 9-10, 32-33, 38-65, 190  
     cláusula de Horn 478, 499, 500-503  
     completude 52  
     conjunção 39  
     consistência 38  
     constante 43  
     convertendo para forma de cláusula 477-478  
     disjunção 39  
     e eliminação 54  
     e introdução 54  
     e planejamento 262-269  
     equivalência 39  
     exemplo de função 43  
     forma de cláusula 482, 485-488  
     forma normal conjuntiva 422-423  
     forma normal prenex 486  
     função 43  
     hipótese de mundo fechado 280, 498, 503  
     implicação 39  
     inconsistente 52-53  
     inferência 52  
     insatisfação 53  
     interpretação 40  
     modelo 52-53  
     *modus ponens* 54  
     *modus tollens* 54  
     negação 46  
     percurso do cavalo 170, 172-175, 244  
     predicado 45  
     procedimentos de prova 53  
     procurando no espaço de inferência 90, 165, 181  
     quantificação 48  
     quantificador existencial 49  
     quantificador universal 49  
     regras de inferência 52-55  
     resolução 52  
     satisfação 52  
     semântica 52  
     semântica declarativa 179  
     semântica procedural 175, 179  
     sentença atômica 45  
     sentenças 45-46  
     ícone verdade 44  
     ícones 42-44  
     ícones impróprios 43  
     skolemização 55, 485-486, 498  
     ícone 44  
     validade 52-53  
     valor verdade 43  
     variável 44
- Cálculo de predicados de primeira ordem (*ver* cálculo de predicados)  
 Cálculo Estocástico Lambda 315  
 Cálculo proposicional 38-42, 89-90  
     conjunção 39  
     disjunção 39  
     equivalência 39  
     fórmula bem formada 39  
     implicação 39  
     interpretação 40  
     negação 39  
     proposições 39  
     semântica 40-41

sentença 39  
 símbolo verdade 40  
 símbolos 38-40  
 sintaxe 42-47  
 tabela verdade 41  
**C**andidate 543  
**CASEY** 255  
 Categorias de nível básico 362  
 Ciclo reconhece-atua 167, 180, 233, 239  
 Ciéncia cognitiva 181, 183, 227, 350, 417, 478, 558, 564, 568, 570  
 Ciéncia de sistemas inteligentes 558, 577  
 Circunscrição 287-289  
 Classificação de distância mínima 381-382, 574  
 Cláusula de Horn 478, 499, 500-503  
 Cláusula sem cabeça 605 (*ver também* cláusula de Horn)  
**CLIPS** 169, 175, 234, 274  
**CLUSTER/2** 323, 347, 361-362, 394, 574  
**COBWEB** 323, 361-366, 394  
 Codificação de Gray 426  
 Combinações 141-142  
 Completa em relação à refutação 483, 485  
 Completeness 89, 485, 494, 507, 508, 510, 560  
 Complexidade de busca 132-135  
 Compreensão de histórias 544-545  
 Compreensão de linguagem natural 20-21, 474-519  
     algoritmo de Viterbi 539  
     analisador de rede de transição estendida 529-533  
     analisador livre de contexto 505  
     analisadores de rede de transição 529-537  
     análise 529-537, 541-543  
     análise de baixo para cima 518-519  
     análise n-grama 157  
     aplicações 509-513  
     combinando sintaxe e semântica 534-537  
     conhecimento do mundo 515  
     decodificação (*ver* reconhecimento de fonema)  
     e árvores de decisão 540-541  
     estrutura profunda 551  
     extração de informações 548-550

ferramentas estocásticas 537-544  
 fonema 155-156, 390, 475  
 fonologia 515  
 geração 480, 515  
 gramática 526-529 (*ver também* análise, sintaxe)  
 gramáticas de caso 551  
 gramáticas de casos e gramáticas funcionais 513  
 gramáticas de estrutura de frase expandidas 529  
 gramáticas de ligações 504, 543  
 gramáticas livres de contexto 517-519  
 gramáticas semânticas 551  
 gramáticas sensíveis ao contexto 525-529  
 gramáticas transformacionais 513  
 hierarquia de Chomsky 526-529  
**ID3** 540-541, 550  
 interface de bancos de dados 545-548  
 marcadores gramaticais 513  
 modelos de Markov 539-540  
 morfologia 515  
 pragmática 515  
 quadro de caso 534-536  
 reconhecimento de fonema 156  
 semântica 515  
 sintaxe 515, 625-639  
**C**omputação emergente 419, 443-447 (*ver também* agentes)  
     agentes teleorreativos 269-271, 443  
     algoritmos genéticos 23-24, 416, 421-429, 444  
     aprendizado social 438-447  
     arquitetura de subsunção 187-188, 216-218, 443  
     autômatos celulares 439-447  
     Jogo da Vida 420, 439-442  
     máquina de estados finitos 71-72, 439  
     programação evolucionária 442-444  
     programação genética 322, 420, 429-438  
     Santa Fe Institute 443  
     sistemas classificadores 420, 429-438  
     vida artificial 14, 23-24, 377, 420, 438-447

Computer Professionals for Social Responsibility 581  
 Condicionamento baseado em silabas 550-551

Conhecimento do mundo 515  
 Conjunção de disjunções 485  
 Conjuntos mínimos 286, 290-291

Consciéncia 566-567  
 Consisténcia 52, 510  
 Consultor financeiro 60-64, 96-97, 120, 175

Contagem 140-142  
 Controle de busca por meio da resolução de conflitos 179  
     atualidade 179  
     especificidade 179  
     refração 179

Copycat, arquitetura 216, 218-220  
**Corpus** de Brown 156  
**Corpus** de Switchboard 156  
**Crosstalk** 408

## D

Decodificação (*ver* reconhecimento de fonema)  
 Decomposição hierárquica de problemas 22

Dedução natural 478, 509  
 Demodulação 507-509  
**DENDRAL** 19, 80  
 Dendrito 23-24

Diagnóstico automotivo 35-36, 243-244, 278

Diagramas de dependência conceitual 196-201

Diferença de edição mínima (*ver* programação dinâmica)

Dimensão de Vapnik Chervonenkis 399  
**Dipmeter Advisor** 19, 80

Distância de Hamming 406-408

Distância de Levenshtein (*ver* programação dinâmica)

d-separação 305-309

**DYNA-Q** 366

## E

**EBL** (*ver* aprendizado baseado em explicação)

Editor da base de conhecimento 234

Excitatório 563-564

Eliminação 54-56

Eliminação de candidatos 329-335

Eliminação do E 54-55

Empacotamento 345-346  
 Encadeamento progressivo (*ver busca guiada por dados*)  
 Encadeamento regressivo (*ver busca dirigida por objetivo*)  
 Engenharia do conhecimento 234-236  
 Epistemologia 558  
 Espaço de conceito 328  
 Especialista no domínio 234-240  
 Esquema 428-429  
 Estimativa de modo 272-273  
 Estratégia da forma de entrada linear 562-563  
 Estratégia da preferência por unidade 494  
 Estratégia de busca em amplitude 493  
 Estratégia de conjunto de suporte 493  
 Estrutura de aprendizado simbólico 323-328  
 Extração de informações 548-551  
 Extração de resposta 489, 496-499, 503

**F**

Fator de ramificação 80, 132-133  
 Ferramentas estocásticas para análise de linguagem 537-543, 550  
 Fonema 155-156, 390, 475  
 Fonologia 515  
 Forma de cláusula 476, 503, 507  
 Forma normal prenex 486  
 Formação de categoria (*ver aglomeração conceitual*)  
 Fórmula bem formada 39  
 Função de aptidão 420-421  
 Função de densidade de probabilidade 312  
 Função de energia da rede 377, 409-416  
 Função sigmoide 387  
 Fundamentação 561-564

**G**

Generalização 150, 183, 211-214, 309, 320, 322-323, 325-329, 351, 572  
 GOFAI 17  
 Grafos conceituais 207-216, 227-228, 529, 532, 535, 546, 552, 559  
 conceito genérico 207  
 conceitos 207-208  
 copiar 211  
 e cálculo de predicados 215  
 e lógica modal 214

especialização 211-214  
 generalização 211-214  
 herança 210-216  
 hierarquia de tipo 210  
 indivíduos 208-210  
 marcador 209  
 nomes 208-210  
 nós proposicionais 214  
 quantificação 214-215  
 quantificação existencial 215  
 quantificação universal 215  
 referente 209  
 regras de formação canônicas 213  
 relações conceituais 207  
 restringir 211-213  
 reticulado de tipo 210-212  
 simplificar 211-212  
 subtipo 210  
 supertipo 210  
 tipo 210  
 tipo universal 210  
 tipos 208-211  
 Grafos E/OU 90-93, 134, 240-242, 572  
 Gramática de lógica estendida 529  
 Gramáticas de caso 551-552  
 Gramáticas de estrutura de frase estendidas 529  
 Gramáticas livres de contexto 517-519, 527, 529  
 Gramáticas sensíveis ao contexto 525-529  
 Grau de informação 121-125, 134-135, 561  
**H**  
 HEARSAY 181-183  
 Heurísticas 17-18, 22, 25, 36-37, 74, 76-77, 89, 100, 103-137, 185-186, 225, 231, 247-248, 257-258, 279, 282, 492, 495, 499, 504, 506, 509  
 admissibilidade 106, 121-125  
 Algoritmo A\* 122-125, 135, 262, 427  
 análise de meios e fins 478-479, 481-482  
 busca heurística 103-137 (*ver também busca pela melhor escolha*)  
 busca pela melhor escolha 112-121  
 efeito de horizonte 128  
 fator de ramificação 132-133  
 jogos 125-132

minimax 106-107, 125-131, 135  
 MOM acoplado 457  
 monotonicidade 106, 121-125, 561  
 poda alfa-beta 106, 131-132, 135

Hierarquia de Chomsky 526-529

Hipergrafos (*ver grafos E/OU*)

Hiper-resolução 506-507

Hipótese de Church/Turing 579

Hipótese de mundo fechado 280, 498, 503

Hipótese de sistema físico 25, 559-562

**I**

ID3 23, 322, 339-346, 350, 372  
 avaliação de desempenho 344-346  
 e subida de encosta 347  
 empacotamento 345-346  
 reforço 345-346  
 teoria da informação 343-345

Inconsistente 52

Inferência lógica 52-55

Informalidade de comportamento 12

Inibitorias 525

Insatisfação 492

Instanciação universal 54

Inteligência artificial, definição 1-2, 25-26, 559

INTERNIST 19

Introdução do E 54-55

Inversão 423-426, 575

**J**

Java 485

JESS 169, 175, 183, 234  
 Jogo da velha 34-35, 69, 73-74, 89, 104-106, 367-370

Jogo da Vida 439-442

Jogos 17, 34-37, 104, 125-132, 135, 570  
 antecipação de n movimentos 127-128

camada 127-130

efeito de horizonte 128

heurísticas 125-132

jogo da velha 34-35, 69, 73-74, 89, 104-106, 367-370

jogo de damas 17, 107, 128, 255

minimax 106-107, 125-131, 135

minimax com nível de

aprofundamento fixo

nim 126-127

poda alfa-beta 106, 131-133

- programa de jogo de damas de Samuel 107-108, 371, 561  
 quebra-cabeça dos 15 17, 74  
 quebra-cabeça dos 8 74-75, 85-86, 88, 107, 114-120, 122, 124, 170, 244  
 xadrez 34-36, 87-88, 124, 128, 132, 161, 169-173, 355, 400, 468-469, 560
- K**  
 Kernel 269
- L**  
 Lei associativa 41  
 Lei comutativa 41  
 Lei contrapositiva 41  
 Lei de de Morgan 41  
 Lei distributiva 41  
 LEX 335-338, 347  
 Léxico 479  
 Lisp 22, 30  
 Livingstone 252-253, 274  
*Logic Theorist* (Teórico Lógico) 41, 185, 478, 532  
 Lógica autoepistêmica 282  
 Lógica cíclica 315, 465-466  
     campo aleatório de Markov 464-467  
     nós de agrupamento 465  
     nós de variáveis 465  
 Lógica nebulosa (*ver* teoria de conjuntos nebulosa)  
 Lógica padrão 282-283  
 LOGO 235  
 LT (*ver Logic Theorist*)
- M**  
 Macro-operadores 266-269  
 MACSYMA 93  
 Manutenção da verdade baseada em suposições 283-286, 290  
 Máquina de estado finito probabilística 155-157, 310  
 Máquina de estados finitos 71-72, 155, 439  
 Máquina de Moore (*ver* aceitador de estado finito)  
 Máquina de Turing 442  
 Máquinas de vetor de suporte 399-400  
 Matriz associativa nebulosa 295  
 Maximização de expectativa (ME) 459-461  
 Baum-Welch 463-464  
 estimativa de máxima verossimilhança 463  
 Maximização de expectativas 463-465  
     Baum-Welch 460  
 Memória associativa 405-409 (*ver também* redes semânticas, grafos conceituais)  
 Memória associativa bidirecional 409-413  
 Memória autoassociativa (*ver* memória associativa, redes de Hopfield)  
 Memória de trabalho 167  
 Memória heteroassociativa (*ver* memória associativa)  
 Meta-DENDRAL 351-352  
 Metáfora 512  
 Método de Monte Carlo 370  
 Método do centroide 297  
 mgu (*ver* unificador mais geral)  
 Mineração de dados 512  
 Minimax 106-107, 125-131, 135  
 Modelagem de desempenho humano 21  
 Modelo 52-53  
 Modelo de Markov de primeira ordem 310-312  
 Modelo de Markov observável 311-313  
 Modelo de McCulloch-Pitts 378-379  
 Modelo gráfico (*ver* rede Bayesiana de crença, modelo de Markov)  
 Modelo gráfico direcionado (*ver* redes Bayesianas de crença, modelo de Markov)  
 Modelo oculto de Markov 451-459  
     autorregressivo 453-455  
     definição 451-452  
     modelo oculto semi-Markov 457  
     modelos ocultos de Markov 455  
     modelos ocultos de Markov fatoriais 455  
     MOM de memória mista 457  
     MOMs de n-gramas 456-457  
     MOMs hierárquicos 456  
 Modelo oculto de Markov de memória mista 457  
 Modelo oculto semi-Markov 457  
 Modelos conceituais 236-239  
 Modelos de Markov 306-315, 451-452, 537-539  
     cadeia de Markov 310-312, 451, 456, 514, 516  
     campo aleatório de Markov 303, 315, 464-467  
     e análise de linguagem natural 536, 540  
     máquina de estado de Markov 309-313  
     máquina de estado finito probabilística 154-155  
     modelo de Markov de primeira ordem 310-312, 450, 455, 460  
     modelo oculto de Markov 313, 451-459, 463, 469  
     modelos semi-Markov 457  
     Monte Carlo da Cadeia de Markov (MCCM) 459, 462  
     observáveis 311-312  
     processo de decisão de Markov (PDM) 460, 468-472  
     processo discreto de Markov 303, 310-311  
     reconhecedor de estado finito probabilístico 155  
     redes lógicas de Markov 315  
     suposições de Markov 539  
 Modelos de Markov ocultos autorregressivos 453-455  
 Modelos ocultos de Markov 451-459  
 Modelos ocultos de Markov n-grama 456-457  
     bigrama 456  
     trígrama 456  
 Modelos ocultos hierárquicos de Markov 456  
     estado de produção 456  
     estado interno 456  
 Modelos relacionais probabilísticos (MRPs) 314  
 Modelos semi-Markov 457  
*Modus ponens* 52-54  
*Modus tollens* 54  
 Monotonicidade 106, 121-125, 561  
 Morfologia 515  
 Motor analítico 8-9, 16  
 Mundo de blocos 20, 23, 32, 262-267, 514  
 MYCIN 19, 222, 249, 274-275, 291, 293  
**N**  
 Navalha de Occam 291, 340, 462  
 Negação como falha 282  
 NETtalk 390-391  
 Neurociência cognitiva 563, 569

Neurônio 23-24, 375-378, 387, 415, 563  
 Neurotransmissores 563  
 Nim 126-127  
 Nível de ativação 377  
 Números de ponto flutuante 31

**O**

Objeção de Lady Lovelace 12  
 Ontologias 216-221  
 Operadores genéticos (*ver* recombração, mutação, inversão, recombração ordenada, permutação)  
 OPS 169, 179, 183  
 Ortonormalidade de vetores 407  
*Outstar* 396-398  
 OWL 220

**P**

PAC, aprendizado aproximadamente correto 349-350  
 Paramodulação 506-508  
 Percurso do cavalo 170, 172-175, 244  
 Permutações 141-142  
 Planejamento 21-22, 262-274  
 Planejamento teleorreativo 269-271, 443  
 Planejando macros 266-269  
 Plausibilidade de uma proposição 298  
 Plausibilidade neural 563  
 Poda alfa-beta 106, 131-132, 135  
 Potencial de ação 525  
 Pragmáticas 515  
 Problema das pontes de Königsberg 8, 67-68  
 Problema de tabuleiro truncado 560-561  
 Problema do percurso do cavalo 170-174  
 Problema mente-corpo 6  
 Procedimentos de prova 53  
 Processamento paralelo distribuído (*ver* redes conexionistas)  
 Processo de decisão de Markov parcialmente observável (PDMPO) 469-470  
 Processo de Markov discreto 310-313  
 Programação dinâmica 106-112, 135, 370, 457, 460, 519-523  
 Programação evolucionária 442-444  
 Programação genética 322, 420, 429-438 (*ver também* algoritmo genético)

Programação lógica 499-503  
 Programação orientada a objeto 22  
 Prolog 17, 22, 449-503  
 Prosódia 515  
 PROSPECTOR 19, 80, 153, 159, 291, 568  
 PROTOS 255  
 Prova de teorema (*ver* raciocínio automatizado)  
 Prova de teoremas por resolução 483-499

**Q**

Quadro de caso 534-536, 552  
 Quadros 204-207  
 Quebra-cabeça dos 15 17, 74  
 Quebra-cabeça dos 8 74-75, 85-86, 88, 107, 114-120, 122, 124, 170, 244

**R**

Raciocinador por crenças múltiplas 316  
 Raciocínio analógico 258, 355-358, 562  
 Raciocínio automatizado 17, 54, 477-511  
 abordagens baseadas em regra 504-506  
 análise de meios e fins 478-479, 481-482  
 cláusula de Horn 478, 499, 500-503  
 cláusula sem cabeça 500 (*ver* também cláusulas de Horn)  
 completude 485, 487, 495, 507, 509  
 completude de refutação 483, 492  
 conjunção de disjunções 485  
 consistência 52, 510  
 convertendo para forma de cláusula 507  
 dedução natural 478, 509  
 demodulação 507-508  
 e cálculo de predicado 488-490  
 e PROLOG 499-503  
 estratégia da preferência por unidade 494, 499  
 estratégia de busca em amplitude 493  
 estratégia de forma de entrada linear 503  
 estratégia do conjunto de suporte 478, 493

extração de respostas 489, 496-498, 503  
 fatoração 489  
 forma de cláusula 482, 485-488  
 forma normal conjuntiva 422-423  
 forma normal prenex 486  
 heurísticas 477-479, 481, 492, 499, 504  
 hiper-resolução 489, 506-507  
 literal 483  
 LT, *Logic Theorist* 41, 185, 478, 532  
 métodos fracos 185-186, 504  
 paramodulação 506-508  
 refutação 478, 483, 487, 492-499  
 refutação de resolução 483-499  
 relação de regra 500 (*ver também* cláusula de Horn)  
 resolução 52, 483-499  
 resolução binária 483, 487-488, 506-507  
 resolução unitária 494  
 Resolvedor de Problema Geral 35, 169, 179, 478  
 skolemização 486  
 subsunção 509  
 tabelas de diferença 478-483  
 teorema de Herbrand 510  
 Raciocínio baseado em casos 200, 248, 254-260  
 adaptação de caso 255  
 recuperação de caso 256-257  
 Raciocínio baseado em modelo 248-260, 275  
 Raciocínio baseado em regra 239-248, 259  
 Raciocínio Bayesiano 152-154, 302-315, 450-472  
 aprendizado 450-472  
 árvore de junção 308-309  
 árvores de grupos 303  
 d-separação 305-307  
 modelos relacionais probabilísticos (MRPs) 314-315  
 passagem de mensagem 309-310  
 programas de lógica Bayesiana 314  
 RBD, rede Bayesiana dinâmica 303, 305, 309-310, 459  
 redes Bayesiana de crença 303-305, 315, 450, 460-467  
 teorema de Bayes 151-154  
 triangulação 308-309

- Raciocínio de senso comum 16, 52, 190, 213, 234, 280, 288, 514
- Raciocínio estocástico 138-139, 302-315 (*ver também* raciocínio Bayesiano, modelos de Markov) aplicações 145-158 aprendizado 450-451 e incerteza 302-315 exemplo de estrada/tráfego 157-158 inferência 145-147
- Raciocínio guiado por padrões 164-166
- Raciocínio não monotônico 280-283, 316 circunscrição 287-289 hipótese de mundo fechado 287-289, 499, 569 lógica autoepistêmica 282 lógica padrão 282-283 modelos mínimos 280, 287-289 operadores modais 281, 287-289 sistema de manutenção da verdade 283-287
- Raciocínio nebuloso (*ver* teoria de conjuntos nebulosa)
- Raciocínio probabilístico (*ver* raciocínio estocástico, raciocínio Bayesiano, modelos de Markov)
- Ramificação e poda 76
- RBC (*ver* raciocínio baseado em caso)
- Recombinação 422-426, 428, 431-438, 575
- Recombinação ordenada 425
- Reconfiguração de modo 273
- Reconhecedor de estado finito probabilístico 155-157
- Reconhecedor de estados finitos 72, 155
- Reconhecimento de fonema 156
- Rede auto-organizável 395
- Rede Bayesiana dinâmica, RBD, 305, 309-310, 461, 472 aprendizado 467-471
- Rede de Kohonen 394-396
- Rede de transição estendida 529-537
- Rede do associador linear 405-409
- Rede multicamadas 386-387
- Redes atratoras 376-377, 409-417
- Redes conexionistas 7, 218, 319, 322, 377-379, 409, 416, 417, 419, 429, 561, 562, 564, 565, 575 (*ver também* aprendizado de máquina) aprendizado competitivo 376, 392-400
- aprendizado de Grossberg 396-398
- aprendizado do perceptron 379-386
- aprendizado hebbiano 400-408
- aprendizado o vencedor-leva-tudo 376, 392-398
- aprendizado por contrapropagação 396
- atrator 376
- BAM, Bidirectional Associative Memory (memória associativa bidirecional) 409-413
- classificação 379-382
- função de limiar 377
- histórico inicial 377-379
- máquinas de vetor de suporte 399-400
- memória associativa 377, 405-408 (*ver também* redes semânticas, grafos conceituais)
- memória autoassociativa (*ver* memória associativa, redes de Hopfield)
- memória heteroassociativa (*ver* memória associativa)
- memória interpolativa 405-406
- modelo de McCulloch-Pitts 378-379
- NETtalk 390-391
- NETtalk e ID3 391
- neurônio 23-24, 375-378, 387, 415, 563
- nível de ativação 377
- rede associadora linear 405-409
- rede auto-organizável 395
- rede de Kohonen 394-396
- redes de atratores 377, 409-416
- redes de feedback 409
- redes de Hopfield 377, 413-416
- regra delta 376, 381, 384-386
- retropropagação 386-392
- regressão e OU-exclusivo 391-392
- topologia de rede 378
- Redes de feedback 409
- Redes de Hopfield 377, 413-416
- Redes neurais (*ver* redes conexionistas)
- Redes semânticas 7, 21, 32-33, 191-206
- Redução de diferença 479-482
- Reforço 345-346
- Regra delta 376, 381, 384-388
- Regressão ao objetivo 353
- Reinício aleatório 463
- Relação de regra 500 (*ver também* cláusula de Horn)
- Representação associacionista 190-192, 227
- Representação do conhecimento 187, 189-230 (*ver também* cálculo de predicados, grafos conceituais) eficiência 226
- esquema 428-429
- exaustividade 226
- herança 210-216
- lógicas de ordem mais elevada 316
- lógicas multivaloradas 315
- lógicas modais 315
- lógicas temporais 316
- ontologias 216-217, 220
- padronização de relacionamentos de rede 195-200
- problema de enquadramento 226, 262-263
- quadros 204-207
- redes semânticas 7, 21, 32-33, 191-206
- representação associacionista 190-192, 227
- representação extensional 360
- representação intencional 360
- roteiros 200-204
- Representações uniformes para soluções por método fraco 504
- Resolução binária 482, 486-492
- Resolução por refutação 483-492
- colisão 552
- completude 483, 482
- completude da refutação 483, 482
- consistência 52, 510
- convertendo para a forma clausular 485-487
- demodulação 507-508
- e programação lógica 499-503
- e PROLOG 499-503
- estratégia de forma de entrada linear 494-495
- estratégia de preferência por unidade 494
- estratégia do conjunto de suporte 493
- estratégia em amplitude 493
- extração de resposta 489, 496-498
- fatoração 489
- forma clausular 483-487
- forma normal prenex 486
- heurísticas 477-479

- hiper-resolução 489, 506-507  
 literal 483  
 paramodulação 506-508  
 refutação 478, 483, 487, 492-499  
 resolução 52, 483-499  
 resolução binária 483, 487-488, 506-507  
 resolução unitária 494  
 subsunção 509
- Resolvedor Geral de Problemas 478-482  
 análise de meios e fins 186, 478, 481, 547  
 tabelas de diferença 478-483
- Resposta a perguntas 508-509
- Resulta logicamente 52
- Resumo de textos 548-550
- RETE 248, 256
- Retrocesso 80-83, 92, 161, 167, 283
- Retrocesso cronológico 283
- Retrocesso guiado por dependência 283
- Retropropagação 376, 386-392
- Retropropagação e OU-exclusivo 391-392
- RGP (*ver* Resolvedor Geral de Problemas)
- Robótica 21-22
- Roteiros 200-204
- S**
- Sala chinesa 560
- Satisfação 52
- Satisfação da FNC 423-426
- Satisfação de uma FNC 423-426
- SCHEME 578
- Seleção proporcional à aptidão 421
- Semântica 515
- Semântica declarativa 314
- Separabilidade linear 349-380, 381-382
- Serviços de conhecimento 220-221
- SHRDLU 20-21, 514
- Simplicidade de conjunto 291
- Sinapse 24 (*ver também* aprendizado conexionista)
- Sintaxe 515, 625-639
- Sistema de manutenção baseado em justificativa 284-285
- Sistema de manutenção da verdade baseado em lógica 283-286, 290
- Sistema especialista 23, 27, 231-234, 237-240, 248-249
- ambiente de sistema especialista 233-234  
 explicação 231-232  
 fatores de certeza 568  
 modelos conceituais 234, 236-239  
 programação exploratória 235, 238  
 raciocínio baseado em modelo 248-254, 260-261  
 separação de conhecimento e controle 180, 233, 259  
 sistemas especialistas baseados em regras 239-248
- Sistemas baseados em agente 221-225, 565  
 definição 222
- Sistemas classificadores 420, 429-433
- Sistemas de manutenção da verdade 283-287  
 manutenção da verdade baseada em justificativa 284  
 manutenção da verdade baseada em lógica 284, 287  
 manutenção da verdade baseada em suposições 283-286, 290  
 raciocinador de múltiplas crenças 287  
 retrocesso cronológico 283  
 retrocesso guiado por dependência 283
- Sistemas de produção 167-181, 578  
 arquitetura de quadro-negro 181-183  
 ciclo reconhece-atua 167  
 controle 175-179  
 quebra-cabeça dos 8 170  
 memória de trabalho 167  
 problema do percurso do cavalo 172-175  
 regra de produção 167  
 resolução de conflito 167, 173-174  
 vantagens 179-181
- Sistemas especialistas baseados em regra 239-248
- Sistemas lineares dinâmicos 460
- Skolemização 486
- SOAR 169
- Sobre aprendizado (*ver* generalização)
- SOFA 220
- Solução de problema distribuída (*ver* agentes)
- Solução de problema incorporado (*ver* agentes)
- Solução de problemas baseada em agente (*ver* sistemas baseados em agente)
- Solução de problemas com uso intenso de conhecimento (*ver* solução de problemas pelo método forte)
- Solução de problemas pelo método forte 231-277
- Solução de problemas pelo método fraco 186
- STRIPS 222, 264, 266-268
- Subida de encosta 104-105, 106-112, 135, 327, 351, 365, 386, 390, 426-427, 561
- Substituição de Monte Carlo 421
- Subsunção 509
- T**
- Tabelas de diferença 479-483
- Tabelas triangulares 266-269
- Taxonomia numérica 359
- Têmpera simulada 413, 463
- Tentativas de Bernoulli 147, 452
- Teoria da certeza de Stanford 188, 292-294
- Teoria da evidência de Dempster-Shafer 298-302
- Teoria da informação 342-344
- Teoria da probabilidade 142-151 (*ver também* raciocínio Bayesiano, modelos de Markov)  
 aprendizado 450-459  
 contagem 140-142  
 definição 143  
 esperança de um evento 148  
 evento independente 144-145, 151  
 eventos 143-145  
 máquina de estado finito  
 probabilística 155-157  
 probabilidade condicional 149-151  
 probabilidade *a posteriori* 149  
 probabilidade *a priori* 149  
 reconhecedor de estado finito  
 probabilístico 155-157  
 rede Bayesiana de crença 303-305  
 teorema de Bayes 151-154  
 variável aleatória 147-148
- Teoria da semelhança familiar 362
- Teoria de conjuntos nebulosos 294-298,
- Teoria do aprendizado 348-350
- Teoria dos grafos 8, 67  
 ancestral 69-70

- arco 67-70  
árvore 70  
caminho 69  
caminho Euleriano 68  
caminho hamiltoniano 101  
ciclo 69-70  
descendente 69-70  
elo 67  
filho 69  
genitor 69  
grafo direcionado 69  
grafo radicado 69  
grafo rotulado 69  
irmão 69  
nó 67-69  
nó folha 70  
nós conectados 70  
Teorias associacionistas 190-192, 227
- Teste de Turing 11  
Texto para fala (text-to-speech) 513  
Tradição empirista 7-8  
Tradição racionalista 7-8, 13-14, 556  
Transformada rápida de Fourier 454-455  
Triangulação 308
- U**  
Unificação 55-61  
Unificador mais geral 56  
Utilidade de categoria 363-365
- V**  
Validade 52-53  
Verificação de ocorrência 56, 499, 503  
Vida artificial 14, 322, 419-420, 438-447, 556 (*ver também* computação emergente)
- Viés induutivo 320, 322-323, 338, 346-347, 349-350, 376, 417, 558, 564, 572, 573-574  
Viés sintático 348  
Viterbi 457-459, 501 (*ver também* programação dinâmica)  
Vizinho mais próximo 76
- W**  
Web semântica 544, 548-550
- X**  
Xadrez 17, 34-36, 87-88, 124, 128, 132, 161, 169-173, 355, 400, 468-469, 560  
XCON 19, 169



George F. Luger

# Inteligência Artificial

*6<sup>a</sup> edição*

Com uma abordagem clara e objetiva, George F. Luger reúne nesta sexta edição as práticas mais recentes do mercado e os tópicos importantes relacionados à inteligência artificial, como computação emergente, representação da incerteza e processamento probabilístico de linguagem natural, entre outros.

Muito rica em exemplos práticos, esta obra é ideal para estudantes de ciência da computação, engenharia da computação e sistemas de informação, pois apresenta equilíbrio entre teoria e prática e os conceitos fundamentais da área, com detalhes e informações necessárias para a implementação dos algoritmos e estratégias discutidos no livro.



[sv.pearson.com.br](http://sv.pearson.com.br)

A Sala Virtual oferece, para professores, apresentações em PowerPoint (em português) e manual de soluções (em inglês); para estudantes, manual complementar de Lisp, Prolog e Java.



Este livro também está disponível para compra em formato e-book.  
Para adquiri-lo, acesse nosso site.

[www.pearson.com.br](http://www.pearson.com.br)

ISBN 978-85-8143-550-3

9 788581 435503