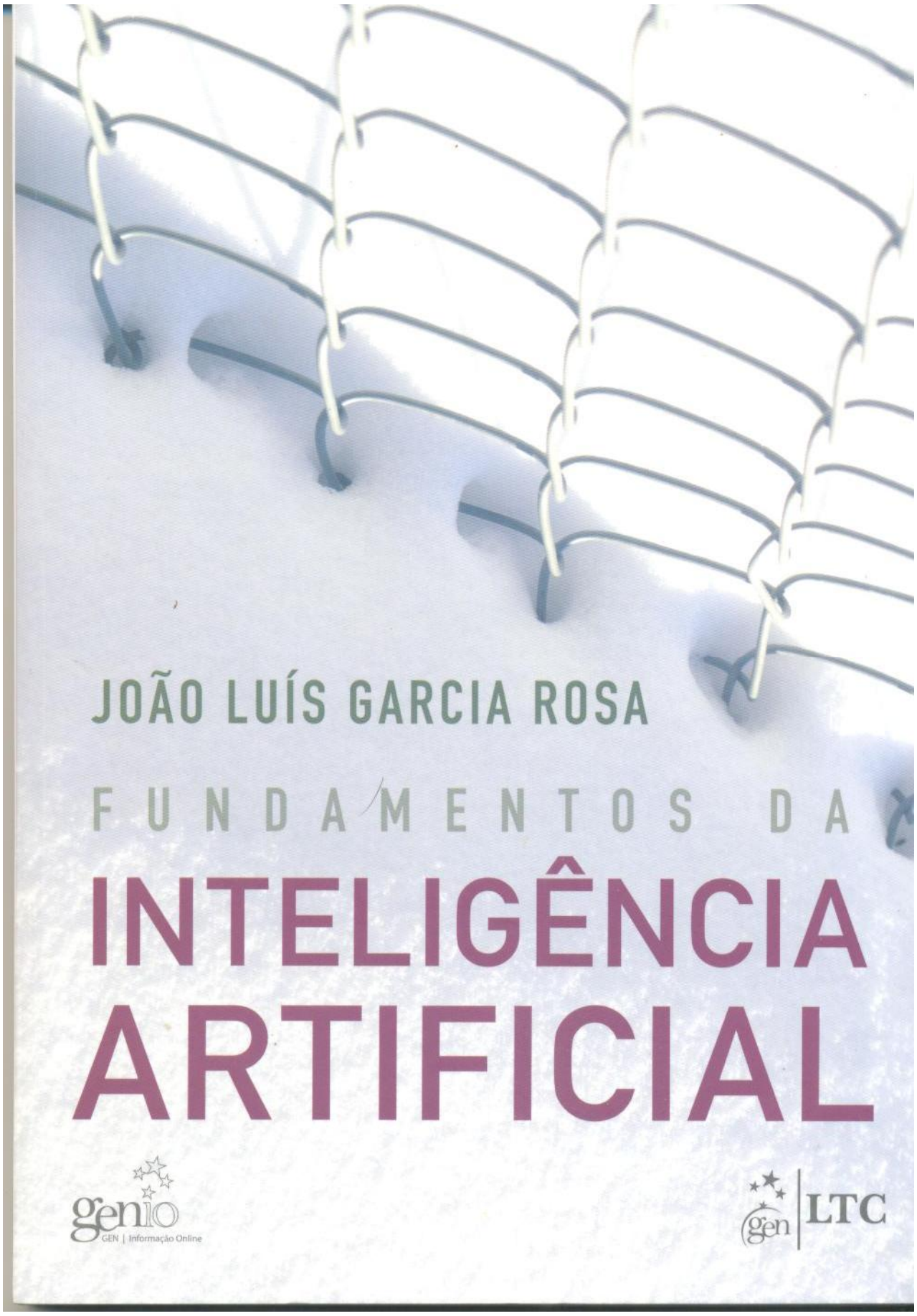


JOÃO LUÍS GARCIA ROSA

FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL

genio
GEN | Informação Online

gen | LTC

O interesse pela construção de mecanismos artificiais inteligentes sempre esteve presente na história da humanidade. Alvo de fascínio e receio, esta área desperta opiniões controversas e instiga o estudo e a pesquisa de diversas áreas do conhecimento.

Mas como definir exatamente Inteligência Artificial (IA)? Num esforço para esclarecer a questão, pode-se entender IA como o desenvolvimento de máquinas capazes de realizar tarefas humanas, simulando a capacidade racional de resolver problemas em situações diversas.

Por esse motivo, a Inteligência Artificial mobiliza tanto cientistas da computação, engenheiros e analistas de sistemas quanto profissionais de outras áreas – filósofos, psicólogos, biólogos e linguistas.

Com o intuito de suprir a carência de livros-texto sobre o tema, João Luís Garcia Rosa elaborou esta obra trazendo fundamentos da IA. A ideia é permitir que o estudante compreenda os conceitos básicos e consiga acessar ferramentas importantes, como a lógica de predicados e as simulações das redes neurais.

Voltado para graduação e pós-graduação, **FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL** utiliza uma linguagem didática e acessível a todos os leitores interessados nesta área de conhecimento, incluindo diversos exercícios propostos ao longo do texto.

LTC

www.grupogen.com.br
http://gen-io.grupogen.com.br

FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL

FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL

JOÃO LUÍS GARCIA ROSA

Professor do Instituto de Ciências Matemáticas e de Computação
da Universidade de São Paulo - USP

gen | LTC

Travessa do Ouvidor, 11
Rio de Janeiro, RJ – CEP 20040-040
Tels.: 21-3543-0770 / 11-5080-0770
Fax: 21-3543-0896
ltc@grupogen.com.br
www.ltceditora.com.br

Capa: Máquina Voadora DG
Editoração Eletrônica: Máquina Voadora DG

CIP-BRASIL. CATALOGAÇÃO-NA-FONTE
SINDICATO NACIONAL DOS EDITORES DE LIVROS, RJ

R694f

Rosa, João Luís Garcia
Fundamentos da inteligência artificial / João Luís Garcia Rosa. - Rio de Janeiro: LTC, 2011.
il. ; 24cm

Inclui bibliografia e índice
ISBN 978-85-216-0593-5

1. Inteligência artificial. I. Título.

11-2680. CDD: 006.32
CDU: 004.891

Dedico este livro aos meus pais,
Helena e Orlando,
e à minha filha
Mariana.

"Tudo é uma questão de manter
A mente quieta,
A espinha ereta
E o coração tranquilo."

Coração Tranquilo, Walter Franco

Agradeço aos meus ex-alunos da Pontifícia Universidade Católica de Campinas,
aos meus atuais alunos da Universidade de São Paulo
e aos colegas e amigos dessas instituições.

Prefácio

Sendo uma área multidisciplinar, a Inteligência Artificial (IA) é motivo de estudo tanto de cientistas da computação, engenheiros e analistas de sistemas, como de linguistas, filósofos, psicólogos e até de biólogos. Daí a importância de um livro de fundamentos básico, técnico, e de acesso fácil aos interessados nesta área de conhecimento.

Este livro teve como origem uma apostila criada para a disciplina *Introdução à Inteligência Artificial* do então Instituto de Informática da Pontifícia Universidade Católica de Campinas (PUC-Campinas), em 1992, em virtude da carência de livros que englobassem o programa proposto para a disciplina. Essa apostila foi atualizada e enriquecida ano a ano até se tornar este livro. Não havia na época (e ainda hoje a oferta é precária) um livro que pudesse ser adotado para essa disciplina, pois se optou por dar ênfase aos fundamentos da IA e não apenas a aplicações. A razão principal é permitir que o estudante, a partir do conhecimento dos fundamentos, possa ter acesso às ferramentas da IA. Por isso, o livro tem uma ênfase muito grande em lógica de predicados de primeira ordem, extremamente necessária para o conhecimento da teoria simbólica da IA, além obviamente de discutir todos os outros conceitos relacionados a uma disciplina de Fundamentos da Inteligência Artificial. Portanto, é um texto indicado para cursos de graduação e pós-graduação, em que haja o interesse em estudar o embasamento teórico dos Sistemas Inteligentes.

J. L. G. Rosa

Sobre o Autor

Graduado em Engenharia Elétrica em 1983, modalidade Eletrônica e Automação, pela Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas – Unicamp, mestre em Engenharia de Computação (dissertação em Processamento de Linguagem Natural – Inteligência Artificial) em 1993 pela mesma faculdade, e doutor em Linguística (Computacional) pelo Instituto de Estudos da Linguagem da Unicamp em 1999. Foi professor titular do Centro de Ciências Exatas, Ambientais e de Tecnologias (CEATec) da Pontifícia Universidade Católica de Campinas (PUC-Campinas), onde lecionou, de 1987 a 2008, disciplinas nos cursos de graduação em Engenharia de Computação e Análise de Sistemas e no mestrado em Sistemas de Computação, como Introdução à Inteligência Artificial, Sistemas Inteligentes, Engenharia do Conhecimento, Redes Neurais Artificiais e Linguagens Formais e Autômatos entre outras. Atualmente é professor doutor do Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP) em São Carlos, onde ministra disciplinas de Inteligência Artificial, Teoria da Computação e Linguagens Formais, Introdução à Ciência da Computação e Algoritmos Avançados, e pesquisa e orienta nas áreas de Inteligência Artificial, Redes Neurais Artificiais e Processamento de Línguas Naturais.

Sumário

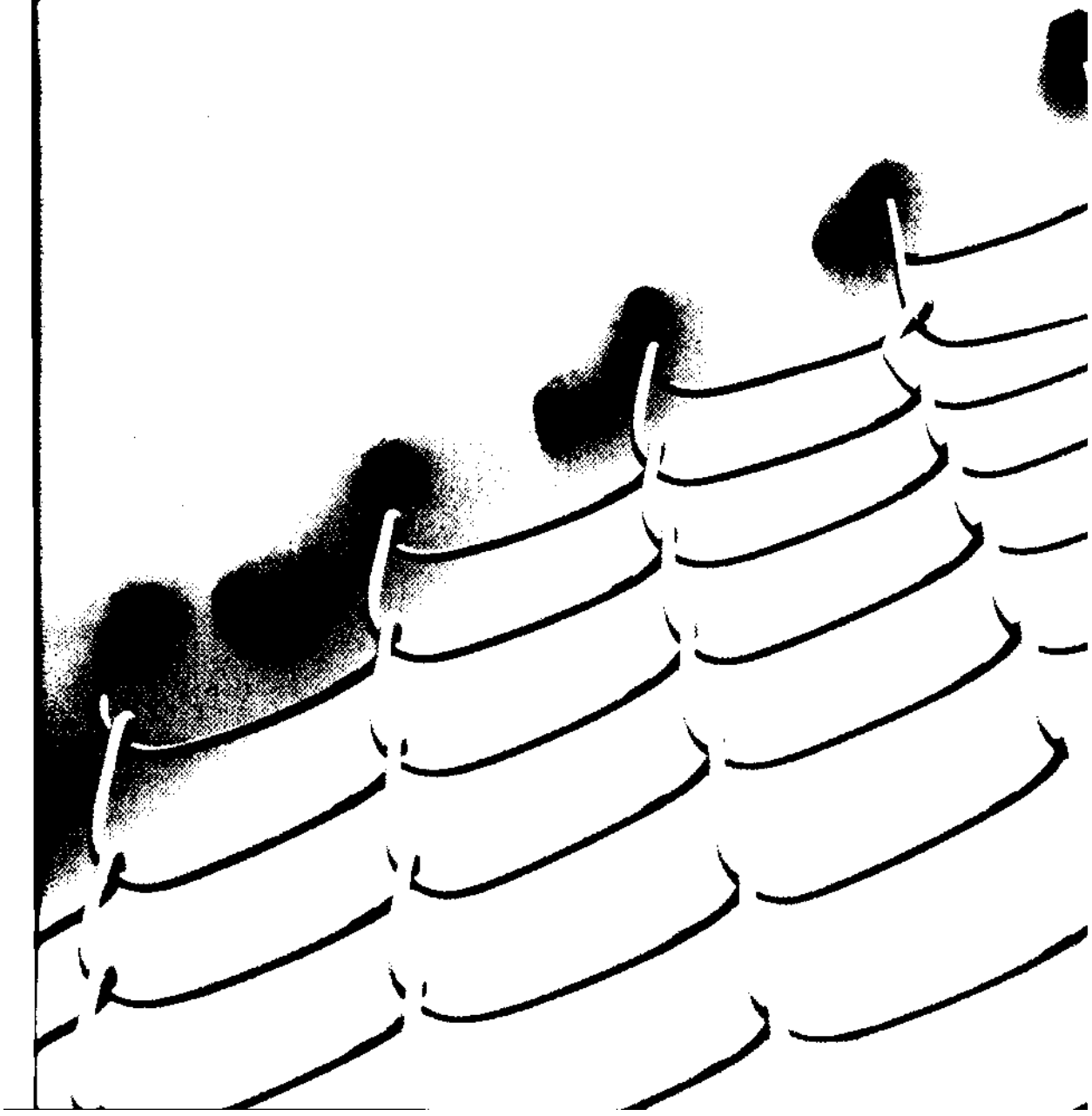
Material Suplementar

Este livro conta com materiais suplementares.

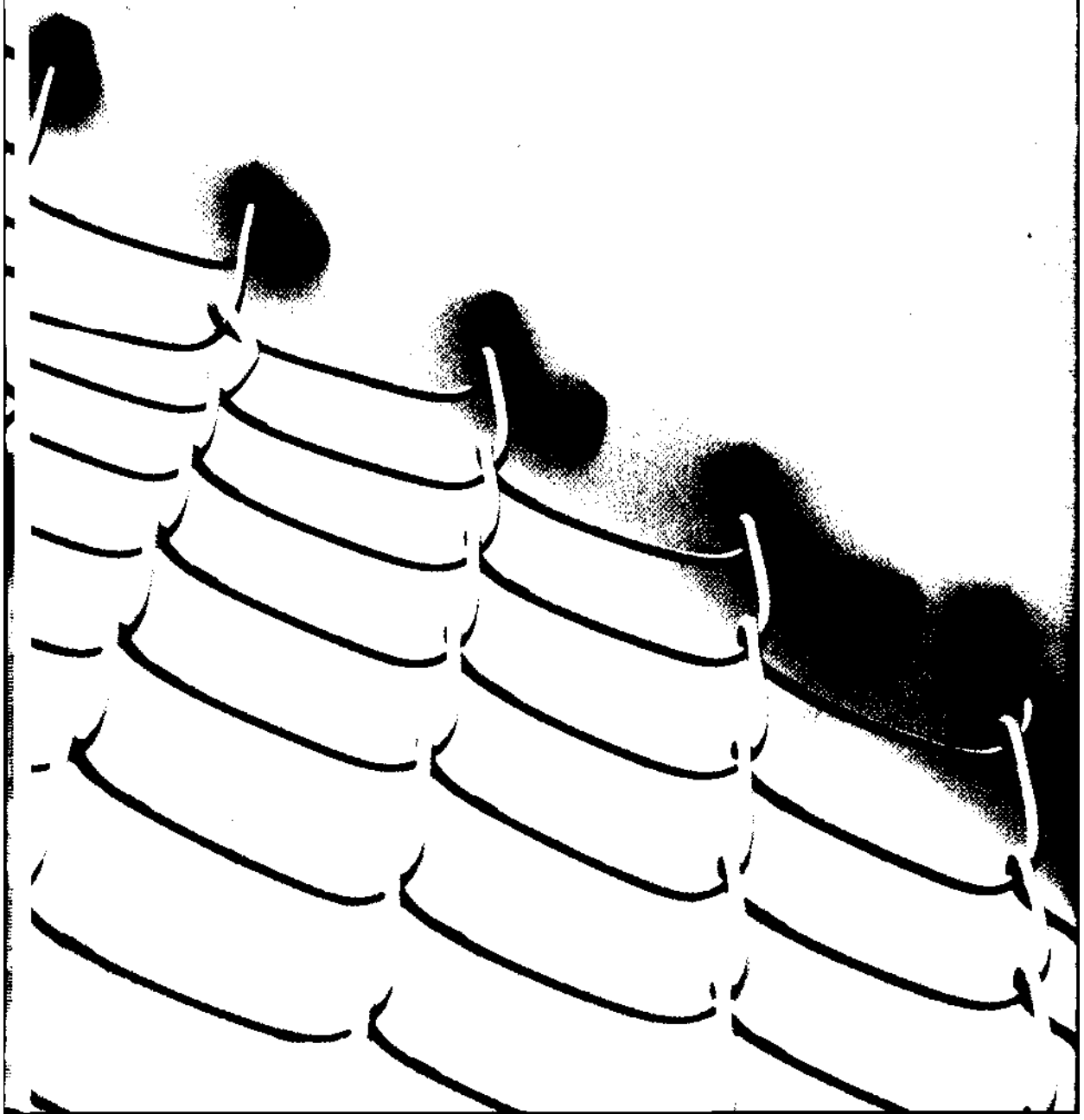
O acesso é gratuito, bastando que o leitor se cadastre em http://gen-io.grupogen.com.br.

GEN-IO (GEN | Informação Online) é o repositório de material suplementar e de serviços relacionados com livros publicados pelo GEN | Grupo Editorial Nacional, o maior conglomerado brasileiro de editoras do ramo científico-técnico-profissional, composto por Guanabara Koogan, Santos, LTC, Forense, Método, E.P.U. e Forense Universitária.

CAPÍTULO 1



Introdução



1.1 O QUE É INTELIGÊNCIA ARTIFICIAL?

O que é exatamente Inteligência Artificial (IA)? A definição exata desse termo é motivo de discussão entre os pesquisadores da área. Definir "artificial" é relativamente simples. "Artificial" é tudo o que é feito pelo homem. E "inteligência"? É bem mais complicado. Uma definição bastante esclarecedora para "inteligência artificial" é: "IA é o estudo de como fazer os computadores realizarem tarefas as quais, até o momento, os homens fazem melhor" (Rich, 1994). Ou ainda, "todo problema para o qual nenhuma solução algorítmica é conhecida é um problema da IA" (Laurière, 1990) (veja mais definições na Figura 1.1 segundo Russell e Norvig (1995)). Ou seja, as tarefas relacionadas com o processamento simbólico, reconhecimento de imagens e tudo o que envolva "aprendizado". Um computador convencional é capaz de realizar cálculos extremamente complexos que se realizados por um homem poderiam levar dezenas de anos, mas, no entanto, não é capaz de distinguir uma cadeira de metal de uma de madeira, coisa que uma criança de três anos é capaz de fazer.

Sistemas que pensam como humanos	**Sistemas que pensam racionalmente**
"O novo e excitante esforço para fazer computadores pensarem... *máquinas com mentes*, no sentido literal e completo" (Haugeland, 1985). "A automação de atividades que nós associamos com o pensamento humano, atividades como tomada de decisões, solução de problemas, aprendizado..." (Bellman, 1978).	"O estudo de faculdades mentais através do uso de modelos computacionais" (Charniak e McDermott, 1985) "O estudo de computações que tornem possível perceber, raciocinar e agir" (Winston, 1992).
Sistemas que agem como humanos	**Sistemas que agem racionalmente**
"A arte de criar máquinas que realizam funções que requerem inteligência quando realizadas por pessoas" (Kurzweil, 1990). "O estudo de como fazer os computadores realizarem tarefas as quais, até o momento, as pessoas fazem melhor" (Rich e Knight, 1994).	"Um campo de estudo que busca explicar e emular comportamento inteligente em termos de processos computacionais" (Schalkoff, 1990). "O ramo da ciência da computação que se preocupa com a automação do comportamento inteligente" (Luger e Stubblefield, 1993).

Figura 1.1 Definições de Inteligência Artificial a partir de vários pontos de vista (Russell e Norvig, 1995).

Com base nisso, conclui-se que a IA tem por objetivo implementar numa máquina a possibilidade de realizar tarefas que uma criança é capaz de realizar, mas o mais poderoso dos supercomputadores ainda não.

1.2 FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial herdou muitas ideias, técnicas e pontos de vista de outras disciplinas (Russell e Norvig, 1995). Entre elas a ciência cognitiva. A hipótese de que as pessoas compre-

endem o mundo através da construção de modelos mentais permite enumerar os itens fundamentais para todos os campos da ciência cognitiva:

- *Psicologia*: Como os modelos são representados no cérebro, como eles interagem com os mecanismos de percepção, memória e aprendizado, e como eles afetam ou controlam o comportamento?
- *Linguística*: Qual é o relacionamento entre um universo, os objetos que ele nomeia e um modelo mental? Quais são as regras de sintaxe e semântica que relacionam modelos às sentenças?
- *Filosofia*: Qual é o relacionamento entre conhecimento, significado e modelos mentais? Como são os modelos usados no raciocínio, e como tal raciocínio está relacionado com a forma lógica?
- *Ciência da computação*: Como um modelo pessoal do mundo pode ser representado em um sistema computacional? Quais as linguagens e ferramentas necessárias para descrever tais modelos e relacioná-los aos sistemas externos? Os modelos podem suportar uma interface de computador que as pessoas achariam simples de usar?

1.3 APLICAÇÕES DA IA

As aplicações da IA vão desde jogos até prova de teoremas. Muitas das tarefas de que a IA trata podem ser divididas em tarefas "corriqueiras", ou seja, tarefas do dia a dia, tarefas formais e tarefas especialistas:

TAREFAS "CORRIQUEIRAS"	
Percepção • visão • fala	**Língua natural** • entendimento • geração • tradução
Raciocínio de senso comum	**Controle de robôs**
TAREFAS FORMAIS	
Jogos • xadrez • etc.	**Matemática** • geometria • lógica • cálculo integral
TAREFAS ESPECIALISTAS	
Engenharia • projeto • descoberta de falhas • planejamento de manufatura	**Análise científica** **Diagnóstico médico** **Análise financeira**

A IA inclui (Rowe, 1988):

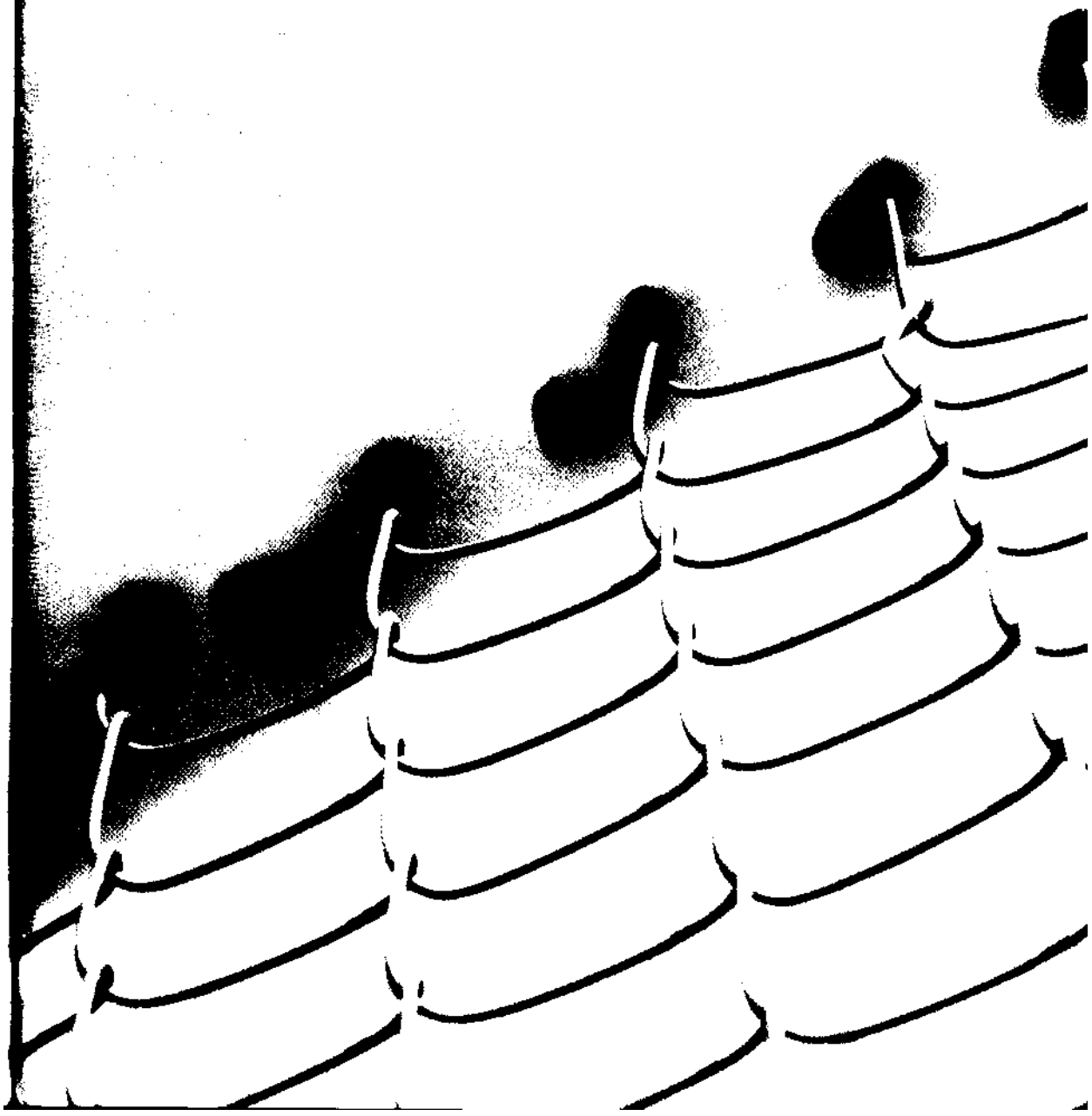
- Fazer o computador se comunicar com o ser humano em línguas naturais (humanas), como o português, através da impressão em um terminal de computador, entendendo o que é digitado em um teclado, gerando fala ou entendendo fala (processamento de línguas naturais).

- Fazer o computador se lembrar de fatos complicados inter-relacionados e obter conclusões a partir deles (inferência lógica).
- Fazer o computador planejar sequências de ações para alcançar metas (planejamento).
- Fazer o computador oferecer ajuda baseada em regras complicadas para várias situações (sistemas especialistas ou sistemas de dedução baseados em regras).
- Fazer o computador olhar através de câmeras e ver o que estiver lá (visão artificial);
- Fazer o computador se mover entre objetos do mundo real (robótica).

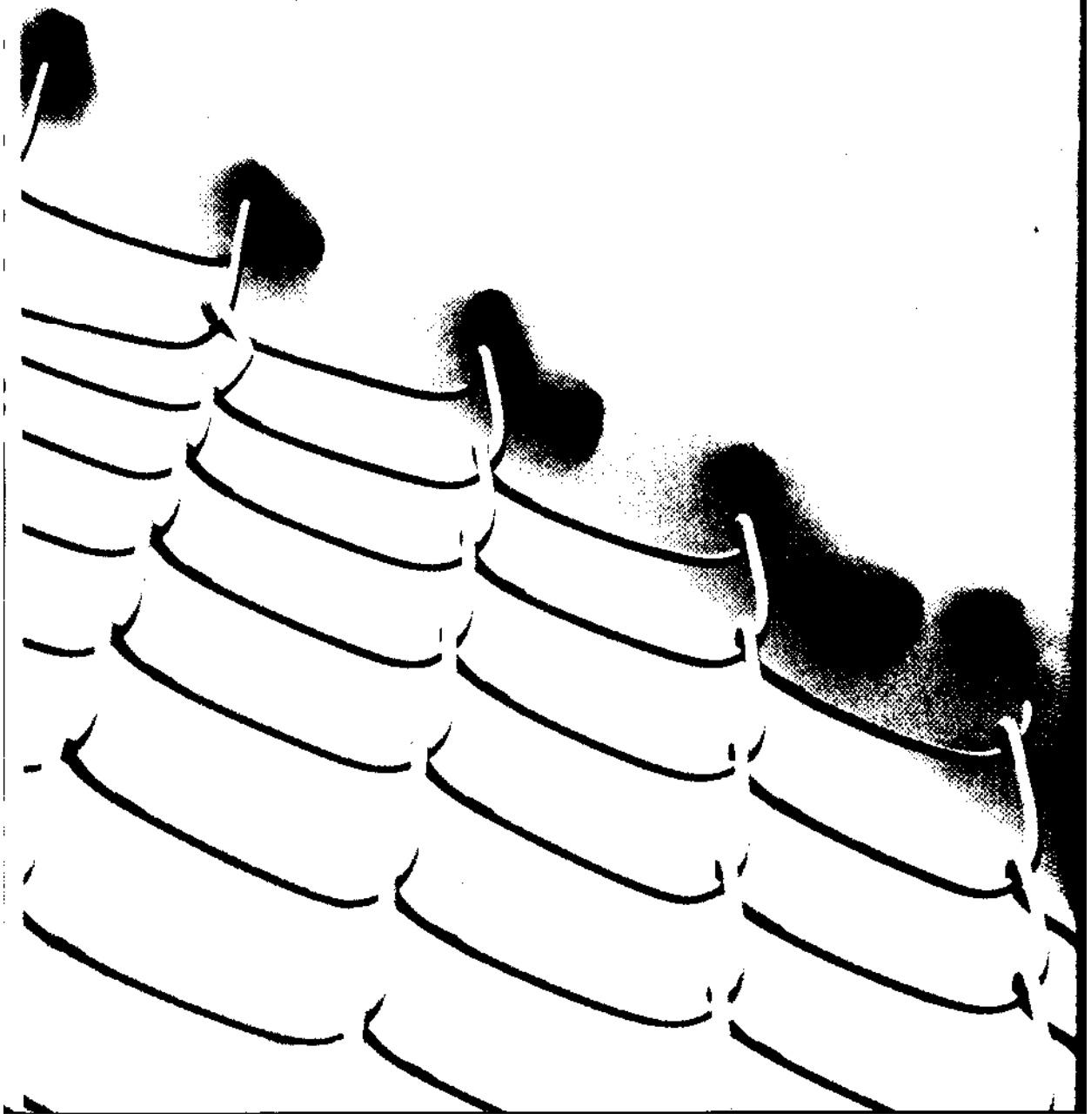
Para realizar essas tarefas, a IA trabalha com várias ferramentas, desde a lógica de predicados (lógica clássica) até simulações das redes neurais, as redes de células nervosas do cérebro. Com a lógica de predicado, podem-se construir os chamados Sistemas de Dedução Baseados em Regras (SDBR), ou Sistemas Especialistas, como é mais conhecido, muito embora seja um nome não muito apropriado.

Neste livro são apresentados os fundamentos da Inteligência Artificial partindo das estratégias de busca para sistemas de produção (Capítulo 2), depois os conceitos e definições da lógica de predicados de primeira ordem (Capítulo 3), ferramenta para representação do conhecimento em sistemas simbólicos. A seguir, no Capítulo 4, vai-se estudar a prova automática de teoremas, através da regra de inferência para os Sistemas de Dedução Baseados em Regras (SDBR), a Regra da Resolução. No Capítulo 5 estudam-se os SDBRs propriamente ditos, e no Capítulo 6 os conhecimentos básicos sobre uma linguagem de programação para implementar os SDBRs, a linguagem Prolog. No Capítulo 7, introduz-se a lógica nebulosa (não clássica) a partir do conceito de conjunto nebuloso (*fuzzy set*). No Capítulo 8 são apresentadas as aplicações do Processamento de Línguas Naturais, e finalmente, no Capítulo 9, as Redes Neurais Artificiais, ferramenta poderosa, cada vez mais usada em Sistemas Inteligentes.

CAPÍTULO 2



Métodos de Busca



2.1 SISTEMAS DE PRODUÇÃO

2.1.1 Introdução

Um sistema de produção é um formalismo computacional em que podem ser identificados claramente três componentes principais: uma base de dados global, um conjunto de regras de produção e um sistema de controle. Os sistemas de produção capturam a essência da operação de muitos sistemas de Inteligência Artificial (IA).

A base de dados global é a estrutura de dados central usada por um sistema de produção em IA. A base de dados global representa o problema que se quer solucionar por meio de um sistema de produção. Dependendo da aplicação, essa base de dados pode ser uma matriz de números ou uma estrutura de arquivo indexado relacional.

As regras de produção operam na base de dados global. Cada regra tem uma precondição que ou é satisfeita ou não pela base de dados global. Se a precondição for satisfeita, a regra pode ser aplicada. A aplicação de uma regra altera a base de dados. O sistema de controle escolhe qual regra aplicável deve ser aplicada e termina a computação quando uma condição de terminação na base de dados global for satisfeita.

2.1.2 Um exemplo: o quebra-cabeça de oito peças

Para o quebra-cabeça de oito peças, a base de dados global pode ser uma matriz quadrada 3×3 de inteiros (pode-se imaginar o -1 representando o espaço vazio). As regras de produção seriam as regras que aplicadas a uma configuração da base de dados modificariam essa configuração. Para este exemplo, em que se têm oito peças distintas e cada peça pode ser movimentada para cima, para baixo, para a esquerda e para a direita, dever-se-ia ter $8 \times 4 = 32$ regras de produção. Mas pode-se optar por "movimentar" o espaço vazio, que é único, em vez de se moverem oito peças diferentes. Nesse caso, o conjunto de regras de produção seria reduzido a quatro regras, a saber, mover o vazio (b) para cima (R1), para baixo (R2), para a direita (R3) e para a esquerda (R4). Suponha que se deseje partir de uma configuração inicial 283-164-7b5 e chegar em 123-8b4-765, como mostra a Figura 2.1.

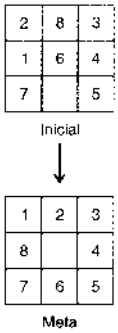


Figura 2.1 Configurações Inicial e Meta para o problema do quebra-cabeça de oito peças (Nilsson, 1982).

2.1.3 O procedimento básico

O algoritmo que vai implementar essa busca de aplicação de regras num sistema de produção é o seguinte (Nilsson, 1982):

Procedimento *PRODUÇÃO*

1. DADOS ← base de dados inicial
2. até DADOS satisfazer a condição de terminação, faça:
3. início
4. selecione alguma regra, R, no conjunto de regras que possa ser aplicada a DADOS
5. DADOS ← resultado da aplicação de R a DADOS
6. fim

2.1.4 Controle

O procedimento mostrado anteriormente é não determinístico porque não se especificou precisamente como se selecionará uma regra aplicável na linha 4. A seleção de regras e a manutenção da trilha das sequências de regras já tentadas e as bases de dados que elas produzem constituem o que se chama de estratégia de controle para sistemas de produção. Em muitas aplicações de IA, a informação disponível para a estratégia de controle não é suficiente para permitir a seleção da maioria das regras apropriadas no passo 4. A operação dos sistemas de produção de IA pode então ser caracterizada como um processo de busca no qual as regras são tentadas até que seja encontrada alguma sequência delas que produza uma base de dados que satisfaça a condição de terminação. As estratégias de controle eficientes requerem conhecimento suficiente sobre o problema a ser resolvido tal que a regra selecionada no passo 4 tenha uma boa chance de ser a mais apropriada. Uma estratégia boa (informada) para o problema do quebra-cabeça de oito peças é a estratégia "subida de encosta" (*hill-climbing*), que consiste em aplicar uma regra que não tire uma peça que já esteja na sua posição final (Nilsson, 1982). Veja a Figura 2.2.

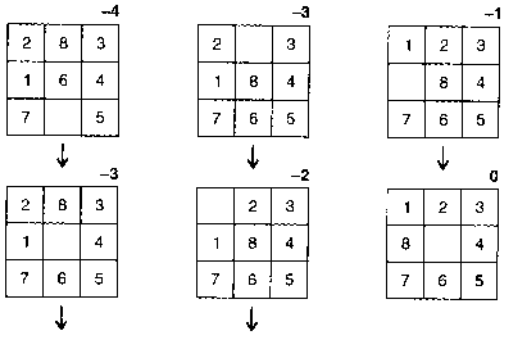


Figura 2.2 Aplicação da estratégia subida de encosta ao problema do quebra-cabeça de oito peças (Nilsson, 1982).

A forma como é conduzido um processo de busca é chamada de regime de controle. Existem dois tipos principais de regimes de controle: irrevogável e tentativo. Num regime de con-

trole irrevogável, uma regra aplicável é selecionada e aplicada irrevogavelmente sem provisão para reconsideração ulterior. Num regime de controle tentativo, uma regra aplicável é selecionada (ou arbitrariamente ou talvez com uma boa razão), a regra é aplicada, mas é feita provisão para retornar mais tarde a esse ponto para aplicar alguma outra regra, caso a regra escolhida produza uma falha. O regime de controle irrevogável não tem interesse para a IA.

Dentro do regime de controle existem as estratégias de controle. Vão-se distinguir duas estratégias de controle dentro do regime de controle tentativo. Numa, chamada de estratégia por *backtracking* (veja aplicação para o problema do quebra-cabeça de oito peças na Figura 2.3 (Nilsson, 1982)), um ponto de *backtracking* é estabelecido quando uma regra é selecionada. Caso a computação subsequente encontre dificuldade de produzir uma solução, o estado da computação reverte ao ponto de *backtracking* prévio, onde uma outra regra é aplicada, e o processo continua. Observe na Figura 2.3 que, além de retornar ao ponto de *backtracking* mais próximo, caso uma configuração se repita (*dead end*), o controle também estabelece que o número máximo de passos é 7, ou seja, se, ao alcançar a sétima configuração, o sistema não tiver atingido a condição de terminação, o sistema força o *backtracking*.

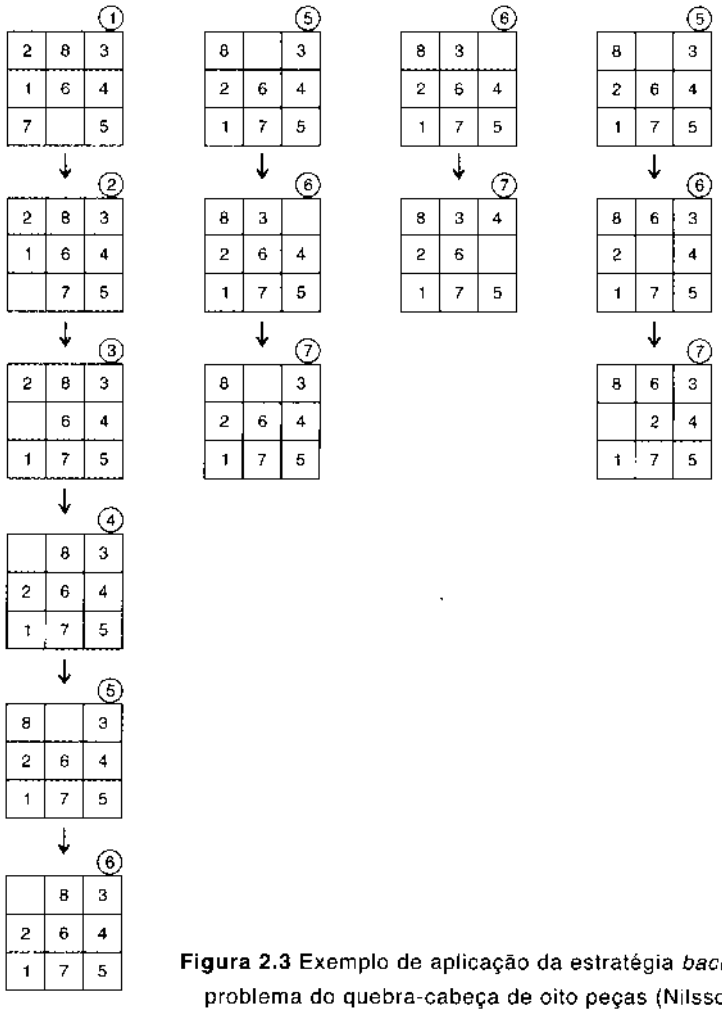


Figura 2.3 Exemplo de aplicação da estratégia *backtracking* ao problema do quebra-cabeça de oito peças (Nilsson, 1982).

Na segunda estratégia de controle do regime de controle tentativo, chamada de controle por busca em grafos, a provisão é feita para manter a trilha dos efeitos de muitas sequências de regras simultaneamente. Vários tipos de estrutura de grafos e procedimentos de busca em grafos são usados nesse tipo de controle.

2.2 ESTRATÉGIAS DE BUSCA PARA SISTEMAS DE PRODUÇÃO DE IA

2.2.1 *Backtracking*

Em muitos problemas de interesse, a aplicação de uma regra imprópria pode atrasar substancialmente ou inibir uma terminação bem-sucedida. Nesses casos, deseja-se uma estratégia de controle que possa tentar uma regra e, se mais tarde, se descobrir que essa regra é imprópria, voltar e tentar uma outra regra em vez daquela.

O processo de *backtracking* é uma forma na qual a estratégia de controle pode ser tentativa. Uma regra é selecionada e, se não levar a uma solução, os passos intermediários são "esquecidos" e uma outra regra é selecionada. Formalmente, a estratégia de *backtracking* pode ser usada a despeito de quanto conhecimento está disponível na hora de selecionar a regra. Se não houver conhecimento nenhum disponível, as regras podem ser selecionadas de acordo com algum esquema arbitrário. Por último, o controle retornará para selecionar a regra apropriada. Obviamente, se puder ser usado conhecimento confiável para seleção de regras, o *backtracking* para considerar regras alternativas ocorrerá com menor frequência, e o processo como um todo será mais eficiente.

Um procedimento recursivo simples captura a essência da operação de um sistema de produção sob controle de *backtracking*. Esse procedimento, chamado BACKTRACK, pega um único argumento, DADOS, inicialmente com o valor igual à base de dados global do sistema de produção. Até a terminação bem-sucedida, o procedimento retorna uma lista de regras, que, se aplicada em sequência à base de dados inicial, produz uma base de dados satisfazendo a condição de terminação. Se o procedimento parar sem achar tal lista de regras, ele retorna FALHA.

2.2.1.1 Procedimento recursivo *BACKTRACK* (DADOS) (Nilsson, 1982)

1. Se TERMO(DADOS), retorne []; TERMO é um predicado verdadeiro para argumentos que satisfazem a condição de terminação do sistema de produção. Até a terminação bem-sucedida, [], a lista vazia, é retornada.
2. Se DEADEND(DADOS), retorne FALHA; DEADEND é um predicado verdadeiro para argumentos que se sabe não pertencerem ao caminho para uma solução. Nesse caso, o procedimento retorna o símbolo FALHA (falha).
3. REGRAS ⇐ REGRASAPL(DADOS); REGRASAPL é uma função que computa as regras aplicáveis a seu argumento e as ordena (ou arbitrariamente ou de acordo com mérito heurístico).
4. LOOP: se NULL(REGRAS), retorne FALHA; se não houver (mais) regras a aplicar, o procedimento falha.
5. R ⇐ PRIMEIRA(REGRAS); a melhor das regras aplicáveis é selecionada.

6. REGRAS ⇐ CAUDA(REGRAS); a lista de regras aplicáveis é diminuída, removendo a recém-selecionada.
7. RDADOS ⇐ R(DADOS); a regra R é aplicada para produzir uma nova base de dados.
8. CAMINHO ⇐ BACKTRACK(RDADOS); BACKTRACK é chamada recursivamente na nova base de dados.
9. Se CAMINHO = FALHA, vá para LOOP; se a chamada recursiva falha, tente uma outra regra.
10. Retorne CONC(R, CAMINHO); de outra forma, passe a lista de regras bem-sucedidas, adicionando R à frente da lista.

2.2.1.2 Exemplo: o problema das oito rainhas

Imagine um tabuleiro de xadrez (matriz 8 por 8). O problema das oito rainhas consiste em colocar uma rainha em cada linha de tal forma que uma rainha não ataque a outra. Para quem não sabe, o movimento da rainha no jogo de xadrez é horizontal, vertical ou diagonal, qualquer número de casas. Suponha a configuração da Figura 2.4. O algoritmo deve agora tentar colocar uma rainha na linha 6. Não é possível. *Backtrack* para realocar a rainha da linha 5, movendo-a para a coluna 8. Ainda assim não é possível realocar a 6. Logo, *backtrack* para a linha 4 e assim por diante. Veja na Figura 2.5 uma solução para o problema, depois de 91 *backtrackings*.

2.2.2 Busca em grafos

Os grafos (ou, mais especialmente, as árvores) são estruturas extremamente úteis para manter a trilha dos efeitos de muitas sequências de regras.

Nas estratégias de *backtracking*, o sistema de controle efetivamente esquece qualquer caminho que resulta em falha. Apenas o caminho correntemente sendo estendido é armazenado explicitamente. Um procedimento mais flexível envolve o armazenamento explícito de todos os caminhos de tal forma que qualquer um deles pode ser candidato a futura extensão.

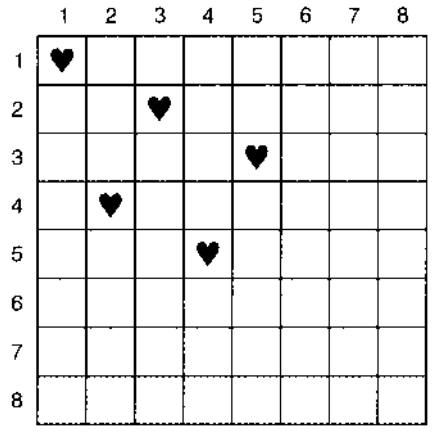


Figura 2.4 O procedimento *backtracking* aplicado ao problema das oito rainhas.

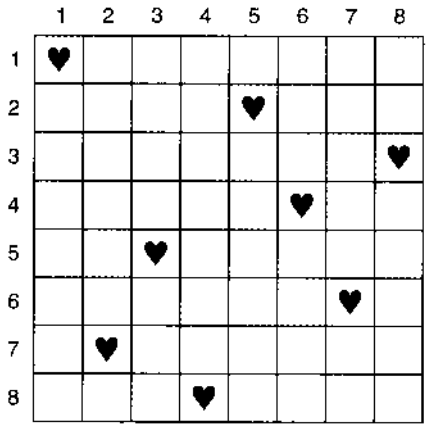


Figura 2.5 Uma solução ao problema das oito rainhas, utilizando o *backtracking*.

2.2.2.1 Notação de grafos

Um grafo consiste em um conjunto (não necessariamente finito) de nós ou vértices. Certos pares de nós são conectados por arcos ou arestas, e esses arcos são direcionados de um membro do par ao outro. Tal grafo é chamado de grafo direcionado. Para esse modelo, os nós são rotulados por base de dados, e os arcos são rotulados por regras. Se um arco é direcionado do nó n_i para o nó n_j, então o nó n_j é o sucessor do nó n_i, e o nó n_i é o pai do nó n_j. Nos grafos de interesse, um nó pode ter apenas um número finito de sucessores. (Os sistemas de produção têm apenas um número finito de regras aplicáveis.) Cada nó de um par pode ser sucessor do outro; nesse caso, o par de arcos direcionados é algumas vezes substituído por uma margem.

Uma árvore é um caso especial de um grafo no qual cada nó tem, no máximo, um pai. Um nó na árvore que não tem pai é chamado de nó raiz. Um nó na árvore que não tem sucessores é chamado de nó folha. Diz-se que o nó raiz é de profundidade zero. A profundidade de qualquer outro nó na árvore é, por definição, a profundidade de seus pais mais 1.

Uma sequência de nós $(n_1, n_2, ..., n_k)$, com cada n_j um sucessor de n_{j-1} para $j = 2, ..., k$, é chamada de um caminho de comprimento k do nó n_1 para o nó n_k. Se um caminho existe entre o nó n_i para o nó n_j, então o nó n_j é acessível a partir do nó n_i. O nó n_j é então um descendente do nó n_i, e o nó n_i é um ancestral do nó n_j. Note que o problema de achar uma sequência de regras para transformar uma base de dados em outra é equivalente ao problema de achar um caminho num grafo.

Frequentemente é conveniente atribuir custos positivos aos arcos, para representar o custo da aplicação da regra correspondente. Usa-se a notação $c(n_i, n_j)$ para denotar o custo de um arco direcionado do nó n_i para o nó n_j. O custo de um caminho entre dois nós é a soma dos custos de todos os arcos que conectam os nós no caminho. Em alguns problemas, deseja-se achar o caminho que tenha *custo mínimo* entre dois nós.

No tipo mais simples de problema, deseja-se achar um caminho (talvez tendo custo mínimo) entre um nó dado s, representando a base de dados inicial, e um outro nó dado t, representando alguma outra base de dados. A situação mais usual envolve achar um caminho entre um nó s e qualquer membro do conjunto de nós $\{t_i\}$ que representa as bases de dados que satisfazem a condição de terminação. Chama-se o conjunto $\{t_i\}$ de o conjunto meta, e cada nó t em $\{t_i\}$ é um nó meta.

2.2.2.2 Um procedimento geral de busca em grafos

Procedimento *BUSCA-EM-GRAFOS (Nilsson, 1982)*

1. Crie um grafo de busca, G, consistindo somente no nó inicial, *s*. Ponha *s* numa lista chamada ABERTOS.
2. Crie uma lista chamada FECHADOS que está inicialmente vazia.
3. LOOP: se ABERTOS está vazia, saia com falha.
4. Selecione o primeiro nó em ABERTOS, remova-o de ABERTOS e ponha-o em FECHADOS. Chame esse nó de *n*.
5. Se *n* é um nó meta, saia com sucesso com a solução obtida traçando um caminho entre os ponteiros de *n* para *s* em G. (Os ponteiros são estabelecidos no passo 7.)
6. Expanda o nó *n*, gerando o conjunto, *M*, de seus sucessores que não são ancestrais de *n*. Instale esses membros de *M* como sucessores de *n* em G.
7. Estabeleça um ponteiro para *n* a partir daqueles membros de *M* que ainda não estejam nem em ABERTOS nem em FECHADOS. Adicione esses membros de *M* a ABERTOS. Para cada membro de *M* que já esteja em ABERTOS ou FECHADOS, decida se direciona ou não seu ponteiro para *n*. Para cada membro de *M* já em FECHADOS, decida para cada um de seus descendentes em G se redireciona ou não o seu ponteiro.
8. Reordene a lista ABERTOS, arbitrariamente ou de acordo com mérito heurístico.
9. Vá para LOOP.

Nessa aplicação, a estratégia de controle gera (torna explícita) parte de um grafo especificado implicitamente. Essa especificação implícita é dada pelo nó inicial s, representando a base de dados inicial e as regras que alteram bases de dados. Será conveniente introduzir a noção de um operador sucessor que é aplicado a um nó para fornecer todos os sucessores desse nó (e os custos dos arcos associados). Chama-se esse processo de aplicação do operador sucessor a um nó, expandindo o nó. Expandindo *s*, os sucessores de *s* tornam explícito o grafo que é implicitamente definido por *s* e pelo operador sucessor. Uma estratégia de controle de busca em grafos então pode ser vista como um processo de tornar explícita uma porção de um grafo implícito suficiente para incluir o nó meta. Veja o procedimento BUSCA-EM-GRAFOS, acima.

Esse procedimento gera um grafo implícito, G, chamado de grafo de busca, e um subconjunto T de G, chamado de árvore de busca. Cada nó de G é também de T. A árvore de busca é definida pelos ponteiros. Cada nó (exceto *s*) em G tem um ponteiro direcionado a apenas um de seus pais em G, que define seu único pai em T. O grafo de busca forma uma ordenação parcial porque nenhum nó em G é um de seus próprios ancestrais (passo 6). Cada caminho possível a um nó descoberto pelo algoritmo é preservado explicitamente em G; um caminho único a qualquer nó é definido por T. Ou seja, os nós em ABERTOS são os nós folha da árvore de busca, e os nós em FECHADOS são os nós não folhas. Mais precisamente, no passo 3 os nós em ABERTOS são aqueles nós (folhas) da árvore de busca que ainda não foram selecionados para expansão. Os nós em FECHADOS ou são nós folha selecionados para expansão que não geraram sucessores no grafo de busca ou são nós não folha da árvore de busca.

O procedimento ordena os nós em ABERTOS no passo 8 tal que o "melhor" deles é selecionado para expansão no passo 4. Essa ordenação pode ser baseada em várias ideias heurísticas ou de forma arbitrária. Quando o nó selecionado para expansão for o nó meta, o processo termina com sucesso. O caminho bem-sucedido do nó inicial ao nó meta pode ser recuperado (na ordem reversa) percorrendo os ponteiros de volta do nó meta a *s*. O processo termina sem sucesso quando a árvore de busca não tem mais nós folha que ainda não foram selecionados para expansão (alguns nós podem não ter sucessores, então é possível a lista ABERTOS estar vazia). No caso da terminação malsucedida, os nós metas devem ser inacessíveis a partir do nó inicial.

O passo 7 requer explicação adicional. Se o grafo implícito sendo buscado é uma árvore, pode-se assegurar que nenhum de seus sucessores, gerados em G, foi gerado previamente: todo nó (exceto o raiz) de uma árvore é o sucessor de apenas um nó e então é gerado uma única vez quando seu único pai é expandido. Nesse caso especial, os membros de M nos passos 6 e 7 não estão em ABERTOS e nem em FECHADOS. Nesse caso, cada membro de M é adicionado a ABERTOS e é instalado na árvore de busca como um sucessor de *n*. O grafo de busca é a árvore de busca na execução do algoritmo, e não há necessidade de trocar os pais dos nós em T.

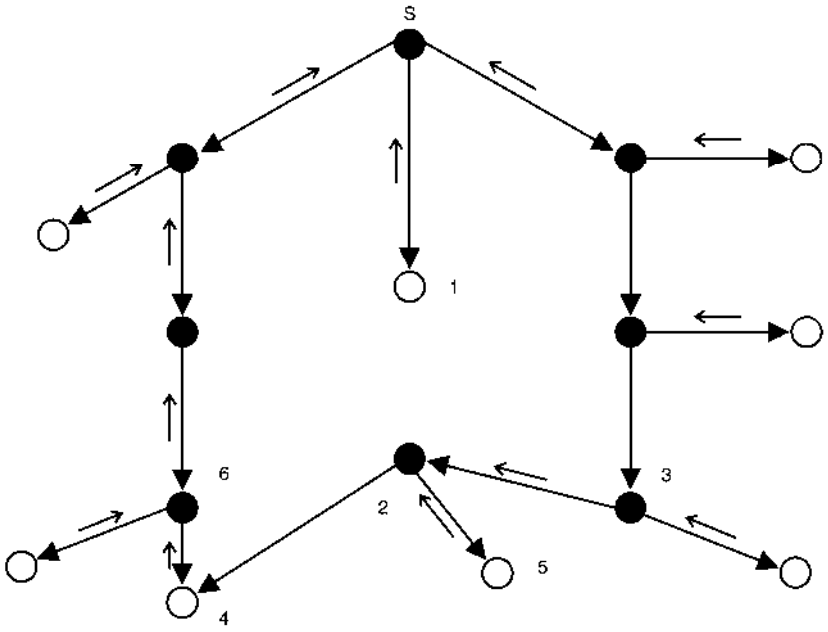


Figura 2.6 Situação de aplicação do algoritmo BUSCA-EM-GRAFOS em que há redirecionamento dos ponteiros ao pai.

Admita a situação da Figura 2.6. Suponha que os nós da lista FECHADOS estão cheios (representados pelo círculo preto) e os nós da lista ABERTOS estão vazados (círculo branco) e suponha que os custos dos arcos são unitários. Quando o nó 1 se expande, o pai de 2 muda de 3 para 1 e 4 muda o pai de 6 para 2, pois surgiu um caminho de comprimento menor (menor custo) para *s*.

2.2.2.3 Procedimentos não informados de busca em grafos

Se nenhuma informação heurística do domínio do problema é usada para ordenar os nós em ABERTOS, algum esquema arbitrário deve ser usado no passo 8 do algoritmo. O procedimento de busca resultante é chamado *não informado*. Em IA, normalmente não há muito interesse por procedimentos não informados. Os procedimentos não informados mais comuns são: busca em profundidade e busca em largura.

O primeiro tipo de busca não informada ordena os nós em ABERTOS na ordem decrescente de sua profundidade na árvore de busca. Os nós mais profundos são colocados primeiro na lista. Nós de profundidade igual são ordenados arbitrariamente. A busca que resulta de tal ordenação é chamada de *busca em profundidade* (*depth-first*) porque o nó mais profundo na árvore de busca é sempre selecionado para expansão. Para prevenir o processo de busca de ficar "rodando" em algum caminho inútil para sempre, um limite de profundidade é necessário. Nenhum nó cuja profundidade na árvore de busca excede esse limite é gerado.

O procedimento de busca em profundidade gera novas bases de dados em uma ordem similar à gerada por uma estratégia de controle *backtracking* não informada. A correspondência seria exata se o processo de busca em grafo gerasse apenas um sucessor de cada vez. Usualmente, a implementação *backtracking* é preferida à versão busca em profundidade do BUSCA-EM-GRAFOS porque o *backtracking* é mais simples de implementar e envolve menos armazenamento. (As estratégias de *backtracking* salvam apenas um caminho ao nó meta; elas não salvam o registro inteiro da busca, como fazem as estratégias de busca em grafo por profundidade.)

Algoritmo 2.1 Busca em Profundidade

1. Se o estado inicial é um estado meta, saia e retorne com sucesso.
2. Caso contrário, faça o seguinte até que sucesso ou falha seja assinalado:
   a. Gere um sucessor, E, do estado inicial. Se não houver mais sucessores, assinale falha.
   b. Chame Busca por Profundidade com E como o estado inicial.
   c. Se sucesso é retornado, assinale sucesso. Caso contrário, continue neste *loop*.

O segundo tipo de procedimento de busca não informada ordena os nós em ABERTOS na ordem crescente de sua profundidade na árvore de busca. (Novamente, para promover o término antecipado, nós metas devem ser postos imediatamente no início de ABERTOS.) A busca que resulta de tal ordenação é chamada de *busca em largura* (*breadth-first*) porque a expansão dos nós na árvore de busca acontece ao longo do "contorno" de mesma profundidade.

A busca em largura garante achar um caminho de comprimento mais curto a um nó meta, se esse caminho existir, tratando-se portanto de uma estratégia *completa*. (Se nenhum caminho existe, o método falhará para grafos finitos ou nunca terminará para grafos infinitos.)

Algoritmo 2.2 Busca em Largura

1. Crie uma variável chamada LISTA-DE-NOS e faça-a igual ao estado inicial.
2. Até que um estado meta seja encontrado ou LISTA-DE-NOS esteja vazia, faça:
   a. Remova o primeiro elemento da LISTA-DE-NOS e chame-o de E. Se LISTA-DE-NOS estiver vazia, saia

3. Para cada forma que cada regra possa unificar com o estado descrito em E, faça:
   a. Aplique a regra para gerar um estado novo.
4. Se o estado novo é um estado meta, saia e retorne esse estado.
5. Caso contrário, adicione o estado novo ao final de LISTA-DE-NOS.

2.2.2.4 Procedimentos heurísticos de busca em grafos

Os métodos de busca não informados, seja a busca em largura ou em profundidade, são métodos exaustivos para achar caminhos para o nó meta. Em princípio, esses métodos proveem uma solução ao problema de achar um caminho, mas eles são frequentemente inviáveis para usar o controle dos sistemas de produção porque a busca expande muitos nós antes que um caminho seja encontrado. Como sempre existem limites práticos à quantidade de tempo e armazenamento disponíveis para usar na busca, alternativas mais eficientes à busca não informada devem ser encontradas.

Para muitas tarefas é possível usar a informação dependente da tarefa para ajudar a reduzir a busca. Informação desse tipo é usualmente chamada de informação heurística,[1] e os procedimentos de busca que a usam são chamados de métodos heurísticos de busca. É possível especificar a heurística que reduza o esforço de busca (abaixo daquele esforço gasto, por exemplo, pela busca em largura) sem sacrificar a garantia de achar um caminho de comprimento mínimo. Algumas heurísticas reduzem bastante o esforço de busca, mas não garantem achar caminhos de custo mínimo. Em muitos problemas práticos, há o interesse em minimizar alguma combinação do custo do caminho e o custo da busca necessário para obter o caminho. Além disso, há interesse em métodos de busca que minimizem essa combinação média sobre todos os problemas a serem encontrados. Se o custo de combinação média do método de busca 1 é menor que o custo de combinação média do método de busca 2, então o método de busca 1 tem mais potência heurística que o método de busca 2. Note que, de acordo com essa definição, não é necessário que um método de busca com mais potência heurística garanta achar um caminho de custo mínimo.

Custos de combinação média nunca são realmente computados, porque é difícil decidir combinar custo de caminho e custo de esforço de busca e porque seria difícil definir uma distribuição de probabilidade sobre o conjunto de problemas a ser encontrado. Além disso, o problema de decidir se um método de busca tem mais potência heurística do que outro é normalmente deixado para a intuição informada, obtida das experiências reais com os métodos.

Considere o seguinte problema do *caixeiro-viajante*: Um vendedor tem uma lista de cidades, que deve visitar apenas uma vez. Existem estradas diretas ligando cada par de cidades da lista. Ache a rota que o vendedor deve seguir para a viagem mais curta possível que começa e termina em uma das cidades.

Um exemplo de uma boa heurística de propósito geral que é útil para vários problemas combinatoriais é a *heurística do vizinho mais próximo*, que trabalha selecionando a alternati-

[1] A palavra *heurística* vem da palavra grega *heuriskein*, que significa "descobrir", que é também a origem de *eureka*, a famosa exclamação de Arquimedes ("Eu achei") (Rich e Knight, 1994).

va localmente superior a cada passo. Aplicando-a ao problema do caixeiro-viajante, produz-se o seguinte procedimento:

1. Arbitrariamente, selecione uma cidade inicial.
2. Para selecionar a próxima cidade, olhe todas as cidades ainda não visitadas e selecione a mais próxima à cidade corrente. Vá a ela.
3. Repita o passo 2 até que todas as cidades tenham sido visitadas.

Sendo N o número de cidades, esse procedimento é executado em tempo proporcional a N^2, uma melhora significativa sobre $N!$, que seria o tempo gasto caso fosse usada uma estratégia não informada (que geraria uma explosão combinatorial).

Existem muitas heurísticas que, ainda que não sejam tão gerais quanto a heurística do vizinho mais próximo, são úteis em uma ampla variedade de domínios. Por exemplo, considere a tarefa de descobrir ideias interessantes em alguma área específica. A seguinte heurística é frequentemente útil:

Se há uma função interessante de dois argumentos $f(x, y)$, olhe para o que acontece se os dois argumentos são idênticos.

No domínio da matemática, essa heurística leva à descoberta do quadrado se f for a função de multiplicação e leva à descoberta da função identidade se f for a função da união de conjuntos. Em domínios menos formais, essa mesma heurística leva à descoberta da introspecção se f for a função *contemplar* ou leva à noção de suicida se f for a função *matar*.

Sem a heurística haveria com certeza uma explosão combinatorial. Só isso já poderia ser um argumento suficiente a favor de seu uso. Mas existem outros argumentos também:

1. Raramente há necessidade da solução ótima; uma boa aproximação normalmente serve muito bem. De fato, existe alguma evidência de que as pessoas, quando resolvem problemas, em vez de serem otimizadoras, elas procuram a satisfação. Em outras palavras, elas procuram qualquer solução que satisfaça algum conjunto de requisitos, e assim que encontram um elas terminam. Um bom exemplo disso é a procura de um espaço no estacionamento. A maioria das pessoas para assim que acha um bom espaço, mesmo que haja um espaço melhor mais à frente.
2. Ainda que as aproximações produzidas por heurísticas possam não ser muito boas no pior caso, os piores casos raramente acontecem na vida real.
3. A tentativa de entender por que uma heurística funciona, ou por que ela não funciona, frequentemente leva a um entendimento mais aprofundado do problema.

EXERCÍCIOS RESOLVIDOS

2.1 Considere a resolução do problema do quebra-cabeça de oito peças usando subida de encosta (*hill-climbing*). Forneça as regras de produção para esse sistema e considere a estratégia heurística mais apropriada para esse caso. Faça com que funcione com o seguinte exemplo: Inicial: 123-856-47b; Meta: 123-456-78b.

Resolução

Regras de Produção:

Mover o vazio para:
1. Cima
2. Baixo
3. Esquerda
4. Direita

Uma heurística boa para esse caso, apesar de incompleta, é a subida de encosta (*hill-climbing*), que consiste em retirar apenas as peças que estão fora de sua posição final.

Exemplo:

Inicial: −3

1	**2**	**3**
8	**5**	**6**
4	7	

Final: 0

1	**2**	**3**
4	**5**	**6**
7	**8**	

Com a subida de encosta não é possível resolver. Possível solução:

−3

1	**2**	**3**
8	**5**	**6**
4	7	

R1 −4

1	**2**	**3**
8	**5**	
4	7	6

R3 −5

1	**2**	**3**
8		5
4	7	6

R3 −5

1	**2**	**3**
	8	5
4	7	6

R2 −4

1	**2**	**3**
4	8	5
	7	6

R4 −3

1	**2**	**3**
4	8	5
7		6

R1 −2

1	**2**	**3**
4		5
7	**8**	6

R4 −1

1	**2**	**3**
4	**5**	
7	**8**	6

R2 0

1	**2**	**3**
4	**5**	**6**
7	**8**	

2.2 Considere o problema do caixeiro-viajante. Discuta pelo menos uma heurística que seja interessante para esse problema. Defina os elementos para um sistema de produção (base de dados global, conjunto de regras de produção e condição de terminação) que o represente.

Resolução

Heurística: vizinho mais próximo.

Sistema de Produção:

- BD: lista de cidades: cabeça = cidade atual
- Regras:
  1. Ir para a cidade A.
  2. Ir para a cidade B.
  3. Ir para a cidade C.
  4. ...

- Condição de terminação: lista vazia

2.3 Considere o problema do jogo da velha. Discuta pelo menos uma heurística que seja interessante para esse problema. Defina os elementos para um sistema de produção (base de dados global, conjunto de regras de produção e condição de terminação) que o represente.

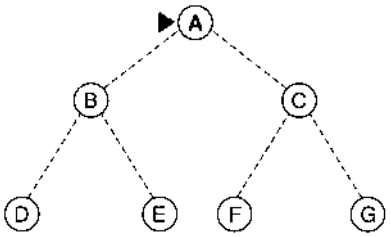
Resolução

BD = matriz 3 por 3 de (X, O, vazio)
Regras =
1. Colocar X em espaço vazio.
2. Colocar O em espaço vazio.

Condição de terminação = uma linha, coluna ou diagonal de X ou O ou velha.
Uma heurística possível:
1. Tentar formar uma linha, coluna ou diagonal de X ou O.
2. Tentar evitar que o adversário consiga.

2.4 Dê a sequência de expansão dos nós deste grafo para os seguintes procedimentos de busca cega:

1. Busca em largura.
2. Busca em profundidade.



Resolução

Parte 1: busca em largura
A-B-C-D-E-F-G.
Parte 2: busca em profundidade
A-B-D-E-C-F-G.

EXERCÍCIOS PROPOSTOS

2.1 Usando busca em profundidade, para o algoritmo BUSCA-EM-GRAFOS, para o quebra-cabeça de oito peças:

a. Escreva o desenvolvimento completo das listas ABERTOS e FECHADOS e dos ponteiros para o pai.

b. Faça a recuperação da trajetória ótima.

2.2 Discuta vantagens e desvantagens dos métodos heurísticos e não informados de busca em grafos.

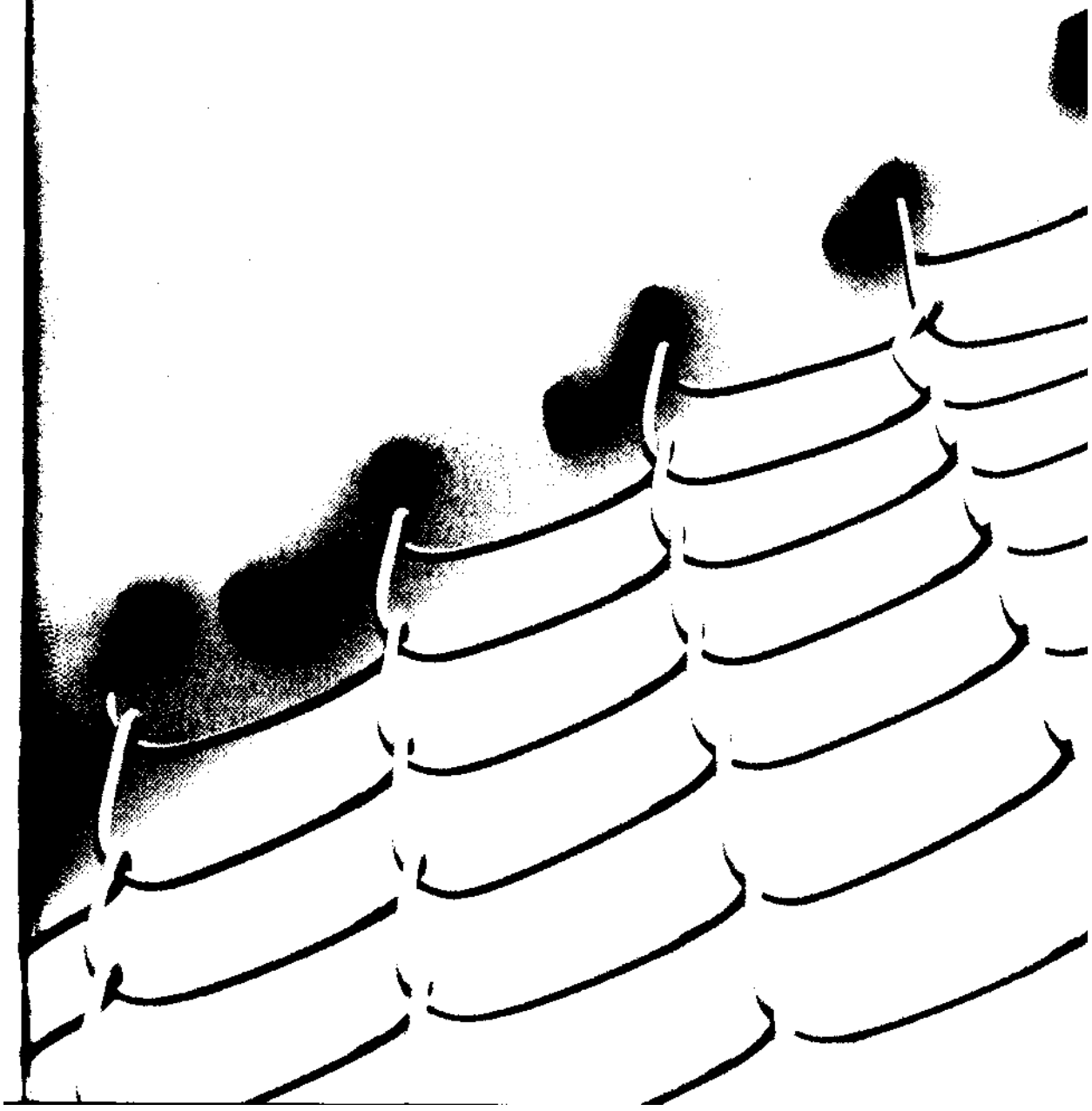
2.3 Descreva algumas sequências de regras que são aplicadas ao problema do quebra-cabeça de oito peças, para o procedimento subida de encosta, para algumas configurações iniciais.

2.4 Represente os elementos de um sistema de produção para o exemplo do caixeiro-viajante usando a notação de grafos. Considerar as possibilidades para a solução desse problema levando em consideração um caminho ótimo (custo mínimo).

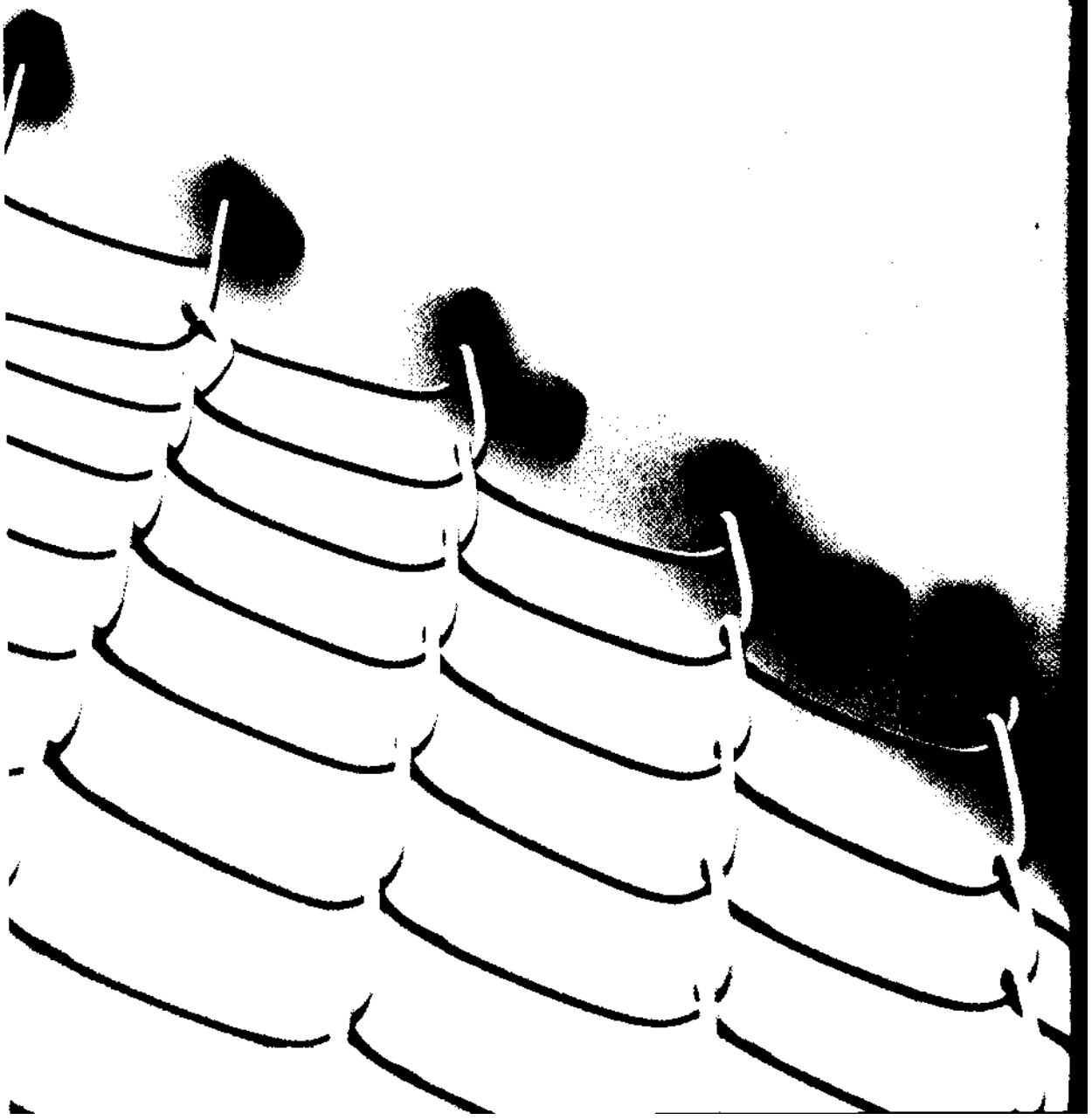
2.5 Especifique uma base de dados global, regras e uma condição de terminação para um sistema de produção para resolver o seguinte problema dos jarros de água:

- Dados um jarro de 5 litros cheio de água e um jarro de 2 litros vazio, como se pode obter precisamente 1 litro no jarro de 2 litros? A água pode ser desperdiçada ou transferida de um jarro para outro, entretanto, só os 5 litros iniciais estão disponíveis.

CAPÍTULO 3



Lógica de Predicados



3.1 O QUE É LÓGICA?

3.1.1 Introdução

Definir lógica não é uma tarefa simples, mesmo se se considerar apenas sistemas formais desenvolvidos pelos lógicos matemáticos. Esses sistemas foram originalmente criados e estudados com a preocupação de caracterizar uma linguagem simbólica dentro da qual todas as proposições matemáticas possam ser expressas. Mais particularmente, os lógicos queriam uma linguagem dentro da qual eles pudessem expressar um conjunto de verdades matemáticas básicas, ou axiomas, a partir dos quais todo o resto poderia ser gerado, aplicando um conjunto finito de regras de prova caracterizadas (combinatoriais), que poderiam ser mostradas como preservadoras da verdade.

Em resumo, as linguagens da lógica matemática não serviam para o uso geral. Seus desenvolvedores não defendem que elas sejam simbolismos universais para aplicações irrestritas — ou seja, que tudo pensável possa ser adequadamente expresso nelas. Certamente, nem tudo que é expresso em língua natural pode ser expresso em uma linguagem lógica formalizada.

Alguns esforços têm sido feitos nessa direção. Dúvidas têm sido levantadas sobre a adequação ou apropriação, por exemplo, da linguagem *básica* da lógica — a linguagem do cálculo de predicados de primeira ordem.

Uma linguagem formal é determinada pela especificação de um vocabulário, constituído de tipos sintáticos, e pelo desenvolvimento de um conjunto de regras de formação para geração de expressões complexas, em particular, termos e/ou fórmulas complexas bem formados da linguagem. Todas essas especificações devem ser mecânicas ou algorítmicas.

Tendo determinado a sintaxe de uma linguagem, pode-se então especificar a semântica de uma forma que reflita a especificação recursiva de sua sintaxe. Podem-se então atribuir valores semânticos às expressões primitivas, não lógicas, da linguagem, com tipos diferentes de valor semântico, associados a tipos sintáticos diferentes. Tais atribuições também são chamadas de modelos ou interpretações. Pode-se então dar um conjunto de regras de interpretação que determina valores semânticos de expressões complexas, como uma função de valores semânticos de seus constituintes e das regras de formação sintática usadas para gerá-los. Por exemplo, o significado de constantes lógicas — conectivos funcionais veritativos e quantificadores — pode ser dado pelas regras de interpretação que atribuem valores a expressões complexas que as contêm.

Finalmente, pode-se especificar um aparato dedutivo para a linguagem. Esse aparato pode tomar várias formas, mas todas envolvem a especificação de um conjunto de regras de transformação cuja aplicabilidade a um conjunto de sentenças pode ser efetivamente determinado e cuja saída, tipicamente uma sentença, pode também ser determinada. Adicionalmente, essas regras devem consistir na semântica especificada para a linguagem. Por exemplo, se as premissas de uma regra são válidas — verdadeiras em toda interpretação, de acordo com a semântica —, então a conclusão da regra também é. Tais regras são ditas consistentes com respeito à validade. Deve-se também requerer que as regras sejam preservadoras da verdade.

3.1.2 Raciocínio e lógica

Para conseguir a liberdade que se tem na escolha de regras, deve-se claramente distinguir raciocínio de prova. O raciocínio ideal pode, frequentemente, conduzir de crenças verdadeiras para falsas. O raciocínio frequentemente faz com que se desista de algumas crenças, a partir das quais se inicia — mesmo quando não se resolve intencionalmente colocar essas crenças em teste (em contraste às provas por refutação, ou por redução ao absurdo, na lógica).

Para considerar um caso simples, suponha que se aceite — entre outras coisas, é claro — alguma sentença na forma "se p, então q" e se aceite o antecedente. Dever-se-ia aceitar o consequente? A resposta é "não necessariamente", pois imagine que se tenham razões suficientemente fortes para acreditar em não q, e isso levaria à desistência das crenças no condicional ou seu antecedente. Entretanto, as regras de prova são locais; elas se aplicam a um dado conjunto de sentenças de acordo com as suas formas sintáticas individuais. O raciocínio, por outro lado, pode frequentemente ser global; deve-se tentar não apenas levar em consideração todas as evidências relevantes, mas também conseguir mais evidências, se a evidência imediata é julgada insuficiente. Esses julgamentos sobre a relevância e pesos de evidência são tipicamente os produtos do raciocínio.

Pode parecer que a prova lógica está sendo colocada em oposição ao raciocínio. A visão correta é que a prova lógica é uma ferramenta usada no raciocínio. Portanto, falar em *raciocínio lógico* não é apropriado.

Os esforços para resolver o problema da representação do conhecimento envolvem dois grandes obstáculos: decidir que conhecimento representar e obter respostas de um computador num tempo razoável.

3.1.3 Lógica default

A noção de *pressuposição* vem da linguística. Em termos pragmáticos, as pressuposições de uma asserção, uma pergunta ou um comando são aquelas proposições que um falante assume como válidas para o seu universo.

Considere o verbo *lamentar* em duas sentenças cujos objetos são cláusulas relativas (ou seja, proposições embutidas):

a. José lamenta que Maria tenha vindo à festa.
b. José não lamenta que Maria tenha vindo à festa.

Como todos os verbos *factuais* (que representam um fato), dada nenhuma informação ao contrário, entende-se que o falante assumiu a proposição embutida — nesse caso, que Maria veio à festa. Por outro lado, na sentença:

c. José não lamenta que Maria tenha vindo à festa, porque ela não veio à festa.

O falante afirma através da cláusula *porque* que Maria não veio à festa. Então não é consistente acreditar que o falante assuma que Maria veio à festa e a pressuposição não seja gerada.

3.1.4 Lógica modal para planejamento de expressão

A lógica e a dedução têm importantes papéis na geração da língua natural. Por exemplo, suponha que haja alguma coisa que uma determinada pessoa A não possa fazer, mas uma outra pessoa B pode e faria para a pessoa A, se soubesse do que se trata. O que a pessoa A pode fazer é formular uma pergunta que informe à pessoa B o que deve ser feito e como fazê-lo. O sucesso de uma expressão, emitida pela pessoa A, tal como "a pessoa B poderia pegar aquela chave inglesa na caixa de ferramentas próxima ao martelo?", vem de (1) direcionando a atenção da pessoa B para a coisa na caixa de ferramentas próxima ao martelo, (2) informando à pessoa B o que é uma chave inglesa (se ela ainda não souber) e (3) pedindo à pessoa B que a pegue para a pessoa A.

A produção de uma expressão bem-sucedida pode requerer raciocínio sobre o que o agente sabe ou não e o que ele deve fazer para saber. Entretanto, o raciocínio sobre o conhecimento, a ação e o efeito da ação no conhecimento não está dentro do escopo da lógica de primeira ordem, assumindo sua semântica padrão, parcialmente porque os predicados de "conhecimento" são obscuros. Ao contrário dos predicados padrões, os predicados de "conhecimento" não permitem a substituição de termos equivalentes, tal como, se *CORRE(Pedro, 2milhas)* é verdadeiro, então *CORRE(Pedro, 1.24km)* também é. Mesmo que *SABE(Pedro, 2milhas = 2milhas)* seja verdadeiro, o comando *SABE(Pedro, 2milhas = 1.24km)* pode não ser, porque Pedro pode não conhecer nada sobre conversão métrica. O problema é que a ação muda o universo e o que é sabido sobre ele, falsificando coisas que eram verdadeiras e vice-versa. A lógica de primeira ordem, com sua semântica padrão, utiliza verdades eternas.

A lógica modal, por outro lado, pode ser aplicada ao raciocínio sobre o conhecimento e a ação, como uma consequência de sua consideração com "necessidade" e "possibilidade". Essas considerações levaram os filósofos a introduzir a noção de um "estado consistente de associações" ou "universo possível". A necessidade corresponde à verdade em todos os universos possíveis, enquanto a possibilidade corresponde à verdade em alguns universos possíveis. Pode-se olhar para o conhecimento e a ação como associação de universos possíveis. Ou seja, os universos *u1* e *u2* podem estar relacionados pela consistência de *u2* com o que algum agente conhece em *u1* ou com o resultado de alguma ação realizada em *u1*.

Um problema mais grave é tornar tal lógica tratável computacionalmente. Uma solução envolve a tradução (isto é, axiomatização) para a lógica de primeira ordem, da interpretação de "universos possíveis". Esses "universos possíveis" então se tornam coisas sobre as quais se pode raciocinar.

3.1.5 Lógica temporal para raciocínio sobre o futuro

O raciocínio sobre a mudança — o que pode ser, o que poderia ser e o que poderia ter sido — tem um papel importante no comportamento conversacional. Quando uma pessoa não sabe a resposta para a pergunta de alguém, raciocina-se sobre se deveria sabê-la depois. Se a resposta deixar a outra pessoa insatisfeita, raciocina-se sobre se deveria ter uma resposta melhor mais tarde. Por exemplo:

A: Érica matriculou-se em Física?
B_1: Eu não sei. Quer que eu o informe quando descobrir?
B_2: Não. Quer que eu o informe se ela se matricular no próximo ano?

É claro que é importante não fazer uma pergunta como esta:

A: Campinas está a menos de 30 quilômetros de Bragança Paulista?

B: Não, mas quer que eu o informe quando estiver?

Nem a lógica de primeira ordem com sua semântica padrão e nem a lógica modal são adequadas para raciocínio sobre mudança. A lógica modal não provê uma boa representação em sequências de eventos. Num sistema, é necessário que se possa raciocinar a partir de eventos passados para o que possa ser verdade mais tarde, incluindo possivelmente o presente. Computacionalmente, já existe uma abordagem para tornar o tempo uma representação implícita da entrada (veja Redes Recorrentes, no Capítulo 9).

3.1.6 O que é importante sobre a representação do conhecimento?

Em ciência da computação, uma boa solução frequentemente depende de uma boa representação. Para a maioria das aplicações em Inteligência Artificial, a escolha da representação é mais difícil, pois as possibilidades são substancialmente maiores e os critérios são menos claros. A escolha da representação torna-se crucial para os estados do raciocínio e conhecimento de agentes inteligentes que podem entender a língua natural, pois as primitivas de representação e o sistema para sua combinação efetivamente limitam o que tais sistemas podem perceber, conhecer ou entender.

Os problemas para a elaboração de programas de computador inteligentes, que usam conhecimento para realizar tarefas, são muitos. Alguns desses problemas são (Woods, 1983):

- como estruturar um sistema de representação que será capaz, a princípio, de fazer todas as distinções importantes;
- como manter-se reservado sobre os detalhes que não podem ser resolvidos;
- como reconhecer, eficientemente, que conhecimento é relevante para o sistema, numa situação particular;
- como adquirir conhecimento dinamicamente, durante o tempo de vida do sistema; e
- como assimilar partes do conhecimento, na ordem em que elas são encontradas, em vez de uma ordem específica de apresentação.

3.1.7 O papel da lógica na representação do conhecimento

Segundo Kolata (1982), os teóricos ainda não chegaram a um consenso sobre como resolver o problema da Inteligência Artificial — como tornar verdadeiras as máquinas pensantes. Existem dois pontos de vistas filosóficos oponentes (A e B).

O ponto de vista A acredita que o caminho para resolver o problema é projetar programas de computador que raciocinem de acordo com as linguagens formais da lógica matemática, não importando se essa é ou não a forma como as pessoas pensam. O ponto de vista B acredita que uma abordagem favorável é tentar fazer com que os computadores imitem a forma como a mente humana trabalha, o que, segundo esse ponto de vista, quase não tem nada a ver com a lógica matemática.

Ambos os lados do debate concordam que o objetivo central da pesquisa é que os computadores devem, de alguma forma, vir a "conhecer" e "entender" o que todo humano conhece

sobre o universo e sobre os organismos, naturais ou artificiais, que o habitam. Esse corpo de conhecimento — indefinido, sem dúvida, nos seus limites — tem o nome de "senso comum". O problema que se enfrenta é como colocar tal conhecimento em um robô. Isto é, como projetar um robô com uma capacidade de raciocínio suficientemente poderosa e favorável que, quando provido com algum subcorpo desse conhecimento, seja capaz de gerar satisfatoriamente o resto desse conhecimento para adaptar-se inteligentemente e explorar seu ambiente? Pode-se assumir que a maioria do conhecimento de senso comum é geral, como no conhecimento de que objetos caem, a menos que estejam com algum suporte, de que objetos físicos não desaparecem de repente e de que se fica molhado quando se toma chuva.

3.1.8 O papel de uma rede de conhecimento para uma máquina inteligente

Um problema fundamental na construção de um agente de computador inteligente é a análise da situação para determinar o que fazer. Por exemplo, muitos sistemas especialistas são organizados em volta de um conjunto de "regras de produção", um conjunto de regras de ações padrão, que caracterizam o comportamento desejado do sistema. Tal sistema opera, determinando em todo passo, que regras são satisfeitas pelo estado corrente do sistema, agindo então sobre esse estado, executando uma dessas regras.

Segundo Woods (1983), a abordagem ao problema de determinar que regras aplicar assume que partes de padrões de todas as regras são organizadas em uma taxonomia estruturada de todas as situações e objetos sobre os quais o sistema tem algum conhecimento. Por *taxonomia* entende-se uma coleção de conceitos ligados por uma relação generalizada, de tal forma que os conceitos mais gerais que um dado conceito são acessíveis a partir dele. Por uma taxonomia *estruturada* entende-se que as descrições dos conceitos têm uma estrutura interna disponível ao sistema de computador tal que, por exemplo, a colocação dos conceitos dentro da taxonomia pode ser computacionalmente determinada. Uma característica de tal taxonomia é que a informação pode ser armazenada em seu nível mais geral de aplicabilidade e acessada indiretamente por conceitos mais específicos que "adquirem" aquela informação.

3.1.9 A necessidade de uma organização taxonômica

Para regras independentes, em um sistema de regras de produção clássico, os conflitos de várias regras serem satisfeitas ao mesmo tempo são descobertos apenas quando uma situação conflitante ocorre. Numa estrutura de classificação taxonômica, entretanto, a absorção das condições de uma regra por outra pode ser descoberta quando a regra é assimilada na taxonomia, quando a pessoa que entra com a regra introduz a questão de como as duas regras podem interagir.

A assimilação de regras em uma estrutura de conhecimento taxonômico não apenas facilita a descoberta de interações na ocasião da entrada, mas também promove uma compactação na especificação das regras. Confiando no fato de que conceitos adquirem informação a partir de conceitos mais gerais, pode-se, usualmente, criar o conceito para a parte padrão de uma regra nova apenas adicionando uma restrição pequena a um conceito existente.

3.1.10 Programação lógica como uma representação do conhecimento

A ideia de que a lógica serviria como uma linguagem de programação foi posta em prática em 1972, na forma do Prolog. Seu valor foi provado em várias áreas da computação, entre as quais o processamento de línguas naturais.

A introdução do Prolog tornou possível representar o conhecimento em termos da lógica e também fazer inferências a partir desse conhecimento, automaticamente.

3.2 LÓGICA SENTENCIAL OU CÁLCULO PROPOSICIONAL

A motivação para se estudar lógica num curso de Inteligência Artificial é de usar essa linguagem artificial como uma ferramenta para fazer dedução lógica em base de conhecimento.

3.2.1 Aspectos da lógica

A Inteligência Artificial (IA) deve ter mecanismos para a representação de fatos. A abordagem mais comum para esse fim é usar a linguagem da *lógica*. A prova de teoremas era um dos primeiros domínios nos quais as técnicas de IA eram exploradas. Vão ser usados neste capítulo os seguintes símbolos lógicos padrões: "$\rightarrow$" (implicação), "$\neg$" (negação), "$\vee$" (disjunção), "$\wedge$" (conjunção), "$\forall$" (quantificação universal = "para todos") e "$\exists$" (quantificação existencial = "existe").

3.2.1.1 Representando fatos simples através da lógica

Para representar o conhecimento do mundo de que um sistema de IA necessita, explora-se o uso da lógica proposicional. Os fatos do mundo real vão ser representados através das *fórmulas bem formadas* ("fbfs") ou *proposições lógicas*, como mostrado a seguir:

Está chovendo.
chovendo

Está ensolarado.
ensolarado

Se está chovendo, então não está ensolarado.
chovendo $\rightarrow$ $\neg$*ensolarado*

Se se quiser representar o fato óbvio de que:
"Sócrates é um homem"
Pode-se escrever:
socrateshomem

Mas se se quiser representar
"Platão é um homem"

seria necessário escrever algo como:
plataohomem

que seria uma asserção totalmente isolada, não sendo possível concluir sobre similaridades entre Sócrates e Platão. Seria melhor, portanto, escrever esses fatos como:

homem(sócrates)
homem(platão)

já que a estrutura da representação reflete a estrutura do conhecimento. Veja a dificuldade em representar a sentença

"Todos os homens são mortais"

Poder-se-ia representá-la por
mortalhomem

Mas essa forma não captura o relacionamento existente entre um indivíduo sendo um homem e um indivíduo sendo mortal. Para fazê-lo, necessita-se de variáveis e quantificação (ver linguagens de 1.ª ordem), a menos que se deseje escrever comandos separados sobre a mortalidade de todos os homens conhecidos.

3.2.2 Sintaxe das linguagens proposicionais

Dado um alfabeto α, uma *cadeia* sobre α é uma sequência de símbolos de α. Uma *linguagem* sobre α é um conjunto de cadeias de α.

Um *alfabeto proposicional* α consiste em:[2]

***símbolos lógicos*:**
pontuação: (,)
conectivos: $\neg$ (negação)
$\wedge$ (conjunção)
$\vee$ (disjunção)
$\rightarrow$ (implicação)
$\equiv$ ou $\Leftrightarrow$ (equivalência ou bi-implicação)

***símbolos não lógicos*:** um conjunto finito **P** de *símbolos proposicionais* diferentes dos símbolos lógicos. Exemplo: p, q etc.

Todas as definições a seguir têm por base o alfabeto proposicional α.

O conjunto de *fórmulas proposicionais* (ou simplesmente *fórmulas*) é o menor conjunto de cadeias satisfazendo as seguintes condições:

i. todo símbolo proposicional é uma fórmula;
ii. se p e q são fórmulas, então $(\neg p)$, $(p \wedge q)$, $(p \vee q)$, $(p \rightarrow q)$ e $(p \equiv q)$ também são fórmulas.

[2] O pontilhado à esquerda do texto será usado para indicar uma definição.

Uma fórmula q é uma *subfórmula* de uma fórmula p se e somente se q é uma subcadeia de p. Em outras palavras, toda subcadeia de p que ainda for uma fórmula é uma subfórmula.

A *linguagem proposicional*, denotada por $L(\alpha)$, é o conjunto das fórmulas proposicionais.

3.2.3 Semântica das linguagens proposicionais

As fórmulas de uma linguagem proposicional, incluindo os símbolos proposicionais, terão como significado os valores-verdade *FALSO* ou *VERDADEIRO*, abreviados *F* e *V*, respectivamente.

Seja **P** o conjunto de símbolos proposicionais de α. Uma *atribuição de valores-verdade* para α (ou simplesmente uma *atribuição* para α) é uma função a: $\mathbf{P} \Rightarrow \{F,V\}$.

Seja a uma atribuição de valores-verdade. A *função de avaliação* para $L(\alpha)$ *induzida* por a é a função v: $L(\alpha) \Rightarrow \{F,V\}$ definida da seguinte forma (Obs.: Note que a *atribuição* é para símbolos proposicionais e a *avaliação* é para fórmulas):

$v(a) = a(a)$, se a é um símbolo proposicional

$v(\neg p) = V$, se $v(p) = F$
$= F$, se $v(p) = V$

$v(p \wedge q) = V$, se $v(p) = v(q) = V$
$= F$, em caso contrário

$v(p \vee q) = F$, se $v(p) = v(q) = F$
$= V$, em caso contrário

$v(p \rightarrow q) = F$, se $v(p) = V$ e $v(q) = F$
$= V$, em caso contrário

$v(p \equiv q) = V$, se $v(p) = v(q)$
$= F$, em caso contrário

Sejam **P** e **Q** conjuntos de fórmulas em $L(\alpha)$ e r uma fórmula em $L(\alpha)$.

a. r é *verdadeira* em uma atribuição de valores-verdade a se e somente se $v(r) = V$. Em caso contrário, r é *falsa*.
b. r é uma *tautologia* se e somente se, para toda atribuição de valores-verdade a, $v(r) = V$. Isso corresponde a uma coluna inteira de V (verdadeiro) na tabela-verdade.
c. uma atribuição de valores-verdade a *satisfaz* **P**, ou a é um *modelo* para **P**, se e somente se, para toda fórmula s em **P**, $v(s) = V$. Isso corresponde a uma linha inteira de V (verdadeiro) na tabela-verdade.
d. **P** é *satisfazível* se e somente se existe uma atribuição de valores-verdade a que satisfaz **P**. Em caso contrário, **P** é *insatisfazível*.

e. r é uma consequência *lógica* de **P**, ou **P** *implica logicamente* r (notação: **P** |= r), se e somente se, para toda atribuição de valores-verdade *a*, *a* satisfaz **P** então *a* satisfaz r.
f. **P** é *tautologicamente equivalente* a **Q** (notação: **P** |=| **Q**) se e somente se toda fórmula de **Q** for consequência lógica de **P** e vice-versa.

3.2.3.1 O método da tabela-verdade

O método da tabela-verdade baseia-se na observação de que, se **P** é um conjunto finito de fórmulas e q é uma fórmula, há um número finito de símbolos proposicionais ocorrendo em q e nas fórmulas em **P**. Logo, há um número finito de atribuições de valores-verdade distintas para esses símbolos. Assim, para decidir se q é uma consequência lógica de **P**, basta enumerar todas essas atribuições e, para cada uma delas que satisfizer todas as fórmulas em **P**, testar se ela também satisfaz q. Se essa condição for sempre observada, então se pode afirmar que q é uma consequência lógica de **P**. Note que, se *a* não satisfizer alguma fórmula em **P**, o valor que *a* atribui a q não importará, pela forma como a implicação lógica foi definida. Esse método também pode ser usado para decidir se uma fórmula é satisfazível ou se dois conjuntos finitos de fórmulas são tautologicamente equivalentes.

EXEMPLO Seja **P** o seguinte conjunto de fórmulas:

1. $a \rightarrow \neg b$
2. $b \wedge a$
3. b,

seja **Q** o seguinte conjunto de fórmulas:

1. $a \vee b$
2. $b \rightarrow a$

e seja r a fórmula $\neg b \rightarrow a$.

Veja a tabela-verdade para esses conjuntos:

		1	2	3	4	5	6
a	b	$a \rightarrow \neg b$	$b \wedge a$	b	$a \vee b$	$b \rightarrow a$	$\neg b \rightarrow a$
F	*F*	*V*	*F*	*F*	*F*	*V*	*F*
F	*V*	*V*	*F*	*V*	*V*	*F*	*V*
V	*F*	*V*	*F*	*F*	*V*	*V*	*V*
V	*V*	*F*	*V*	*V*	*V*	*V*	*V*

Observe que, para o conjunto **P** (colunas 1, 2 e 3), não existe nenhuma atribuição (linha da tabela-verdade) que o satisfaça, ou seja, não existe nenhuma atribuição que resulta sempre em verdadeiro para **P**. Logo, **P** é insatisfazível. Já para o conjunto **Q** (colunas 4 e 5), existem duas atribuições (a = *V*, b = *F* e a = *V*, b = *V*) que satisfazem esse conjunto, logo **Q** é satisfazível. Como a fórmula r (coluna 6) também é verdadeira para as duas atribuições que satisfazem **Q**, diz-se que r é consequência lógica de **Q**, ou que **Q** implica logicamente r. Observe também que **P** e **Q** não são tautologicamente equivalentes.

3.3 LÓGICA DE PRIMEIRA ORDEM

A lógica de primeira ordem, ou cálculo de predicados de primeira ordem (CPPO), pode ser caracterizada como um sistema formal apropriado à definição de teorias do universo de discurso da Matemática. A motivação para se estudar essa lógica é que a lógica sentencial não dá conta da representação de frases do tipo:

"Sócrates é homem."
"Platão é homem."
"Todos os homens são mortais."

3.3.1 Sintaxe das linguagens de primeira ordem

Um *alfabeto de primeira ordem* α consiste em:

símbolos lógicos:
pontuação: (,)
conectivos: $\neg, \wedge, \vee, \rightarrow, \equiv$
quantificadores: $\forall$ (quantificador universal)
$\exists$ (quantificador existencial)

variáveis: um conjunto de símbolos distintos dos demais, por convenção, representadas por letras maiúsculas: *X*, *Y*, *Z* etc.

símbolo de igualdade (opcional): =

símbolos não lógicos: Um conjunto, possivelmente vazio, de *constantes* distintas dos demais símbolos, por convenção, representadas por letras minúsculas: a, b, c etc.

Para cada n > 0, um conjunto, possivelmente vazio, de *símbolos funcionais n-ários* distintos dos demais símbolos (o símbolo funcional representa a função da computação, que efetua um cálculo e retorna um valor), por convenção, representados pelas letras minúsculas a partir da letra f (de funcional): f, g, h etc.

Para cada n > 0, um conjunto, possivelmente vazio, de *símbolos predicativos n-ários* distintos dos demais símbolos (o símbolo predicativo, ou *predicado*, representa uma relação entre

objetos, ou seja, sua avaliação será verdadeira ou falsa), por convenção, representados por letras minúsculas: p, q, r etc.

Todas as definições a seguir fazem referência ao alfabeto de primeira ordem α.

O conjunto de *termos de primeira ordem* (ou simplesmente *termos*) é o menor conjunto satisfazendo as seguintes condições:

i. toda variável é um termo;
ii. toda constante é um termo;
iii. se $t_1, ..., t_n$ são termos e f é um símbolo funcional n-ário, então $f(t_1, ..., t_n)$ também é um termo.

O conjunto de *fórmulas* é o menor conjunto satisfazendo as seguintes condições:

i. se $t_1, ..., t_n$ são termos e p é um símbolo predicativo n-ário, então $p(t_1, ..., t_n)$ é uma fórmula, chamada de *fórmula atômica*. (Observe que os argumentos do predicado são termos, ou seja, podem ser constantes, variáveis ou símbolos funcionais, mas *nunca* outros predicados.)
ii. se $t_1, ..., t_n$ são termos e "=" é um símbolo de a, então $(t_1 = t_2)$ é uma fórmula, também chamada de *fórmula atômica*.
iii. se p e q são fórmulas, então $(\neg p)$, $(p \wedge q)$, $(p \vee q)$, $(p \rightarrow q)$ e $(p = q)$ também são fórmulas.
iv. se p é uma fórmula e X é uma variável, então $\forall X$ (p) e $\exists X$ (p) também são fórmulas.

Uma fórmula q é uma *subfórmula* de uma fórmula p se e somente se q é uma subcadeia de p.

A *linguagem de primeira ordem*, denotada por $L(\alpha)$, é o conjunto de termos e fórmulas de primeira ordem.

a. Em uma fórmula da forma $\forall X$ (q) (ou da forma $\exists X$ (q)), q é o *escopo* de $\forall X$ (ou de $\exists X$).
b. Uma ocorrência de uma variável X em uma fórmula p é *ligada* em p, se a ocorrência se dá em uma subfórmula de p da forma $\forall X$ (q) ou da forma $\exists X$ (q). Caso contrário, a ocorrência de X é *livre*. Ou seja, a ocorrência de X é ligada se X ocorrer dentro do escopo de seu quantificador.
c. Uma variável X é livre em p se existe uma ocorrência livre de X em p.
d. Uma fórmula p é uma *sentença* se e somente se nenhuma variável ocorre livre em p. Por exemplo, $\forall X$ $(a(X) \wedge b(Y)) \rightarrow c(X)$ não é uma sentença, pois a variável Y ocorre livre (não existe quantificador para ela, e a segunda ocorrência de X também é livre (está fora do escopo de $\forall X$).

a. Dada uma fórmula p, com variáveis livres $x_1, ..., x_n$, o *fecho universal* de p é a fórmula $\forall x_1 ... \forall x_n$ (p) e o *fecho existencial* de p é a fórmula $\exists x_1 ... \exists x_n$ (p).
b. Uma fórmula p está na *forma normal prenex* se e somente se p for da forma q(M), em que

q, o *prefixo* de p, é uma cadeia de quantificadores e M, a *matriz* de p, é uma fórmula sem ocorrências de quantificadores. Ou seja, forma normal prenex é quando se têm todos os quantificadores à esquerda da fórmula. Por exemplo, a fórmula, que é uma sentença, $\forall X \exists Y (a(X,Y) \rightarrow b(X))$ está na forma normal prenex, mas a fórmula, que também é uma sentença, $\forall X\, a(X) \rightarrow \exists Y\, b(Y)$ não está.

c. Uma fórmula p é uma *conjunção* se e somente se, omitindo-se os parênteses, for da forma $P_1 \wedge ... \wedge P_n$.

d. Uma fórmula p é uma *disjunção* se e somente se, omitindo-se os parênteses, for da forma $P_1 \vee ... \vee P_n$.

e. Uma fórmula p está em *forma normal conjuntiva* se e somente se estiver em forma normal prenex e a sua matriz for uma conjunção de disjunções de fórmulas atômicas, negadas ou não. Por exemplo, $\forall X \exists Y ((a(X,Y) \vee b(X)) \wedge (b(Y) \vee a(Y,Y))$, está na forma normal conjuntiva. E $\forall X \exists Y (a(X,Y) \vee b(X))$ também está. E $\forall X \exists Y (a(X,Y) \wedge b(Y))$ também, assim como $\forall X \exists Y (a(X,Y))$. Por quê?

*Regra de Inferência (**Modus Ponens**):*

MP: a partir de p e de (p → q), deduza q.

3.4 NOTAÇÃO CLAUSAL

a. Um *literal positivo* é uma fórmula atômica.

b. Um *literal negativo* é a negação de uma fórmula atômica.

c. Um *literal* é ou um literal positivo ou um literal negativo.

d. Dois literais têm *sinais opostos* se e somente se um deles for positivo e o outro for negativo.

e. Dois literais são *complementares* se e somente se um deles for a negação do outro.

f. Uma fórmula atômica *F* é o *átomo* de um literal L, denotado por |L|, se e somente se L for *F* ou ¬*F*.

Uma *cláusula* é ou uma sequência não vazia de literais ou a *cláusula vazia*, denotada por "[]".

A *linguagem de cláusulas* é o conjunto de todas as cláusulas.

Uma interpretação I *satisfaz* uma cláusula não vazia c (denotado por I |= c) se e somente se I satisfaz a sentença *F* definida como

$$\forall X_1 ... \forall X_m (L_1 \vee ... \vee L_n)$$

em que $X_1, ..., X_m$ são as variáveis ocorrendo em c e $L_1, ..., L_n$ são os literais de c. Diz-se ainda que c e *F* são *equivalentes*. Por convenção, a cláusula vazia é sempre insatisfazível. Com essa definição, está-se querendo dizer que, apesar de a cláusula, por definição, ser

uma sequência de literais, esses literais estão ligados através de disjunções, muito embora alguns autores não as tornem explícitas na sua representação. Pode-se dizer então que uma *cláusula é uma disjunção de literais.*

3.4.1 Representação clausal de fórmulas

Um conjunto de cláusulas **S** é uma *representação clausal* para uma fórmula p se e somente se p é satisfazível se e somente se **S** é satisfazível.

A obtenção da representação clausal de uma fórmula é um processo mecânico, como descrito a seguir:

Algoritmo 3.1 Algoritmo de Representação Clausal

Entrada: uma fórmula p
Saída: uma representação clausal **S** para p

1. Tome o fecho existencial de p.

   Se p contiver uma variável livre X, substitua p por $\exists X(p)$. Repita esse passo até que a fórmula não tenha variáveis livres.

2. Elimine quantificadores redundantes.

   Elimine todo quantificador "$\forall X$" ou "$\exists X$" que não contenha nenhuma ocorrência livre de X no seu escopo. (Ou seja, elimine todo quantificador desnecessário.)

3. Renomeie variáveis quantificadas mais de uma vez.

   Se houver dois quantificadores governando a mesma variável, substitua a variável de um deles e todas as suas ocorrências livres no escopo do quantificador por uma nova variável que não ocorra na fórmula. Repita o processo até que todos os quantificadores governem variáveis diferentes.

4. Elimine os conectivos "$\rightarrow$" e "$\equiv$".

   Substitua:

$(q \rightarrow r)$	por	$(\neg q \vee r)$
$(q \equiv r)$	por	$(\neg q \vee r) \wedge (q \vee \neg r)$
$\neg(q \rightarrow r)$	por	$(q \wedge \neg r)$
$\neg(q \equiv r)$	por	$(q \wedge \neg r) \vee (\neg q \wedge r)$

5. Mova "$\neg$" para o interior da fórmula.

Até que cada ocorrência de "$\neg$" preceda imediatamente uma fórmula atômica, substitua:

$\neg\forall X(q)$ por $\exists X(\neg q)$
$\neg\exists X(q)$ por $\forall X(\neg q)$
$\neg(q \wedge r)$ por $(\neg q \vee \neg r)$
$\neg(q \vee r)$ por $(\neg q \wedge \neg r)$
$\neg\neg q$ por q

6. Mova os quantificadores para o interior da fórmula. Objetivo: diminuir os escopos dos quantificadores.

Substitua, caso X não seja livre em r,
$\forall X(q \vee r)$ por $\forall X(q) \vee r$
$\forall X(r \vee q)$ por $r \vee \forall X(q)$
$\forall X(q \wedge r)$ por $\forall X(q) \wedge r$
$\forall X(r \wedge q)$ por $r \wedge \forall X(q)$
$\exists X(q \vee r)$ por $\exists X(q) \vee r$
$\exists X(r \vee q)$ por $r \vee \exists X(q)$
$\exists X(q \wedge r)$ por $\exists X(q) \wedge r$
$\exists X(r \wedge q)$ por $r \wedge \exists X(q)$

7. Elimine os quantificadores existenciais.

Seja q a fórmula corrente. Crie a nova fórmula corrente substituindo a subfórmula de q da forma "$\exists Y(r)$", que se situa mais à esquerda, por "$r[Y/f(x_1, ..., x_n)]$", em que $x_1, ..., x_n$ é uma lista de todas as variáveis livres de "$\exists Y(r)$" e "f" é qualquer símbolo funcional n-ário que não ocorre em q. Quando não houver variáveis livres em "$\exists Y(r)$", substitua "$\exists Y(r)$" por "$r[Y/c]$", em que "c" é uma constante que não ocorre em q. Repita o processo até que todos os quantificadores existenciais tenham sido eliminados.

Na verdade, deve-se pegar o escopo do quantificador existencial e verificar se além da variável quantificada existencialmente existe alguma variável ocorrendo livre nesse escopo. Se existir, substitui-se a variável existencial por uma função de todas as variáveis que ocorrem livres. Se não, substitui-se a variável existencial por uma constante. A explicação para esse procedimento pode ser entendida através do seguinte exemplo. Suponha a sentença ambígua da língua natural:
"Todo homem ama uma mulher."
Essa sentença tem pelo menos duas leituras possíveis. Na primeira, cada homem ama a sua mulher, portanto, para cada homem (são todos) existe uma mulher que ele ama. Já numa outra leitura, existe uma única mulher, que é representada por uma constante a, que todos os homens amam. Logo as leituras gerariam as seguintes fórmulas:
$\forall X \exists Y ((h(X) \wedge m(Y)) \rightarrow a(X, Y))$
$\exists Y \forall X ((h(X) \wedge m(Y)) \rightarrow a(X, Y))$
Observe que a diferença das interpretações está apenas nos escopos dos quantificadores. A primeira fórmula gerará uma representação, em que Y será substituída por um $f(X)$, ou seja, cada homem tem a sua mulher, portanto Y, que é mulher, é função de X, que é homem. Na segunda fórmula, o Y será substituído por uma constante, pois é a mesma mulher que todos os homens amam.

Exemplo: Na fórmula $\exists X \forall Y ((p(X) \wedge q(Y))$, substitui-se a variável X por uma constante a, pois no escopo do quantificador existencial, que é $\forall Y ((p(X) \wedge q(Y))$, nenhuma variável além do X ocorre livre. Logo, a fórmula fica $\forall Y ((p(a) \wedge q(Y))$. Já na fórmula $\forall Y \exists X (p(X) \vee q(Y))$, substitui-se X por $f(Y)$, pois no escopo do quantificador existencial, que é $(p(X) \vee q(Y))$, além do X, o Y ocorre livre, logo substitui-se a variável X por uma função da variável livre Y, ou seja, $f(Y)$. Assim, a fórmula fica $\forall Y (p(f(Y)) \vee q(Y))$.

Os novos símbolos funcionais assim introduzidos são chamados de *funções de Skolem*, e o processo de substituição é chamado de *skolemização*.

8. Obtenha a forma normal prenex.
   Mova os quantificadores universais para a esquerda.

9. Obtenha a forma normal conjuntiva.
   Até que a matriz da fórmula seja uma conjunção de disjunções, substitua:

   $(q \wedge r) \vee s$ por $(q \vee s) \wedge (r \vee s)$
   $q \vee (r \wedge s)$ por $(q \vee r) \wedge (q \vee s)$

10. Simplifique (opcional)
    Transforme a fórmula q resultante do passo 9 em outra mais simples s tal que s ainda esteja em forma normal conjuntiva (sem quantificadores existenciais) e q seja satisfazível se e somente se s o for.

11. Obtenha a representação clausal
    A representação clausal **S** da fórmula inicial p será o conjunto das cláusulas da forma $L_1 \ldots L_n$, tais que $L_1 \vee \ldots \vee L_n$ é uma disjunção da matriz da fórmula resultante do passo anterior. Cada cláusula deve ficar numa linha (disjunção de literais), e cláusulas em linhas diferentes pressupõem conjunção de cláusulas.

EXERCÍCIO RESOLVIDO

3.1 Seja o seguinte problema das casas:

- Há cinco casas de cinco diferentes cores.
- Em cada casa mora uma pessoa de uma diferente nacionalidade.
- Esses cinco proprietários bebem diferentes bebidas, fumam diferentes tipos de cigarro e têm um certo, e diferente dos demais, animal de estimação.
- Nenhum deles tem o mesmo animal, nem fuma o mesmo cigarro e nem bebe a mesma bebida.

A questão é: Quem tem um peixe?

Dicas:

1. O inglês vive na casa vermelha.
2. O sueco tem cachorros como animais de estimação.
3. O dinamarquês bebe chá.

4. A casa verde fica à esquerda da casa branca.
5. O dono da casa verde bebe café.
6. A pessoa que fuma Pall Mall cria pássaros.
7. O dono da casa amarela fuma Dunhill.
8. O homem que vive na casa do centro bebe leite.
9. O norueguês vive na primeira casa.
10. O homem que fuma Blends vive ao lado do que tem gatos.
11. O homem que cria cavalos vive ao lado do que fuma Dunhill.
12. O homem que fuma Bluemaster bebe cerveja.
13. O alemão fuma Prince.
14. O norueguês vive ao lado da casa azul.
15. O homem que fuma Blend é vizinho do que bebe água.

Resolução

Predicados p com seis argumentos: P: posição da casa; Co: cor da casa; N: nacionalidade do proprietário; B: bebida preferida; Ci: cigarro; A: animal. Preenchem-se os argumentos dos predicados através de associações (espécie de casamento de padrões). Para simplificar a representação e facilitar o entendimento, vai-se usar uma tabela:

Posição	1	2	3	4	5
Cor					
Nacionalidade					
Bebida					
Cigarro					
Animal					

Começa-se pela sentença 8, pois ela não é relacionada com nenhuma outra:

8. O homem que vive na casa do centro bebe leite.

p (3, Co3, N3, leite, Ci3, A3)
pois 3 é a posição do centro (5 casas)

Posição	1	2	3	4	5
Cor					
Nacionalidade					
Bebida			leite		
Cigarro					
Animal					

A sentença 9 também não depende de outras:

9. O norueguês vive na primeira casa.

p (1, Co1, norueguês, B1, Ci1, A1)

Posição	1	2	3	4	5
Cor					
Nacionalidade	norueguês				
Bebida			leite		
Cigarro					
Animal					

Posso pegar as sentenças que dizem respeito aos itens já levantados, leite ou norueguês:

14. O norueguês vive ao lado da casa azul.

p (2, azul, N2, B2, Ci2, A2)

Posição	1	2	3	4	5
Cor		azul			
Nacionalidade	norueguês				
Bebida			leite		
Cigarro					
Animal					

As sentenças 4 e 5 fazem referências à casa verde. Como ela fica à esquerda da casa branca (sentença 4), não pode ser nem a posição 1 nem a 2 (que é azul). Como o seu dono bebe café (sentença 5), não pode ser a posição 3 (cujo dono bebe leite — veja sentença 8). Logo, só sobrou a posição 4:

4. A casa verde fica à esquerda da casa branca.

5. O dono da casa verde bebe café.

p (4, verde, N4, café, Ci4, A4)
p (5, branca, N5, B5, Ci5, A5)

Posição	1	2	3	4	5
Cor		azul		verde	branca
Nacionalidade	norueguês				
Bebida			leite	café	
Cigarro					
Animal					

Pegando agora a sentença 1:

1. O inglês vive na casa vermelha.

Só sobrou a posição 3 para o vermelho, logo pode-se atualizar a posição 3:
p (3, vermelho, inglês, leite, Ci3, A3)

Posição	1	2	3	4	5
Cor		azul	vermelha	verde	branca
Nacionalidade	norueguês		inglês		
Bebida			leite	café	
Cigarro					
Animal					

Só falta a amarela, que é a de posição 1. Com a sentença

7. O dono da casa amarela fuma Dunhill.

atualiza-se a posição 1:
p (1, amarelo, norueguês, B1, Dunhill, A1)

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês		inglês		
Bebida			leite	café	
Cigarro	Dunhill				
Animal					

Depois a sentença 11, que se associa a Dunhill:

11. O homem que cria cavalos vive ao lado do que fuma Dunhill.

Como quem fuma Dunhill está na casa 1, o que está ao lado só pode estar na 2:
p (2, azul, N2, B2, Ci2, cavalo)

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês		inglês		
Bebida			leite	café	
Cigarro	Dunhill				
Animal		cavalo			

Com relação à sentença

3. O dinamarquês bebe chá.

só há duas possibilidades: ou ele está na casa 2 ou na 5: escolhe-se a 2 (caso não se chegasse ao final, dever-se-ia fazer um *backtracking* para esse ponto e escolher a 5).

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês	dinamarquês	inglês		
Bebida		chá	leite	café	
Cigarro	Dunhill				
Animal		cavalo			

Agora a sentença

12. O homem que fuma Bluemaster bebe cerveja.

Esse homem só pode estar na casa 5:

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês	dinamarquês	inglês		
Bebida		chá	leite	café	cerveja
Cigarro	Dunhill				Bluemaster
Animal		cavalo			

Agora a sentença

13. O alemão fuma Prince.

diz que o alemão só pode estar na casa 4, e, por consequência, a casa 5 é ocupada pelo sueco:

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês	dinamarquês	inglês	alemão	sueco
Bebida		chá	leite	café	cerveja
Cigarro	Dunhill			Prince	Bluemaster
Animal		cavalo			

Com a sentença

15. O homem que fuma Blends é vizinho do que bebe água.

conclui-se que esse homem está na casa 2 e que quem bebe água é o norueguês da casa 1:

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês	dinamarquês	inglês	alemão	sueco
Bebida	água	chá	leite	café	cerveja
Cigarro	Dunhill	Blends		Prince	Bluemaster
Animal		cavalo			

Agora é só ir preenchendo as outras posições:

2. O sueco tem cachorros como animais de estimação.

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês	dinamarquês	inglês	alemão	sueco
Bebida	água	chá	leite	café	cerveja
Cigarro	Dunhill	Blends		Prince	Bluemaster
Animal		cavalo			cachorro

6. A pessoa que fuma Pall Mall cria pássaros.

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês	dinamarquês	inglês	alemão	sueco
Bebida	água	chá	leite	café	cerveja
Cigarro	Dunhill	Blends	Pall Mall	Prince	Bluemaster
Animal		cavalo	pássaro		cachorro

10. O homem que fuma Blends vive ao lado do que tem gatos.

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês	dinamarquês	inglês	alemão	sueco
Bebida	água	chá	leite	café	cerveja
Cigarro	Dunhill	Blends	Pall Mall	Prince	Bluemaster
Animal	gato	cavalo	pássaro		cachorro

Portanto, a resposta da pergunta "Quem tem um peixe?", por exclusão, é o alemão.

Posição	1	2	3	4	5
Cor	amarela	azul	vermelha	verde	branca
Nacionalidade	norueguês	dinamarquês	inglês	alemão	sueco
Bebida	água	chá	leite	café	cerveja
Cigarro	Dunhill	Blends	Pall Mall	Prince	Bluemaster
Animal	gato	cavalo	pássaro	PEIXE	cachorro

EXERCÍCIOS PROPOSTOS

3.1 Considere as seguintes frases:

a. Josualdo gosta de todos os alimentos.

b. Maçã é alimento.

c. Galinha é alimento.

d. Qualquer coisa que qualquer um coma e não morra é alimento.

e. Velásquez come amendoim e está vivo.

f. Solange come tudo o que Velásquez come.

Traduza essas frases em fórmulas da lógica de 1.ª ordem e converta estas fórmulas em cláusulas.

3.2 O que está errado com o seguinte argumento?

a. Os homens estão amplamente distribuídos sobre a Terra.

b. Sócrates é um homem.

c. Portanto, Sócrates está amplamente distribuído sobre a Terra.

Como os fatos representados por essas frases devem ser colocados na lógica para que esse problema não surja?

3.3 Converta as seguintes sentenças em forma clausal:

a. $\forall X \forall Y\, (\neg q(X,Y) \rightarrow \neg p(X,Y))$

b. $\forall X \forall Y\, (p(X,Y) \rightarrow (q(X,Y) \wedge r(X,Y)))$

c. $\forall X \forall Y\, (p(X,Y) \rightarrow q(X,Y))$

d. $\neg \forall X \exists Y\, (p(X,Y) \rightarrow q(X,Y))$

e. $\neg \forall X\, (p(X) \rightarrow (\forall Y\, (p(Y) \rightarrow p(f(X,Y))) \wedge \neg \forall Y\, (q(X,Y) \rightarrow p(Y))))$

f. $\forall X\, (p(X) \rightarrow p(X))$

g. $\neg \forall X\, p(X) \rightarrow \exists X\, \neg p(X)$

3.4 Considere a interpretação a em lógica proposicional dada por:

$$a(p) = F \text{ e } a(q) = a(r) = V$$

Qual o valor verdade da fórmula a seguir segundo essa interpretação?

$$(\neg p \wedge q) \vee (p \rightarrow (q \vee r))$$

3.5 Mostre que cada uma das seguintes fórmulas é uma tautologia:

a. $(p \rightarrow q) \rightarrow ((r \vee p) \rightarrow (r \vee q))$

b. $(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$

3.6 Mostre que:

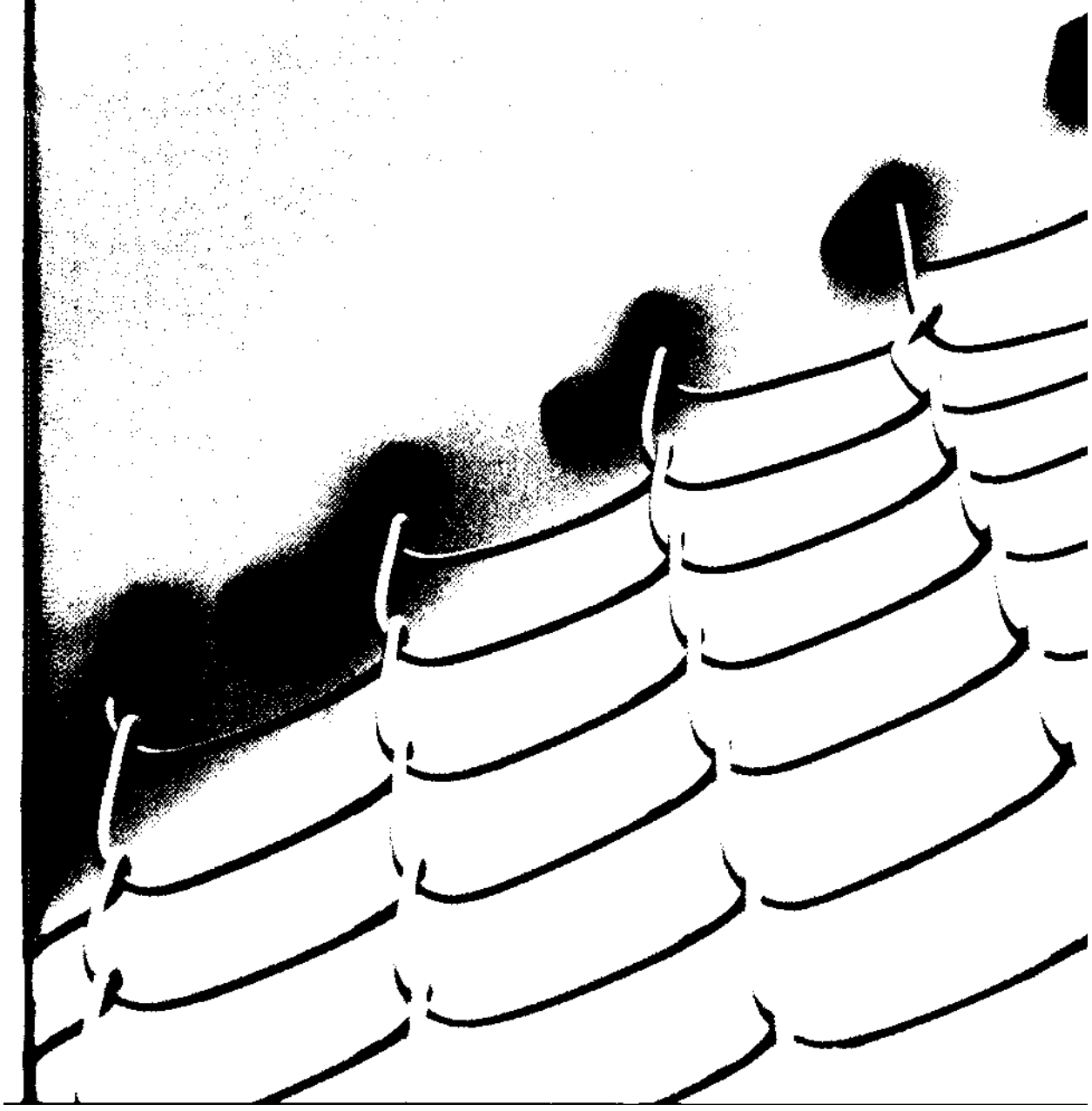
a. $(p \rightarrow (q \rightarrow r)) \equiv ((p \wedge q) \rightarrow r)$

b. $(p \rightarrow q) \equiv (\neg q \rightarrow \neg p)$

c. $(p \vee (\neg q \vee r)) \equiv ((\neg p \wedge q) \rightarrow r)$

d. $((p \rightarrow q) \rightarrow p) \equiv p$

CAPÍTULO 4



Prova Automática de Teoremas

4.1 REPRESENTAÇÃO DO CONHECIMENTO

Vai-se explorar o uso da lógica de predicados como uma forma de representar conhecimento. Considere o seguinte conjunto de sentenças (Rich e Knight, 1994):

1. Marco era um homem.
2. Marco era um pompeiano.
3. Todos os pompeianos eram romanos.
4. César era um soberano.
5. Todos os romanos ou eram leais a César ou o odiavam.
6. Todos são leais a alguém.
7. As pessoas somente tentam assassinar soberanos aos quais elas não são leais.
8. Marco tentou assassinar César.

Os fatos descritos por estas sentenças podem ser representados como um conjunto de fórmulas bem formadas (fbfs) na lógica de predicados como se segue:

1. Marco era um homem.
   $homem(marco)$

Essa representação captura o fato crítico de *Marco* ser um homem. Não há, entretanto, a informação de tempo verbal. Mas para esse exemplo isso não é relevante.

2. Marco era um pompeiano.
   $pompeiano(marco)$

3. Todos os pompeianos eram romanos.
   $\forall X\ (pompeiano(X) \rightarrow romano(X))$

4. César era um soberano.
   $soberano(césar)$

5. Todos os romanos ou eram leais a César ou o odiavam.
   $\forall X\ (romano(X) \rightarrow (leal(X,césar) \vee odiar(X,césar)))$

6. Todos são leais a alguém.
   $\forall X \exists Y\ leal(X,Y)$

7. As pessoas somente tentam assassinar soberanos aos quais elas não são leais.
   $\forall X \forall Y\ ((pessoa(X) \wedge tentarassassinar(X,Y) \wedge soberano(Y)) \rightarrow \neg leal(X,Y))$

8. Marco tentou assassinar César.
tentarassassinar(marco,césar)

Através desse breve exemplo, foi possível perceber que não é tão simples passar sentenças do português para a forma de comandos lógicos. Suponha que se deseje usar esses comandos para responder à questão

"Marco era leal a César?"

Parece que usando 7 e 8 dá para concluir que Marco não era leal a César (ignorando a distinção entre passado e presente). Agora, vai-se tentar produzir uma prova formal, raciocinando "para trás" (*encadeamento regressivo* — ver Capítulo 5), a partir da meta desejada:

¬leal(marco,césar)

A fim de provar a meta, necessita-se de regras de inferência para transformá-la em outra meta (ou possivelmente um conjunto de metas), que pode, por sua vez, ser transformada, e assim por diante, até que não haja mais metas insatisfeitas. A Figura 4.1 mostra uma tentativa para produzir uma prova da meta, reduzindo o conjunto de metas até o conjunto vazio.

O problema é que, ainda que se saiba que Marco era um homem, não há maneira de concluir que Marco era uma pessoa. Precisa-se adicionar a representação de um outro fato ao sistema:

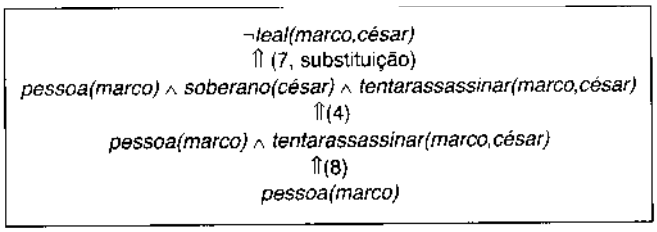


Figura 4.1 Uma tentativa de prova para "¬leal(marco,césar)".

9. Todos os homens são pessoas.

$\forall X\ (homem(X) \rightarrow pessoa(X)$

Agora dá para satisfazer a última meta e produzir uma prova de que Marco não era leal a César.

Desse exemplo simples podem-se perceber três pontos importantes, na conversão de sentenças do português a comandos lógicos:

- Muitas sentenças do português são ambíguas (por exemplo, 5, 6 e 7 anteriores). A escolha da interpretação correta pode ser difícil.
- Existe frequentemente uma escolha de como representar o conhecimento. Representações simples são desejáveis, mas elas podem impedir certos tipos de raciocínio.
- Mesmo em situações muito simples, um conjunto de sentenças não parece conter toda a informação necessária para raciocinar sobre o tópico em questão. Para ser capaz de usar um conjunto de comandos efetivamente, é muitas vezes necessário ter acesso a um

outro conjunto de comandos que representam fatos considerados óbvios demais para mencionar.

Um outro problema surge em situações em que não se conhecem de antemão quais comandos deduzir. No exemplo apresentado, o objeto era responder à questão "Marco era leal a César?". Como um programa poderia decidir se ele deveria tentar provar:

leal(marco,césar)

ou

¬leal(marco,césar)

4.1.1 Funções e predicados computáveis

No exemplo visto, todos os fatos simples eram expressos como combinações de predicados individuais, tais como:

tentarassassinar(marco,césar)

Isso é interessante se o número de fatos não é muito grande ou se os fatos são suficientemente desestruturados. Mas suponha que se deseja expressar fatos simples, tais como os seguintes relacionamentos "maior-que" e "menor-que":

ma(1,0)	*me*(0,1)
ma(2,1)	*me*(1,2)
ma(3,2)	*me*(2,3)
	...

É óbvio que não é desejável ter que escrever a representação de cada um desses fatos individualmente. Nesse caso é útil ampliar a representação usando os *predicados computáveis*.

É também interessante ter *funções computáveis*, assim como os predicados. Então pode-se calcular a verdade de

ma(mais (2,3),1)

Para isso, é necessário primeiro calcular o valor da função *mais*, dados os argumentos 2 e 3, e então enviar os argumentos 5 e 1 a *ma*.

O próximo exemplo mostra como essas ideias de funções e predicados computáveis podem ser úteis. Ele também faz uso da noção de igualdade e permite que objetos iguais sejam substituídos um pelo outro quando isso for interessante durante uma prova.

Considere o seguinte conjunto de fatos, novamente envolvendo Marco:

1. Marco era um homem.
   homem(marco)

2. Marco era um pompeiano.
   pompeiano(marco)

3. Marco nasceu em 40 d.C.
   $nasceu(marco,40)$

4. Todos os homens são mortais.
   $\forall X\ (homem(X) \rightarrow mortal(X))$

5. Todos os pompeianos morreram quando o vulcão entrou em erupção em 79 d.C.
   $erupcao(vulcão,79) \wedge \forall X\ (pompeiano(X) \rightarrow morreu(X,79))$

6. Nenhum mortal vive mais de 150 anos.
   $\forall X \forall T1 \forall T2((mortal(X) \wedge nasceu(X,T1) \wedge ma(T2\text{-}T1,150)) \rightarrow morto(X,T2))$

7. Agora é 2010.
   $agora = 2010$

8. Vivo significa não morto.
   $\forall X \forall T((vivo(X,T) \rightarrow \neg morto(X,T)) \wedge (\neg morto(X,T) \rightarrow vivo(X,T)))$

9. Se alguém morre, então ele está morto para sempre.
   $\forall X \forall T1 \forall T2((morreu(X,T1) \wedge ma(T2,T1)) \rightarrow morto(X,T2))$

1. homem(marco)
2. pompeiano(marco)
3. nascer(marco,40)
4. ∀X (homem(X) → mortal(X))
5. erupção(vulcão,79)
6. ∀X(pompeiano(X) → morreu(X,79))
7. ∀X∀T1∀T2((mortal(X) ∧ nasceu(X,T1) ∧ ma(T2-T1,150)) → morto(X,T2))
8. agora = 2010
9. ∀X∀T((vivo(X,T) → ¬morto(X,T)) ∧ (¬morto(X,T) → vivo(X,T)))
10. ∀X∀T1∀T2 ((morreu(X,T1) ∧ ma(T2,T1)) → morto(X,T2))

Figura 4.2 Um conjunto de fatos sobre "Marco".

Um conjunto de todos os fatos representados está na Figura 4.2. (A numeração mudou um pouco, pois a sentença 5 foi quebrada em duas partes.) Agora pode-se tentar responder à questão "Marco está vivo agora?" provando:

$$\neg vivo(marco,agora)$$

As provas são mostradas nas Figuras 4.3 e 4.4. O termo *nil* no final de cada prova indica que a lista de condições restantes a serem provadas está vazia, e, portanto, a prova foi bem-sucedida. Note que nessas provas, quando um comando da forma

$$(a \wedge b) \rightarrow c$$

foi usado, *a* e *b* são estabelecidos como submetas independentes. Pode-se satisfazer a meta

nasceu(marco,T1)

usando o comando 3 fazendo a ligação de *T1* a 40, mas deve-se também ligar *T1* a 40 em

ma(agora-T1,150)

já que os dois *T1*s são a mesma variável no comando 4, da qual as duas submetas vieram. Um bom procedimento de prova computacional deve incluir uma forma de determinar que uma unificação existe e também uma forma de garantir substituições uniformes durante a prova.

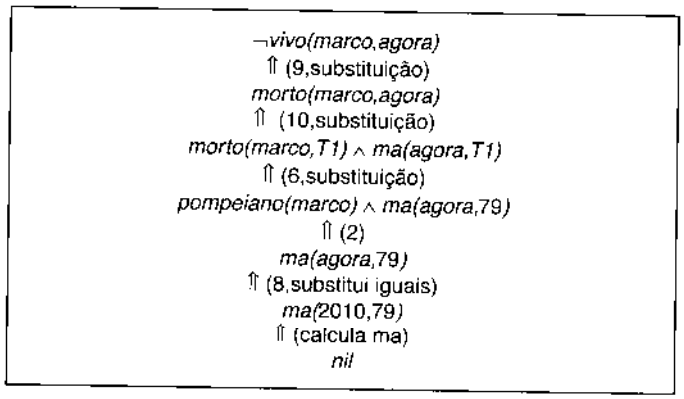


Figura 4.3 Uma forma de provar que "Marco Está Morto".

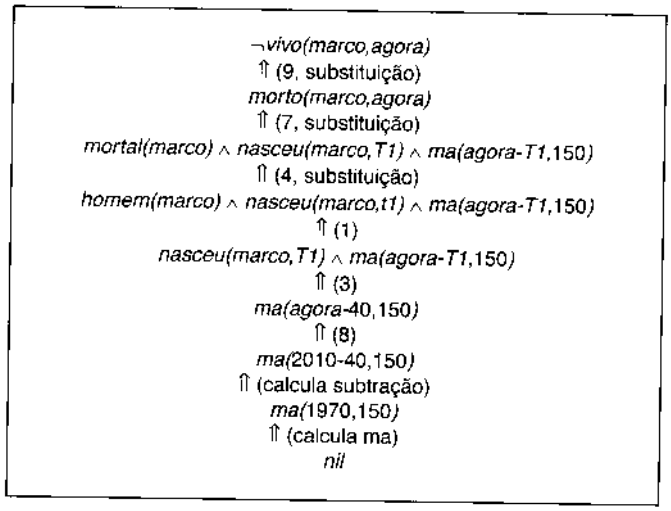


Figura 4.4 Uma outra forma de provar que "Marco Está Morto".

Olhando as provas mostradas, duas coisas devem ficar claras:

- Mesmo conclusões simples podem precisar de muitos passos para provar.
- Vários processos, tais como unificação, substituição e aplicação de *modus ponens* são envolvidos na produção de uma prova.

4.2 RESOLUÇÃO

4.2.1 Unificação

a. Um par (X,T) é uma *substituição simples* (lê-se "X substituído por T") se e somente se X é uma variável e T é um termo.

b. Um conjunto finito β de substituições simples é uma *substituição* se e somente se duas substituições simples em β não coincidem no primeiro elemento.

c. Uma substituição β é uma *substituição básica* se e somente se, para todo (X,T) em β, T é um termo sem ocorrências de variáveis.

d. β é a *substituição vazia* se e somente se β for o conjunto vazio.

e. Uma substituição β é uma *renomeação de variáveis* ou, simplesmente, uma *renomeação*, se e somente se cada par (X,T) em β for tal que T é uma variável e não existirem dois pares (X,U) e (Y,V) em β tais que $X \neq Y$ e $U = V$.

A expressão "X/T" denotará uma substituição simples (X,T). Letras gregas denotarão substituições, e, em especial, "ε" denotará a substituição vazia.

Uma *expressão* é qualquer sequência de símbolos de um alfabeto de primeira ordem, e uma *expressão simples* é qualquer literal ou termo sobre o alfabeto.

Sejam C uma cláusula e β uma substituição. A *instanciação* de C por β, denotada por Cβ, é a cláusula obtida instanciando-se C por β e eliminando-se as ocorrências repetidas do mesmo literal, exceto a ocorrência mais à esquerda.

A *composição* de substituições é a função, denotada por "$\circ$", que mapeia pares de substituições em uma substituição e é definida da seguinte forma:

Para todo par de substituições

$$\beta = \{X_1 / T_1, ..., X_n / T_n, Y_1 / S_1, ..., Y_k / S_k\}$$
$$\theta = \{Y_1 / R_1, ..., Y_k / R_k, Z_1 / Q_1, ..., Z_m / Q_m\}$$

em que $X_1, ..., X_n, Y_1, ..., Y_k, Z_1, ..., Z_m$ são variáveis distintas, a composição de β com θ será a substituição:

$$\beta \circ \theta = \{X_1 / (T_1)\theta, ..., X_n / (T_n)\theta, Y_1 / (S_1)\theta, ..., Y_k / (S_k)\theta, Z_1 / Q_1, ..., Z_m / Q_m\}$$

EXEMPLO Sejam $\beta = \{X/f(Y), Y/Z\}$ e $\theta = \{X/a, Y/b, Z/Y\}$. Então a composição das substituições β com θ é a substituição

$$\beta \circ \theta = \{X/f(b), Y/Y, Z/Y\}$$

que é obtida aplicando-se θ aos termos das substituições simples em β, formando o conjunto $\{X/f(b), Y/Y\}$ e acrescentando a única substituição simples de θ, "Z/Y", cujo primeiro elemento não coincide com o primeiro elemento de uma substituição simples em β.

Propriedades:

a. $\theta \circ \varepsilon = \varepsilon \circ \theta = \theta$
b. $(E\theta)\beta = E(\theta \circ \beta)$, qualquer que seja a expressão E
c. Se $E\theta = E\beta$ então $\theta = \beta$, qualquer que seja a expressão E
d. $(\theta \circ \beta) \circ \varphi = \theta \circ (\beta \circ \varphi)$

Sejam $C = l_1 l_2 \ldots$ uma cláusula com conjunto de literais $\mathbf{L} = \{l_1, l_2\}$ e β uma substituição.

(a) β é um *unificador* para **L** se e somente se $(l_1)\beta = (l_2)\beta$.

(b) β é um *unificador mais geral* (ou, abreviadamente, um *u.m.g.*) de **L** se e somente se β é um unificador de **L** e, para todo unificador θ de **L**, existe uma substituição φ tal que $\theta = \beta \circ \varphi$.

(c) O conjunto **L** é *unificável* se e somente se existe um unificador para **L**.

4.2.1.1 O algoritmo da unificação

Um conjunto de termos **D** é o *conjunto de discórdia* de um conjunto de literais $\mathbf{L} = \{l_1, \ldots, l_n\}$ se e somente se

i. $\mathbf{D} = \varnothing$, se $n = 1$;
ii. $\mathbf{D} = \{T_1, \ldots, T_n\}$, se $n > 1$ e todas as expressões em **L** são idênticas até o i-ésimo símbolo, exclusive, e T_j é o termo ocorrendo em **L** que começa no i-ésimo símbolo, para $j = 1, \ldots, n$.

a. Uma substituição simples X/T *satisfaz o teste de ocorrência* se e somente se X não ocorre em T.
b. Um conjunto de discórdia **D** *satisfaz o teste de ocorrência* se e somente se existem uma variável X e um termo T em **D** tais que X não ocorre em T.

EXEMPLO Sejam os seguintes conjuntos de literais e seus conjuntos de discórdia:

Conjunto de literais	Conjunto de discórdia
{p(a, X, f(g(Y))), p(Z, h(Z,W), f(W))}	{a,Z}
{p(a, X, f(g(Y))), p(a, h(a,W), f(W))}	{X, h(a,W)}
{p(a, h(a,W),f(g(Y))), p(a, h(a,W), f(W))}	{g(Y), W}

Em todos os exemplos anteriores, o conjunto de discórdia satisfaz o teste de ocorrência.

Algoritmo 4.1 Algoritmo da Unificação

Algoritmo para determinar um *u. m. g.*, caso exista, para um conjunto de expressões simples.

entrada: um conjunto **L** de literais
saída: • um u.m.g. β de **L**, se **L** for unificável
• 'NÃO', se **L** não for unificável

início

β := ε;
W := **L**;
D := conjunto de discórdia de W;
enquanto (número de literais de W) > 1 e **D** satisfaz o teste de ocorrência *faça*
início
selecione uma variável *X* e um termo *T* em **D** tais que *X* não ocorre em *T*;
β := β ° {*X*/*T*};
W := W{*X*/*T*};
D := conjunto de discórdia de W;
fim;
se (número de literais de W) = 1
então retorne β
senão retorne 'NÃO'
fim

EXEMPLO Determine um *u. m. g.* para o seguinte conjunto de literais:

$$\mathbf{L} = \{p(a,f(X)), p(Y,Z)\}$$

O primeiro passo consiste em localizar os termos mais à esquerda em que os literais diferem da seguinte forma:

- considere cada um dos literais em **L** simplesmente uma cadeia de símbolos. Compare os literais para localizar a posição *i* mais à esquerda em que diferem;
- extraia de cada um dos literais os termos que começam na posição *i*, formando o *conjunto de discórdia* dos literais.

No caso, os literais em **L** diferem na terceira posição. Logo, o conjunto de discórdia de **L** é:

$$\mathbf{D} = \{a, Y\}$$

Como **D** contém uma variável *Y* e um termo *a*, em que *Y* não ocorre, pode-se construir a substituição θ = {Y/a}, que, aplicada a **L**, torna os seus literais "mais semelhantes":

$$\mathbf{L}\theta = \{p(a,f(X)), p(a,Z)\}$$

Note que, por força da substituição, e por *Y* não ocorrer em *a*, os dois literais em **L**θ coincidem no primeiro termo. Repita agora o processo sobre **L**θ. O novo **D** será:

$$\mathbf{D'} = \{f(X), Z\}$$

Novamente, como **D'** contém uma variável *Z* e um termo f(X) em que *Z* não ocorre, pode-se construir β = {Z/f(X)}, que, aplicada a **L**θ, torna os seus literais idênticos:

$$(\mathbf{L}\theta)\beta = \{p(a,f(X))\}$$

O *u. m. g.* de **L** será então a composição de θ = {Y/a} com β = {Z/f(X)}:

$$\theta \circ \beta = \{Y/a, Z/f(X)\}$$

Dado um conjunto de literais, esse processo anterior consiste essencialmente em tornar idênticas as expressões, caminhando da esquerda para a direita, através de substituições sucessivas.

4.2.2 O que é resolução

O sistema formal da resolução trabalha exclusivamente com cláusulas e contém apenas uma regra de inferência, chamada de regra da resolução, que gera uma nova cláusula a partir de duas outras. Dados um conjunto **S** de cláusulas e uma cláusula C, uma dedução de C a partir de **S** nesse sistema formal consiste em uma sequência de cláusulas terminando em C e gerada aplicando-se repetidamente a regra da resolução. Uma refutação a partir de **S** é uma dedução da cláusula vazia a partir de **S**. A regra da resolução é definida de tal forma que **S** é insatisfazível se e somente se existe uma refutação a partir de **S**.

Convém recordar alguns pontos antes de prosseguir. Na definição da regra da resolução, tratar-se-á uma cláusula não vazia "$l_1 \ldots l_n$" como o conjunto finito $\{l_1, \ldots, l_n\}$ e a cláusula vazia "[]" como o conjunto vazio. Assim, utilizar-se-ão as operações usuais de teoria dos conjuntos para definir novas cláusulas a partir de outras. Por exemplo, se "l m n" e "n p" são cláusulas, a expressão "(l m n) ∪ (n p)" denota a cláusula "l m n p" (a ordem dos literais no resultado é irrelevante em face da semântica das cláusulas).

Uma cláusula A é uma *instância* de B se e somente se existir uma substituição $\beta = \{X_1/T_1, \ldots, X_n/T_n\}$ de variáveis por termos tal que A é obtida substituindo-se simultaneamente X_i por T_i em B, para $i = 1, \ldots, n$. Usar-se-á Bβ para denotar o resultado da substituição.

EXEMPLO Seja a cláusula B = p(*X*) q(*X*,*Y*). Seja uma substituição β = {*X*/a,*Y*/f(b)}. A instanciação de B por β, denotada por Bβ, é a cláusula instância A = p(a) q(a,f(b)).

A regra da resolução combina:

- uma adaptação para cláusulas da regra *Modus Ponens* (regra R1)

- um processo de "unificação" de literais de duas cláusulas (regra R2)
- um processo de "unificação" de literais de uma mesma cláusula (regra da resolução RE).

Para o primeiro passo, recorde que a regra *Modus Ponens* ditava que de p e de $p \rightarrow q$, pode-se derivar q ou, equivalentemente, de p e de $\neg p \vee q$ pode-se derivar q. Uma adaptação imediata de *Modus Ponens* para cláusulas seria então:

Regra R1: se A' possui um literal l e A" possui um literal $\neg$l, derive $A = (A' - l) \cup (A'' - \neg l)$.

Ou seja, A é uma lista de literais contendo, em qualquer ordem, os literais em A' e A", exceto l e $\neg$l.

EXEMPLO Seja o seguinte conjunto de cláusulas:

1. p(*X*) $\neg$q(*Y*)
2. q(*Y*) r(*Z*)

dá para obter, usando a regra R1, a cláusula

3. p(*X*) r(*Z*)

Agora, imagine o seguinte conjunto de cláusulas, parecido com o anterior:

1. p(*X*) $\neg$q(*Y*)
2. q(*W*) r(*Z*)

não é possível mais obter a cláusula 3, pois as variáveis são diferentes. A regra R2 vai resolver esse problema.

Regra R2: se A' possui um literal l' e A" possui um literal $\neg$l" e existe uma substituição β tal que $l'\beta = l''\beta$, derive $A = (A'\beta - l'\beta) \cup (A''\beta - \neg l''\beta)$.

Ou seja, A é uma lista de literais formada tomando-se, em qualquer ordem, os literais em A' e A", exceto l' e $\neg$l", e aplicando-se a substituição β a todos os literais restantes.

EXEMPLO Retomando o conjunto anterior

1. p(*X*) $\neg$q(*Y*)
2. q(*W*) r(*Z*)

existe uma substituição $\beta = \{W/Y\}$ que, aplicada às duas cláusulas, resulta no seguinte:

1. p(*X*) $\neg$q(*Y*)
2. q(*Y*) r(*Z*)

que obviamente produz a cláusula a seguir:

3. p(*X*) r(*Z*)

O processo de tornar idênticos os literais em uma cláusula C através de uma substituição de variáveis por termos é chamado de *unificação*, e a substituição é chamada de um *unificador* de C. Um *unificador mais geral* é aquele que, intuitivamente, especifica as substituições mais simples possíveis. O processo de unificação deverá então utilizar sempre um unificador mais geral para não bloquear outras unificações. Por exemplo, $\beta = \{X/f(a), Y/a\}$ e $\theta = \{X/f(Y)\}$ unificam $C = p(X)\ p(f(Y))$, pois $C\beta = p(f(a))$ e $C\theta = p(f(Y))$. Porém θ é um unificador mais geral do que β. A substituição θ é então preferível a β, pois, por exemplo, o literal "p(f(b))" é unificável com o literal em $C\theta$, mas não é unificável com o literal em $C\beta$.

Em geral, diz-se que uma cláusula B é um *fator* de uma cláusula A se e somente se existe um conjunto L de literais de A e existe um unificador mais geral β para L tal que $B = A\beta$. Note que uma cláusula A é um fator dela mesma. O processo de obter fatores de cláusulas é chamado de *fatoração*.

> *Regra RE*: se B' e B" são fatores de cláusulas A' e A" tais que B' possui um literal l' e B" um literal $\neg$l" e existe um unificador mais geral β para l' e l", derive $A = (B'\beta - l'\beta) \cup (B''\beta - \neg l''\beta)$

Nesse caso, diz-se que a cláusula A é um *resolvente* de A' e A", que são as *cláusulas pais*. A Tabela 4.1 mostra alguns casos interessantes de resolução.

EXEMPLO Seja o seguinte conjunto de cláusulas:

1. $p(X)\ q(Z)$
2. $\neg r(Y)\ p(T) \neg r(W)$
3. $r(V) \neg q(U)$

Se for usada a regra R1, nada será obtido, pois essa regra não permite substituições. Se for usada a regra R2, obtêm-se as seguintes cláusulas:

4. $p(T) \neg r(W) \neg q(U)$	R2: 2,3 $\beta = \{V/Y\}$
5. $p(X)\ p(T) \neg r(W)$	R2: 1,4 $\beta = \{Z/U\}$

e assim por diante. Mas se for usada a Regra da Resolução (RE), é possível simplificar a cláusula 2, fatorando-a:

2'. $\neg r(Y)\ p(T)$	fator de 2, com $\beta = \{W/Y\}$

e aí continua-se aplicando RE:

4. $p(T) \neg q(U)$	RE: 2',3 $\beta = \{Y/V\}$
5. $p(X)$	RE: 1, 4 $\beta = \{U/Z\}$ e fator de $p(X)\ p(T)$

Em resumo, o sistema formal de resolução trabalha apenas com cláusulas, que são objetos com uma sintaxe bem simples, e possui apenas uma regra de inferência, a regra da resolução. Um procedimento de *refutação baseado em resolução* é então um procedimento que, dado um con-

junto qualquer de cláusulas **S**, procura sistematicamente derivar a cláusula vazia utilizando apenas a regra da resolução e tendo como ponto de partida as cláusulas em **S**. A construção de tais procedimentos requer, porém, métodos especiais para tentar contornar a explosão combinatorial gerada pela liberdade de escolha de cláusulas, fatores e literais.

TABELA 4.1 Cláusulas e resolventes

Cláusulas pais	Resolventes(s)	Comentários
p e ¬p q (isto é, p → q)	q	*Modus Ponens*
p q e ¬p q	q	A cláusula q q se reduz a q
p q e ¬p ¬q	q ¬q ou p ¬p	Aqui, existem dois resolventes possíveis: ambos tautologias
¬p e p	[]	Esta cláusula vazia é sinal de contradição
¬p q (ou p → q) e ¬q r (ou q → r)	¬p r (ou p → r)	Encadeamento

4.2.3 O sistema formal da resolução

Uma *renomeação* para B em presença de A é uma renomeação de variáveis β tal que A e Bβ não possuem variáveis em comum. Recorde que, se l é um literal da forma p ou da forma ¬p, então p é o átomo de l, denotado por |l|.

Uma cláusula A é um *fator* de uma cláusula A' se e somente se existem um conjunto **L'** de literais de A' e um u.m.g. β de **L'** tais que A = A'β.

Uma cláusula A é um *resolvente binário* de cláusulas A' e A'' se e somente se existem literais l' e l'' de A' e A'', respectivamente, e uma substituição θ tais que
i. l' e l'' têm sinais opostos e θ é um u.m.g. de $\{|l'|,|l''|\}$
ii. $A = (A'\theta - l'\theta) \cup (A''\theta - l''\theta)$
Diz-se ainda que l' e l'' são os *literais resolvidos* e que θ é a *substituição de resolução*.

Portanto, por convenção, o resolvente A é formado:
- eliminando $l'\theta$ de $A'\theta$ e $l''\theta$ de $A''\theta$;
- listando os literais restantes de $A'\theta$ e $A''\theta$ em qualquer ordem;
- eliminando os literais repetidos.

Sejam A' e A'' cláusulas e β uma renomeação para A'' em presença de A'. Uma cláusula A é um *resolvente* de A' e A'' se e somente se A é um resolvente binário de fatores de A' e A''β.

O *sistema formal da resolução, RE*, consiste em:
Classe de Linguagens: linguagens de cláusulas
Axiomas: nenhum
Regra de Inferência: Regra da Resolução (RE)
RE: se A' e A'' são cláusulas e A é um resolvente de A' e A'', então derive A de A' e A''.

Sejam **S** um conjunto de cláusulas e C uma cláusula.

a. Uma *dedução* de C a partir de **S** no sistema formal da resolução ou, simplesmente, uma *R-dedução* de C a partir de S é uma sequência $\mathbf{D} = (D_1, ..., D_n)$ de cláusulas tal que:
   i. $D_n = C$
   ii. para todo $i \in [1, n]$, D_i pertence a **S** ou D_i é um resolvente de D_j e D_k, para algum $j, k < i$.
   Para cada $i \in [1, n]$, D_i é uma *cláusula de entrada* em **D** se e somente se D_i pertence a **S**; caso contrário, D_i é uma *cláusula derivada*.

b. Uma *refutação* a partir de **S** no sistema formal da resolução ou, simplesmente, uma *R-refutação* a partir de **S** é uma R-dedução de [] a partir de **S**.

4.3 REFUTAÇÃO POR RESOLUÇÃO

4.3.1 Introdução

No problema da prova de teorema tem-se um conjunto de fórmulas **S** a partir das quais se deseja provar alguma fórmula meta, W. Os sistemas baseados em resolução produzem provas por contradição, ou *refutação*. Em uma refutação por resolução, primeiro nega-se a fórmula meta e então adiciona-se a negação ao conjunto **S**. Esse conjunto expandido é então convertido a um conjunto de cláusulas, e usa-se resolução para derivar uma contradição, representada pela cláusula vazia, [].

Um simples argumento pode ser dado para justificar o processo de prova por refutação. Suponha uma fórmula W, que segue logicamente de um conjunto de fórmulas S; então, por definição, nenhuma interpretação que satisfaz **S** pode satisfazer ¬W, e, portanto, nenhuma interpretação pode satisfazer a união de **S** e {¬W}. Portanto, se W segue logicamente de S, o conjunto $\mathbf{S} \cup \{\neg W\}$ é insatisfazível.

Se a resolução é aplicada repetidamente a um conjunto de cláusulas insatisfazíveis, eventualmente a cláusula vazia, [], será produzida. Portanto, se W segue logicamente de **S**, então a resolução eventualmente produzirá a cláusula vazia a partir da representação da cláusula $\mathbf{S} \cup \{\neg W\}$. Por outro lado, se a cláusula vazia é produzida a partir da representação de cláusula $\mathbf{S} \cup \{\neg W\}$, então W segue logicamente de **S**.

Considere um exemplo simples. Observe as seguintes frases:

1. Qualquer um que possa ler é alfabetizado.
   $\forall X \ (l(X) \rightarrow a(X))$

2. Os golfinhos não são alfabetizados.
   $\forall X \ (g(X) \rightarrow \neg a(X))$

3. Alguns golfinhos são inteligentes.
   $\exists X \ (g(X) \wedge i(X))$

A partir deles quer-se provar a frase:

4. Alguns que são inteligentes não podem ler.
   $\exists X \ (i(X) \wedge \neg l(X))$

O conjunto de cláusulas que correspondem às frases 1 a 3 é:

1. ¬l(X) a(X)
2. ¬g(Y) ¬a(Y)

3a. g(*a*)

3b. i(*a*)

em que *a* é a constante de Skolem. A negação do teorema a ser provado, convertido à forma de cláusula, é:

4'. ¬i(*Z*) l(*Z*).

Provar esse teorema através da refutação por resolução envolve gerar resolventes a partir do conjunto de cláusulas 1-3 e 4', adicionando esses resolventes ao conjunto e continuando até que a cláusula vazia seja produzida. Uma prova possível (existe mais de uma) produz a seguinte sequência de resolventes:

5. l(a) resolvente de 3b e 4'
6. a(a) resolvente de 5 e 1
7. ¬g(a) resolvente de 6 e 2
8. [] resolvente de 7 e 3a.

4.3.2 Sistemas de produção para refutação por resolução

Suponha um conjunto S de cláusulas chamado de *conjunto-base*. O algoritmo básico para um sistema de produção de refutação por resolução pode ser escrito como:

Procedimento *RESOLUÇÃO*

1. *CLÁUSULAS* ⇐ **S**
2. *até que* [] seja um membro de *CLÁUSULAS*, *faça*:
3. *início*
4. selecione duas cláusulas distintas C_i e C_j em *CLÁUSULAS*
5. calcule um resolvente, R_{ij} de C_i e C_j
6. *CLÁUSULAS* ⇐ o conjunto produzido adicionando R_{ij} a *CLÁUSULAS*
7. *fim*

Observe que o procedimento Resolução anterior é muito similar ao procedimento Produção do Capítulo 2.

4.3.3 Estratégias de controle para métodos de resolução

As decisões sobre quais cláusulas em *CLÁUSULAS* resolver (comando 4) e qual resolução dessas cláusulas realizar (comando 5) são tomadas através da estratégia de controle.

É útil para a estratégia de controle usar uma estrutura chamada de *grafo de derivação*. Os nós nesse grafo são rotulados pelas cláusulas; inicialmente, existe um nó para toda cláusula no conjunto-base. Quando duas cláusulas C_i e C_j produzem um resolvente R_{ij}, cria-se

um novo nó, *descendente*, rotulado R_{ij}, ligado com os nós pais C_i e C_j.

Uma refutação por resolução pode ser representada como uma *árvore de refutação* (dentro do grafo de derivação) tendo um nó raiz rotulado por [].

A estratégia de controle busca por uma refutação crescendo o *grafo* de derivação até que uma *árvore* seja produzida com um nó raiz rotulado pela cláusula vazia, []. Uma estratégia de controle para um sistema de refutação é *completa* se seu uso resulta num procedimento que achará uma contradição (eventualmente) onde existir. (A completude de uma *estratégia* não deve ser confundida com a completude lógica de uma regra de inferência.)

4.3.3.1 Estratégias para provar uma meta através da refutação

A estratégia de busca em largura

Na estratégia de busca em largura, todos os resolventes de primeiro nível são calculados primeiro, depois os resolventes de segundo nível e assim por diante. (Um *resolvente de primeiro nível* está entre as cláusulas do conjunto-base; um *resolvente do i-ésimo nível* é aquele cujos pais são resolventes do $(i - 1)$-ésimo nível.) A estratégia de busca em largura é completa, mas é muito ineficiente.

EXEMPLO Exemplo do golfinho (item 4.3.1):

Veja a árvore de refutação na Figura 4.5.

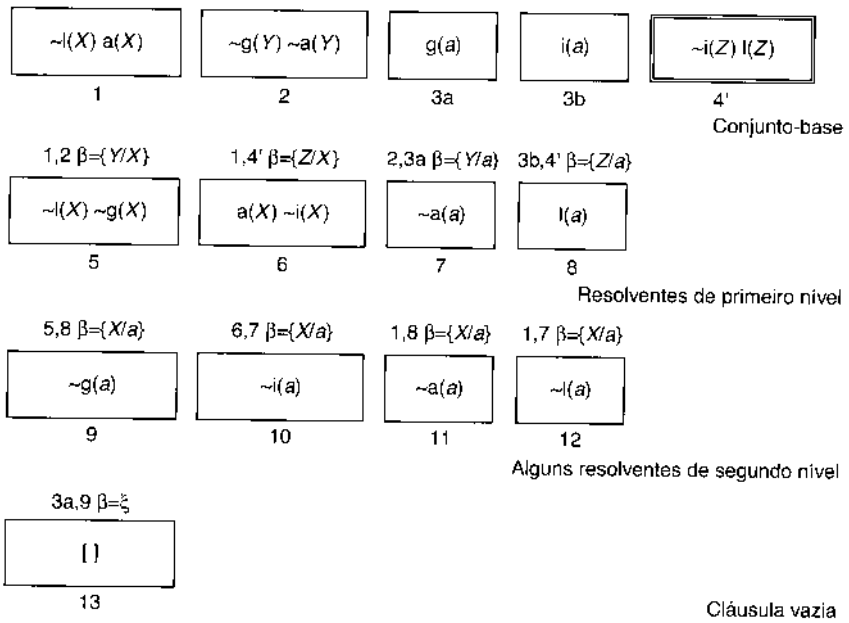


Figura 4.5 Ilustração da estratégia de busca em largura.

A estratégia do conjunto de suporte

Uma refutação por conjunto de suporte é aquela na qual no mínimo um pai para cada resolvente é selecionado entre as cláusulas resultantes da negação da fórmula meta ou dos seus descendentes (o *conjunto de suporte*). A estratégia precisa garantir a busca de todos as refutações por conjunto de suporte possíveis (na forma por largura). Além de completa, a estratégia do conjunto de suporte é mais eficiente que a busca em largura (Figura 4.6).

EXEMPLO Exemplo do golfinho (item 4.3.1):

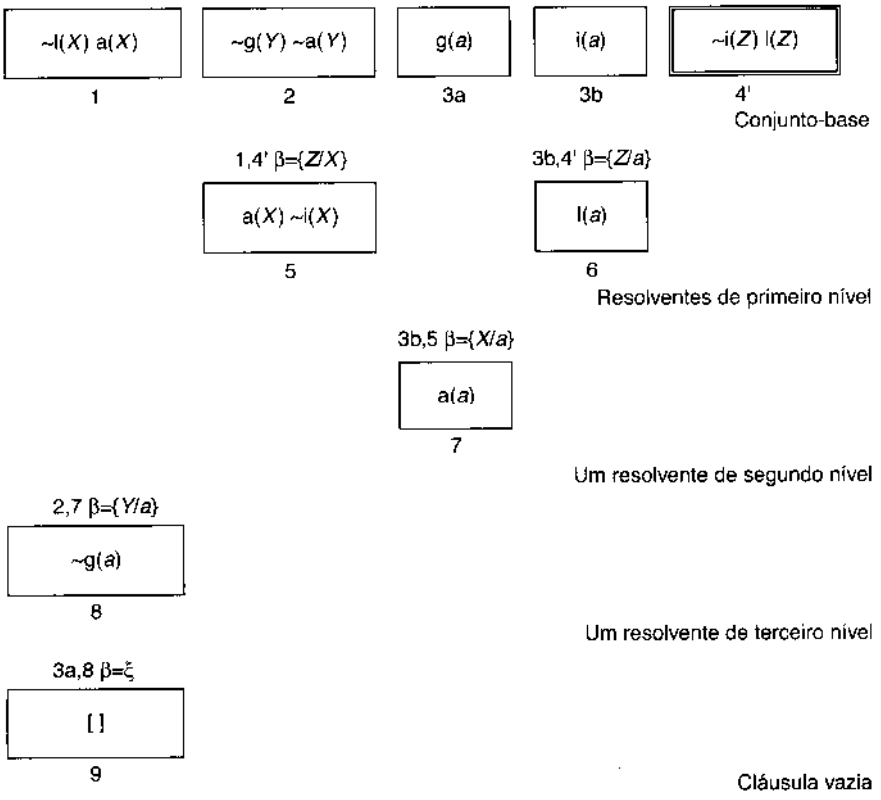


Figura 4.6 Ilustração da estratégia de conjunto de suporte.

A estratégia por preferência unitária

A estratégia por preferência unitária é uma modificação da estratégia por conjunto de suporte na qual, em vez de preencher cada nível na forma por largura, tenta-se selecionar uma cláusula de um único literal (chamado de *unidade*) para ser um pai numa resolução. Cada vez

que as unidades são usadas na resolução, os resolventes têm menos literais do que seus outros pais. Esse processo ajuda a dirigir a busca para produzir a cláusula vazia e, então, tipicamente, aumentar a eficiência. Mas não é completa (Figura 4.7).

EXEMPLO Exemplo do golfinho (item 4.3.1):

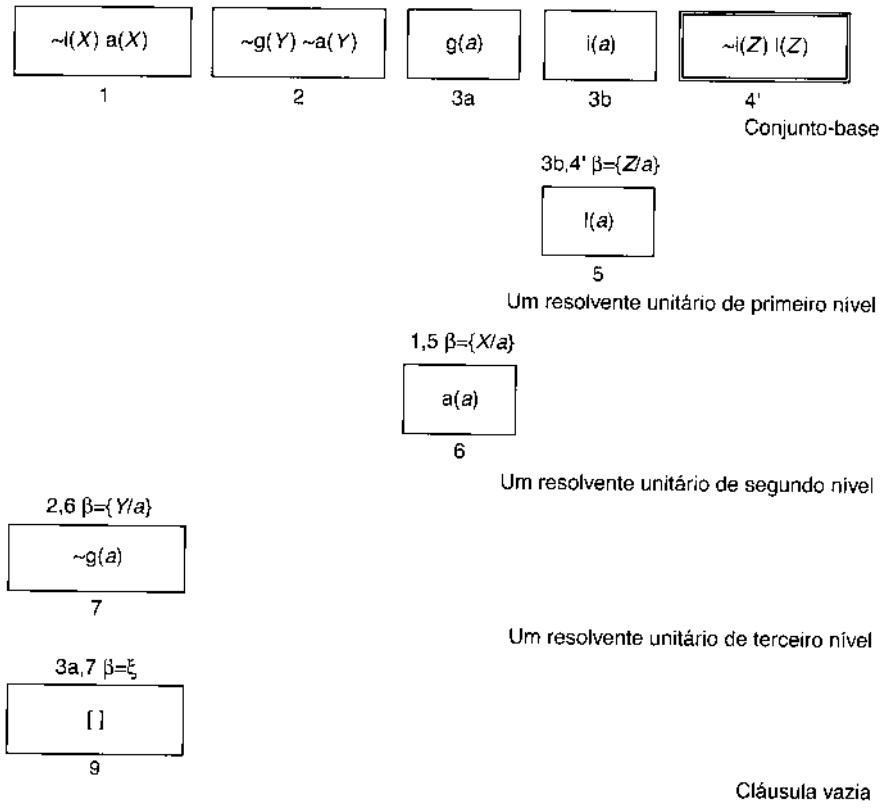


Figura 4.7 Ilustração da estratégia de preferência unitária.

4.3.3.2 Estratégias para provar que um conjunto de cláusulas é insatisfazível

A estratégia por forma de entrada linear

Uma refutação por forma de entrada linear é aquela na qual cada resolvente tem no mínimo um pai pertencente ao conjunto-base. Essa estratégia não é completa, ou seja, existem casos nos quais uma refutação existe, mas uma refutação por forma de entrada linear não (Figura 4.8).

EXEMPLO Exemplo do golfinho (item 4.3.1):

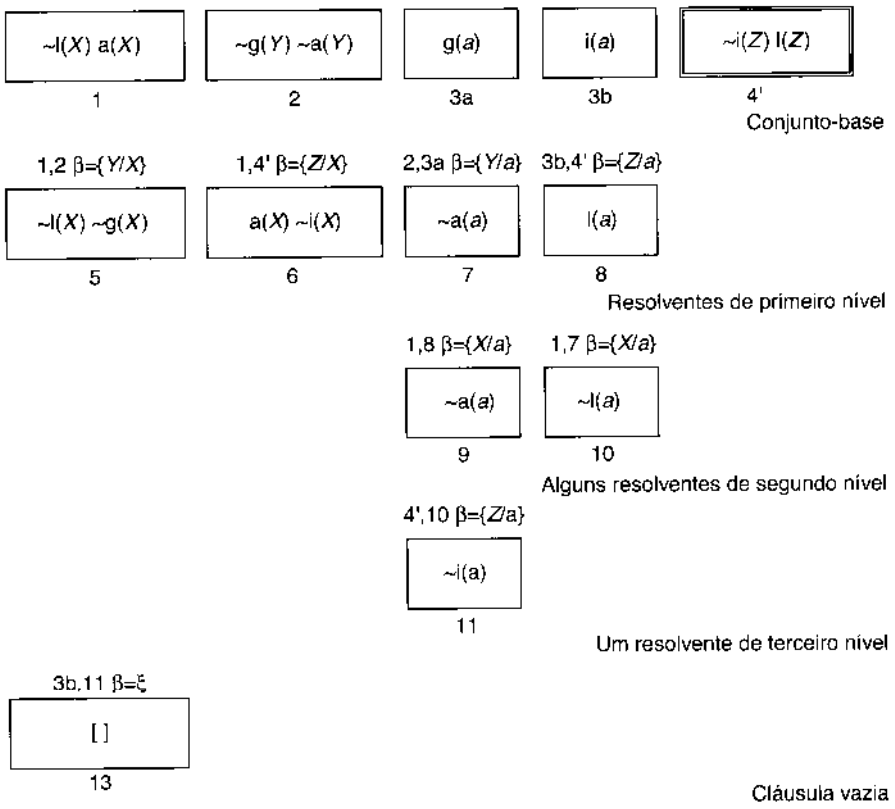


Figura 4.8 Ilustração da estratégia forma de entrada linear.

A estratégia por forma "ancestral filtrada"

Uma refutação por forma "ancestral filtrada" é aquela em que cada resolvente tem um pai que está no conjunto-base ou que é um ancestral do outro pai. Portanto, a forma "ancestral filtrada" é muito parecida com a forma linear. É uma estratégia completa.

EXEMPLO Conjunto-base **S**:

1. $\neg q(X)\ \neg p(X)$
2. $q(Y)\ \neg p(Y)$
3. $\neg q(W)\ p(W)$
4. $q(u)\ p(a)$

Obs.: Para a estratégia *forma de entrada linear*, não se chega à cláusula vazia, pois o conjunto **S** deve ter pelo menos uma cláusula unitária. A árvore de refutação a seguir está simplificada, ou seja, não estão explícitas todas as derivações possíveis (Figura 4.9).

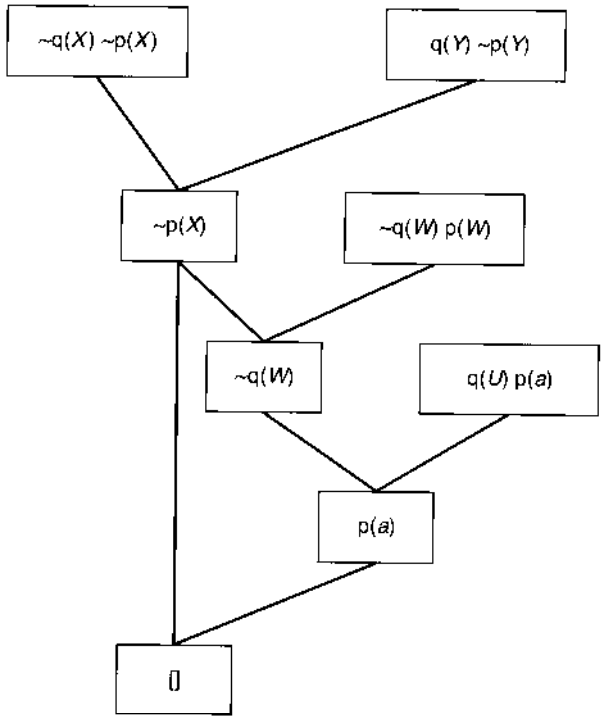


Figura 4.9 Ilustração da estratégia forma ancestral filtrada.

4.3.4 Estratégias de simplificação

Algumas vezes um conjunto de cláusulas pode ser simplificado pela eliminação de certas cláusulas ou pela eliminação de certos literais dentro das cláusulas. Essas simplificações são tais que o conjunto de cláusulas simplificado é insatisfazível se e somente se o conjunto original for insatisfazível. Portanto, o emprego dessas estratégias de simplificação ajuda a reduzir a taxa de crescimento de novas cláusulas.

Eliminação de tautologias

Qualquer cláusula contendo um literal e sua negação (chama-se tal cláusula uma *tautologia*) pode ser eliminada, desde que qualquer conjunto insatisfazível contendo uma tautologia ainda seja insatisfazível depois de sua remoção e vice-versa.

Incorporação procedimental

Algumas vezes é possível e mais conveniente *calcular* os valores verdade de literais do que incluir esses literais, ou suas negações, no conjunto-base. Tipicamente, os cálculos são realizados para instâncias concretas. Uma *instância concreta* é uma instância de uma expressão em que não ocorrem variáveis.

Mas o que significa realmente "calcular" uma expressão? As expressões do cálculo de predicados são construções linguísticas que denotam valores-verdade, elementos, funções ou relações num domínio. Tais expressões podem ser interpretadas com referência a um modelo que associa entidades linguísticas a entidades de domínio apropriadas. O resultado final é que os valores *V* ou *F* se tornam associados a sentenças na linguagem.

Eliminação por subjugação

Por definição, uma cláusula A_i *subjuga* uma cláusula B_i se existe uma substituição β tal que A_i β é um subconjunto de B_i. Como exemplos:

p(*X*) subjuga p(*Y*) q(*Z*), para β = { *X* / *Y* }
p(*X*) subjuga p(a), para β = { *X* / a }
p(*X*) subjuga p(a) q(*Z*), para β = { *X* / a }
p(*X*) q(a) subjuga p(f(a)) q(a) r(*Y*), para β = {*X* / f(a)}

Uma cláusula num conjunto insatisfazível que é subjugada por uma outra cláusula no conjunto pode ser eliminada sem afetar a insatisfazibilidade do resto do conjunto. A eliminação de cláusulas subjugadas por outras frequentemente leva a reduções substanciais no número de resoluções necessárias para encontrar uma refutação.

EXERCÍCIOS RESOLVIDOS

4.1 Aplique o algoritmo da unificação nos seguintes pares de literais, ou seja, diga para cada par de literais se ele é unificável ou não, e, se for, qual é o seu unificador mais geral:

a. f(a) e f(b) b. f(*X*) e f(g(*Y*)) c. f(a,g(*X*,*Y*)) e f(*X*,g(b,a))

Resolução

a. Não é unificável, pois o conjunto de discórdia {a, b} não satisfaz o teste de ocorrência, pois não há nenhuma variável.
b. β = { *X* / g(*Y*) }
c. Não é unificável, pois o conjunto de discórdia {a, b} não satisfaz o teste de ocorrência, pois não há nenhuma variável.

4.2 Usando os fatos a seguir, através de refutação, prove o teorema "Marco odiava César".

1. Marco era um homem.
2. Marco era um pompeiano.
3. Todos os pompeianos eram romanos.
4. César era um soberano.
5. Todos os romanos ou eram leais a César ou o odiavam.
6. Todos são leais a alguém.
7. Os homens somente tentam assassinar soberanos aos quais eles não são leais.
8. Marco tentou assassinar César.

Nota: Todos os fatos necessários para a solução do problema estão explicitados nessas oito sentenças anteriores. Explicite para cada resolvente quais são as cláusulas pais e qual a substituição usada. Tente aplicar as estratégias de simplificação. Use qualquer estratégia de controle. A sentença 7 poderia ser lida assim: Se x é homem e y é soberano e x tentou assassinar y então x não é leal a y.

Resolução

Vocabulário:
Predicados:
h(X) = X é homem
p(X) = X é pompeiano
r(X) = X é romano
s(X) = X é soberano
l(X, Y) = X é leal a Y
o(X, Y) = X odeia Y
t(X, Y) = X tentou assassinar Y

Constantes:
a = Marco
b = César
1. Marco era um homem: h(a)
2. Marco era um pompeiano: p(a)
3. Todos os pompeianos eram romanos: $\forall X(p(X) \rightarrow r(X))$
4. César era um soberano: s(b)
5. Todos os romanos ou eram leais a César ou o odiavam: $\forall X(r(X) \rightarrow (l(X,b) \vee o(X,b))$
6. Todos são leais a alguém: $\forall X \exists Y\, l(X, Y)$
7. Os homens somente tentam assassinar soberanos aos quais eles não são leais: $\forall X \forall Y((h(X) \wedge s(Y) \wedge t(X,Y)) \rightarrow \neg l(X,Y))$
8. Marco tentou assassinar César: t(a, b)

Meta: o(a, b)

Cláusulas:
1. h(a)
2. p(a)
3. $\neg p(X_3)\ r(X_3)$
4. s(b)
5. $\neg r(X_5)\ l(X_5, b)\ o(X_5, b)$
6. $l(X_6, f(X_6))$
7. $\neg h(X_7)\ \neg s(Y_7) \neg t(X_7, Y_7)\ \neg l(X_7, Y_7)$
8. t(a, b)
9. ¬o(a, b) :negação da meta

10. ¬r(a) l(a, b) : 5, 9 β = { X_5 / a }
11. ¬h(a) ¬s(b) ¬t(a, b) ¬r(a) : 7, 10 β = { X_7 / a; Y_7 / b }
12. ¬h(a) ¬s(b) ¬t(a, b)¬p(a) : 3, 11 β = { X_3 / a }
13. ¬h(a) ¬s(b) ¬t(a, b) : 2, 12 ξ
14. ¬h(a) ¬s(b) : 8, 13 ξ
15. ¬h(a) : 4, 14 ξ
16. [] :1, 15 ξ

4.3 A partir das cláusulas (axiomas) a seguir, conectadas conjuntivamente:

1. r j
2. ¬p ¬r
3. m ¬j k
4. ¬s ¬t ¬m
5. ¬p ¬r m
6. ¬p s
7. ¬u ¬n
8. ¬q t
9. ¬p p
10. ¬q u
11. ¬k n l

Simplifique o conjunto de cláusulas (se possível) e prove o teorema $(p \wedge q) \rightarrow l$, usando refutação e estratégia de controle *por conjunto de suporte*.

Resolução

Simplificação: a cláusula 5 deve ser eliminada, pois é subjugada por 2, e a cláusula 9 deve ser eliminada pois é uma tautologia.

Meta: $(p \wedge q) \rightarrow l$
Negação da meta:
$\neg((p \wedge q) \rightarrow l)$
$\neg(\neg(p \wedge q) \vee l)$
$((p \wedge q) \wedge \neg l)$

Logo, a negação da meta produz três cláusulas: p, q e ¬l

1. r j
2. ¬p ¬r
3. m ¬j k
4. ¬s ¬t ¬m
5. ¬p s
6. ¬u ¬n
7. ¬q t
8. ¬q u

9. ¬k n l
10. p
11. q
12. ¬l
13. s 5, **10**
14. ¬t ¬m 4, **13**
15. ¬t ¬j k 3, **14**
16. ¬t ¬j n l 9, **15**
17. ¬t ¬j n **12**, **16**
18. ¬t ¬j ¬u 6, **17**
19. ¬t ¬j ¬q 8, **18**
20. ¬t ¬j **11**, **19**
21. ¬t r 1, **20**
22. ¬t ¬p 2, **21**
23. ¬t **10**, **22**
24. ¬q 7, **23**
25. [] **11**,**24**

Obs.: As cláusulas em **negrito** pertencem ao conjunto de suporte.

4.4 Através de um exemplo, mostre que a resolução é consistente, ou seja, mostre que o resolvente de duas cláusulas segue logicamente das duas cláusulas.
Obs.: Para a → b, diz-se que b segue logicamente de a.

Resolução

Usando o seguinte exemplo:
((a ∨ ¬b) ∧ (c ∨ b)) → (a ∨ c)
Através da tabela-verdade, prova-se que essa expressão é uma tautologia, logo, ela é verdadeira, qualquer que seja a interpretação.

a	b	c	X = a ∨ ¬b	Y = c ∨ b	Z = a ∨ c	W = X ∧ Y	W → Z
F	F	F	V	F	F	F	V
F	F	V	V	V	V	V	V
F	V	F	F	V	F	F	V
F	V	V	F	V	V	F	V
V	F	F	V	F	V	F	V
V	F	V	V	V	V	V	V
V	V	F	V	V	V	V	V
V	V	V	V	V	V	V	V

4.5 Mostre que $\exists Z \forall X\ (p(X) \rightarrow q(Z))$ é equivalente a $\exists Z\ (\exists X\ p(X) \rightarrow q(Z))$.

$\exists Z \forall X\ (p(X) \rightarrow q(Z))$
$\exists Z \forall X\ (\neg p(X) \vee q(Z))$
$\forall X\ (\neg p(X)\ \neg q(a))$
Cláusula: $\neg p(X)\ q(a)$
$\exists Z\ (\exists X\ p(X) \rightarrow q(Z))$
$\exists Z\ (\neg \exists X\ p(X) \vee q(Z))$
$\exists Z\ (\forall X\ \neg p(X) \vee q(Z))$
$(\forall X\ \neg p(X) \vee q(a))$
Cláusula: $\neg p(X)\ q(a)$

4.6 Suponha que se queira resolver as seguintes cláusulas:

1. ama(pai(a),a)
2. $\neg$ama(Y, X) $\vee$ ama(X, Y)

Sabendo-se que *ama* é um predicado binário e *pai* é um símbolo funcional unário, responda:

(a) Qual será o resultado do algoritmo de unificação quando aplicado ao átomo do literal da cláusula 1 e ao átomo do primeiro literal da cláusula 2?

(b) O que deve ser gerado como um resultado da resolução dessas duas cláusulas?

(a) Y/pai(a), X/a

(b) ama(a,pai(a))

EXERCÍCIOS PROPOSTOS

4.1 Considere as cláusulas obtidas no Exercício 3.1 (Capítulo 3):

a. Prove que Josualdo gosta de amendoim utilizando a resolução.
b. Utilize a resolução para responder à pergunta "O que Solange come?".

4.2 Suponha os seguintes fatos:

a. Elesbão gosta apenas de cursos fáceis.
b. Os cursos de ciência são difíceis.
c. Todos os cursos do departamento de fabricação de cestas são fáceis.
d. BK301 é um curso de fabricação de cestas.

Utilize a resolução para responder à pergunta "De que curso Elesbão gosta?".

4.3 Considere os fatos seguintes:

a. Os sócios do Clube de Bolinha de Gude de São Petersburgo são Paul, Linda, Maguila e Sharon Stone.
b. Paul é casado com Linda.
c. Maguila é irmão de Sharon Stone.
d. O esposo de toda pessoa sócia do clube também é sócio do clube.
e. A última reunião do clube foi na casa de Paul.

1. Represente esses fatos na lógica de predicados.
2. Desses fatos anteriores, a maioria das pessoas seria capaz de decidir sobre a verdade das declarações adicionais seguintes:

f. A última reunião do clube foi na casa de Linda.
g. Sharon Stone não é casada.

Podem-se construir provas de resolução para demonstrar a verdade de cada uma dessas declarações, dados os cinco fatos listados anteriormente? Faça-o, se possível. Caso contrário, acrescente os fatos de que você precisar e depois construa as provas.

4.4 Determine se cada um dos seguintes conjuntos de cláusulas é satisfazível:

a. { ¬p q r, ¬q s, p s, ¬r, ¬s }
b. { p ¬q, p q, ¬p }
c. { ¬p q, p ¬r, ¬q, ¬r }
d. { p q, ¬p q, p ¬q, ¬p ¬q }

4.5 Determine se cada um dos seguintes conjuntos de expressões é unificável:

a. { p(X, f(Y), Y), p(W, Z, g(a,b)) }
b. { p(X, Z, Y), p(X, Z, X), p(a, X, X) }
c. { p(a, X, f(X)), p(X,Y,Z) }
d. { p(Z, f(X), b), p(X, f(a), b), p(g(X), f(a), Y) }

4.6 Procure uma refutação para os seguintes conjuntos (inconsistentes) de cláusulas conectadas conjuntivamente usando a estratégia de controle chamada "forma de entrada linear":

P:
1. ¬p ¬q r
2. ¬s t
3. ¬t p
4. s
5. ¬r
6. ¬s u
7. ¬u q

Q:
1. ¬p r
2. ¬q ¬r
3. ¬s t
4. ¬t p
5. s
6. ¬s j
7. ¬j q

4.7 Se um curso é fácil, alguns estudantes no curso são felizes. Se um curso tem exame, nenhum estudante no curso é feliz. Use resolução para mostrar que, se um curso tem exame, o curso não é fácil.

4.8 Usando refutação, mostre que o conjunto **S** de cláusulas:

1. ¬a(X) f(X) g(f(X))
2. ¬f(X) b(X)
3. ¬f(X) c(X)
4. ¬g(X) b(X)
5. ¬g(X) d(X)
6. a(g(X)) f(h(X))

implica a "meta": $\exists X \exists Y ((b(X) \wedge c(X)) \vee (d(Y) \wedge b(Y)))$

Não deixe de individualizar a variável de cada cláusula. Desenhe a árvore de refutação indicando claramente cada substituição.

4.9 Use resolução para mostrar que o conjunto de cláusulas a seguir é insatisfazível (inconsistente):

a) { p(X,Y) q(a,f(Y)) p(a,g(Z)), ¬p(a,g(X)) q(a,f(g(b))), ¬q(X,Y)}

4.10 Indique quais das seguintes cláusulas são subjugadas por p(f(X),Y):

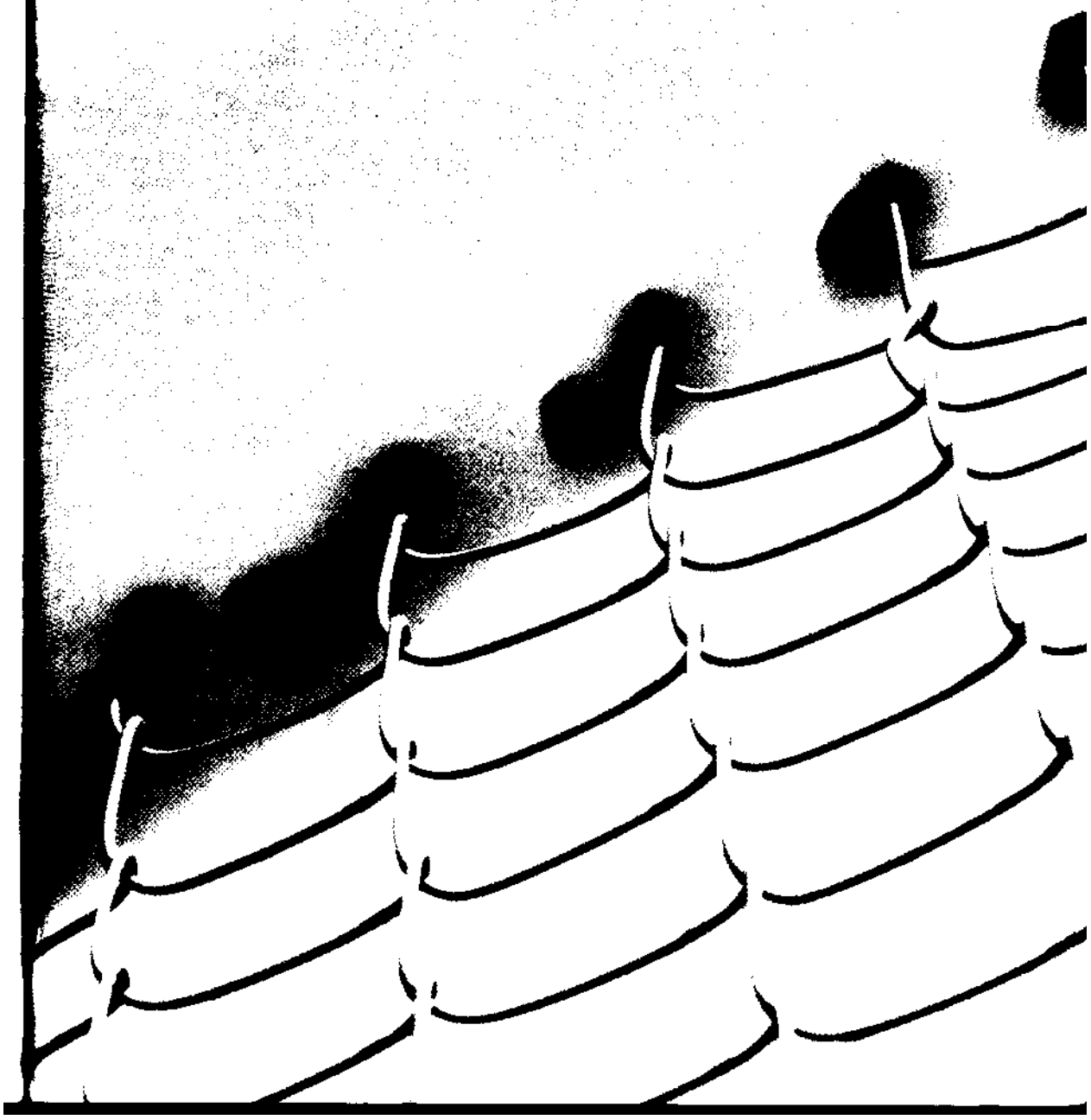
a. p(f(a), f(X)) p(Z, f(Y))
b. p(Z,a) ¬p(a,Z)
c. p(f(f(X)), Z)
d. p(f(Z), Z) q(X)
e. p(a,a) p(f(X), Y)

4.11 Prove por refutação que: $((a \rightarrow b) \rightarrow c) \rightarrow (a \rightarrow (b \rightarrow c))$.

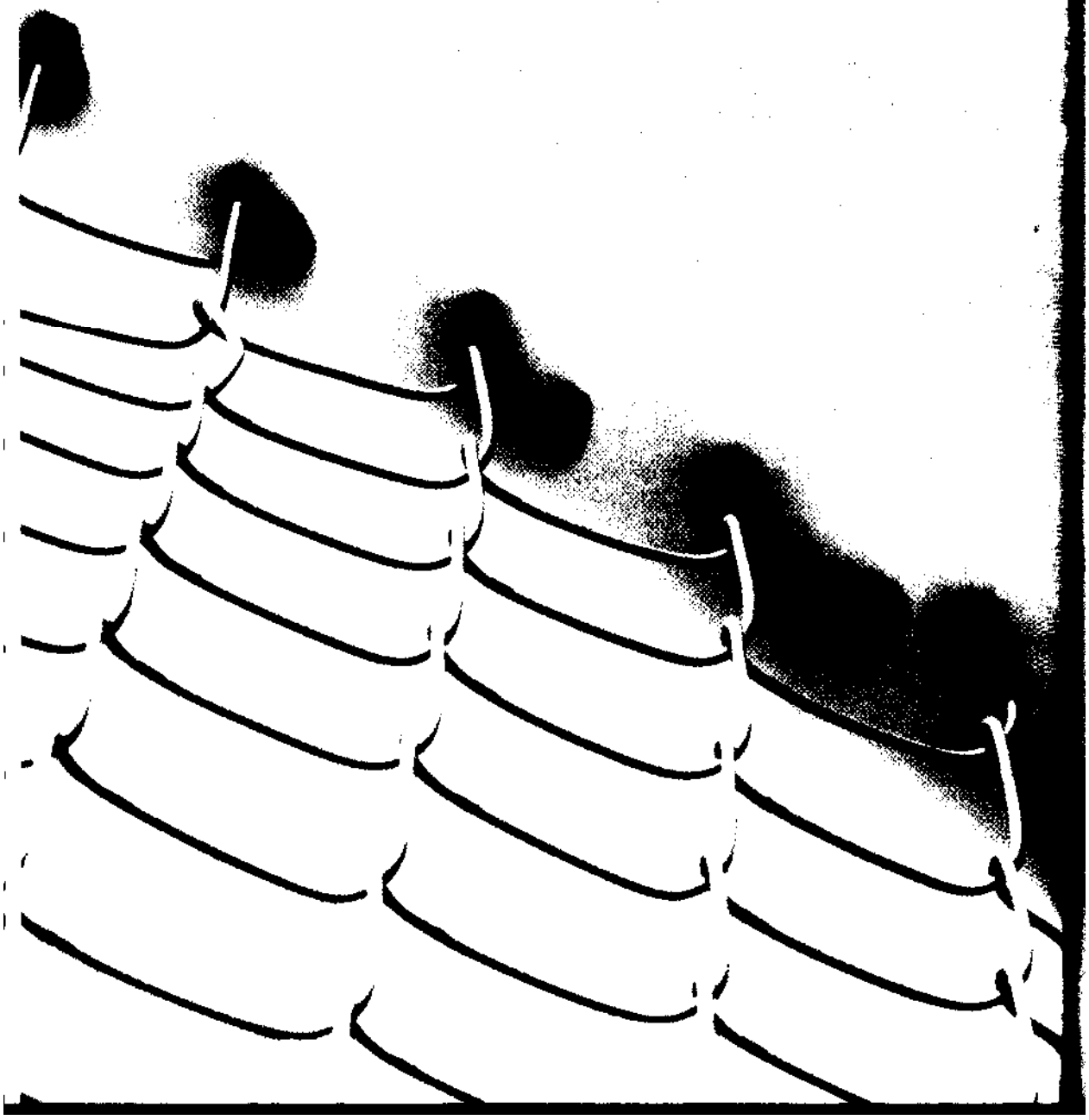
4.12 Mostre por refutação que a fórmula "$\exists X$ p(X)" segue logicamente da fórmula "$p(a1) \vee p(a2)$". Entretanto, a forma skolemizada de "$\exists X$ p(X)", denominada "p(a)", não segue logicamente de "$p(a1) \vee p(a2)$". Explique.

4.13 Prove que: $\forall Z (q(Z) \rightarrow p(Z)) \rightarrow (\exists X ((q(X) \rightarrow p(a)) \wedge (q(X) \rightarrow p(b))))$.

CAPÍTULO 5



Raciocínio Baseado em Regras



5.1 INTRODUÇÃO

A forma como um corpo de conhecimento sobre um certo campo é expresso por um especialista desse campo frequentemente contém informação importante sobre como esse conhecimento pode ser usado da melhor forma. Suponha, por exemplo, que um matemático diga:

"Se X e Y são ambos maiores que zero, o produto de X e Y também é maior que zero."

No cálculo de predicados essa sentença ficaria:

$$\forall X \forall Y ((m(X,0) \wedge m(Y,0)) \rightarrow m(vezes(X,Y),0)).$$

Entretanto, pode-se usar a seguinte fórmula equivalente:

$$\forall X \forall Y ((m(X,0) \wedge \neg m(vezes(X,Y),0)) \rightarrow \neg m(Y,0)).$$

O conteúdo lógico da sentença do matemático é independente das muitas formas equivalentes do cálculo de predicados que poderiam representá-la. Mas, na forma que são construídas, as sentenças do português frequentemente contêm informação de controle extralógica ou heurística. No exemplo anterior, a sentença parece indicar que se está habituado ao fato de que se X e Y são individualmente maiores que zero, então é óbvio provar que X multiplicado por Y é maior que zero.

Muito do conhecimento usado por sistemas de IA é diretamente representável por expressões gerais de implicação. Veja as seguintes sentenças:

1. Todos os vertebrados são animais.

$$\forall X \text{ (vertebrado } (X) \rightarrow \text{animal } (X))$$

2. Todos do departamento de computação acima de 30 anos são casados.

$$\forall X \forall Y ((\text{trabalha\_em(dep\_computação,}X) \wedge \text{idade } (X,Y) \wedge m(Y,30)) \rightarrow \text{casado } (X))$$

3. Existe um cubo acima de todo cilindro vermelho.

$$\forall X ((\text{cilindro } (X) \wedge \text{vermelho } (X)) \rightarrow \exists Y \text{ (cubo } (Y) \wedge \text{acima } (Y,X)))$$

O sistema descrito aqui não converte fórmulas em cláusulas; ele as usa numa forma perto da sua forma original dada. As fórmulas que representam conhecimento de asserção sobre o problema são separadas em duas categorias: *regras* e *fatos*. As regras consistem nas asserções dadas na forma implicacional. Tipicamente, elas expressam conhecimento *geral* sobre uma área particular e são usadas como regras de produção. Os fatos são as asserções que não são expressas como im-

plicações. Eles representam conhecimento *específico* relevante a um caso particular. A tarefa dos sistemas de produção é provar uma *fórmula meta* a partir desses fatos e regras.

Em sistemas por encadeamento *progressivo* (*forward* ou *data-driven*), as implicações usadas como regras-P operam numa base de dados global de fatos até que uma condição de terminação envolvendo a fórmula meta seja alcançada. Em sistemas por encadeamento regressivo (*backward* ou *goal-driven*) as implicações usadas como regras-R operam numa base de dados global de metas até que uma condição de terminação envolvendo os fatos seja alcançada. Combinar operação progressiva e regressiva também é possível.

Esse tipo de sistema de prova de teorema é um sistema *direto*, ao contrário do sistema de refutação. Um sistema direto não é necessariamente mais eficiente que um sistema de refutação, mas sua operação parece ser intuitivamente mais fácil para as pessoas entenderem.

Sistemas desse tipo são chamados de *sistemas de dedução baseados em regras*, para enfatizar a importância de se usar regras para fazer deduções. A pesquisa em IA tem produzido muitas aplicações de sistemas baseados em regras.

5.2 UM SISTEMA DE DEDUÇÃO PROGRESSIVO

5.2.1 A forma E/OU para expressões de fatos

O sistema progressivo tem como sua base de dados global inicial uma representação para o conjunto de fatos dado. Em particular, não se pretende converter esses fatos em forma de cláusulas. Os fatos são representados como fórmulas do cálculo de predicados que foram transformadas em formas livres de implicações chamadas *formas E/OU*. Para converter uma fórmula na forma E/OU, os símbolos "$\rightarrow$" (se existirem) são eliminados, usando a equivalência $W1 \rightarrow W2 \equiv \neg W1 \vee W2$. (Tipicamente, existem poucos símbolos "$\rightarrow$" entre os fatos porque as implicações são preferivelmente representadas como regras.) Depois, os símbolos de negação são movidos para dentro (usando as leis de De Morgan) até que seus escopos incluam, no máximo, um único predicado. A expressão resultante é então skolemizada e prenexada; as variáveis dentro dos escopos dos quantificadores universais são padronizadas através da renomeação, as variáveis quantificadas existencialmente são substituídas por funções de Skolem, e os quantificadores universais são eliminados. Assume-se que qualquer variável restante tem quantificação universal. Ou seja, na verdade é aplicado o algoritmo da representação clausal até o passo imediatamente anterior à obtenção da forma normal conjuntiva, complementando com a eliminação dos quantificadores universais.

Por exemplo, a expressão fato:

$$\exists U \forall V \, (q(V,U) \wedge \neg \, ((r(V) \vee p(V)) \wedge s(U,V)))$$

é convertida para

$$q(V,a) \wedge ((\neg r(V) \wedge \neg p(V)) \vee \neg s(a,V)))$$

As variáveis podem ser renomeadas de tal forma que a mesma variável não ocorra em conjunções diferentes (principais) da expressão fato. A renomeação de variáveis nesse exemplo

leva à expressão:

$$q(W,a) \wedge ((\neg r(V) \wedge \neg p(V)) \vee \neg s(a,V)).$$

Uma expressão na forma E/OU consiste em subexpressões de literais conectados por símbolos "$\wedge$" e "$\vee$". Note que uma expressão na forma E/OU não está na forma de cláusula. Está muito mais perto da expressão original.

5.2.2 Usando grafos E/OU para representar expressões de fatos

Um grafo E/OU pode ser usado para representar uma expressão fato na forma E/OU. Por exemplo, a árvore E/OU da Figura 5.1 representa a expressão fato posta na forma E/OU anterior. Cada subexpressão da expressão fato é representada por um nó no grafo. As subexpressões relacionadas disjuntivamente, $E_1, ..., E_k$, de um fato, $(E_1 \vee ... \vee E_k)$, são representadas por nós descendentes conectados a seus nós pais por um conector-k (que contém um arco de ligação entre os descendentes). Cada subexpressão conjuntiva, $E_1, ..., E_n$, de uma expressão, $(E_1 \wedge ... \wedge E_n)$, é representada por um único nó descendente conectado ao nó pai por um conector-l.

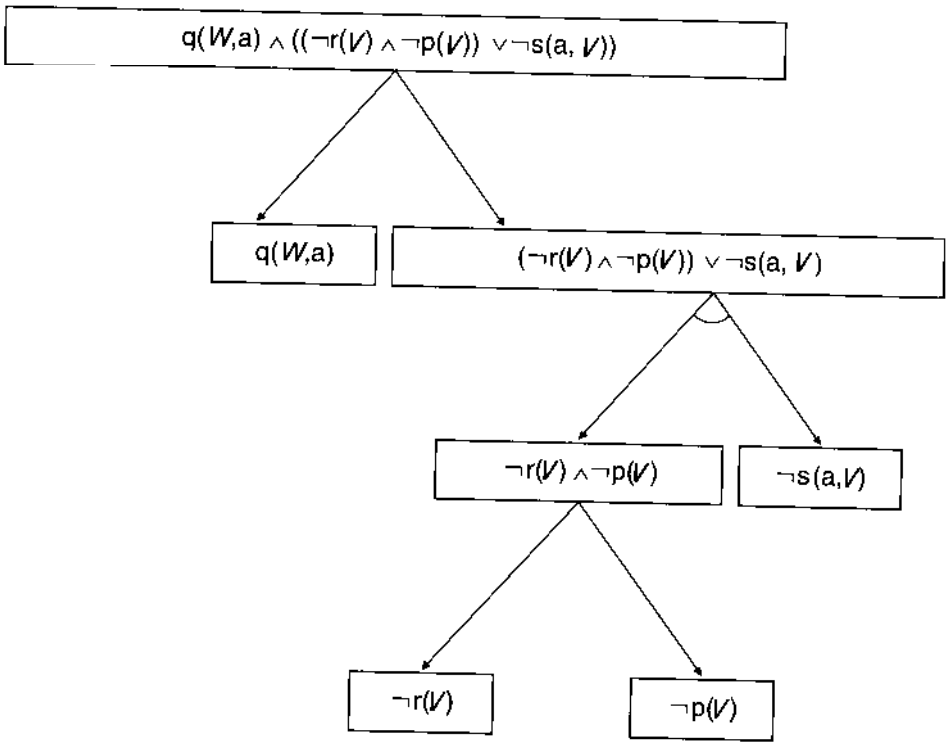


Figura 5.1 Uma representação em árvore E/OU de uma expressão fato.

Os nós folhas da representação de grafo E/OU de uma expressão fato estão rotulados pelos literais que ocorrem na expressão. Chama-se de *nó raiz* o nó no grafo que rotula a expressão fato inteira. Ele não tem nenhum ancestral no grafo.

Uma propriedade interessante da representação de grafo E/OU de uma fórmula é que o conjunto de cláusulas no qual a fórmula pode ser convertida pode ser visto como o conjunto de grafos solução (terminando em nós folhas) do grafo E/OU. Então, as cláusulas que resultam da expressão $q(W,a) \wedge ((\neg r(V) \wedge \neg p(V)) \vee \neg s(a,V))$ são:

$q(W,a)$
$\neg s(a,V)\ \neg r(V)$
$\neg s(a,V)\ \neg p(V)$

5.2.3 Usando regras para transformar grafos E/OU

As regras de produção usadas pelo sistema de produção progressivo são aplicadas a estruturas de grafos E/OU para produzir estruturas de grafos transformadas. Essas regras são baseadas em fórmulas implicacionais que representam conhecimento assercional geral sobre um domínio de problema. Para simplificar, limitaram-se os tipos de fórmulas permitidas como regras àquelas da forma:

$$L \rightarrow W,$$

em que L é um literal único, W é uma fórmula arbitrária (que se assume estar na forma E/OU), e quaisquer variáveis ocorrendo na implicação são assumidas como tendo quantificação universal sobre a implicação inteira. As variáveis nos fatos e regras são padronizadas de tal forma que nenhuma variável ocorra em mais de uma regra e que as variáveis das regras sejam diferentes das dos fatos.

A restrição a antecedentes de literal único simplifica consideravelmente o processo de casamento na aplicação de regras aos grafos E/OU. Por exemplo, pode-se transformar a implicação $(L1 \vee L2) \rightarrow W$ em um par de regras $L1 \rightarrow W$ e $L2 \rightarrow W$.

5.2.4 Usando a fórmula meta para terminação

O objetivo do sistema de produção progressivo descrito é provar alguma fórmula meta a partir de uma fórmula fato e de um conjunto de regras. Esse sistema progressivo é limitado em relação ao tipo de expressões metas que ele pode provar; especificamente, ele pode provar apenas as fórmulas metas cuja forma seja uma *disjunção* de literais, conhecida como cláusula-P. Representa-se essa fórmula meta por um conjunto de literais e assume-se que os membros desse conjunto estão relacionados disjuntivamente. Os literais metas (assim como as regras) podem ser usados para adicionar descendentes ao grafo E/OU. Quando um dos literais metas casa com um literal rotulando um nó literal, n, do grafo, adiciona-se um novo descendente do nó n, rotulado pelo literal meta casado, ao grafo. Esse descendente é chamado de *nó meta*. Os nós metas são conectados aos seus pais por arcos de casamento. O sistema de produção termina de forma bem-sucedida quando produz um grafo E/OU contendo um grafo de solução que termina em nós metas. (Na terminação, o sistema inferiu uma cláusula idêntica a alguma subparte da cláusula meta.)

5.2.5 Receita para resolução por encadeamento progressivo

Para resolver um problema através do encadeamento progressivo, é necessária a execução de alguns passos:

1. Verificar se os elementos estão no formato adequado: o fato deve estar na forma E/OU; as regras devem ter apenas um literal como antecedente, e a meta deve ser uma disjunção de literais. Se a meta não obedecer a essa exigência, não é possível resolver através do encadeamento progressivo.
2. Construir o grafo E/OU para a expressão fato.
3. Construir o grafo do conhecimento para a expressão fato, aplicando as regras no grafo E/OU.
4. Obter as cláusulas-P a partir do grafo do conhecimento.
5. Verificar se a meta foi alcançada a partir do conjunto de cláusulas obtidas. Lembre-se de que esse conjunto é uma conjunção de cláusulas-P, ou seja, uma conjunção de disjunções.

5.3 UM SISTEMA DE DEDUÇÃO REGRESSIVO

Uma propriedade importante da lógica é a dualidade entre asserções e metas em sistemas de prova de teoremas. Já foi vista uma instância desse princípio de dualidade nos sistemas de refutação por resolução. Lá a fórmula meta era negada, convertida na forma de cláusula e adicionada às asserções em forma de cláusulas também. A dualidade entre asserções e metas permite que a meta negada seja tratada como se fosse uma asserção. Os sistemas de refutação por resolução aplicam resolução ao conjunto de cláusulas combinadas até que a cláusula vazia (denotando F) seja produzida.

Pode-se também descrever um sistema de resolução dual que opera nas expressões metas. Para preparar as fórmulas para tal sistema, deve-se primeiro negar a fórmula representando as asserções, converter essa fórmula negada ao dual da fórmula de cláusula (uma disjunção de conjunções de literais) e adicionar essas cláusulas à forma de cláusula dual da fórmula meta. O sistema deve então aplicar uma versão dual da resolução até que a cláusula vazia (agora denotando V) seja produzida.

Pode-se também imaginar sistemas mistos nos quais três formas diferentes de resolução são usadas: a resolução entre asserções, a resolução entre expressões metas e a resolução entre uma asserção e uma meta. O sistema progressivo descrito no último item deveria ser apontado como um desses sistemas mistos porque ele envolve o casamento de um literal fato no grafo E/OU com um literal meta. O sistema de produção regressivo, descrito aqui, é também um sistema misto que é, de alguma forma, dual ao sistema progressivo. Sua operação envolve os mesmos tipos de representações e mecanismos que são usados no sistema progressivo.

5.3.1 Expressões metas na forma E/OU

O sistema regressivo é capaz de tratar de expressões metas de forma arbitrária. Primeiro, converte-se a fórmula meta na forma E/OU pelo mesmo tipo de processo usado para converter uma expressão fato. Eliminam-se os símbolos $\rightarrow$, movem-se os símbolos de negação para dentro, skolemizam-se as variáveis *universais* e eliminam-se os quantificadores exis-

tenciais. As variáveis restantes na forma E/OU de uma expressão meta assumem a quantificação existencial.

Por exemplo, a expressão meta:

$$\exists Y \forall X\, (p(X) \rightarrow (q(X,Y) \wedge \neg(r(X) \wedge s(Y))))$$

é convertida para

$$\neg p(f(Y)) \vee (q(f(Y),Y) \wedge (\neg r(f(Y)) \vee \neg s(Y))),$$

em que $f(Y)$ é uma função de Skolem.

A padronização das variáveis nas disjunções (principais) da meta leva a:

$$\neg p(f(z)) \vee (q(f(Y),Y) \wedge (\neg r(f(Y)) \vee \neg s(Y))).$$

(Note que a variável Y não pode ser renomeada *dentro* da subexpressão disjuntiva.)

As fórmulas metas na forma E/OU podem ser representadas como grafos E/OU. Mas com expressões metas os conectores-k nesses grafos são usados para separar *conjuntivamente* subexpressões relacionadas. A representação do grafo E/OU para a fórmula meta do exemplo anterior é mostrada na Figura 5.2. Os nós folhas desse grafo são rotulados pelos literais da expressão meta. Nos grafos metas E/OU, chama-se qualquer descendente do nó raiz de *nó submeta*. As expressões que rotulam tais nós descendentes são chamadas de *submetas*.

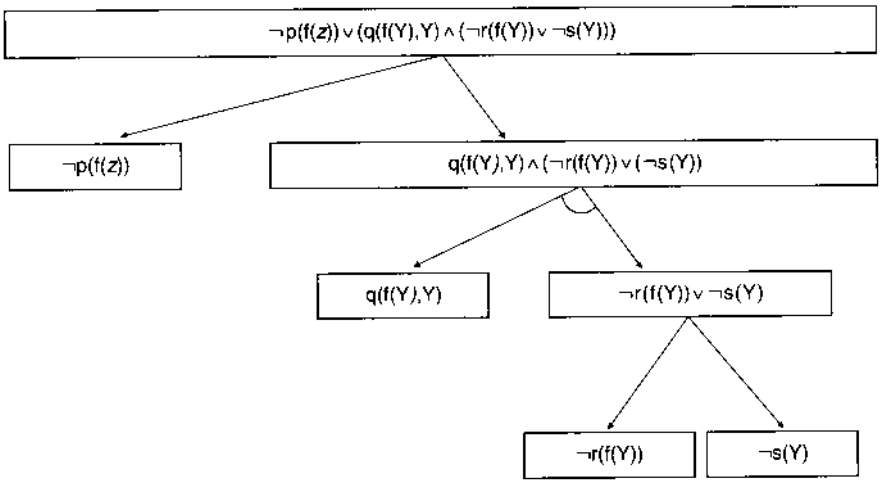


Figura 5.2 Uma representação de grafo E/OU de uma fórmula meta.

O conjunto de cláusulas na representação de forma de cláusula dessa fórmula meta pode ser lido a partir do conjunto de grafos solução terminando em nós folhas:

$\neg p(f(z))$
$q(f(Y),Y) \wedge \neg r(f(Y))$
$q(f(Y),Y) \wedge \neg s(Y)$

As cláusulas metas são *conjunções* de literais, e a disjunção dessas cláusulas é a forma de cláusula da fórmula meta.

5.3.2 Aplicando regras num sistema regressivo

As regras-R para esse sistema são baseadas em implicações assercionais. Elas são asserções assim como são as regras-P no sistema progressivo. Agora, entretanto, as regras-R serão restritas a expressões da forma

$$W \rightarrow L,$$

em que W é qualquer fórmula (que se assume estar na forma E/OU), L é um literal, e o escopo da quantificação de quaisquer variáveis na implicação é a implicação inteira. Novamente, a restrição de regras-R a implicações dessa forma simplifica o casamento e não causa dificuldades práticas importantes. Também, uma implicação tal como $W \rightarrow (L1 \wedge L2)$ pode ser convertida a duas regras $W \rightarrow L1$ e $W \rightarrow L2$. Tal regra-R é aplicável a um grafo E/OU representando uma fórmula meta se esse grafo contém um nó literal rotulado por L' que unifica com L. O resultado da aplicação da regra é adicionar um arco de casamento a partir do nó rotulado por L' para um nó do novo descendente rotulado por L. Esse novo nó é o nó raiz da representação do grafo E/OU de Wu em que u é o u.m.g. de L e L'. Esse u.m.g. rotula o arco de casamento no grafo transformado.

5.3.3 A condição de terminação

As expressões fatos usadas pelo sistema regressivo são limitadas àquelas na forma de uma conjunção de literais, também chamada de cláusula-R. Tais expressões podem ser representadas como um conjunto de literais. Análogo ao sistema progressivo, quando um literal fato casa com um literal rotulando um nó literal do grafo, um descendente correspondente *nó fato* pode ser adicionado ao grafo. Esse nó fato é ligado ao nó literal da submeta casada por um arco de casamento rotulado pelo u.m.g. O mesmo literal fato pode ser usado várias vezes (com variáveis diferentes em cada uso) para criar nós fatos múltiplos.

A condição para terminação bem-sucedida para o sistema regressivo é que o grafo E/OU contém um grafo de solução *consistente* terminando em nós fatos. Novamente, um grafo de solução consistente é aquele em que as substituições do arco de casamento têm uma composição unificadora.

5.3.4 Receita para resolução por encadeamento regressivo

Para resolver um problema através do encadeamento regressivo, é necessária a execução de alguns passos:

1. Verificar se os elementos estão no formato adequado: a meta deve estar na forma E/OU; as regras devem ter apenas um literal como consequente e o fato deve ser uma conjunção de literais. Se o fato não obedecer a essa exigência, não é possível resolver através do encadeamento regressivo;
2. Construir o grafo E/OU para a expressão meta;
3. Construir o grafo do conhecimento para a expressão meta, aplicando as regras no grafo E/OU;
4. Obter as cláusulas-R a partir do grafo do conhecimento;
5. Verificar se o fato pode ser verificado a partir do conjunto de cláusulas obtidas. Lembre-se de que esse conjunto é uma disjunção de cláusulas-R, ou seja, uma disjunção de conjunções.

5.4 "RESOLVENDO" DENTRO DOS GRAFOS E/OU

O sistema regressivo descrito não é capaz de provar expressões de metas válidas ou tautológicas como $(\neg p \vee p)$, a menos que ele possa provar $\neg p$ ou p separadamente. O sistema progressivo não pode reconhecer expressões fatos contraditórias como $(\neg p \wedge p)$. Com a finalidade de superar essas deficiências, os sistemas devem ser capazes de realizar inferências intrameta ou intrafato.

Vai-se descrever como certas inferências intrameta podem ser realizadas. Considere, por exemplo, as seguintes expressões usadas num sistema regressivo:

Meta

$(p(X,Y) \vee q(X,Y)) \vee v(X,Y)$

Regras

R1: $(r(V) \wedge s(U,b)) \rightarrow p(U,V)$

R2: $(\neg s(a,S) \wedge w(R)) \rightarrow q(R,S)$

Fatos

$r(b) \wedge w(b) \wedge v(a,b) \wedge v(b,b)$

Depois de aplicar as regras R1 e R2, tem-se o grafo E/OU mostrado na Figura 5.3. Esse grafo tem dois literais complementares cujos predicados unificam com u.m.g. {*X*/a, *Y*/b}. O procedimento *Resolução de Meta Restrita ("RGR" — Restricted Goal Resolution)* permite eliminar nós folha complementares numa disjunção, para efeito de simplificação do grafo A/O. Um outro procedimento possível, caso o grafo gerado não resolva o problema, é gerar um outro grafo, partindo da expressão obtida através da aplicação da propriedade distributiva.

5.5 UMA COMBINAÇÃO DE SISTEMAS PROGRESSIVO E REGRESSIVO

Ambos os sistemas de dedução baseados em regras, progressivo e regressivo, têm limitações. O sistema regressivo pode manipular expressões metas de forma arbitrária, mas está restrito a expressões fatos que consistem em conjunções de literais. O sistema progressivo pode manipular expressões fatos de forma arbitrária, mas está restrito a expressões metas que consistem em disjunções de literais. Pode-se combinar os dois sistemas e tirar vantagens de cada um sem as limitações de ambos?

Na verdade, quatro fatores influenciam a razão de se escolher raciocinar progressivamente ou regressivamente (Rich e Knight, 1994):

1. Existem mais estados iniciais ou metas? É preferível mover de um conjunto de estados menor para um maior (e, portanto, mais fácil de achar).
2. Em qual direção o fator de ramificação é maior? (Fator de ramificação é o número médio de nós que podem ser alcançados diretamente de um único nó.) É preferível mover na direção com o fator de ramificação menor.
3. O programa terá que justificar seu processo de raciocínio para o usuário? Se tiver, é importante mover na direção que corresponde à forma como o usuário está pensando.
4. Que tipo de evento irá iniciar um episódio de resolução de problema? Se for a chegada de um novo fato, o raciocínio progressivo faz sentido. Se for desejada a resposta de uma pergunta, o raciocínio regressivo é mais natural.

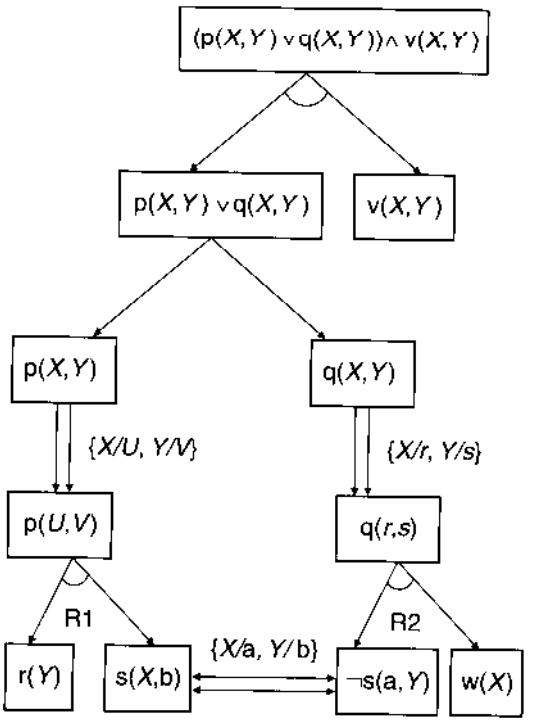


Figura 5.3 Um grafo E/OU com nós literais complementares.

EXERCÍCIOS PROPOSTOS

5.1 A partir do fato

$(l \vee b) \wedge c$

e das regras

1. $a \rightarrow (g2 \wedge p)$
2. $b \rightarrow (a \wedge m)$
3. $c \rightarrow (h \vee (d \wedge e))$
4. $d \rightarrow g1$
5. $e \rightarrow n$
6. $l \rightarrow \neg h$

deduza a meta

$g1 \vee g2$

usando técnica progressiva e com o auxílio de grafo de conhecimento. Se necessário, use RGR.

5.2 Com o auxílio de um grafo de conhecimento, mostre, usando técnica regressiva, que os fatos apoiam a meta:

Meta:

$(k \wedge l) \vee (m \wedge r)$

Regras:

1. $(j \vee h) \rightarrow k$
2. $(f1 \wedge p) \rightarrow l$
3. $(n \vee (q \wedge \neg p)) \rightarrow m$
4. $f2 \rightarrow h$
5. $f3 \rightarrow q$
6. $(f4 \wedge s) \rightarrow r$
7. $f4 \rightarrow s$

Fatos:

$f1 \wedge f2 \wedge f3 \wedge f4$

Use RGR, se necessário.

5.3 Usando as seguintes regras

1. $(p \wedge q) \rightarrow r$
2. $s \rightarrow t$
3. $t \rightarrow p$

4. $s \rightarrow v$
5. $v \rightarrow q$

e técnica progressiva, demonstre *r* a partir de *s*. Monte grafo de conhecimento. Em seguida, num estilo demonstração por refutação, reproduza os mesmos passos, deixando claras as resoluções efetuadas.

5.4 Repita o exercício anterior usando técnica regressiva (encadeamento *backward*).

5.5 Considere a seguinte base de dados

1. $\forall X \forall Y ((h(X) \wedge d(Y)) \rightarrow f(X,Y))$
2. $\exists Y(g(Y) \wedge (\forall Z(r(Z) \rightarrow f(Y,Z))))$
3. $\forall Y(g(Y) \rightarrow d(Y))$
4. $\forall X \forall Y \forall Z((f(X,Y) \wedge f(Y,Z)) \rightarrow f(X,Z))$

e com base nela prove a seguinte sentença:

$\forall X \forall Z ((h(X) \wedge r(Z)) \rightarrow f(X,Z))$

5.6 Considere o sistema

Fato:
s

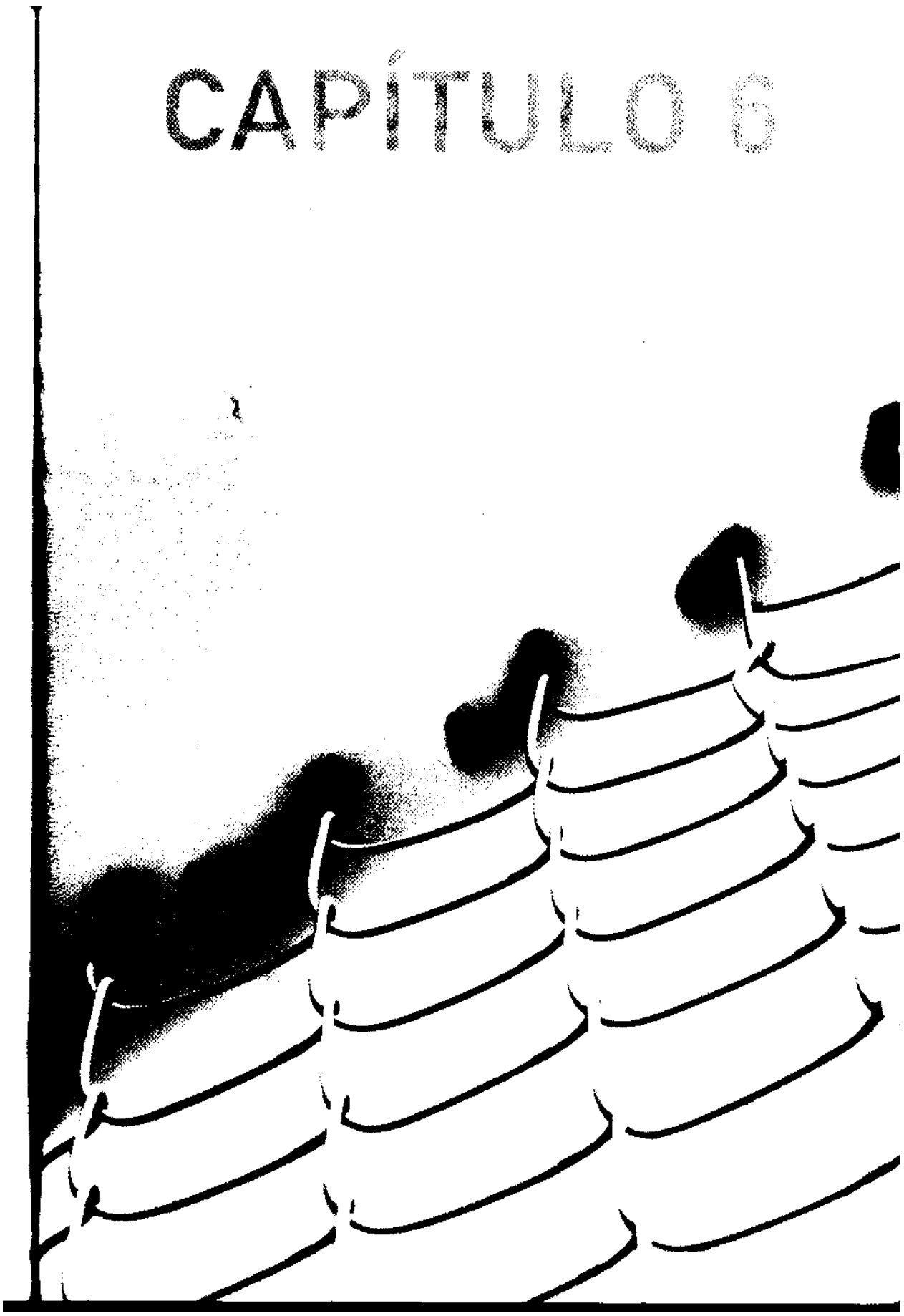
Regras:
1. $r \rightarrow \neg p$
2. $\neg j \rightarrow r$
3. $s \rightarrow (\neg m \vee \neg t)$
4. $\neg m \rightarrow (\neg j \vee k)$
5. $\neg t \rightarrow \neg q$
6. $\neg u \rightarrow \neg q$
7. $n \rightarrow \neg u$
8. $k \rightarrow (n \vee l)$

Meta:
$\neg p \vee \neg q \vee l$

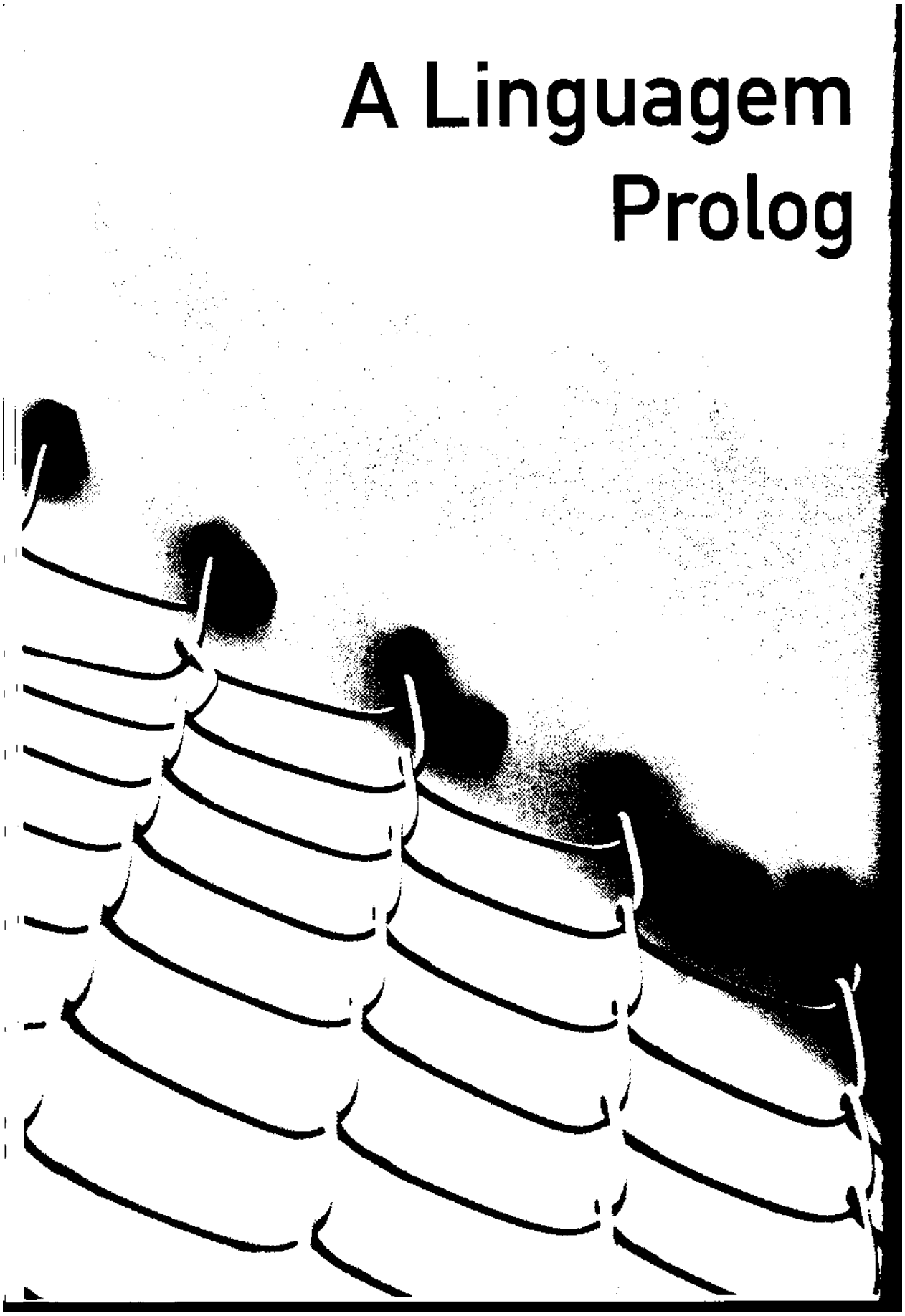
Prove a meta utilizando técnica progressiva.

5.7 É possível resolver o problema anterior usando técnica regressiva (encadeamento *backward*)? No caso afirmativo, indique um grafo solução.

CAPÍTULO 6



A Linguagem Prolog



6.1 PROGRAMAÇÃO LÓGICA

Programas lógicos são essencialmente conjuntos de cláusulas da forma:

$$b \leftarrow a_1, a_2, \ldots, a_n$$

chamadas de cláusulas de Horn, em que b e a_j são fórmulas atômicas e todas as variáveis nas fórmulas atômicas são assumidas como universalmente quantificadas. "←" é lido "se", e as vírgulas significam conjunção. Um lado direito vazio denota uma asserção não condicional do fato. Por exemplo,

1. gosta(mãe(*X*),*X*) ←
   "todo *X* é gostado(a) por sua mãe"

2. gosta(roberto,*Y*) ← gosta(*Y*,roberto)
   "Roberto gosta de todo *Y* que gosta dele"

As variáveis aparecem em itálico para distinguir das constantes.

Com respeito a um dado conjunto de cláusulas (isto é, um programa), o usuário pode perguntar por relações a serem calculadas, colocando chamadas de procedimentos, isto é, cláusulas da forma:

$$\leftarrow a_1, a_2, \ldots, a_n$$

Isso inicia um processo de demonstração automático, durante o qual as variáveis na chamada tomam valores que "a_1 e a_2 e ... a_n" contêm. Aqui "←" pode ser interpretada como uma interrogação. Uma terminação malsucedida implica que não existem tais valores.

Portanto, com respeito às cláusulas 1 e 2 anteriores, a seguinte chamada:

3. ← gosta(*Z*,roberto)
   "Quem gosta de Roberto?"

resulta em *Z* sendo unificado (ligado) a "mãe(roberto)". O mesmo resultado teria sido obtido da chamada:

4. ← gosta(*Z*,roberto),gosta(roberto,*Z*)
   "Quem gosta de e é gostado por Roberto?"

Resultados alternativos para a mesma chamada podem ser obtidos com programas não determinísticos. Por exemplo, se se adicionasse a cláusula:

gosta(sonia,roberto) ←

então chamar 3 pode alternativamente resultar em *Z* sendo ligado a "sonia".

Interpretadores práticos de programas lógicos, tal como Prolog, também incluem algumas características extralógicas para funções de entrada e saída e de controle.

6.1.1 Características notáveis de programas lógicos

A simplicidade da sintaxe e da semântica de programas lógicos esconde várias características notáveis não encontradas em linguagens de programação convencionais. Segundo o trabalho de Pereira e Warren, as características especialmente relevantes à escrita de gramática são:

1. Casamento de padrões (unificação) substitui o uso convencional de funções de seletor e construtor para operações sobre dados estruturados.
2. Os argumentos de um procedimento podem servir não apenas para este receber um ou mais valores como entrada, mas também para retornar um ou mais valores como saída.
3. Os argumentos de entrada e saída de um procedimento não têm que ser distinguidos antes, mas podem variar de uma chamada para outra.
4. Os procedimentos podem gerar (via *backtracking*, no caso do Prolog) um conjunto de resultados alternativos. Tais procedimentos são chamados *não determinísticos*. O *backtracking* é equivalente a uma forma de iteração de alto nível.
5. Os procedimentos podem retornar resultados "incompletos", isto é, o termo ou termos retornados como resultado de um procedimentos podem conter variáveis, que são preenchidas apenas mais tarde por chamadas de outros procedimentos. O efeito é similar ao uso de atribuição numa linguagem convencional para preencher campos de uma estrutura de dados. Note, entretanto, que podem existir muitas ocorrências de uma variável não instanciada e que todas elas são preenchidas simultaneamente (num único passo) quando a variável é finalmente instanciada. Note também que, quando duas variáveis são unificadas, elas se tornam uma única. O efeito é como um ponteiro invisível, ou referência, ligando uma variável a outra. Refere-se a esse fenômeno como "variável lógica".
6. "Programa" e os "dados" são idênticos na forma. Um procedimento consistindo somente em fatos está mais perto de uma matriz, ou tabela de dados, em uma linguagem convencional.

6.2 INTRODUÇÃO AO PROLOG

A ideia de usar lógica como um formalismo executável em computador recebeu um grande ímpeto com o advento da linguagem Prolog ("PROgrammation en LOGic"). Desenvolvida na década de 1970 em Edimburgo, a linguagem Prolog tem sua aplicação dirigida à computação simbólica (não numérica). Trata-se de uma evolução das linguagens de computador, pois Prolog tem características de linguagens procedimentais (o como), e de linguagens declarativas (o quê).

6.2.1 Inferência lógica do Prolog

Prolog é uma linguagem de programação lógica. Um programa Prolog é uma base de conhecimento em que cada asserção é uma tradução de uma implicação da lógica. A máquina de inferência do Prolog é capaz de realizar deduções lógicas a partir dessa base de conhecimento e produzir conhecimento novo. Considere, por exemplo, as seguintes proposições (Pereira e Shieber, 1987):

1. Todos os homens são mortais.
2. Platão é um homem.
3. Platão é mortal.

Para a lógica, as duas primeiras proposições são premissas, e a terceira é a conclusão lógica dessas premissas. Na lógica de predicados de primeira ordem, se o predicado h significar *homem*, o predicado m *mortal*, e a constante p *Platão*, pode-se ter:

1. $\forall X$ (h(X) $\rightarrow$ m(X))
2. h(p)
3. m(p)

em que a terceira fórmula seria facilmente obtida a partir das duas primeiras usando qualquer ferramenta lógica de dedução, como por exemplo a regra da resolução (regra de inferência derivada da regra *Modus Ponens*).

Toda implicação da lógica da forma $a \rightarrow b$, equivalente a $\neg a \vee b$, gera uma regra Prolog da forma *b* :- *a*. Logo, um programa Prolog para essa base de conhecimento seria:

```
m(X) :- h(X).
h(p).
```

A partir desse programa, ao se perguntar se *m(p)* é verdadeiro, a máquina de inferência do Prolog responde que sim (*yes*).

A linguagem Prolog trabalha no estilo encadeamento regressivo (*backward*). Por exemplo, seja o seguinte programa:

```
p(X,Y) :- q(X,Y).
q(a,b).
```

E se coloca a seguinte questão:

```
?- p(a,b).
```

Prolog faz as instanciações `X = a e Y = b` e dispara a primeira regra, ou seja, troca `p(a,b)` por `q(a,b)`. (Na verdade, como `p(X,Y) :- q(X,Y)` corresponde à implicação da lógica q(X, Y) $\rightarrow$ p(X,Y), então o Prolog está trocando o consequente da implicação p(a,b) pelo seu antecedente q(a,b), ou seja, estratégia *backward*.)

6.2.2 Cláusulas definidas

Uma cláusula (disjunção de literais) consiste em literais positivos e negativos:

$$p_0 \vee p_1 \vee \ldots \vee \neg n_0 \vee \neg n_1 \vee \ldots$$

Usando a lei de De Morgan:

$$\neg p \vee \neg q \equiv \neg(p \wedge q)$$

podem-se expressar as cláusulas como uma única implicação:

$$(n_0 \wedge n_1 \wedge ...) \rightarrow (p_0 \vee p_1 \vee ...)$$

Prolog não é baseado na forma clausal completa, mas sim num subconjunto bem menos expressivo, as *cláusulas de Horn*, que são cláusulas com no máximo um literal positivo. Existem então apenas três tipos de cláusulas de Horn:

- cláusulas unitárias — com um literal positivo da forma p_0 (ou, equivalentemente, $\rightarrow p_0$).
- cláusulas não unitárias — com um literal positivo e um ou mais literais negativos, isto é, da forma $p_0 \vee \neg n_0 \vee \neg n_1 \vee ...$ (ou, equivalentemente, $(n_0 \wedge n_1 \wedge ...) \rightarrow p_0$).
- cláusulas negativas — sem literais positivos, com um ou mais literais negativos, isto é, da forma $\neg n_0 \vee \neg n_1 \vee ...$ (ou, equivalentemente, $(n_0 \wedge n_1 \wedge ...) \rightarrow$).

Os dois primeiros tipos de cláusulas de Horn são conhecidos como *cláusulas definidas* porque elas têm exatamente um literal positivo — uma única conclusão definida para a implicação —, ao contrário de cláusulas gerais, com seu consequente potencialmente disjuntivo e indefinido.

6.2.3 Exemplo: relações familiares

Prolog é adequada para resolver problemas que envolvem objetos e relações entre objetos. Veja o exemplo da árvore familiar na Figura 6.1 (Bratko, 2000).

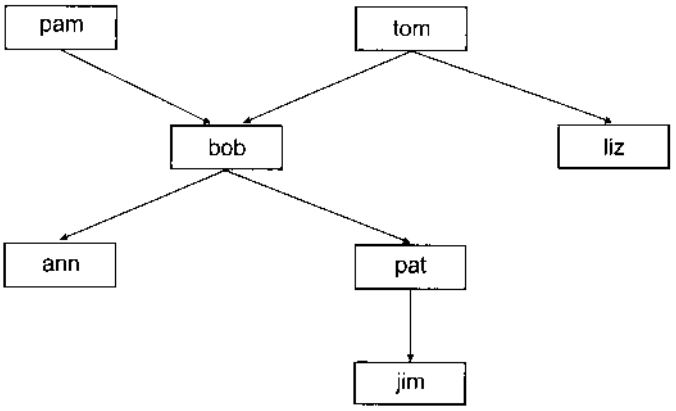


Figura 6.1 Uma árvore familiar.

O fato de que Tom é "pais" (pai/mãe) de Bob se escreve:

```
pais(tom,bob).
```

pais ⇒ relação
tom, bob ⇒ argumentos

Os fatos na árvore familiar do exemplo são:

```
pais(pam,bob).
pais(tom,bob).
pais(bob,ann).
pais(bob,pat).
pais(tom,liz).
pais(pat,jim).
```

São seis cláusulas declarando fatos sobre a relação "pais". Uma vez comunicados ao sistema Prolog os fatos sobre a relação "pais", podem-se colocar questões:

```
? - pais(bob,pat).
```

yes

```
? - pais(liz,pat).
```

no

```
? - pais(X,liz).
X = tom
```

Outra relação, avós:

```
? - pais(Y,jim),pais(X,Y).
X = bob
Y = pat
```

RESUMO

- Uma relação é definida pelo estabelecimento das n-uplas de objetos que satisfazem a relação.
- Um programa Prolog é constituído de cláusulas. Cada cláusula termina com um ponto.
- Os argumentos das relações podem ser (entre outras coisas) constantes ou variáveis.
- Perguntas ao sistema Prolog são constituídas de um ou mais objetivos. Uma sequência de objetivos separados por vírgulas significa a conjunção desses objetivos.

6.2.4 Estendendo o exemplo através de regras

Vejamos uma nova informação para a base de dados: o sexo das pessoas. Pode-se usar uma relação unária:

```
feminino(pam).
masculino(tom).
```

ou usar uma relação binária:

```
sexo(pam,feminino).
sexo(tom,masculino).
```

Para a também nova relação "filhos" (filho/filha), que é o inverso da relação pais.

cabeça	corpo
`filhos(Y,X)`	`:- pais(X,Y).`
(conclusão)	(condição)

em Linguagem de Predicados de Primeira Ordem:

$$\forall X \forall Y(\text{pais}(X,Y) \rightarrow \text{filhos}(Y,X))$$

A diferença principal entre fatos e regras é que o fato é uma cláusula incondicionalmente verdadeira e a regra é uma cláusula que pode ser verdade dependendo de se alguma condição for verdade.

Como as regras são usadas em Prolog? Para a verificação da veracidade do seguinte fato:

```
? - filhos(liz,tom). % liz é filha de tom?
```

o procedimento do Prolog, para executar a prova de tal fato, é o seguinte:

1. Procura tal fato na "base de conhecimento". Não existe tal fato.

2. Procura se existe uma regra sobre a relação filhos. Como existe, aplica a particular instanciação `X = tom` e `Y = liz`, ou seja,

   ```
   filhos(liz,tom) :- pais(tom,liz).
   ```

3. O objetivo original `filhos(liz,tom)` é substituído por um novo objetivo:

   ```
   pais(tom,liz).
   ```

4. O novo objetivo é encontrado como um fato na "base de conhecimento", e o Prolog responde *yes*.

RESUMO

- Cláusulas em Prolog são de três tipos: fatos, regras e questões.
- Fatos declaram coisas verdadeiras. Regras declaram coisas que são verdade dependendo de uma condição. Questões: através delas o usuário pode perguntar ao programa sobre a verdade de certas coisas.
- Cláusulas em Prolog são constituídas de cabeça e corpo. O corpo é uma lista de objetivos separados por vírgulas, a lista entendida como conjunção de objetivos.
- Fatos são cláusulas que possuem um corpo vazio. Questões são cláusulas que possuem somente corpo. Regras são cláusulas que possuem cabeça e corpo (não vazio).
- Durante a execução, uma variável pode ser instanciada por algum objeto.

- Variáveis são assumidas como universalmente quantificadas e lidas como *para todo*. Leituras alternativas são possíveis para variáveis que aparecem somente no corpo.

6.2.5 Definição de regra recursiva

Para definir uma nova relação, predecessor, vai-se fazê-lo da seguinte forma:

```
predecessor(X,Z) :- pais(X,Z).
predecessor(X,Z) :- pais(X,Y), predecessor(Y,Z).
```

Note que a definição do predicado predecessor usa recursividade. A programação recursiva é um dos princípios fundamentais da programação em Prolog. A relação predecessor é definida por duas cláusulas, o que consiste em um procedimento.

6.2.6 Como Prolog responde questões

Uma questão em Prolog é uma sequência de objetivos (um ou mais). Prolog tenta satisfazer os objetivos, isto é, demonstrar que os objetivos seguem logicamente a partir dos fatos e regras do programa. Se as questões contêm variáveis, Prolog também tenta achar a particular instanciação que satisfaz os objetivos. Os fatos e regras são aceitos como um conjunto de axiomas (hipóteses). A questão do usuário é aceita como uma possível tese de um teorema. Prolog tenta provar esse teorema (demonstrar que ele segue logicamente dos axiomas). Prolog realiza encadeamento lógico regressivo (*backward*), ou seja, a partir da meta, aplica as regras (no sentido *backward*) e termina nos fatos. Pode-se dizer também que Prolog parte da negação (implícita) da meta até chegar à cláusula vazia, utilizando a estratégia por preferência unitária (demonstração por absurdo): *refutação*.

EXEMPLO sequência de prova da conjectura (tese):

```
?- predecessor(tom,pat).
```

(*Obs.*: Note o sinal "–" após a interrogação no *prompt* do interpretador Prolog. Esse sinal representa que a cláusula meta deve ser *negada* antes do início da prova. Essa negação é feita internamente pelo mecanismo de inferência do Prolog. Mesmo porque a *questão* Prolog deve ser um (ou mais) literal negativo!)

1. Prolog procura um fato que combine com o objetivo:
   se sim: *yes*
   se não: vai para o passo 2.

Quando Prolog encontra o fato que unifica com a meta, aplica-se a regra da resolução (literal negativo da meta e literal positivo do fato se cancelam, resultando na cláusula vazia). A razão de o Prolog procurar primeiro o fato é que a estratégia usada é a preferência unitária e o fato é cláusula unitária. Apesar de a estratégia por preferência unitária não ser completa, é a mais eficiente de todas. Caso a busca não seja bem-sucedida, o Prolog realiza o *backtracking* e uma nova busca é efetuada.

2. Prolog procura uma regra cuja cabeça combine com o objetivo:
   se não: *no*
   se sim: vai para o passo 3.

Se não há fato e nem regra cuja cabeça (literal positivo) combina o objetivo, então não há como aplicar a regra da resolução. Portanto não há solução.

3. Prolog "dispara" a regra para a particular instanciação
   ```
   predecessor(X,Z) :- pais(X,Z).
   X = tom
   Z = pat
   ```
   e substitui o objetivo corrente por
   ```
   ? - pais(tom,pat).
   ```

   Lembre-se de que na regra Prolog a cabeça corresponde ao consequente da implicação lógica (conclusão) e o corpo, ao antecedente (condição). Quando Prolog *dispara* a regra, na verdade "troca" a cabeça pelo corpo, ou seja, "retorna" do consequente para o antecedente na implicação lógica, exatamente como deve ser no encadeamento *backward*. Além disso, pode-se pensar em termos de regra da resolução: trocar a cabeça pelo corpo significa "cancelar" o literal negativo da meta com o literal positivo da cabeça, resultando no literal negativo do corpo da cláusula Prolog.

4. Prolog procura uma cláusula cuja cabeça combina com o objetivo (novo)
   se sim e cláusula = fato: *yes*
   se sim e cláusula = regra: "dispara" a regra
   se não: *backtrack*

   Repete-se o processo. Caso encontre um fato, termina, pois no sistema *backward* o encadeamento termina no fato. E na refutação encontrar o fato significa chegar à cláusula vazia (lembre-se: havia um literal negativo e o fato é seu literal complementar).

5. Prolog "dispara" a segunda regra
   ```
   predecessor(X,Z) :- pais(X,Y), predecessor(Y,Z).
   X = tom
   Z = pat
   ```
   e substitui o objetivo corrente
   ```
   ? - pais (tom,Y), predecessor (Y,pat).
   ```

6. Prolog procura o primeiro objetivo, que combina com o fato `pais(tom,bob)`, instanciando `Y = bob.`

7. Resta analisar
   ```
   ? - predecessor(bob,pat).
   ```

8. Prolog "dispara" a primeira regra novamente e substitui o objetivo

```
? - pais(bob,pat).
```

que finalmente é encontrado como fato no banco de conhecimento.

6.2.7 Significados declarativo e procedimental

É possível distinguir entre os significados declarativo (o quê) do procedimental (o como) de um programa Prolog. A habilidade do Prolog de realizar muitos detalhes procedimentais por si mesmo é considerada uma de suas vantagens.

RESUMO

- A programação em Prolog consiste em definir relações e fazer questões sobre essas relações.
- Prolog consiste em cláusulas de três tipos: fatos, regras e questões.
- Uma relação pode ser especificada por fatos simplesmente estabelecendo as n-uplas de objetos que satisfazem a relação, ou estabelecendo regras a respeito da relação.
- Um procedimento é um conjunto de cláusulas a respeito de uma mesma relação.
- Questionar sobre relações, através de perguntas, lembra questionar um banco de dados. A resposta consiste em um conjunto de objetos que satisfazem a questão.
- A resposta é obtida através de um complexo processo que envolve inferência lógica, exploração entre alternativas, *backtracking*.
- Existem dois significados nos programas Prolog: declarativo e procedimental. O declarativo é vantajoso do ponto de vista da programação, apesar de que os detalhes procedimentais muitas vezes devem ser considerados pelo programador.

6.3 SINTAXE E SIGNIFICADO DE PROGRAMAS PROLOG

6.3.1 Predicados

Os predicados Prolog, expressões que denotam relações entre objetos, têm a sintaxe dos átomos, ou seja, inicia-se um predicado sempre com letras minúsculas:

```
pai (X, tom)
mãe (maria, Y)
```

6.3.2 Objetos de dados

A Figura 6.2 mostra a classificação de objetos em Prolog. O sistema Prolog reconhece o tipo de um objeto no programa por sua forma sintática. Isso é possível porque a sintaxe do Prolog especifica formas diferentes para cada tipo de objetos de dados. Os objetos de dados da linguagem são chamados de *termos*. Um termo ou é uma constante, uma variável ou um termo composto.

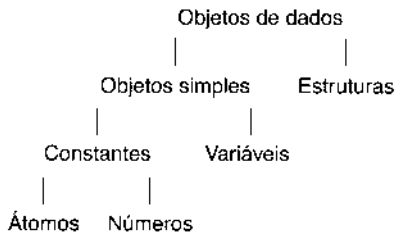


Figura 6.2 Objetos de dados em Prolog.

6.3.2.1 Átomos e números

Como nas linguagens de programação convencionais, as constantes denotam objetos elementares definidos. Os átomos podem ser construídos de três formas:

1. Cadeias de letras, dígitos e o caractere "underscore" ('_'), começando com uma letra minúscula:

```
ana        nil        x25        x_25
```

2. Cadeias de caracteres especiais:

```
<--->
=====>
```

Obs.: Ao usar átomos dessa forma, deve-se tomar cuidado, pois algumas cadeias de caracteres especiais já têm um significado predefinido; um exemplo é ':-'.

3. Cadeias de caracteres entre apóstrofos. Isso é útil quando se deseja, por exemplo, ter um átomo que começa com uma letra maiúscula. Colocar entre apóstrofos torna-os distintos de variáveis:

```
'Tom'
'America_do_Sul'
```

Os números usados no Prolog incluem números inteiros e reais:

```
1     1313  0     -97   3.14  -0.0035
```

6.3.2.2 Variáveis

Uma variável deve ser vista como algum objeto particular mas não identificado. Note que uma variável não é simplesmente uma posição de armazenamento à qual se pode atribuir um valor, como na maioria das linguagens de programação; em vez disso, ela é um nome local para algum objeto de dado. As varáveis são cadeias de letras, dígitos e o caractere "underscore". Começam com uma letra maiúscula ou um "underscore":

```
X     Resultado   _23
```

Quando uma variável aparece apenas uma vez em uma cláusula, não há necessidade de nomeá-la. Por exemplo, na regra seguinte:

```
temcrianca(X) :- pais(X,Y).
```

Essa regra diz que para todo X X tem uma criança se X é um pais (pai/mãe) de algum Y. Essa cláusula pode ser reescrita:

```
temcrianca(X) :- pais(X,_).
```

O escopo lexical de uma variável é uma cláusula. Isso significa que, por exemplo, se o nome X15 ocorre em duas cláusulas, então se trata de duas variáveis diferentes. Mas cada ocorrência de X15 dentro da mesma cláusula significa a mesma variável.

6.3.2.3 Estruturas

Estruturas são objetos que têm muitos componentes. Os componentes podem, por sua vez, ser estruturas. Por exemplo, a data pode ser vista como uma estrutura com três componentes: dia, mês e ano. As estruturas são tratadas como objetos simples. Para combinar os componentes em um único objeto, deve-se escolher um functor. Por exemplo, a data 1.º de maio de 1998 pode ser escrita assim (Figura 6.3):

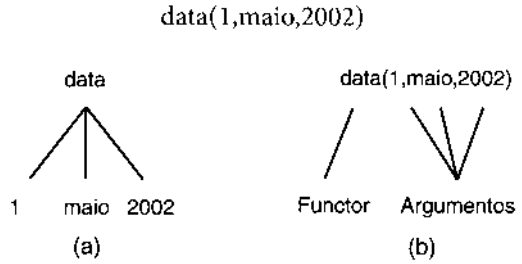


Figura 6.3 Data é um exemplo de objeto estruturado: (a) representado como uma árvore; (b) como é escrito em Prolog.

Os componentes de um functor podem ser constantes, como no exemplo anterior, ou variáveis, como em:

```
data(D,maio,2002)
```

Cada functor é definido por duas coisas:
1. o nome, cuja sintaxe é a dos átomos;
2. a aridade, ou seja, o número de argumentos.

Todos os objetos estruturados em Prolog são árvores, representados nos programas por termos. A Figura 6.4 mostra a estrutura de árvore que corresponde à expressão aritmética

```
(a + b) * (c - 5)
```

Isso pode ser escrito usando-se os símbolos '*', '+' e '-' como functors, da seguinte forma:

```
*(+(a,b),-(c,5))
```

Figura 6.4 Uma estrutura de árvore que corresponde à expressão aritmética (a + b) * (c - 5).

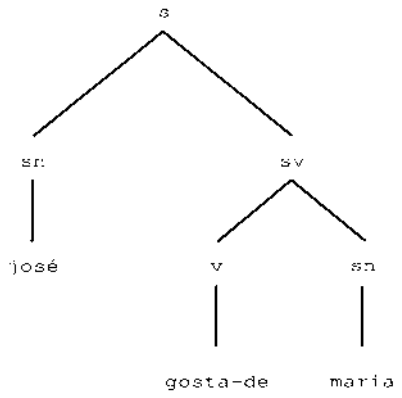
Na nossa base de dados da família, pode-se incluir um predicado chamado data_nasc, de aridade 2, em que o primeiro argumento é o nome e o segundo, o functor **data**, que representa a data de nascimento da cada membro da família:

```
data_nasc(tom, data(D,M,A)).
```

Pode-se imaginar que um functor seja um tipo registro e os argumentos de um termo composto sejam os campos do registro. Os termos compostos são usualmente representados graficamente como árvores. Por exemplo, o termo:

```
s(sn(josé),sv(v(gosta-de),sn(maria)))
```

seria representado como a estrutura:



Algumas vezes é conveniente escrever um termo composto usando uma notação infix opcional, por exemplo:

X + Y (P;Q) X < Y

em vez de:

+(X,Y) ;(P,Q) <(X,Y)

Note que um átomo é considerado um functor de aridade 0.

6.3.3 "Matching" (unificação)

Dados dois termos, diz-se que eles "unificam" se:

- são idênticos, ou
- as variáveis em ambos são instanciadas a objetos, de modo a serem idênticos.

EXEMPLO

data(D,M,2002) e **data(D1,maio,A1)** UNIFICAM, pois:
D = D1
M = maio
A1 = 2002, que são as *instanciações*.

A unificação (*matching*) é um processo que pega como entradas dois termos e checa se eles unificam. Se os termos não unificam, diz-se que esse processo falha. Se eles unificam, então o processo é bem-sucedido e as variáveis são instanciadas em ambos os termos para valores tais que os tornem idênticos. Considere a unificação de duas datas:

?- data(D,M,2002) = data(D1,maio,A1).

Já foi mencionado que a instanciação **D = D1, M = maio** e **A1 = 2002** satisfaz a unificação. Existem, entretanto, outras instanciações que também satisfazem. Duas delas são:

```
D = 1
D1 = 1
M = maio
A1 = 2002
D = três
D1 = três
M = maio
A1 = 2002
```

Diz-se que essas duas instanciações são menos gerais que a primeira, pois elas atribuem constantes para as variáveis `D` e `D1`. A unificação em Prolog sempre resulta na instanciação mais geral. Essa instanciação mantém, na medida do possível, o maior número de variáveis, para que elas possam ser instanciadas mais tarde (unificação mais geral).

As regras gerais para decidir se dois termos, **S** e **T**, unificam são as seguintes:

1. Se **S** e **T** são constantes então **S** e **T** unificam apenas se são o mesmo objeto.
2. Se **S** é uma variável e **T** é qualquer coisa, então eles unificam, e **S** é instanciado a **T**. Se **T** é uma variável, então **T** é instanciado a **S**.
3. Se **S** e **T** são estruturas então eles unificam apenas se
   a. S e T têm o mesmo functor principal e
   b. Todos os seus componentes correspondentes unificam. A instanciação resultante é determinada pela unificação dos componentes. Veja o seguinte exemplo:

```
?- triangulo(ponto(1,1),A,ponto(2,3)) = triangulo(X,ponto(4,Y),ponto(2,Z)).
```

Nesse caso, têm-se duas estruturas. Para que seja possível unificá-las, é necessário que elas possuam o mesmo functor principal (no caso, triângulo). Seus componentes devem ser instanciados da seguinte forma: `X = ponto(1,1)`; `A = ponto(4,Y)` e `Z = 3`. Ou seja, essa é a instanciação mais geral que satisfaz a unificação.

6.3.4 Cláusulas

Uma unidade fundamental de um programa lógico é a meta ou chamada de procedimento. São exemplos:

```
dá(tom,maçã,professor)
reverso([1,2,3],1)
X < Y
```

Uma meta é um tipo especial de termo, distinguido apenas pelo contexto no qual ele aparece no programa. O functor (principal) de uma meta é chamado de *predicado*. Ele corresponde a um nome de procedimento numa linguagem de programação convencional.

Um *programa* lógico consiste simplesmente em uma sequência de comandos chamados de *cláusulas*. Uma cláusula compreende uma *cabeça* e um *corpo*. A cabeça ou consiste em uma meta simples ou é vazia. O corpo consiste em uma sequência de zero ou mais metas (isto é, ele também pode ser vazio).

Se nem a cabeça e nem o corpo de cláusula são vazios, a cláusula é chamada de não unitária (*regra*), e escrita na forma:

```
P :- Q,R,S.
```

em que P é a meta cabeça e Q, R e S são as metas que fazem parte do corpo. Pode-se ler tal cláusula ou *declarativamente* como:

"P é verdadeiro se Q e R e S são verdadeiros."

ou *procedimentalmente* como:

"Para satisfazer a meta P, satisfaça as metas Q, R e S."

Se o corpo da cláusula está vazio, a cláusula é chamada unitária (*fato*), e escrita na forma:

```
P.
```

em que P é a meta cabeça. Isso é interpretado declarativamente como:

"P é verdadeiro."

e procedimentalmente como:

"Meta P é satisfeita."

Finalmente, se a cabeça da cláusula é vazia, chama-se a cláusula de *questão* e escreve-se na forma:

```
?-P,Q.
```

em que P e Q são as metas do corpo. Tal questão é lida declarativamente como:

"P e Q são verdadeiros?"

e procedimentalmente como:

"Satisfaça as metas P e Q."

As cláusulas geralmente contêm variáveis. Note que as variáveis em cláusulas diferentes são completamente independentes, mesmo que elas tenham o mesmo nome — isto é, o "escopo lexical" de uma variável é limitado a uma única cláusula. Cada variável distinta em uma cláusula deveria ser interpretada como contendo um valor arbitrário. Para ilustrar isso, têm-se alguns exemplos de cláusulas contendo variáveis, com possíveis leituras declarativas e procedimentais, segundo Pereira e Warren (1980):

1. `empregado(X) :- emprega(Y,X).`
"Para quaisquer `X` e `Y`,`X` é empregado se `Y` emprega `X`."
"Para descobrir se `X` é empregado, ache um `Y` que emprega `X`."

2. `derivativo(X,X,1).`
"Para qualquer `X`, o derivativo de `X` com respeito a `X` é `1`."
"A meta de achar um derivativo para a expressão `X` com respeito a `X` é satisfeita pelo resultado `1`."

3. `?-tem_casco(X),aquático(X).`
"É verdadeiro para qualquer **X**, que **X** tem casco e **X** é aquático?"
"Ache um **X** que tem casco e é aquático."

Num programa lógico, o *procedimento* para um predicado particular é a sequência de cláusulas no programa cujas metas cabeça têm aquele predicado como functor principal. Por exemplo, o procedimento para o predicado ternário 'concatenar' deve consistir em duas cláusulas:

```
concatenar([X|L1],L2,[X|L3]) :-
        concatenar(L1,L2,L3).
concatenar([],L,L).
```

em que `'concatenar(L1,L2,L3)'` significa "a lista **L1** concatenada com a lista **L2** é a lista **L3**".

Como se viu, as metas no corpo de uma cláusula são ligadas pelo operador `','` que pode ser interpretado como conjunção. Por conveniência, algumas vezes pode-se usar um operador `';'` significando disjunção. (A precedência de `';'` é tal que ele domina `','` mas é dominado por `':-'`.) Um exemplo é a cláusula:

```
avô(X,Z) :-
   (mãe(X,Y); pai(X,Y)), pai(Y,Z).
```

que pode ser lida como:

"Para quaisquer **X**, **Y** e **Z**,
X tem **Z** como um avô se
ou a mãe de **X** é **Y** ou o pai de **X** é **Y**,
e o pai de **Y** é **Z**."

Pode-se eliminar o uso de disjunção, definindo um predicado extra — ou seja, o exemplo anterior é equivalente a:

```
avô(X,Z) :- pais(X,Y), pai(Y,Z).
pais(X,Y) :- mãe(X,Y).
pais(X,Y) :- pai(X,Y).
```

– e tal disjunção não será mencionada na descrição da semântica de cláusulas mais formal que se segue.

6.4 SEMÂNTICAS DECLARATIVA E PROCEDIMENTAL

A semântica das cláusulas definidas deveria estar clara a partir das interpretações informais já dadas. Entretanto, é útil ter uma definição precisa. A *semântica declarativa* de cláusulas definidas revela quais metas podem ser consideradas verdadeiras de acordo com um dado programa e é definida recursivamente como se segue.

> Uma meta é *verdadeira* se ela é a cabeça de alguma instância da cláusula e cada uma das metas (se existir) no corpo dessa instância da cláusula for verdadeira, em que uma *instância* de uma cláusula (ou termo) é obtida, pela substituição, para cada zero ou mais variáveis da cláusula, de um novo termo para todas as ocorrências da variável.

Por exemplo, segundo Pereira e Warren (1980), se um programa contém o procedimento precedente para 'concatenar', então a semântica declarativa diz que

```
concatenar([a],[b],[a,b])
```

é verdadeiro, porque sua meta é a cabeça de certa instância da primeira cláusula para 'concatenar', ou seja,

```
concatenar([a],[b],[a,b]) :- concatenar([],[b],[b])
```

e sabe-se que a única meta no corpo dessa instância de cláusula é verdadeira, já que é uma instância do fato, que é a segunda cláusula para 'concatenar'.

Já foi dito que os programas Prolog podem ser entendidos de duas formas: declarativamente e procedimentalmente. Seja a seguinte cláusula:

```
p :- q, r.
```

em que p, q e r têm a sintaxe dos termos. Leituras declarativas alternativas dessa cláusula seriam:

p é verdadeiro se **q** e **r** são verdadeiros.

A partir de **q** e **r**, obter **p**.

Duas leituras procedimentais alternativas dessa cláusula são:

Para resolver o problema **p**, primeiro resolva o subproblema **q** e então o subproblema **r**.

Para satisfazer **p**, primeiro satisfaça **q** e então **r**.

Portanto, a diferença entre as leituras declarativa e procedimental é que a última define não apenas as relações lógicas entre a cabeça da cláusula e as metas no corpo, mas também a ordem na qual as metas são processadas.

O significado declarativo dos programas determina se uma dada meta é verdadeira, e, se for, para quais valores de variáveis ela é verdadeira. Para definir precisamente o significado declarativo, é necessário introduzir o conceito de instância de uma cláusula. Uma instância de uma cláusula C é a cláusula C com cada uma de suas variáveis substituídas por algum termo. Uma variante de uma cláusula C é uma instância da cláusula C em que cada variável é substituída por outra variável. Por exemplo, considere a cláusula:

```
temcrianca(X) :- pais(X,Y).
```

Duas variantes dessa cláusula são:

```
temcrianca(A) :- pais(A,B).
temcrianca(X1) :- pais(X1,X2).
```

Instâncias dessa cláusula são:

```
temcrianca(pedro) :- pais(pedro,Z).
temcrianca(paulo) :-pais(paulo,pequeno(carolina)).
```

Dados um programa e uma meta G, o significado declarativo diz:

Uma meta G é verdadeira (isto é, satisfatível, ou segue logicamente do programa) se e somente se
1. Existe uma cláusula C no programa tal que
2. Existe uma cláusula instância I de C tal que

a. a cabeça de I é idêntica a G, e
b. todas as metas no corpo de I são verdadeiras.

Em geral, uma questão ao sistema Prolog é uma lista de metas separadas por vírgulas. Uma lista de metas é verdadeira se todas as metas na lista são verdadeiras para a mesma instanciação de variáveis. Os valores das variáveis resultam da instanciação mais geral.

Uma vírgula entre metas denota a conjunção de metas: todas elas devem ser verdadeiras. Mas Prolog também aceita a disjunção de metas: basta qualquer uma das metas numa disjunção ser verdadeira. A disjunção é indicada por um ponto e vírgula. Por exemplo,

```
p :- q; r.
```

que é lido como: p é verdadeiro se q é verdadeiro ou r é verdadeiro. O significado dessa cláusula é equivalente ao das seguintes cláusulas juntas:

```
p :- q.
p :- r.
```

Note que a semântica declarativa não faz referência à sequência de metas dentro do corpo de uma cláusula, nem à sequência de cláusulas dentro de um programa. Essa informação de sequen-

ciamento é, entretanto, muito relevante para a *semântica procedimental* que Prolog dá às cláusulas definidas. A semântica procedimental define exatamente como o sistema Prolog executará uma meta, e a informação de sequenciamento é a forma pela qual o programador Prolog direciona o sistema para executar seu programa. O efeito de executar uma meta é enumerar, uma por uma, suas instâncias verdadeiras. Aqui está uma definição informal da semântica procedimental.

> Para *executar* uma meta, o sistema procura a primeira cláusula cuja cabeça *casa* ou *unifica* com a meta. O processo de *unificação* acha a instância comum mais geral de dois termos, que é única se existir. Se um casamento é encontrado, a instância da cláusula casada é então *ativada*, executando por sua vez, da esquerda para a direita, cada uma das metas (se existir) em seu corpo. Se, alguma vez, o sistema falha ao achar um casamento para uma meta, ele retorna (*backtracks*), isto é, ele rejeita a cláusula ativada mais recente, desfazendo quaisquer substituições feitas pelo casamento com a cabeça da cláusula. Depois ele reconsidera a meta original que ativou a cláusula rejeitada e tenta achar uma cláusula subsequente que também case com a meta.

Por exemplo, considere a questão:

```
?- concatenar(X,Y,[a,b])
```

que pode ser lida declarativamente como:

"Existem as listas **X** e **Y** que quando concatenadas fornecem a lista **[a,b]**?"

Se se executar a meta expressa nessa questão, acha-se que ela casa com a cabeça da primeira cláusula para 'concatenar', com **X** instanciado a **[a|X1]**. A nova variável **X1** é restrita pela nova meta (ou chamada de procedimento recursivo) que é produzida:

```
concatenar(X1,Y,[b])
```

Novamente essa meta casa com a primeira cláusula, instanciando **X1** a **[b|X2]** e fornecendo a nova meta:

```
concatenar(X2,Y,[])
```

Agora essa meta casará apenas com a segunda cláusula, instanciando **X2** e **Y** a **[]**. Já que não existem mais metas a serem executadas, tem-se a solução:

```
X = [a,b]
Y = []
```

isto é, uma instância verdadeira da meta original é:

```
concatenar([a,b],[],[a,b])
```

Se for rejeitada essa solução, o *backtracking* gerará outras soluções:

```
                                X = [a]    Y = [b]
X = []          Y = [a,b]
```

nessa ordem, recasando, com a segunda cláusula para 'concatenar', metas já resolvidas uma vez usando a primeira cláusula.

O significado procedimental especifica como Prolog responde às questões. Responder a uma questão significa tentar satisfazer uma lista de metas, isto é, encontrar uma instanciação particular das variáveis que ocorrem nas metas de tal modo que os objetivos sigam logicamente do programa.

Esquematicamente:

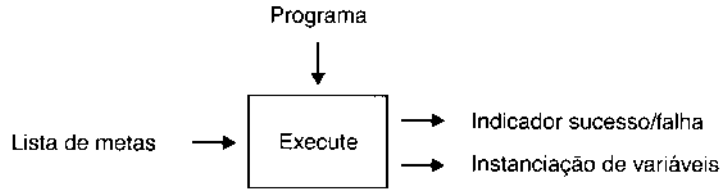


Figura 6.5 Visão da entrada/saída do procedimento que executa uma lista de metas.

O significado das duas saídas na Figura 6.5 resulta em: (1) o indicador de sucesso/falha é 'yes' se as metas são satisfatíveis e 'no' em caso contrário. 'Yes' significa uma terminação bem-sucedida, e 'no', uma falha. (2) uma instanciação de variáveis é produzida apenas no caso de uma terminação bem-sucedida; no caso de falha não há instanciação.

Veja o seguinte exemplo:

PROGRAMA

```
grande(urso).          % cláusula 1
grande(elefante).      % cláusula 2
pequeno(gato).         % cláusula 3
marrom(urso).          % cláusula 4
preto(gato).           % cláusula 5
cinza(elefante).       % cláusula 6
escuro(Z) :-           % cláusula 7
   preto(Z).           % qualquer coisa preta é escura
escuro(Z) :-           % cláusula 8
   marrom(Z).          % qualquer coisa marrom é escura
```

QUESTÃO

```
?- escuro(X), grande(X).
   % quem é escuro e grande?
```

Execução ("Trace")

1. Lista de metas iniciais: **escuro(X), grande(X).**

2. Percorrer o programa de cima para baixo, procurando uma cláusula cuja cabeça combina com **escuro(X)**, o primeiro objetivo. A cláusula 7 é encontrada:
   escuro(Z) :- preto(Z).

   Trocar a primeira meta pelo corpo instanciado da cláusula 7, obtendo uma nova lista de metas:

   preto(X), grande(X).

3. Percorrer o programa para achar uma cláusula cuja cabeça combine com **preto(X)**. A cláusula 5 é encontrada:

   preto(gato).

   Essa cláusula não tem corpo, tal que a lista meta, propriamente instanciada, se reduz para:

   grande(gato).

4. Percorrer o programa para a meta **grande(gato)**. Nenhuma cláusula encontrada. Portanto retorne (*backtrack*) ao passo (3) e desfaça a instanciação **X = gato**. Agora a lista meta é novamente:

   preto(X), grande(X).

   Continuar percorrendo o programa para baixo da cláusula 5. Nenhuma cláusula encontrada. Portanto, retornar (*backtrack*) para o passo (2), restituindo a lista de metas original **escuro(X), grande(X)** e continuar percorrendo abaixo da cláusula 7. A cláusula 8 é encontrada:

   escuro(Z) :- marrom(Z).

   Substituir a primeira meta na lista de metas por **marrom(X)**, dando:

   marrom(X), grande(X).

5. Percorrer o programa para combinar **marrom(X)**, achando **marrom(urso)**. Não há corpo, então a lista de metas se reduz a **grande(urso)**. Como essa cláusula é encontrada como fato (cláusula 1), então a lista de metas se reduz ao vazio. Isso indica a terminação bem-sucedida, e a instanciação da variável correspondente é:

```
X = urso
```

6.5 LISTAS

6.5.1 Representação de listas

A *lista* é uma estrutura de dados simples muito usada em programação não numérica. Uma lista é uma sequência de qualquer número de itens, tais como **ana**, **tenis**, **tom**, **esqui**. Tal lista pode ser escrita em Prolog como:

```
[ana,tenis,tom,esqui]
```

Essa é, entretanto, apenas a aparência externa das listas. Como já foi visto, todos os objetos estruturados em Prolog são árvores.

Como se pode representar uma lista como um objeto Prolog padrão? Devem-se considerar dois casos: ou a lista está vazia ou não vazia. No primeiro caso, a lista é simplesmente escrita como o átomo do Prolog, **[]**. No segundo caso, a lista pode ser vista como consistindo em duas coisas:

1. o primeiro item, chamado a *cabeça* da lista;
2. a parte remanescente da lista, chamada de *cauda*.

Para o exemplo anterior, a cabeça é **ana** e a cauda é a lista

```
[tenis,tom,esqui]
```

Em geral, a cabeça pode ser qualquer coisa (qualquer objeto Prolog, por exemplo, uma árvore ou uma variável); a cauda deve ser uma lista. A cabeça e a cauda podem ser combinadas em uma estrutura por um functor especial. A escolha desse functor depende da implementação Prolog; assume-se aqui que é o ponto:

```
.(Cabeca,Cauda)
```

Cauda é por sua vez uma lista, que pode estar vazia ou ter sua própria cabeça e cauda. Portanto, para representar listas de qualquer tamanho é só seguir a estrutura estabelecida, como no exemplo a seguir:

```
.(ana,.(tenis,.(tom,.(esqui,[]))))
```

A Figura 6.6 mostra a estrutura de árvore correspondente. Note que a lista vazia aparece no termo anterior. Isso ocorre porque a última cauda é uma lista de um único item:

```
[esqui]
```

Esta lista tem a lista vazia como cauda:

```
[esqui]=.(esqui,[])
```

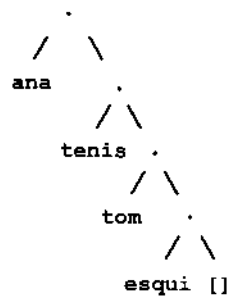


Figura 6.6 Representação em árvore para a lista `[ana, tenis, tom, esqui]`.

6.5.2 Exemplos de aplicações de listas

1. Concatenação de duas listas `L = L1+L2`:

```
conc([],L,L).                %seL1 é vazia,
                             %L = L2
conc([X|L1],L2,[X|L3])       :-
                             %senão,X(cabeça
                             % de L1) será a cabeça
conc(L1,L2,L3).              % de L e a cauda de L1
                             % concatenada com
                             % L2 resultará na
                             % cauda de L
```

2. X é membro de uma lista:

```
membro (X,[X|T]).            % X é membro se for
                             % cabeça
membro (X,[H|T]) :-          % senão, X é membro da
                             % cauda membro (X, T).
```

3. adicionar um elemento na cabeça da lista:

```
ad (X,L,[X|L]).              % X vira cabeça da nova
                             % lista
```

4. apagar um elemento X de uma lista:

```
apag(X,[X|T],T).          % se X for a cabeça,
                          % retira-a
apag(X,[Y|T],[Y|T1]) :-
                          % senão apaga X da cauda
            apag (X,T,T1).
```

6.6 ALGUMAS CONSIDERAÇÕES RELEVANTES

6.6.1 O predicado not

Para o seguinte exemplo:

X está em casa se *X* não está fora.
Tina está em casa.
José é o marido de Tina.

Em Prolog:

```
casa (X) :- not (fora(X)).
fora(tina).
marido(josé, tina).
```

Caso se coloquem as questões:

```
?- casa (josé).
```
yes
```
?- casa (X).
```
no

Por que isso acontece? Em Prolog, metas negadas podem apenas ser testadas como metas testes e não como metas de unificação.

6.6.2 Loop infinito

Suponha a seguinte cláusula Prolog:

```
p :- p.
```

Agora suponha a seguinte questão para essa cláusula:

```
?- p.
```

Nesse caso, Prolog entra em *loop* infinito. Por quê? Porque a cláusula `p :- p.` diz que "`p` é verdadeiro se `p` é verdadeiro". Está declarativamente perfeitamente correta, mas procedimentalmente pode causar problemas ao Prolog.

6.6.3 Concatenação

Para o predicado **conc**:

```
conc([],L,L).
conc([H|T1],L2,[H|T]) :- conc(T1,L2,T).
```

a seguinte questão:

```
?- append(X,Y,[a,b]).
```

daria quais respostas através da solicitação de *backtracking* (;)?

```
X = []
Y = [a,b] → ;
X = [a]
Y = [b]   → ;
X = [a,b]
Y = []    → ;
no
```

6.7 CONCLUSÕES

Prolog é a linguagem para *programação lógica* e pode fazer muitas coisas. Mas tem quatro fraquezas lógicas fundamentais (Rowe, 1988):

- Prolog não permite fatos ou conclusões disjuntivos, ou seja, sentenças em que uma ou mais coisas são verdadeiras, mas não se sabe quais.
- Prolog não permite fatos ou conclusões negativos, isto é, sentenças diretas em que alguma coisa é falsa.
- Prolog não permite que muitos fatos, conclusões ou regras tenham quantificação existencial, isto é, sentenças em que exista algum valor de variável, ainda que não se saiba qual, tal que o predicado que a contém seja verdadeiro.
- Prolog não permite diretamente *lógica de segunda ordem*, nomes de predicados como variáveis, isto é, sentenças sobre P em que P significa um nome de predicado.

RESUMO

- Uma lista é uma estrutura de dados que ou está vazia ou consiste em duas partes: uma *cabeça* e uma *cauda*. A cauda por sua vez também é uma lista.
- As listas são manipuladas pelo Prolog com um caso especial de *árvores binárias*. Para melhorar a legibilidade, Prolog provê uma notação especial para listas:

```
[Item1, Item2, ...]
```

ou

```
[Cabeca | Cauda]
```

ou

```
[Item1, Item2, ... | Outros]
```

6.8 AMBIENTES PROLOG DISPONÍVEIS

Há vários ambientes Prolog disponíveis, desde o Arity Prolog, com interpretador e compilador, cuja interface é simples (em sistema DOS), mas que tem a grande vantagem de trabalhar com o Prolog puro (não há necessidade de definição de diretivas). O Arity Prolog você encontra de graça no site: http://www.arity.com/www.pl/products/ap.htm.

Há também o Visual Prolog com interface mais elaborada, porém com a necessidade de aprender a trabalhar com esse ambiente, além do conhecimento da linguagem Prolog. Disponível uma versão para uso não comercial em: http://www.visual-prolog.com/.

6.9 EXEMPLOS DE PROGRAMAS COMPLETOS PROLOG

1. Example.ari

```
/*EXAMPLE.ARI - Interpreta expressões aritméticas. Digite uma
expressão aritmética válida no Prolog no prompt "fácil>", se-
guido por um ponto final e tecle Enter.Para sair do programa
digite Ctrl-c ou Ctrl-Break. */

:- public main/0, restart/0.

main :- repeat, fácil, fail.
restart :- write('Programa terminado'), halt.
fácil :- nl, write('fácil>'), read(X), compute(X).
compute(X) :- Ans is X,
    case([(Ans = err) -> write('Não posso avaliar':X)
    | write(X = Ans)]).
```

2. Towers.ari

```
/* TOWERS.ARI : Torres de Hanói */
:- module towers.
:- public towers/0.
towers :- repeat,
   write('Número de anéis (ou ctrl-c para terminar): '),
   read(X), hanoi(X), fail.
hanoi(N) :- move(N,esquerda,centro,direita), !.
move(0,_,_,_) :- !.
move(N,A,B,C):-M is N-1,move(M,A,C,B),inform(A,B),move(M, C,B,A).
inform(A, B) :- write([move, disco, de, A, para, B]), nl.
```

Explicações de alguns predicados usados nesses exemplos:

repeat = sempre sucede, e, quando encontrado durante o *backtracking*, sempre sucede novamente.

fail = essa meta sempre falha
halt = sai do interpretador ou da aplicação compilada.
X is E = avalia **E** e unifica o valor com **X**. Se não for possível a avaliação de **E**, retorna **err**.
case (+[A1 → B1, A2 → B2, ..., | C]) = executa **B1** se **A1** sucede, senão executa **B2** se **A2** sucede etc. Se nenhuma das situações sucede, **C** é executado.
T1 = T2 : tenta unificar **T1 e T2**.
! = o predicado de controle *cut* sempre sucede e previne o *backtracking*. *Cut* é uma meta que sempre tem sucesso, mas se o *backtracking* encontra um *cut*, então a meta que contém o *cut* imediatamente falha, e a meta que chamou essa meta também falha.

3. O seguinte exemplo traz um programa Prolog para Processamento de Linguagem Natural, com as diretivas (*entry-points*) para compilação no Prolog:

```
% ANALISADOR SINTÁTICO
% Autor: João Luís Garcia Rosa
%  SCC-ICMC-USP
%  NILC
%  e-mail: joaoluis@icmc.usp.br
%
% definindo os "entry-points" para o compilador
:- module anasin.
:- public main/0.
main :- repeat, anasin, fail.
anasin :- nl, write('Entre com a frase: '),
   read(X),
   frase(X,A),
   nl, write('Frase CORRETA sintaticamente'), nl, nl.
% as letras são transformadas em minúsculas
letra(C,D) :-
   C >= "A", C =< "Z",
   D is C + 32.
letra(C,C) :-
   C >= "a", C =< "z".
   identificador(C,D) :- letra(C,D).
% o predicado append coloca uma lista no
% final de outra lista
append([],L,L).
append([H|T],L,[H|T1]) :- append(T,L,T1).
% o predicado pega_resto_pal pega o restante da
% palavra e vai criando uma nova lista, até
% encontrar o espaço ou o ponto
pega_resto_pal([H|T],Lista,Pal,X) :-
   identificador(H,Id), !,
   append(Lista,[Id],Nlista),
   pega_resto_pal(T,Nlista,Pal,X).
```

```
pega_resto_pal([32|T],Lista,Pal,T) :-
   name(Pal,Lista), !.
pega_resto_pal(["."|T],Lista,Pal,T) :-
   name(Pal,Lista), !.
pega_resto_pal([],Lista,Pal,[]) :-
   !,name(Pal,Lista).
% tokenizar cria a partir de uma sequência de
% caracteres ASCII teclados, uma lista com todas
% as palavras separadas por vírgulas
tokenizar([H|T],Lista,L) :-
   letra(H,Letra), !,
   pega_resto_pal(T,[Letra],Pal,Res),
   append(Lista,[Pal],NovaLista),
   tokenizar(Res,NovaLista,L).

tokenizar([32|T],Lista,L) :-
   !, tokenizar(T,Lista,L).
tokenizar([],Lista,Lista).
% gramática GCD do F. Pereira e D. Warren
% (1980), com gênero e número
sen([NP,VP],S0,S) :-
   frsu(_,N,NP,S0,S1),frve(N,VP,S1,S2).
frsu(G,N,[SUBST,SUBST2],S0,S) :-
   det1(G,N,S0,S1),
   determ(G,N,S1,S2),
   expsubst(G,N,SUBST,S2,S3),
   expcom(G1,N1,SUBST2,S3,S),!.
frsu(G,N,[SUBST],S0,S) :-
   expsubst(G,N,SUBST,S0,S),!.
frsu(_,sing,[SUBST],S0,S) :-
   nome(SUBST,S0,S).
expsubst(G,N,[SUBST],S0,S) :-
   subst(G,N,SUBST,S0,S1),
   expadj(G,N,S1,S).
expsubst(G,N,[SUBST],S0,S) :-
   expadj(G,N,S0,S1),
   expsubst(G,N,SUBST,S1,S).
expsubst(G,N,[SUBST],S0,S) :-
   subst(G,N,SUBST,S0,S).

expadj(G,N,S0,S) :-
   adj(G,N,S0,S).
expadj(G,N,S0,S) :-
   adj(G,N,S0,S1),
```

```
    expadj(G,N,S1,S).
expcom(G,N,[SUBST],S0,S) :-
    pcom(S0,S1),
    frsu(G,N,SUBST,S1,S).
expcom(G,N,x,S,S).
frve(N,[VTD,NP],S0,S) :-
    vtd(N,VTD,S0,S1), frsu(_,N1,NP,S1,S),!.
frve(N,[VTI,PP],S0,S) :-
    vti(N,VTI,S0,S1), fprep(G,N1,PP,S1,S),!.
frve(N,[VR],S0,S) :-
    part(S0,S1),vr(N,VR,S1,S).
fprep(G,N,[NP],S0,S) :-
    prep(G,N,S0,S1),frsu(G,N,NP,S1,S).
det1(masc,pl,S0,S) :- conecta(S0,todos,S).
det1(fem,pl,S0,S) :- conecta(S0,todas,S).
det1(_,_,S,S).
determ(masc,sing,S0,S) :- conecta(S0,o,S).
determ(fem,sing,S0,S) :- conecta(S0,a,S).
determ(masc,pl,S0,S) :- conecta(S0,os,S).
determ(fem,pl,S0,S) :- conecta(S0,as,S).
determ(_,_,S,S).
subst(masc,sing,alimento,S0,S) :- conecta(S0,alimento,S).
subst(masc,pl,alimento,S0,S) :- conecta(S0,alimentos,S).
adj(_,sing,S0,S) :- conecta(S0,elegante,S).
adj(_,pl,S0,S) :- conecta(S0,elegantes,S).
adj(masc,sing,S0,S) :- conecta(S0,belo,S).
adj(fem,sing,S0,S) :- conecta(S0,bela,S).
nome(homem,S0,S) :- conecta(S0,joao,S).
nome(mulher,S0,S) :- conecta(S0,maria,S).

vtd(sing,comer,S0,S) :- conecta(S0,comeu,S).
vtd(sing,quebrar,S0,S) :- conecta(S0,quebrou,S).
vtd(sing,mover,S0,S) :- conecta(S0,moveu,S).
vr(sing,mover,S0,S) :- conecta(S0,moveu,S).
vti(sing,bater,S0,S) :- conecta(S0,bateu,S).
part(S0,S) :- conecta(S0,se,S).
prep(_,_,S0,S) :- conecta(S0,em,S).
prep(masc,sing,S0,S) :- conecta(S0,no,S).
prep(fem,sing,S0,S) :- conecta(S0,na,S).
pcom(S0,S) :- conecta(S0,com,S).
conecta([W|S],W,S).
frase(X,A) :-
   tokenizar(X,Y,Z),!,sen(A,Z,[]),write(A),
   create(H1,frase),write(H1,A),close(H1).
```

EXERCÍCIOS RESOLVIDOS

6.1 Suponha a seguinte base de conhecimento:

a. Joaquim possui um cão.
b. Todos que possuem cães gostam de animais. ("se x possui y e y é cão e z é animal, então x gosta de z").
c. Ninguém que goste de animais mata um animal.
d. Sardinha é gato.
e. Ou Joaquim ou a Curiosidade matou Sardinha.
f. Os gatos são animais.

Represente o conhecimento anterior através dos fatos e regras do Prolog. Faça o *trace* para provar que a Curiosidade matou o gato (Sardinha)?

Resolução

Lógica:

a. $\exists X$ (possui(joaquim, X) $\wedge$ cão(X))
b. $\forall X \forall Y \forall Z$ ((possui(X,Y) $\wedge$ cão (Y) $\wedge$ animal (Z)) $\rightarrow$ gosta(X,Z))
c. $\forall X$ ((gosta(X,Y) $\wedge$ animal(Y)) $\rightarrow$ $\neg$mata(X,Y))
d. gato(sardinha)
e. mata(joaquim, sardinha) $\vee$ mata(curiosidade, sardinha)
f. $\forall X$ (gato(X) $\rightarrow$ animal(X))

Prolog:

a.
```
possui(joaquim, a).
cão(a).
```
b.
```
gosta(X,Z) :- possui(X,Y), cão(Y), animal(Z).
```
c.
```
n_mata(X,Y) :- gosta(X,Y), animal(Y).
```
d.
```
gato(sardinha).
```
e.
```
mata(curiosidade, sardinha) :- n_mata(joaquim, sardinha).
```
f.
```
animal(X) :- gato(X).
```

Questão: ?- mata(curiosidade, sardinha).

Trace:

a. Procura fato que combine com o objetivo. Não há.
b. Procura regra que combine com o objetivo 1. Acha a regra (e). Troca a cabeça pelo corpo. Novo objetivo (2): `n_mata(joaquim, sardinha)`.
c. Procura fato que combine com o objetivo 2. Não há.
d. Procura regra que combine com o objetivo 2. Acha a regra (c). Faz a instanciação `X = joaquim e Y = sardinha`. Troca a cabeça pelo corpo. Novo objetivo (3): `gosta(joaquim, sardinha), animal (sardinha)`.
e. Procura fato que combine com o primeiro predicado do objetivo (3). Não há.
f. Procura regra que combine com o primeiro predicado do objetivo (3). Acha a regra (b). Faz a instanciação `X = joaquim` e `Z = sardinha`. Troca a cabeça pelo corpo.

Novo objetivo (4): `possui(joaquim, Y), cão(Y), animal(sardinha), animal(sardinha).`

g. Procura fato que combine com o primeiro predicado do objetivo (4). Acha o fato (a1). Instancia `Y = a`.

h. O objetivo (4) se reduz a `cão(a), animal(sardinha), animal(sardinha).`

i. Procura fato para o segundo predicado do objetivo (4): acha o fato (a2). O objetivo (4) se reduz a `animal(sardinha), animal(sardinha).`

j. Procura fato para o terceiro predicado do objetivo (4). Não há fato.

k. Procura regra para o terceiro predicado do objetivo (4). Acha a regra (f). Instancia `X = sardinha`. Troca a cabeça pelo corpo. Novo objetivo (5): `gato(sardinha), animal(sardinha).`

l. Procura fato para o objetivo (5). Acha o fato (d). Portanto o objetivo (5) se reduz a `animal(sardinha).`

m. Repete-se os passos k) e l). Logo, Prolog responde *yes*.

6.2 Admita que um retângulo é representado pelo functor `retangulo(P1,P2,P3,P4)` em que os `Pi` são dados pelo functor `ponto(x,y)`. Defina a relação `regular(R)` em Prolog, que é verdadeiro se `R` é um retângulo no plano cartesiano, cujos lados são verticais (paralelos ao eixo *y*) e horizontais (paralelos ao eixo *x*).

Resposta

```
regular(retangulo(ponto(X1,Y1), ponto(X2,Y2), ponto(X3,Y3),
ponto(X4,Y4))) :- Y1 = Y2, X2 = X3, Y3 = Y4, X4 = X1.
```

6.3 Considere o seguinte problema:

Fato: `s`

Regras:
```
1. r :- p, q.
2. r :- k.
3. t :- s.
4. p :- m.
5. p :- t.
6. m :- v.
7. m :- n.
8. n :- l.
9. v :- s.
10. q :- v.
11. q :- h.
12. h :- s.
13. h :- g.
14. j :- h.
15. k :- j.
```

Meta: `r`

Sabendo-se que o Prolog trabalha com encadeamento regressivo e usa técnica de busca *backtracking*, dê a sequência de passos do Prolog para resolver esse problema. Utilize os números das regras.

1. Procura fato **r**. Não encontra.
2. Procura regra com cabeça **r**. Encontra a regra **1**. Dispara a regra 1. Novo objetivo: **p, q**.
3. Procura fato **p**. Não encontra.
4. Procura regra com cabeça **p**. Encontra a regra **4**. Dispara a regra. Novo objetivo: **m, q**.
5. Procura fato **m**. Não encontra.
6. Procura regra com cabeça **m**. Encontra a regra **6**. Dispara a regra. Novo objetivo: **v, q**.
7. Procura fato **v**. Não encontra.
8. Procura regra com cabeça **v**. Encontra a regra **9**. Dispara a regra. Novo objetivo: **s, q**.
9. Procura fato **s**. Encontra. Novo objetivo: **q**.
10. Procura fato **q**. Não encontra.
11. Procura regra com cabeça **q**. Encontra a regra **10**. Dispara a regra. Novo objetivo: **v**.
12. Procura fato **v**. Não encontra.
13. Procura regra com cabeça **v**. Encontra a regra **9**. Dispara a regra. Novo objetivo: **s**.
14. Procura fato **s**. Encontra. *yes*.

6.4 Considere a seguinte base de conhecimento:

1. Os gatos gostam de comer peixes.
2. Siameses são gatos.
3. Sardinhas são peixes.
4. Carlos é uma sardinha.
5. Hércules é uma sardinha.
6. Frederico é um siamês.
   a. Converta essa base de conhecimento para fórmulas da lógica de primeira ordem.
   b. Usando refutação, responda à pergunta: "O que Frederico gostaria de comer?".
   c. Converta essa base de conhecimento para um programa Prolog. Como esse programa daria uma resposta diferente da encontrada no item (b)?

(a)

a. $\forall X \forall Y$ ((gato(X) $\wedge$ peixe(Y)) $\rightarrow$ gosta_comer(X,Y))
b. $\forall X$ (siamês(X) $\rightarrow$ gato(X))
c. $\forall X$ (sardinha(X) $\rightarrow$ peixe(X))
d. sardinha(carlos)
e. sardinha(hércules)
f. siamês(frederico)

(b)

1. ¬gato($X1$) ¬peixe(Y) gosta_comer($X1$,Y)
2. ¬siamês($X2$) gato($X2$)
3. ¬sardinha($X3$) peixe($X3$)
4. sardinha(carlos)
5. sardinha(hércules)
6. siamês(frederico)

7. ¬gosta_comer(frederico,*X*) // negação da meta
8. ¬gato(frederico) ¬peixe(*X*) RE: 1,7 β = {*X1*/frederico, *Y*/*X*}
9. ¬siamês(frederico) ¬peixe(*X*) RE: 2,8 β = {*X2*/frederico}
10. ¬peixe(*X*) RE: 6,9 ε
11. ¬sardinha(*X*) RE: 3,10 β = {*X3*/*X*}
12. RE: 4,11 β = {*X*/**carlos**}

(c)

```
1. gosta_comer(X,Y) :- gato(X), peixe(Y).
2. gato(X) :- siamês(X).
3. peixe(X) :- sardinha(X).
4. sardinha(carlos).
5. sardinha(hércules).
6. siamês(frederico).
```

Através do *backtracking*. Resposta: **hércules**.

6.5 Escreva um programa Prolog para reconhecimento de animais, com a seguinte base de conhecimento:

a. animal que tem pena não é mamífero.
b. animal que tem pelo é mamífero e não é ave.
c. animal que não é mamífero é ave.
d. animal que não tem pena tem pelo.
e. animal que não é mamífero e voa é sabiá.
f. animal que é ave e não voa é pinguim.
g. animal que é mamífero e não voa é vaca.
h. animal que voa e não é ave é morcego.

Consulte o seu programa para identificar qual é o animal que voa e não tem pena. Qual é a sequência de prova usada?

Resolução

Uma solução possível para este exercício é a seguinte:

```
/* ANIMAIS.ARI */
n_pena(X) :- mam(X).          % animal que tem pena não é
                              % mamífero
mam(X) :- pelo(X).            % animal que tem pelo é
                              % mamífero e não é ave
n_pelo(X) :- ave(X).
mam(X) :- n_ave(X).           % animal que não é mamífero é
                              % ave
pena(X) :- n_pelo(X).         % animal que não tem pena tem
                              % pelo
```

```
n_mam(sabia). voa(sabia). ave(pinguim). n_voa(pinguim).
mam(vaca). n_voa(vaca). voa(morcego). n_ave(morcego).
```

EXERCÍCIOS PROPOSTOS

6.1 Traduza as seguintes declarações em regras do Prolog:

a. "Todo mundo que tem criança é feliz."

b. "Para todo X, se X tem uma criança que tem uma irmã então X tem duas crianças."

6.2 Defina a relação **netos** usando a relação **pais**.

6.3 Defina um predicado Prolog para concatenar duas listas.

6.4 Defina um predicado Prolog para achar o mínimo entre quatro números fornecidos.

6.5 Quais dos seguintes objetos é sintaticamente correto em Prolog? Que tipo de objetos são eles?

a. `Diana`
b. `diana`
c. `'Diana'`
d. `'diana'`
e. `_diana`
f. `ir(diana,sul)`
g. `45`
h. `'Diana ir sul'`
i. `5(X,Y)`
j. `+(north,west)`
k. `tres(Preto(Gatos))`

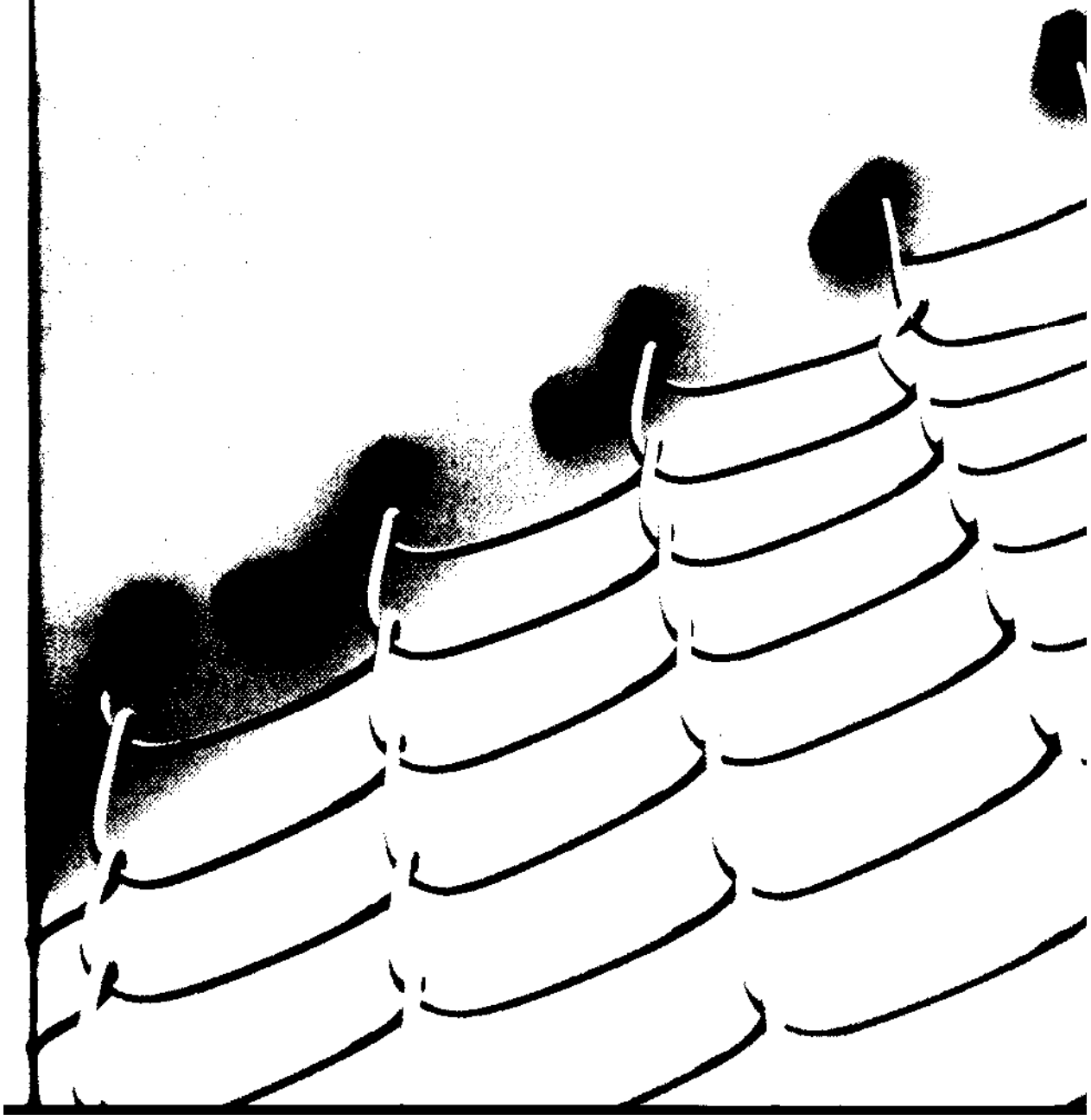
6.6 As seguintes operações de *matching* terão sucesso ou falharão? Se tiverem sucesso, quais são as instanciações das variáveis?

a. `ponto(A,B) = ponto(1,2)`
b. `ponto(A,B) = ponto(X,Y,Z)`
c. `mais(2,2) = 4`
d. `+(2,D) = +(E,2)`
e. `triangulo(ponto(-1,0),P2,P3)=triangulo(P1,ponto(1,0),ponto(0,Y))`

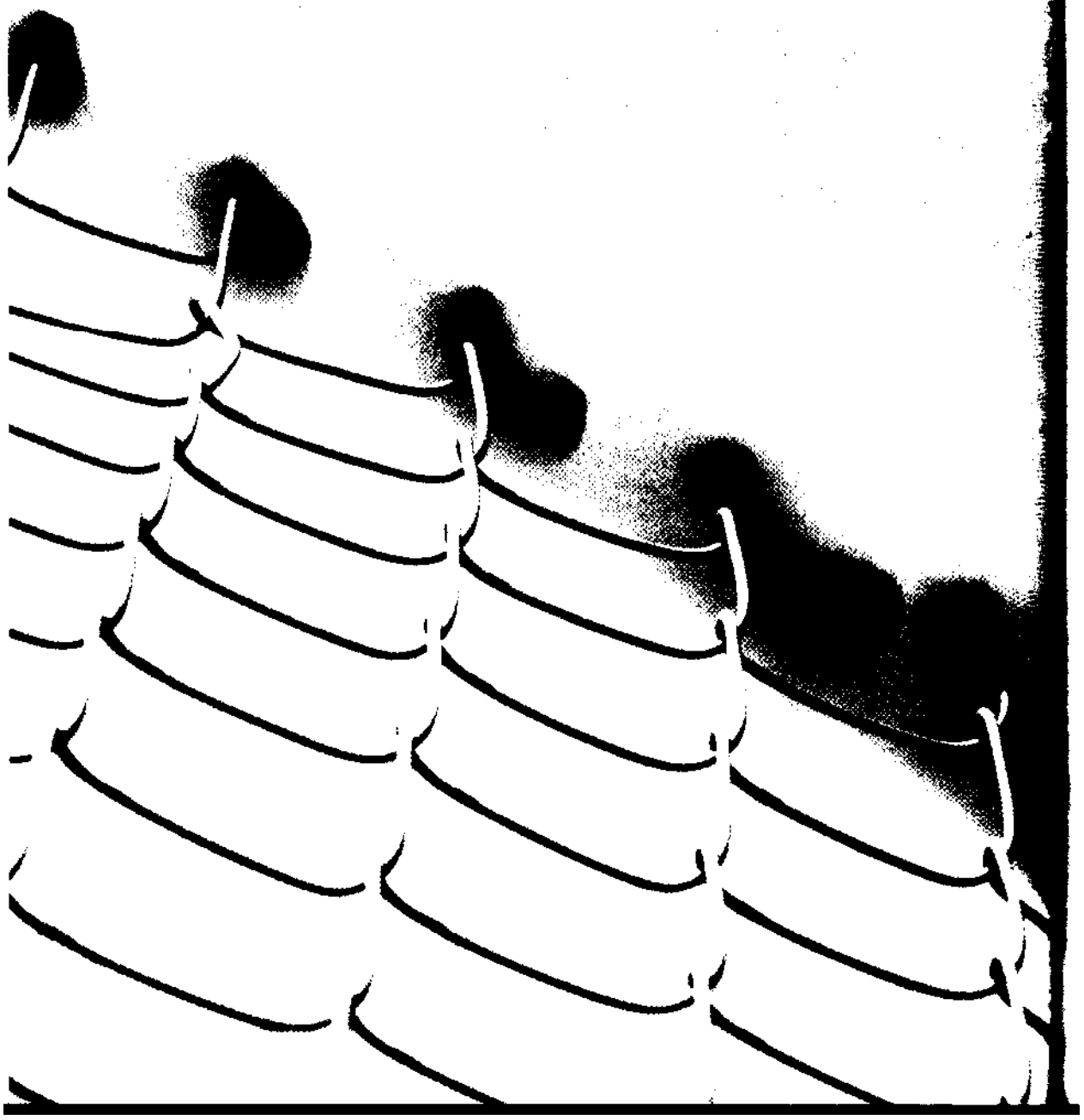
Obs.: Para o item (e), a instanciação resultante define uma família de triângulos. Como você descreveria essa família?

6.7 Defina a relação **gêmeos(Criança1,Criança2)**, para achar gêmeos na base de dados da família.

CAPÍTULO 7



Conjuntos Nebulosos



7.1 INTRODUÇÃO

Os conjuntos nebulosos (*fuzzy sets*), concebidos por L. Zadeh (1965), são a base para os sistemas nebulosos muito usados em controle inteligente. Neste capítulo, será estudado um pouco da base da lógica nebulosa, também conhecida como lógica multivalorada ou lógica da incerteza.

7.2 CONJUNTOS ORDINÁRIOS (*CRISP*) E NEBULOSOS (*FUZZY*)

7.2.1 Conjuntos ordinários

Para indicar que um elemento individual x é um membro ou elemento de um conjunto A, escreve-se

$$x \in \text{A}.$$

Quando x não é um elemento de um conjunto A, escreve-se

$$x \notin \text{A}.$$

Um conjunto pode ser descrito pelos nomes de seus elementos. Suponha que um conjunto A tenha os elementos $a_1, a_2, ..., a_n$. Esse conjunto pode ser descrito como:

$$\text{A} = \{ a_1, a_2, ..., \text{a}_n\},$$

Como para a teoria clássica de conjuntos um elemento pertence a A ou não pertence a A, pode-se definir uma *função de pertinência* $\mu_A(x)$, que será igual a 1 se $x \in$ A e igual a 0 se $x \notin$ A. Suponha agora que X é o conjunto universo. A função de pertinência pode ser descrita como:

$$\mu_A: X \Rightarrow \{0, 1\}.$$

Pode-se representar o conjunto A em termos da função de pertinência de seus elementos. Suponha que A = {1, 3, 4, 6, 8} em X = [1,10]. Logo A pode ser descrito como A = $\{\mu_A(x_i)/x_i\}$, ou seja,

$$\text{A} = \{1/1, 0/2, 1/3, 1/4, 0/5, 1/6, 0/7, 1/8, 0/9, 0/10\}.$$

Todas as definições, teoremas e propriedades da lógica de predicados clássica valem para os conjuntos ordinários. O complemento de um conjunto A pode ser definido como:

$$\neg\text{A} = \{x \mid x \notin \text{A}\}.$$

A partir dessa definição, conclui-se que

$$\neg\varnothing = X,$$

e

$$\neg X = \varnothing.$$

em que $\varnothing$ representa o conjunto vazio.

A união de dois conjuntos A e B é o conjunto contendo todos os elementos que pertencem só ao conjunto A, só ao B ou a ambos. Isso é denotado por

$$A \cup B = \{ x \mid x \in A \text{ ou } x \in B \}.$$

A partir disso pode-se concluir que

$$A \cup X = X$$

e

$$A \cup \varnothing = A.$$

A *lei do terceiro excluído* pode ser representada por:

$$A \cup \neg A = X.$$

A interseção dos conjuntos A e B é o conjunto contendo todos os elementos que pertencem a A e a B. É denotado por

$$A \cap B = \{ x \mid x \in A \text{ e } x \in B \}.$$

Pode-se então concluir que

$$A \cap X = A$$

e

$$A \cap \varnothing = \varnothing.$$

A *lei da contradição* estabelece que

$$A \cap \neg A = \varnothing.$$

7.2.2 Conjuntos nebulosos

Para os conjuntos nebulosos (*fuzzy*), a função de pertinência dos elementos de um conjunto varia entre 0 e 1, incluindo os extremos, ou seja:

$$\mu_A : X \Rightarrow [0, 1].$$

Ou seja, nesse caso não tem mais sentido dizer que um elemento pertence ou não a um conjunto. Deve-se dizer que um elemento pertence com grau de pertinência de 0.9, por exemplo, se

ele *quase* pertencer, ou com grau de pertinência de 0.1, se ele *quase* não pertencer. Os extremos do intervalo correspondem ao *pertence* e ao *não pertence* da lógica clássica. Para um conjunto nebuloso A pode-se ter:

$$A = \{0.1/1, 0.8/2, 1/3, 1/4, 0.3/5, 0/6, 0/7, 0.2/8, 0.1/9, 1/10\}.$$

Para esse conjunto pode-se dizer que os elementos 3, 4 e 10 pertencem a A (grau de pertinência igual a 1), 6 e 7 não pertencem a A (grau de pertinência 0), 1 e 9 pertencem com grau de pertinência 0.1, 8 pertence com grau 0.2, 5 pertence com grau 0.3 e 2 pertence com grau 0.8.

As operações com os conjuntos nebulosos são diferentes das operações com conjuntos ordinários. As leis do terceiro excluído e da contradição não são satisfeitas para conjuntos nebulosos.

7.2.2.1 Operações com conjuntos nebulosos

Para os conjuntos nebulosos existem diversas definições para o complemento, a união e a interseção de conjuntos. As mais aceitas são as seguintes (em termos da função de pertinência de seus elementos):

$$\mu_{\neg A}(x) = 1 - \mu_A(x),$$
$$\mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)],$$
$$\mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)].$$

Exemplos: Para

$$A = \{0.1/1, 0.7/2, 1/3, 0/4, 0.5/5, 0.2/6, 0.1/7, 0.9/8, 0/9, 1/10\} \text{ e}$$
$$B = \{0.9/1, 1/2, 1/3, 0/4, 0/5, 0.1/6, 0.9/7, 1/8, 0.3/9, 0.1/10\},$$

tem-se

$$\neg A = \{0.9/1, 0.3/2, 0/3, 1/4, 0.5/5, 0.8/6, 0.9/7, 0.1/8, 1/9, 0/10\}$$
$$\neg B = \{0.1/1, 0/2, 0/3, 1/4, 1/5, 0.9/6, 0.1/7, 0/8, 0.7/9, 0.9/10\}$$
$$A \cup B = \{0.9/1, 1/2, 1/3, 0/4, 0.5/5, 0.2/6, 0.9/7, 1/8, 0.3/9, 1/10\}$$
$$A \cap B = \{0.1/1, 0.7/2, 1/3, 0/4, 0/5, 0.1/6, 0.1/7, 0.9/8, 0/9, 0.1/10\}$$

7.3 A RESOLUÇÃO DOS "PARADOXOS" DA LÓGICA CLÁSSICA

Pelo exemplo apresentado anteriormente, dá para concluir que a lei do terceiro excluído não é satisfeita, ou seja, $A \cup \neg A \neq X$, pois

$$A \cup \neg A = \{0.9/1, 0.7/2, 1/3, 1/4, 0.5/5, 0.8/6, 0.9/7, 0.9/8, 1/9, 1/10\}$$

que é diferente do conjunto universo X, a menos dos valores de pertinência para 3, 4, 9 e 10. A lei da contradição também não é satisfeita, pois $A \cup \neg A \neq \varnothing$, como pode ser visto a seguir:

$$A \cap \neg A = \{ 0.1/1, 0.3/2, 0/3, 0/4, 0.5/5, 0.2/6, 0.1/7, 0.1/8, 0/9, 0/10\}$$

que é diferente do conjunto vazio, a menos dos valores de pertinência para 3, 4, 9 e 10.

Observe agora o valor de pertinência para 5: $\mu_A(5) = 0.5$ e $\mu_{\neg A}(5) = 0.5$. Ou seja, para o elemento 5, $\mu_{\neg A}(x) = \mu_A(x)$. Isso resolve os chamados "paradoxos" da lógica clássica como o do barbeiro: "Numa cidade existia uma barbearia onde se lia em uma tabuleta: 'Aqui se barbeiam apenas e tão somente aqueles que não se barbeiam.'" Pergunta: quem faz a barba do barbeiro? Se ele se barbeia, então pela definição ele não se barbeia. Se ele não se barbeia, então pela definição ele se barbeia. Na verdade, o que se tem aqui são duas proposições que são, ao mesmo tempo, contraditórias e equivalentes, ou seja, se P é a proposição que significa que o barbeiro faz sua própria barba e ¬P que ele não faz, tem-se que:

$$P \rightarrow \neg P \text{ e}$$
$$\neg P \rightarrow P.$$

Logo,

$$\mu p(x) = \mu \neg p(x).$$

Para a lógica nebulosa, pela definição do complemento, vem

$$\mu \neg p(x) = 1 - \mu p(x).$$

conclui-se que

$$\mu p(x) = \mu \neg p(x) = 0.5.$$

EXERCÍCIOS PROPOSTOS

7.1 As lógicas de três valores foram concebidas como um passo intermediário entre a lógica clássica (booleana) e a lógica nebulosa. Nessa lógica, três valores são possíveis: 0, 1/2 e 1. Para essa lógica, proposta por Lukasiewicz, definem-se as primitivas pelas seguintes equações:

$\neg a = 1 - a$
$a \wedge b = \min(a,b)$
$a \vee b = \max(a,b)$
$a \rightarrow b = \min(1, 1 + b - a)$
$a \leftrightarrow b = 1 - |a - b|$

Para essa lógica, determine os valores-verdade para cada uma das seguintes expressões lógicas para todas as combinações de valores-verdade das variáveis lógicas *a*, *b* e *c*:

a. $(\neg a \wedge b) \rightarrow c$;
b. $(\neg a \vee \neg b) \leftrightarrow \neg (a \wedge b)$;
c. $(a \rightarrow b) \rightarrow (\neg c \rightarrow a)$.

7.2 Seja a tabela a seguir:

Propriedades das operações de conjuntos ordinários	
Involução	$\neg\neg A = A$
Comutatividade	$A \cup B = B \cup A$ $A \cap B = B \cap A$
Associatividade	$(A \cup B) \cup C = A \cup (B \cup C)$ $(A \cap B) \cap C = A \cap (B \cap C)$
Distributividade	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
Idempotência	$A \cup A = A$ $A \cap A = A$
Absorção	$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$
Absorção do complemento	$A \cup (\neg A \cap B) = A \cup B$ $A \cap (\neg A \cup B) = A \cap B$
Absorção por X e $\varnothing$	$A \cup X = X$ $A \cap \varnothing = \varnothing$
Identidade	$A \cup \varnothing = A$ $A \cap X = A$
Lei da contradição	$A \cap \neg A = \varnothing$
Lei do terceiro excluído	$A \cup \neg A = X$
Leis de De Morgan	$\neg(A \cap B) = \neg A \cup \neg B$ $\neg(A \cup B) = \neg A \cap \neg B$

Para cada uma das propriedades das operações de conjuntos ordinários listadas nessa tabela, determine se as propriedades valem para as operações de complemento, união e interseção originalmente propostas para conjuntos nebulosos.

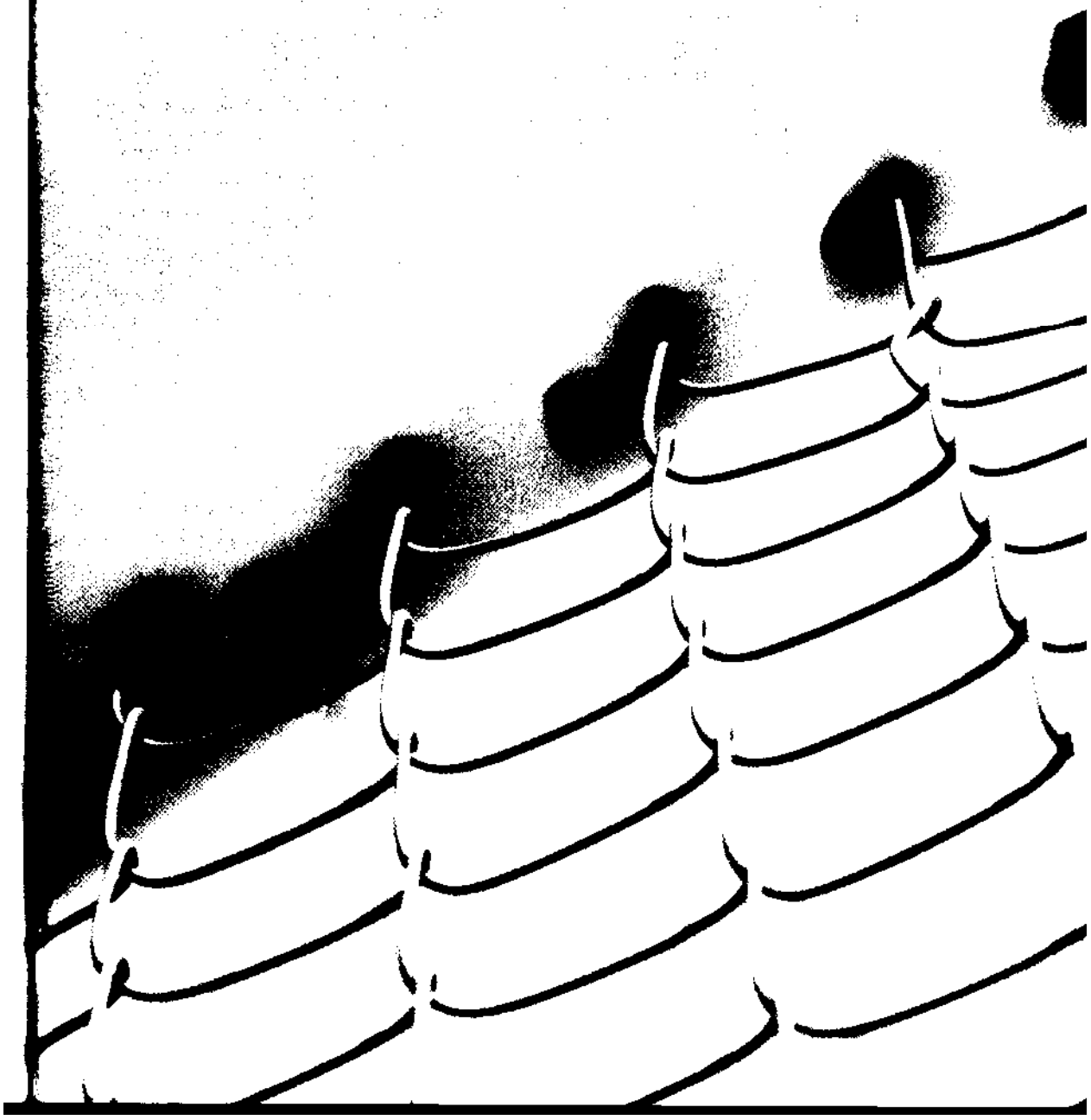
7.3 Considerando os conjuntos nebulosos A, B e C definidos no intervalo $X = [0,10]$ de números reais pelas funções de pertinência

$$\mu_A(x) = \frac{x}{x+2} \qquad \mu_B(x) = 2^{-x} \qquad \mu_C(x) = \frac{x}{1 + 10(x-2)^2}$$

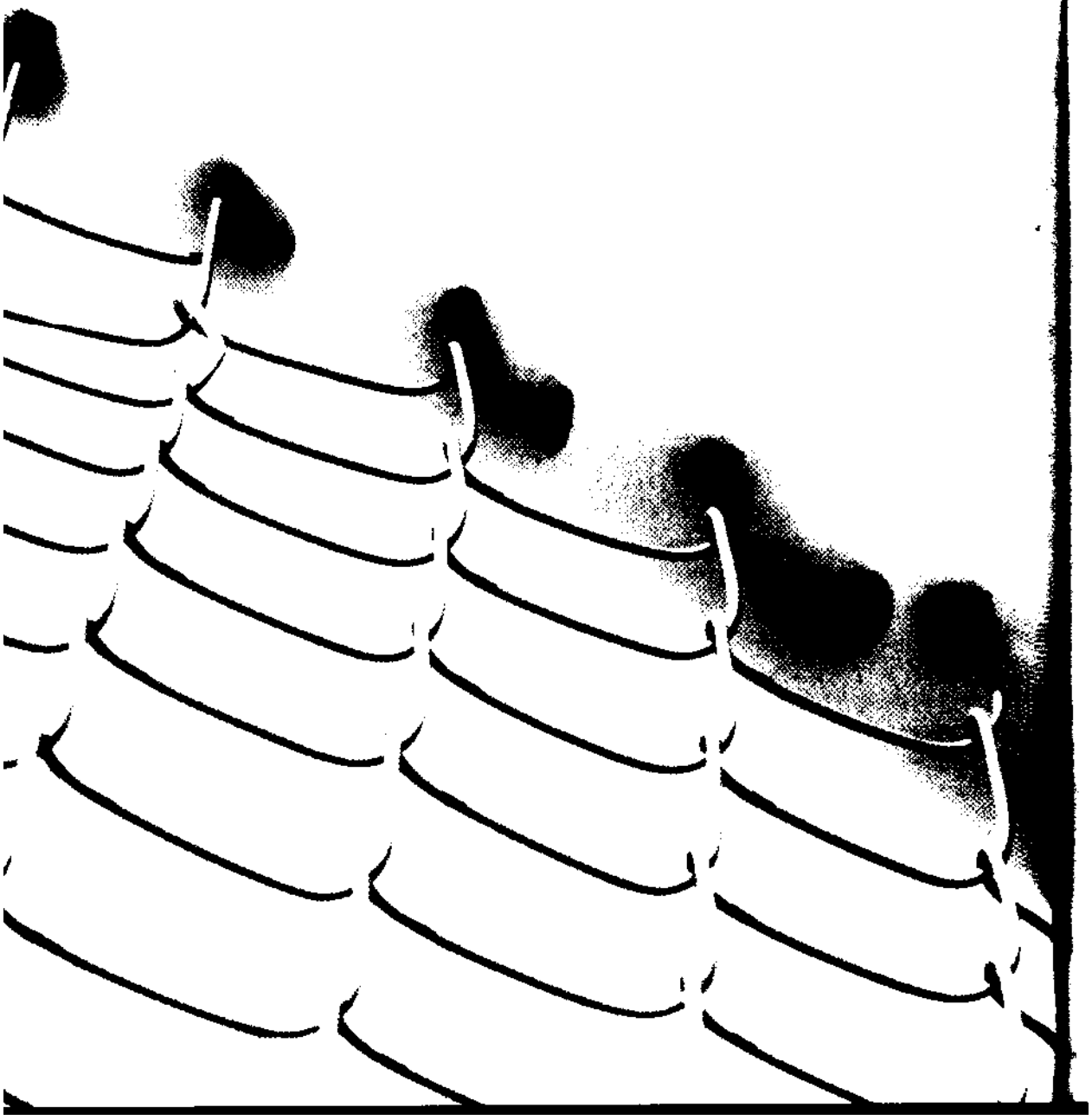
Determine fórmulas matemáticas e gráficos das funções de pertinência de cada uma das seguintes expressões:

a. $\neg A, \neg B, \neg C$;
b. $A \cup B, A \cup C, B \cup C$
c. $A \cap B, A \cap C, B \cap C$
d. $A \cup B \cup C, A \cap B \cap C$
e. $A \cap \neg C, \neg(\neg B \cap C), \neg(A \cup C)$

CAPÍTULO 8



Processamento de Línguas Naturais



8.1 INTRODUÇÃO

O Processamento de Línguas Naturais (PLN) pode ser definido de formas diferentes. Todas as definições incorporam a noção de armazenamento em computador e manipulação de dados linguísticos. Em outras palavras, o Processamento de Línguas Naturais pode ser definido como a habilidade de um computador em processar a mesma linguagem que os humanos usam no dia a dia.

O Processamento de Línguas Naturais é geralmente dividido em seis grandes áreas (Obermeier, 1987): (1) interfaces em língua natural para base de dados, (2) tradução de máquina — isto é, de uma língua natural para outra, (3) programas de indexação inteligentes para sumarização de grandes quantidades de texto, (4) geração de texto para produção automática de documentos padrões, (5) sistemas de fala para permitir interação de voz com computadores, e (6) ferramentas para desenvolver sistemas de Processamento de Línguas Naturais para aplicações específicas.

8.2 UMA ODISSEIA PARA O PLN

– David Bowman: Abra as portas da nave, HAL.
– HAL: Sinto muito, Dave, mas temo não ser capaz de abri-las.

Esse diálogo ocorre entre o comandante de uma nave espacial e seu computador no filme *2001 – Uma Odisseia no Espaço*. O fictício computador HAL 9000 era um agente, com as seguintes capacidades:

- reconhecimento de fala,
- entendimento de língua natural (leitura de lábios),
- geração de língua natural,
- síntese de fala,
- recuperação de informação,
- extração de informação,
- inferência.

A arte ficcional científica (cinema, literatura), que normalmente tenta prever o futuro, mostra, nesse filme de Stanley Kubrick, que o computador do futuro deve ser capaz de processar a língua natural. Ironicamente, o ano de 2001, futuro longínquo para um filme de 1968, que hoje é passado, apresenta uma máquina inteligente que ainda está nos sonhos de qualquer pesquisador da IA.

8.3 O TESTE DE TURING

O Teste de Turing, proposto por Alan Turing (1950), foi projetado para fornecer uma definição operacional satisfatória de *inteligência* (Russell e Norvig, 1995). Turing definiu comportamento inteligente como a habilidade de alcançar *performance* "humana" em todas as tarefas cogniti-

vas. O computador deveria ser interrogado por um ser humano através de um terminal, e esse programa passaria no Teste de Turing, isto é, seria considerado inteligente se o interrogador não pudesse dizer se havia um homem ou um outro computador do outro lado da linha (veja a Figura 8.1). O computador precisaria possuir as seguintes capacidades (Russell e Norvig, 1995):

- *Processamento de línguas naturais* para habilitá-lo a se comunicar com sucesso em inglês (ou qualquer outra língua humana);
- *Representação de conhecimento* para armazenar informação necessária antes ou durante o interrogatório;
- *Raciocínio automático* para usar a informação armazenada para responder a questões e para formular novas conclusões;
- *Aprendizado de máquina* para se adaptar a novas circunstâncias e para detectar e extrapolar padrões.

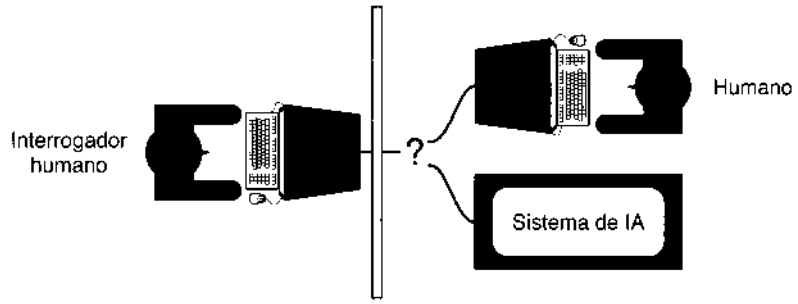


Figura 8.1 O Teste de Turing.

8.4 ALGUNS CONCEITOS DE LINGUÍSTICA

As *palavras* são as expressões básicas da língua. Elas são símbolos que servem para denotar seres, propriedades de seres, relações entre seres, propriedades de relações etc. As palavras são escritas como cadeias de letras do alfabeto e têm uma estrutura interna de subcadeias concatenadas que são chamadas de *morfemas*.

O morfema básico de uma palavra é chamado de *raiz*. A uma raiz podem-se afixar outros morfemas, por exemplo, *prefixos* e *sufixos*.

Cada um desses morfemas tem uma função definida na composição da palavra. O processo de composição é chamado de *análise morfológica*.

8.4.1 Classes de palavras

Substantivos, adjetivos, numerais, pronomes, verbos, advérbios e conectivos são as classes de palavras da língua portuguesa. Os determinantes, normalmente, são os artigos, definidos e indefinidos, mas podem ser numerais, pronomes etc. Os *sintagmas* são subdivisões das sentenças de uma língua natural em que se percebe um significado claro. Os tipos de sintagmas mais importantes são (Savadovsky, 1988):

- *Sintagma nominal* (SN): é um trecho da oração que define completamente uma entidade ou conjunto de entidades do mundo do falante.

- *Sintagma verbal* (SV): essa parte da oração define uma atividade ou estado no tempo, como os verbos. Aliás, o verbo é o centro do sintagma verbal. O sintagma verbal afirma algo sobre algum sintagma nominal externo, na maioria das vezes.
- *Sintagma preposicional* (SP): é composto de uma preposição seguida de um sintagma nominal.
- *Sintagma adjetival* (SAdj): ocorre com um verbo de ligação atribuindo qualidades a um sintagma nominal, como em "Paulo é inteligente", ou qualificando um sintagma nominal.
- *Sintagma adverbial* (SAdv): similar ao sintagma adjetival, é composto de um ou mais advérbios seguidos, opcionalmente, por um sintagma preposicional.
- *Oração subordinada adjetiva restritiva* (OSAR): é uma oração em que ocorre um verbo principal cujo objeto — ou sujeito — está faltando e é referenciado externamente por meio de um SN, SP ou SAdv. Exemplos:
  "O gato *que comeu o rato* dormiu."
  "O rato *que o gato comeu* morreu."
- *Locução verbal* (LV): é definida simplesmente como um verbo, chamado principal, precedido opcionalmente por verbos auxiliares.

8.4.2 As camadas da língua

A *língua* é um meio de comunicação. Ela é organizada em um sistema com níveis de regras complexos, dos mais baixos níveis dos sons e ritmos, aos mais altos, do significado e do relacionamento entre linguagem e o mundo. Cada nível trata de um aspecto do processo de comunicação e forma um subsistema completo com seus próprios elementos e regras de combinação. São (Sowa, 1984):

- *Prosódia* – os padrões de ritmo e entonação da língua.
- *Fonologia* – os sons, ou fonemas, da língua.
- *Morfologia* – os elementos significativos, ou morfemas, que constroem as palavras.
- *Sintaxe* – as regras para combinar palavras em frases ou sentenças. A sintaxe estuda as regras da gramática para expressar o significado em uma cadeia de palavras.
- *Semântica* – o significado e sua expressão.
- *Pragmática* – o uso da língua e seus efeitos no ouvinte. A pragmática estuda como o significado básico está relacionado com o contexto corrente e com as expectativas do ouvinte.

8.4.2.1 A sintaxe

A sintaxe é a camada que combina palavras em sentenças. *Parsing* (análise) é o ato de aplicar regras gramaticais para determinar como as palavras são combinadas. A *ambiguidade* de palavras individuais não é o problema mais sério, desde que o contexto seja suficiente para resolver a parte do discurso.

Ambiguidades sintáticas podem ser resolvidas pela semântica. O componente sintático de um analisador percorre a cadeia de entrada e o componente semântico junta grafos canônicos para representar o significado. Quando o componente semântico falha ao achar uma junção satisfatória, executa-se o componente sintático de forma a rejeitar a regra gramatical corrente e tentar uma opção diferente.

As línguas usam meios sintáticos diferentes para expressar as mesmas relações. Uma língua pode expressar certas relações com grande precisão e tolerar ambiguidades em outras.

De todos os níveis da língua, a sintaxe é a mais bem entendida. A *gramática tradicional* consiste nas regras informais que são ensinadas em escolas. A *gramática transformacional* é uma teoria formal, mais detalhada, cujo proponente mais notável é Noam Chomsky (1972).

8.4.2.2 A semântica

A base semântica depende do que o falante sabe sobre o assunto. A forma como o falante apresenta o assunto depende da pragmática — contexto, circunstâncias externas e expectativas do ouvinte. Não há razão para acreditar que todos esses aspectos do significado se originam de uma estrutura de base simples. Ao invés disso, uma sentença é derivada de seis tipos diferentes de informação:

- *Grafos conceituais* são a forma lógica que estabelece os relacionamentos entre pessoas, coisas, atributos e eventos.
- *Tempo verbal e modalidade* descrevem como os grafos conceituais relacionam-se com o mundo real. Eles estabelecem se alguma coisa aconteceu, pode acontecer, acontecerá ou deveria acontecer.
- *Pressuposição* é a informação de fundo que o falante e o ouvinte assumem.
- *Foco* é o ponto novo que o falante está tentando mostrar.
- *Ligações de correferência* mostram quais conceitos se referem às mesmas entidades. Numa sentença essas ligações são expressas como pronomes ou outras referências anafóricas.
- *Conotações emocionais* são determinadas pelas associações nas mentes do falante e do ouvinte.

A *percepção* é o processo de construção de um *modelo de trabalho* que representa e interpreta a entrada sensorial. O modelo tem dois componentes: uma parte sensorial formada a partir de um mosaico de elementos chamados "perceptos", cada um dos quais unifica com algum aspecto da entrada; e uma parte mais abstrata chamada de *grafo conceitual*, que descreve como os perceptos juntam-se para formar o mosaico.

Um grafo conceitual é uma combinação de nós de conceito e nós de relação, em que todo arco de toda relação conceitual é ligado a um conceito. Mas nem todas as combinações fazem sentido. Para distinguir os grafos significativos que representam situações reais ou possíveis no mundo externo, certos grafos são declarados canônicos. Através da experiência, cada pessoa desenvolve uma visão do mundo representada em grafos canônicos.

Como exemplo, considere a sentença *O homem mordeu um cachorro*. O grafo conceitual da Figura 8.2 representa o significado básico dessa sentença. O tempo verbal passado e o modo indicativo mostram que o evento realmente ocorreu em algum tempo no passado. A pressuposição é que o sintagma *o homem* é uma referência anafórica a um indivíduo específico no contexto — mesmo que ele tenha acabado de ser mencionado, ou que ele esteja presente na cena. O foco é a nova informação sobre a mordida no cachorro. Desde que *o homem* se refira a alguma coisa previamente mencionada, uma ligação de correferência conecta o conceito [HOMEM] na Figura 8.2 a uma ocorrência prévia de [HOMEM]. Essas cinco primeiras estruturas constituem o significado literal ou conceitual de uma sentença. O sexto, conotação emocional, não é expresso diretamente por palavras numa sentença, nem por conceitos e nem por um grafo. Ele é determinado por associações a uma experiência prévia que depende das pessoas envolvidas.

Figura 8.2 Grafo conceitual para a frase "homem mordendo cachorro".

Os grafos conceituais formam uma linguagem de representação do conhecimento baseada na linguística, na psicologia e na filosofia. Nos grafos, os nós de conceitos representam entidades, atributos, estados e eventos, e os nós de relação mostram como os conceitos são interconectados.

As regras de formação canônica são *regras de especialização*. A *generalização*, que procede na ordem reversa da especialização, não é necessariamente canônica, mas é importante porque forma a base para a lógica e para a teoria do modelo.

8.4.3 Gramática gerativa e transformacional

Segundo Dubois-Charlier (1976), "uma teoria linguística moderna que se atribui explicitamente o objetivo de dar conta de todos os aspectos de uma língua e que, além disso, compreende hipóteses sobre a faculdade da linguagem, sobre a aquisição linguística e sobre a universalidade da essência dos mecanismos linguísticos em todas as línguas" chama-se *gramática gerativa e transformacional*.

Nessa gramática, é importante a função da frase, que é uma organização, combinação de elementos, agrupados segundo certos princípios que a caracterizam como estrutura. A gramática é um conjunto de regras que permite organizar as palavras de uma língua em frases. É um sistema finito de regras que o falante de uma língua interiorizou, na maioria das vezes inconscientemente, e que lhe permite entender e produzir frases dessa língua.

8.4.4 Integrando sintaxe e semântica

Os analisadores que fazem apenas a análise sintática se defrontam com um número enorme de ambiguidades. Sem a semântica, um analisador não teria como escolher a melhor das análises ambíguas. Sem sintaxe, um programa perderia distinções. Todos os analisadores, mesmo os semânticos, usam alguma sintaxe.

A sintaxe determina que posições as palavras ocupam na sentença e a semântica determina que posições elas ocupam conceitualmente. Os sistemas com semântica mais complexa frequentemente são tão especializados que não poderiam ser utilizados para uma outra aplicação sem terem que ser completamente reescritos.

Em vez de usar regras sintáticas gerais que relacionam categorias como SN e SV, os sistemas baseados em gramática semântica usam regras orientadas à aplicação que relacionam categorias específicas. A gramática semântica permite acomodar palavras que não estão no dicionário.

Muitos analisadores têm sido projetados para mapear o inglês em grafos de dependência conceitual. Esses analisadores usam sintaxe. A maioria deles trata a sintaxe como "conhecimento de como combinar significados de palavras baseado nas suas posições na expressão". Alguns usam as informações sintáticas e semânticas concorrentemente, enquanto constroem uma representação conceitual. Poucos usam ligar a sintaxe e a semântica em "pacotes de palavras" associados a palavras individuais. Ter uma regra ou procedimento separado para cada palavra consome memória e perde generalizações importantes.

Em vez de um valor de interesse na própria definição da palavra, um *fator de relevância* pode calcular o nível de interesse dinamicamente. O envolvimento emocional, a atenção e o contexto podem afetar o nível de interesse. Ter um nível de interesse fixo em cada definição de palavra não é uma solução geral ou flexível.

8.4.5 Interações entre os níveis da língua

Existem interações entre níveis linguísticos. Ainda que a fonologia seja um processo de baixo nível, inconsciente, um fonema que é suficientemente estranho pode exigir uma decisão consciente.

Para alcançar uma velocidade ótima, cada aspecto da língua deveria ser interpretado no nível apropriado mais baixo. Os fonemas são interpretados no nível fonológico, e as gramáticas, por regras sintáticas. O que for ambíguo em um nível, entretanto, deve ser resolvido por retroalimentação de um nível mais alto: a sintaxe ajuda a resolver ambiguidades fonológicas, e esquemas conceituais ajudam a resolver ambiguidades sintáticas. Num caso extremo, o conhecimento de fundo do nível mais alto pode ser necessário para resolver uma ambiguidade fonológica num nível mais baixo.

Os conceitos e as percepções são blocos para construção de modelos mentais. Mas regras ou padrões são necessários para organizar os blocos de construção em estruturas maiores. Um "esquema" é uma regra que organiza percepções em uma coisa só.

Segundo Sowa (1984), a língua é processada em estágios com interações complexas:

- Fonologia, sintaxe e semântica são independentes, mas interagem entre si.
- Retroalimentação de nível semântico pode ou encorajar ou vetar interpretações nos níveis fonológico ou sintático.
- Um analisador psicologicamente realista requer retroalimentação imediata entre os níveis.
- Com grandes dicionários, um analisador, que empacota toda a informação sintática e semântica com cada palavra, é impraticável. Psicologicamente, ele falha se se considerar a forma com que as pessoas aprendem novas palavras sem ter que modificar sua gramática.

8.4.6 Contexto e conhecimento de fundo

O significado de uma sentença é apenas parcialmente determinado pelo significado de suas palavras. Uma parte grande, se não a maior, do significado vem do contexto, das intenções do falante e das expectativas do ouvinte.

A semântica determina o significado literal. Os outros fatores, que relacionam a linguagem ao mundo, são chamados de *pragmática*. A pragmática é o estudo das relações do usuário da língua com a própria língua, tendo em vista que esse usuário é um interlocutor. Nessa abordagem, terá um papel importante o estudo da *intenção* do locutor e do reconhecimento dessa intenção pelo ouvinte.

Os *esquemas* contêm o conhecimento de fundo que determina o que é razoável. Como um analisador sintático traduz a sentença de entrada em um grafo conceitual, os mecanismos de inferência juntam esquemas ao grafo e geram preferências e expectativas para combinações prováveis.

Os linguistas têm especulado que para qualquer fato no universo é possível construir uma sentença gramatical do inglês que é ambígua para alguém que não conhece o fato. Quando encontrar uma ambiguidade sem solução, o computador pode fazer a mesma coisa que as pessoas fazem — uma pergunta.

O *princípio cooperativo* consiste em adaptar cada contribuição ao propósito ou direção da conversação. Deve-se dizer o essencial, nem de menos nem demais. Tentar fazer da sua contribuição uma verdade. Ser relevante e perspicaz. O princípio cooperativo é uma importante ajuda à comunicação. Ainda que as pessoas nem sempre sejam cooperativas. Algumas vezes elas podem estar brincando, ser evasivas, ou mesmo estar mentindo.

Para ser cooperativo, um sistema deve reconhecer e responder às *pressuposições* básicas. O cheque de pressuposições tem sido implementado em muitos sistemas de perguntas em língua natural. Quando os termos não estão no dicionário do sistema, ele usa padrões semânticos para determinar o tipo de conceito esperado.

As regras da implicação conversacional requerem que ambos os participantes formem modelos mentais, não apenas do mundo externo, mas também dos modelos que eles supõem que o outro falante está formando.

Um sistema baseado em regras conversacionais forma um modelo de cada falante e procura por um tópico de interesse comum. Ele combina o que foi dito ao conhecimento de fundo. Além de combinar a informação nova com o conhecimento de fundo, ele forma um modelo do tópico da conversação e o conhecimento do outro falante, desenha inferências plausíveis e introduz informação nova no tempo apropriado.

A *metáfora* é um meio normal de adaptar palavras existentes a novas situações. Usar metáfora é como dar a uma coisa um nome que pertence a alguma outra coisa. A metáfora não é um dispositivo poético obscuro, mas um aspecto fundamental da língua que reflete os mecanismos do pensamento.

Os passos para o reconhecimento da metáfora por um computador são (Sowa):

- Tentar analisar cada sentença literalmente. Se nenhum grafo canônico puder ser formado para uma sentença gramatical, considere-a uma possível metáfora.
- Checar o catálogo das analogias comuns para achar uma possível transferência de esquemas para um ou mais conceitos na sentença.
- Determinar se um grafo conceitual que representa a sentença pode ser canonicamente derivado do esquema transferido.
- Se uma interpretação satisfatória for achada, lembrar do esquema que foi transferido, desde que seja esperado ser usado novamente dentro da mesma história ou conversação.

Um catálogo padrão de metáforas poderia ser usado para interpretar o discurso do dia a dia. Quando aprendem a falar, as crianças cometem erros. Mas os seus erros tendem a aparecer de simples desentendimentos, não dos "erros calculados" das metáforas.

8.4.7 Problemas de ambiguidade

A ambiguidade pode ser *lexical* quando a mesma palavra, ou expressão básica, denota entidades diferentes no mundo. Normalmente, a ambiguidade lexical é resolvida pelo contexto da frase. Na ambiguidade *sintática*, pode-se ter mais de uma árvore sintática para derivar a mesma frase.

Pode acontecer em frases com mais de um verbo, em que não está claro qual é o verbo principal:

"Pedro viu Maria passeando."

Em alguns casos, a análise semântica pode resolver ambiguidades sintáticas. Pode-se ter também uma ambiguidade sintática, provocada pela forma das regras, em uma frase com um só significado semântico. Isto é, tem-se várias árvores sintáticas, mas um só significado.

A ambiguidade *semântica* é o caso em que se tem mais de um significado para uma frase. Ocorre acompanhada da ambiguidade sintática, quando as diversas árvores sintáticas produzem análises semânticas válidas, como na frase anterior.

8.4.8 Grafos de derivação

Ainda que algumas gramáticas possam ser entendidas declarativamente, algumas vezes é útil seguir uma derivação completa da sentença, construindo um grafo que descreva a história das aplicações de regra de cima para baixo (*top-down*) e da esquerda para a direita. Isso é ilustrado através da gramática:

1. Sentença(s) ⇒ Nome-próprio(k),Verbo(k,s).
2. Nome-próprio(tom) ⇒ tom
3. Nome-próprio(joão) ⇒ gustavo
4. Verbo(k,rir(k)) ⇒ ri

O grafo de derivação para "Gustavo ri" é mostrado na Figura 8.3. Os números identificam a regra aplicada. As substituições necessárias para aplicar as regras aparecem do lado direito dos rótulos. Através delas pode-se reconstruir a estrutura profunda "rir(gustavo)".

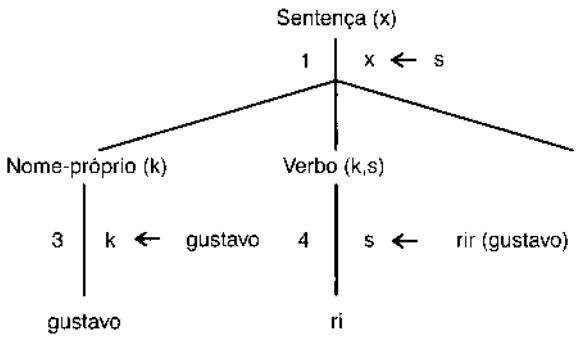


Figura 8.3 Grafo de derivação para "Gustavo ri" (Dahl, 1981).

Note que quando a variável recebe um valor esse valor é propagado para cada uma das suas ocorrências no grafo. Então, quando se aplica a regra 4, sabe-se que o valor de k=gustavo. Também, a renomeação de variáveis deve acontecer sempre que necessário a fim de assegurar que a regra aplicada e a cadeia que ela aplica não compartilhem nenhuma variável.

8.5 ABORDAGENS DO PROCESSAMENTO DE LÍNGUAS NATURAIS

O problema central dos sistemas de Processamento de Línguas Naturais é a transformação de uma frase de entrada potencialmente ambígua em uma forma não ambígua que possa ser usada internamente por um sistema de computador. Essas representações internas variam, é claro, de uma aplicação para outra.

A transposição de uma frase potencialmente ambígua para uma representação interna é conhecida como *parsing* (análise). A palavra *parse* é derivada do latim: *pars orationis* (parte do discurso). No Processamento de Línguas Naturais, *parsing* é usualmente um processo de combinar os símbolos de uma frase em um grupo que pode ser substituído por um outro símbolo mais geral. Esse novo símbolo pode, por sua vez, ser combinado em um outro grupo, e assim por diante, até que uma estrutura permitida apareça.

Existem cinco tipos diferentes de *parsers* (analisadores): casamento de padrões, baseado em gramática, semântico, baseado em conhecimento e analisadores por redes neurais. Cada um tem uma abordagem única, sem igual, no Processamento de Línguas Naturais.

8.5.1 Abordagens por casamento de padrões

Os primeiros programas de língua natural eram baseados na ideia de que os analisadores podiam procurar padrões linguísticos numa sentença sem usar nenhum formalismo gramatical explícito. Durante a análise da sentença, o sistema apenas procura por um possível casamento com um número fixo de padrões. Se um casamento é encontrado, o sistema executa determinada ação (por exemplo, rearranjar a entrada de acordo com um outro padrão). Ex.: ELIZA, o programa de casamento de padrões mais conhecido, foi projetado para simular um psicólogo rogeriano. O programa foi escrito por Joseph Weizenbaum em 1966 e consiste em um conjunto de padrões em que cada padrão tem um número de respostas associadas. Quando um determinado padrão é encontrado, o programa seleciona uma resposta do conjunto e faz todas as substituições necessárias nessa resposta. Um pouco do diálogo com ELIZA (Rich e Knight, 1994):

Pessoa: Homens são todos iguais.
ELIZA: De que forma?
Pessoa: Eles sempre estão nos aborrecendo com uma coisa ou outra.
ELIZA: Você pode dar um exemplo específico?
Pessoa: Bem, meu namorado me fez vir aqui.
ELIZA: Seu namorado a fez vir aqui.
Pessoa: Ele diz que eu ando deprimida a maior parte do tempo.
ELIZA: Sinto ouvi-la dizer que está deprimida.
Pessoa: É verdade. Estou infeliz.
ELIZA: Você acha que vir aqui a ajudará a se sentir menos infeliz?
Pessoa: Eu preciso de ajuda, isso parece certo.
ELIZA: O que significa para você conseguir ajuda?
Pessoa: Talvez eu pudesse aprender a conviver melhor com minha mãe.
ELIZA: Fale-me mais sobre sua família.

Os programas por casamento de padrões sem uma base gramatical provaram ser de uso limitado. Eles são úteis apenas se se necessita de análise parcial ou se outros componentes do sistema podem compensar a perda de informação sintática. O conceito básico de casamento de padrões, entretanto, foi usado nas gramáticas semânticas.

8.5.2 Abordagens baseadas em gramática

O termo gramática se refere a um conjunto de regras que descrevem quais sentenças são parte de uma determinada língua. Essas regras são frequentemente chamadas de regras de reescrita ou produções. Duas regras simples de reescrita são:

S → SN SV
S → SV

Essas regras estipulam que uma sentença (S) deve ter um sintagma nominal (SN) e um sintagma verbal (SV), ou apenas um sintagma verbal. Com essas regras pode-se construir uma estrutura em árvore mostrando sem ambiguidade como as palavras interagem numa sentença (veja Figura 8.4). As folhas da árvore são chamadas de palavras terminais. Os outros símbolos, em maiúscula, são não terminais. As regras de rescrita usualmente estabelecem como um não terminal pode ser reescrito como uma cadeia de terminais ou outros não terminais.

O linguista Noam Chomsky dividiu a gramática em quatro tipos, com base nos tipos de regras que elas usavam (Rosa, 2010). A gramática mais simples, a de tipo 3, é conhecida como *gramática de estado finito* ou *regular*. Ela pode produzir apenas sentenças simples.

A gramática do tipo 2 é conhecida como *gramática livre de contexto*. Nessa gramática, o lado esquerdo de cada regra de rescrita pode consistir em apenas um único símbolo não terminal. Note que esse símbolo sempre pode ser reescrito como o lado direito da regra, não importando o contexto no qual esse não terminal aparece.

A outra gramática complexa é a do tipo 1, ou gramática *sensível ao contexto*. Nesse tipo, mais de um símbolo pode aparecer no lado esquerdo da regra de reescrita. Existe apenas um requisito para essas regras: deve haver mais símbolos do lado direito que do esquerdo.

A gramática mais complexa, do tipo 0, tem regras que não seguem nenhum padrão. Essa gramática é muito difícil de analisar. Realmente, os pesquisadores têm provado que uma *Máquina de Turing* é necessária para processar linguagens com essa gramática.

Chomsky argumenta que uma língua natural como o inglês, e por consequência o português, não poderia ser completamente descrita por uma gramática livre de contexto, mas uma sensível ao contexto ou do tipo 0.

Gramática:

S → SN SV
SN → Det SN1
SN → SN1
SN1 → SN1 Adj
SN1 → Subst
SV → V Adv
Subst → menino

V → estudou
Adj → esperto
Det → o
Adv → muito

Sentença:
"O menino esperto estudou muito."

Árvore:

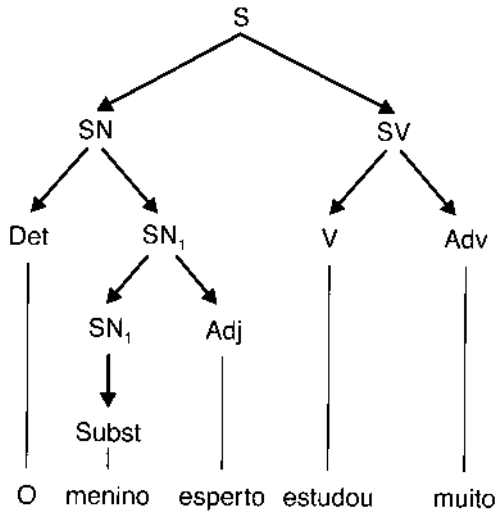


Figura 8.4 Abordagem baseada em gramática.

Um argumento favorável ao português ser uma linguagem sensível ao contexto envolve o fato de que substantivos singulares e plurais requerem verbos singulares e plurais, respectivamente. Por exemplo, a gramática da Figura 8.4 é uma gramática simples sensível ao contexto. Pode-se expandir essa gramática adicionando três regras de reescrita, todas permitidas:

SUBST → meninos
VERBO → estudaram
DET → os

Agora, portanto, a gramática permitirá uma sentença que não é permitida no português:
"Os meninos esperto estudaram muito".

8.5.3 Abordagens semânticas

O papel central do significado no entendimento da língua tem levado os pesquisadores a considerar mais a abordagem semântica do que a sintática. Esses pesquisadores não negam a necessidade de algum processamento estrutural; eles usam a análise sintática para complementar suas considerações semânticas.

Duas abordagens orientadas semanticamente em Processamento de Línguas Naturais são a gramática de caso e a gramática semântica.

A ideia da *gramática de caso* é que toda sentença tem uma representação "implícita" de seu significado. Essa representação inclui o verbo e os vários sintagmas nominais relacionados com o verbo. Por exemplo, na sentença "Joaquim abriu a porta com uma chave" designa-se *Joaquim* como o agente, *porta* como o objeto e *chave* como o instrumento para o verbo *abriu*. Note que os casos permanecem os mesmos para a sentença "A porta foi aberta por Joaquim com uma chave".

O descobridor da teoria de caso, C. Fillmore, referiu-se aos relacionamentos entre esses substantivos e os verbos como casos. Fillmore estabeleceu vários casos, nenhum deles ocorrendo mais de uma vez em uma sentença.

A *gramática semântica* consiste em um léxico e uma série de regras de reescrita. É similar a uma gramática sintática, exceto que classes de palavras (ex., *substantivo, verbo*) são substituídas por classes semânticas específicas (ex., *navios, propriedades_de_navio*). A vantagem dessa abordagem é que o tamanho dessas classes semânticas é muito menor do que o tamanho de uma classe de palavra equivalente. Isso resulta em uma estratégia de análise muito mais eficiente, desde que o programa tem que checar um número menor de possibilidades. A desvantagem da gramática semântica é a dificuldade de transferir regras de reescrita de um domínio de aplicação para outro.

8.5.4 Abordagens baseadas em conhecimento

Em vez de contar apenas com a informação semântica ou estrutural de uma sentença, alguns sistemas de Processamento de Línguas Naturais também têm acesso a uma base de conhecimento para um domínio específico do conhecimento. Isso contrasta com a maioria das teorias baseadas em gramática, que olham o Processamento de Línguas Naturais simplesmente em termos de um conjunto de regras de reescrita para o processamento em nível da sentença.

Uma abordagem baseada em conhecimento é chamada de *análise especialista em palavra*. Nessa abordagem, a palavra é considerada a unidade linguística básica. O conhecimento linguístico é distribuído entre um grupo de "especialistas" procedimentais que sabem como a interpretação de uma palavra muda em determinados contextos.

Um argumento em favor dessa abordagem é o fato de que as palavras têm uma estrutura linguística e conceitual rica. Também é pouco provável que a linguagem possa ser reduzida somente a várias regras de reescrita, como tratam muitas teorias baseadas em gramática.

Outra abordagem baseada em conhecimento é chamada de *teoria da dependência conceptual*. A ideia central por trás dessa teoria é criar uma representação canônica de uma sentença, baseada em certas primitivas semânticas. Uma representação canônica é simplesmente uma forma básica de representar o significado de uma sentença. Sentenças diferentes que signifiquem a mesma coisa terão a mesma representação canônica. Por exemplo, "Alfredo come doce" e "o doce foi comido por Alfredo" têm ambas a mesma representação canônica:

"Alfredo ↔ INGEST ↔ doce."

Nessa teoria, as primitivas semânticas são as entidades mais básicas usadas para descrever o mundo. Palavras individuais podem sempre ser analisadas de novo, mas primitivas semânticas não.

8.5.5 Abordagem por rede neural

Uma abordagem mais recente em Processamento de Línguas Naturais consiste em estabelecer uma rede de unidades de computação parecidas com o neurônio. Cada unidade tem várias entradas, um conjunto pequeno de estados possíveis, e uma saída que é uma função das entradas. Cada entrada para a unidade de computação tem um peso de conexão (peso sináptico), que normalmente pode variar de –1 a 1. Quando uma unidade é ativada, ela analisa todas as suas entradas e as pondera de acordo com seus respectivos pesos de conexão. Se certas condições são encontradas, a unidade gera um valor de saída que é usado como entrada por outras unidades. Note que apenas os pesos de conexão das entradas podem ser mudados durante o "aprendizado"; o padrão de conexão é estabelecido previamente.

Esse tipo de sistema é usualmente chamado de abordagem por rede neural ou *conexionista*. A premissa fundamental dessa abordagem é que as unidades individuais não transmitem grandes quantidades de dados, mas o processamento ocorre porque grande número de unidades similares são conectadas entre si.

Um modelo de análise por rede neural contém três níveis de "neurônios" (camadas). O primeiro nível é o nível lexical, que serve como nível de entrada da rede. Aqui, os neurônios são mapeados em determinadas palavras. No segundo nível, o nível do sentido da palavra, as entradas do nível lexical são combinadas para ativar neurônios que representam o significado das palavras. No terceiro nível, de lógica de caso, os significados são combinados para formar predicados e objetos.

A análise por rede neural se aproxima do modelo do processamento de informação linguística humano, baseado na evidência neurológica (veja o Capítulo 9).

8.6 NÍVEIS DE ANÁLISE DA LÍNGUA

Os dados estruturais a serem identificados e analisados correspondem aos níveis de análise da língua. Como a maioria dos programas trata apenas de material escrito, o nível fonológico (sistema sonoro) é deixado de lado. Cinco níveis podem ser descritos, exemplificados pela seguinte descrição de processamento da sentença simples "O sistema recuperou os documentos" (Warner, 1988).

1. *Morfológico* – palavras (conjunto de letras separadas por espaço) são decompostas em raízes e terminações. Por exemplo, o termo *documentos* seria quebrado na raiz *documento* e na terminação (plural) *s*.
2. *Lexical* – usando um dicionário, a cada raiz é atribuído um conjunto de categorias lexicais. Por exemplo, ao termo *documento* deve ser atribuída a categoria lexical *substantivo*.
3. *Sintático* – usando um módulo de programa chamado de analisador sintático, uma estrutura gramatical é atribuída à sentença. O programa pega o nível de entrada da palavra do componente lexical e decide como as palavras individuais se ligam para formar sintagmas, cláusulas e sentenças inteiras. A seguinte análise da sentença resultaria (Figura 8.5):

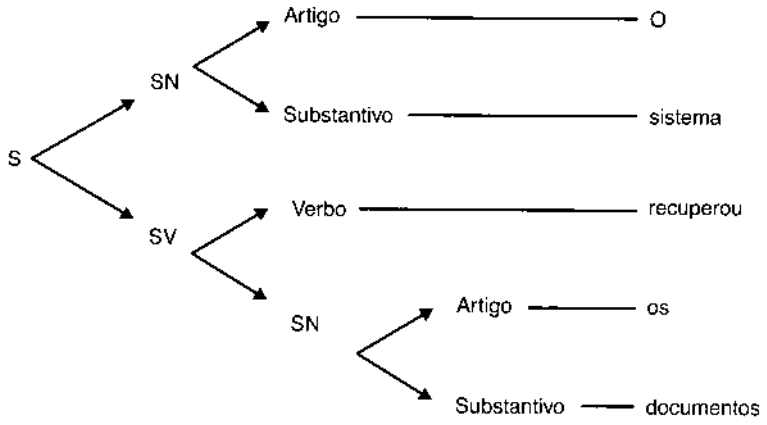


Figura 8.5 Análise sintática da sentença "O sistema recuperou os documentos".

4. *Semântico* – a estrutura sintática é traduzida para alguma forma lógica que representa o significado da sentença e permitirá se fazer certas inferências. Por exemplo, dada uma representação semântica apropriada, a questão "Os documentos estão no sistema?" poderia ser respondida — o sistema poderia inferir que a recuperação de documentos significa que havia documentos a serem recuperados.
5. *Pragmático* – este analisa a sentença no contexto, levando em consideração um certo corpo de conhecimento sobre o domínio e sobre os planos e metas do falador e do ouvidor na conversação. Por exemplo, a informação pragmática permitiria ao sistema inferir que um computador foi envolvido na operação de recuperação, ainda que isso não esteja explícito na sentença.

Os sistemas de Processamento de Línguas Naturais diferem nas quantidades dos cinco tipos de conhecimento que eles podem explorar e em como o conhecimento é configurado em uma arquitetura de sistemas global.

8.7 PROCESSAMENTO DE LÍNGUAS NATURAIS BASEADO EM LÓGICA

São descritos os aspectos teóricos e de aplicação da pesquisa em entendimento e Processamento de Línguas Naturais. No nível teórico, o sistema é apresentado em *lógica*, e no nível de implementação ele é programado em *Prolog*. Esse sistema é concebido como um módulo de acesso a uma base de conhecimento. Dessa forma, o Processamento de Línguas Naturais é visto como o entendimento de asserções e questões nos limites de um domínio específico.

O sistema inteiro está descrito em lógica. Entretanto, em momentos diferentes, serão introduzidos predicados não lógicos, os quais são necessários para a descrição das partes diferentes do sistema (predicados lexicais e sintáticos, predicados de *frame* e semânticos, metapredicados para diálogo e descrição do contexto). É limitado à lógica de predicados clássica para a representação da língua natural. Algumas vezes não parece, porque se dispõem predicados de segunda ordem para relações de *frames* e semânticas. Mas esses predicados de "segunda ordem"

junto com seus axiomas tornam-se representáveis na lógica de primeira ordem; isto é, pode-se defini-los em primeira ordem de tal forma que seus axiomas são dedutíveis dessa definição. Portanto, essa linguagem lógica é uma extensão conservadora da clássica lógica de predicados de primeira ordem.

O processo de entendimento centraliza sua atenção na adaptação de sentenças da língua natural para *frames* predefinidos. Estes são inspirados em estruturas de casos semânticos e são formulados em volta de verbos. Claramente, não é possível achar *frames* gerais. Formularam-se completamente *frames* de domínio específico em volta de verbos de domínio específico. Foi possível e necessário proceder dessa forma porque nosso entendimento de verbo é baseado no fato de que um verbo representa uma ação na base de conhecimento e observa-se que essas ações nunca são as mesmas para dois domínios diferentes. Os *frames* definidos são mínimos em dois aspectos: por um lado, sempre se tenta achar os atributos semânticos mais gerais para os complementos do sintagma nominal e para deduzir mais *frames* específicos por leis de dedução de frame. Por outro lado, tenta-se achar um conjunto mínimo de complementos de sintagma nominal e preposicional para todo verbo. O dispositivo de adaptação de sentenças para *frames* permite então mais complementos com sentenças em certas condições.

8.8 TÉCNICAS DE ANÁLISE

Existem dois grupos de técnicas de análise: não determinística e determinística (Obermeier, 1987). Os não determinísticos podem ser divididos em analisadores *top-down* e *bottom-up*.

Os analisadores *top-down* tentam casar as regras da gramática com a entrada, começando na regra de reescrita mais alta (que usualmente envolve o símbolo inicial ou símbolo da sentença S) e recursivamente se move em direção à mais baixa, a regra de reescrita mais específica. O analisador é bem-sucedido se uma sentença pode ser construída a partir da sentença de entrada (casando).

Os analisadores *top-down* são fáceis de escrever e de modificar. Regras que são mais usadas podem ser colocadas na frente de regras menos usadas, aumentando a *performance*. E o número das sentenças geradas pode ser limitado arbitrariamente.

Contudo, os analisadores *top-down* podem ser lentos. Se todas as regras de um nível falham, o analisador volta (*backtracks*) para o nível anterior para tentar uma outra regra lá. Durante o *backtracking*, os mesmos componentes podem ser analisados muitas vezes. Os analisadores *top-down* também têm problemas para manipulação de entrada disforme e requerem um módulo separado para decidir qual dos muitos analisadores bem-sucedidos é o melhor.

Os analisadores *bottom-up* começam combinando os elementos do nível mais baixo primeiro e então constroem para cima componentes maiores. Por exemplo, na Figura 8.4, os primeiros passos de um analisador *bottom-up* seriam: substituir o não terminal Det por "o", Subst por "menino", Adj por "esperto" e SN1 por Subst Adj.

Os analisadores *bottom-up* podem, ao menos parcialmente, analisar entrada disforme. Também, mecanismos podem ser aplicados para reduzir a explosão combinatorial de possíveis análises. Como os analisadores *bottom-up* não são direcionados à meta, eles geram numerosas análises espúrias. E a correção da análise só pode ser determinada depois que todas as análises forem realizadas.

A análise *determinística* não tem *backtracking*. É também chamada de análise "espere e veja". A técnica cria novos nós de um modo *bottom-up*, mas usa uma característica limitada de "olhar para a frente" (*look ahead*) para determinar qual nó usar. Uma vantagem do determinístico é o aumento de velocidade, porque ele evita a explosão combinatorial para possíveis análises. A desvantagem é que o algoritmo é baseado somente na informação sintática.

8.9 REDES DE TRANSIÇÃO

8.9.1 Introdução

Uma *rede de transição* (Figura 8.6) consiste em uma série de estados conectados por arcos. Cada arco é rotulado por uma categoria de palavra (ex., substantivo ou verbo) ou uma palavra específica. O programa começa num dado estado e então checa a próxima palavra na cadeia de entrada com o propósito de "casar" com um dos arcos. Se um casamento ocorre, o programa vai para o próximo arco e "atravessa" a rede.

A vantagem das redes de transição é que elas podem ser facilmente implementadas em um computador. Cada estado pode ser implementado como uma função que checa suas entradas com os arcos daquele estado. Se um casamento é encontrado, a função (ou estado) no final desse arco é chamada.

SN simples:

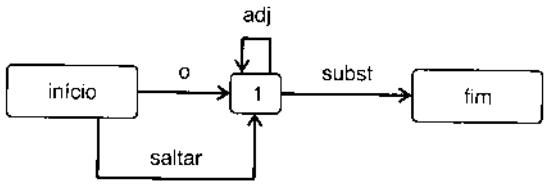


Figura 8.6 Rede de transição.

8.9.2 Redes de transição recursivas

As redes de transição recursivas (Figura 8.7) têm uma característica adicional segundo a qual alguns arcos podem ser rotulados com outra rede de transição subordinada:

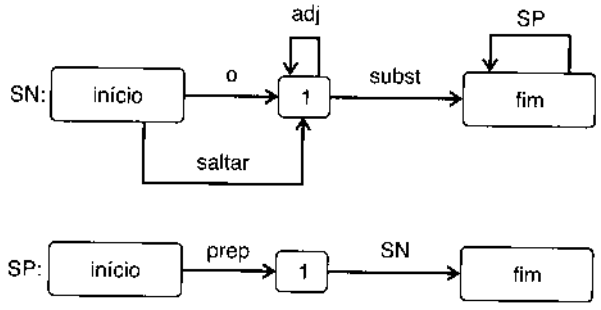


Figura 8.7 Redes de transição recursivas.

Esses arcos permitem que uma dada rede chame uma outra ou mesmo que se chame recursivamente. Usualmente, esses arcos são rotulados com símbolos não terminais (ex., SP para sintagma preposicional). Essas redes são consideravelmente mais poderosas que as redes de transição.

8.9.3 Redes de transição aumentadas

Para a manipulação de línguas naturais, vários pesquisadores em Inteligência Artificial estenderam e refinaram as redes de transição para as *redes de transição aumentadas* (*Augmented Transition Networks* — ATNs), que realizam testes e operações de estrutura de construção durante a análise. As ATNs são similares às redes de transição recursivas, mas têm três características adicionais: registros, que podem armazenar condições ou informação sobre uma base global; condições, que permitem que arcos sejam selecionados se os registros indicam certas condições; e ações, que permitem que arcos modifiquem a estrutura dos dados. A ideia básica é traduzir uma gramática em uma coleção de máquinas de estado finito, ou redes de transição de estado, que podem chamar uma à outra, recursivamente.

Uma vantagem prática das ATNs é sua notação gráfica para gramáticas, mas essa notação não tem importância teórica. As ATNs têm duas notações: uma é a forma gráfica baseada em coleções de máquinas de estado finito, e a outra é a forma executável.

Note que os dados em uma ATN podem ser rotulados não apenas com palavras, classes de palavras e não terminais, mas também com testes arbitrários que dependem do estado dos registros globais. Esses registros globais e seus testes associados tornam possível para o programa checar apenas elementos adjacentes. Obviamente, sendo capaz de armazenar e agir nessas condições, torna-se possível uma análise muito mais eficiente do que as redes de transição anteriores faziam. Por exemplo, a presença de uma forma do verbo ser antes do verbo principal numa sentença pode disparar o cheque para a preposição *por*, o que por sua vez pode confirmar evidência para uma sentença na voz ativa ou passiva.

Ainda que sejam poderosas, as ATNs encontram problemas com sentenças gramaticalmente inválidas, para as quais não existem redes. Se o programa encontra uma construção para a qual não tem uma descrição estrutural, ele simplesmente para.

Como as redes de transição podem ser aumentadas com operações para construção e teste de estruturas, a gramática de estrutura de frase ordinária pode ser estendida a *gramática de estrutura de frase aumentada* (APSG). As gramáticas de metamorfose baseadas em Prolog são equivalentes às APSGs, que são executadas num modo *top-down*.

Ainda que as APSGs e as ATNs sejam dirigidas a regras, suas regras podem ser vistas como programas miniaturas escritos em uma linguagem altamente especializada.

8.10 A REPRESENTAÇÃO DA LÍNGUA NATURAL

Para usar língua natural nas interfaces homem-computador, deve-se primeiro construir uma estrutura abrangente para representar sentenças em língua natural. Usando Prolog pode-se construir uma estrutura de lista sintática e uma estrutura sintático-semântica; pode-se usar Prolog também para incorporar a estrutura semântica em uma estrutura de fundo que transporte os significados de determinadas palavras da sentença para o contexto do conhecimento

do mundo geral. Essa estrutura semântica é então colocada no contexto da sentença sendo analisada. Pode-se usar essa técnica de entendimento da língua natural no desenvolvimento de interfaces amigáveis.

Pode-se usar a gramática de cláusulas definidas, que é um método de expressar regras livres de contexto como comandos lógicos de um tipo restrito, na análise sintática de sentenças em língua natural. Esse método tem muitas vantagens, incluindo suporte para dependência de contexto, a habilidade de permitir que estruturas sejam construídas durante a análise e a habilidade de permitir que condições extras sejam especificadas nas regras gramaticais.

8.11 GRAMÁTICAS DE CLÁUSULAS DEFINIDAS: INTRODUÇÃO

Uma forma de se definir uma *linguagem*, com precisão, seja uma língua natural ou uma linguagem de programação, é através de uma coleção de *regras* chamada de *gramática*. As regras de uma gramática definem quais cadeias de palavras ou símbolos são *sentenças* válidas da linguagem. Além disso, a gramática geralmente fornece algum tipo de *análise* da sentença, em uma estrutura que torna seu significado mais explícito.

Uma classe fundamental de gramática é a *gramática livre de contexto* (GLC), ou "forma de Backus-Naur" (BNF). Nas GLCs, as palavras, ou símbolos básicos, da linguagem são identificadas por símbolos *terminais*, enquanto sintagmas da linguagem são identificados por símbolos *não terminais*. Cada regra de uma GLC expressa uma forma possível para um não terminal, como uma sequência de terminais e não terminais. A análise de uma cadeia de acordo com a GLC é uma *árvore de análise*, mostrando os sintagmas constituintes da cadeia e seus relacionamentos hierárquicos.

Uma ideia importante é traduzir o formalismo de propósito especial de GLCs em um formalismo de propósito geral, chamado de lógica de predicado de primeira ordem. Foi elaborado um método particular para expressar regras livres de contexto como comandos lógicos de um tipo restrito, conhecido como *cláusulas definidas*, ou "cláusulas de Horn". O problema de reconhecer, ou analisar, uma cadeia de uma linguagem é então transformado no problema de provar que um certo teorema segue dos axiomas de cláusulas definidas que descrevem a linguagem.

Essas ideias poderiam apenas ser de interesse teórico. Entretanto, ao mesmo tempo, originou-se uma ideia de maior alcance. Uma coleção de cláusulas definidas pode ser considerada um *programa*. Isso resulta que a dedução automática pode exibir todas as características associadas à computação efetiva, na condição de que a dedução é obtida numa forma direcionada à meta.

Uma realização prática desse conceito de "programação em lógica" foi desenvolvida na forma da linguagem de programação Prolog. Prolog é baseada num procedimento de prova bem simples, mas muito eficiente. Muitas implementações da linguagem existem, e essas implementações têm mostrado que Prolog é tão eficiente quanto as linguagens de programação de alto nível convencionais.

Agora, se uma GLC é expressa em cláusulas definidas de acordo com esse método e executada como um programa Prolog, o programa comporta-se como um analisador *top-down* eficiente, para a linguagem que a GLC descreve.[1] Esse fato torna-se particularmente insignificante

[1] A eficiência do analisador também depende de uma escolha apropriada da GLC para descrever a linguagem.

quando combinado a uma outra descoberta — a de que a técnica para traduzir as GLCs em cláusulas definidas tem uma generalização simples, resultando em um formalismo mais poderoso que as GLCs, mas igualmente pronto para execução pelo Prolog. Esse formalismo chama-se *gramáticas de cláusulas definidas* (GCDs). As GCDs são um caso especial das "gramáticas de metamorfose", que estão para as gramáticas de Chomsky do tipo 0 assim como as GCDs estão para as GLCs. Ainda que as gramáticas de metamorfose possam ser traduzidas em cláusulas definidas, a correspondência não é tão direta quanto para as GCDs.

As GCDs são uma extensão natural das GLCs. Como tais, as GCDs herdaram as propriedades que fazem as GLCs tão importantes para a teoria da linguagem: as formas possíveis para as sentenças de uma linguagem são descritas numa forma clara e modular; é possível representar as características recursivas dos sintagmas, comuns a quase todas as línguas de interesse; existe um corpo estabelecido de resultados nas GLCs que é muito útil no projeto de algoritmos de análise.

É sabido que as GLCs não são totalmente adequadas para descrever língua natural, nem mesmo muitas linguagens de programação. As GCDs compensam esse problema estendendo as GLCs em três importantes formas (Pereira e Warren, 1980).

Primeiro, as GCDs proveem dependência de contexto numa gramática, tal que as formas permissíveis para um sintagma podem depender do contexto no qual esse sintagma ocorre na cadeia. Segundo, as GCDs permitem que estruturas de árvore arbitrárias sejam construídas no decorrer da análise, numa forma que não é forçada pela estrutura recursiva da gramática; tais estruturas de árvore podem prover uma representação do "significado" da cadeia. Terceiro, as GCDs permitem condições extras a serem incluídas nas regras da gramática; essas condições fazem o curso da análise depender de cálculos auxiliares.

Na gramática de cláusulas definidas (GCD) implementadas em muitas versões do Prolog, para análise da sentença:

"O menino chutou a bola"

dever-se-ia ter um SN (sintagma nominal) sujeito ("o menino") e um SV (sintagma verbal) complemento ("chutou a bola"), que por sua vez possui um SN objeto ("a bola"). Ou seja:

```
s(S0,S2) :- sn(S0, S1), sv(S1,S2).
```

s é uma sentença que começa em **S0** e termina em **S2** se contiver um **sn**, que começa em **S0** e termina em **S1** seguido de um **sv** que começa em **S1** e termina em **S2**. Na notação de cláusulas definidas, não há necessidade de se definir os parâmetros internos de controle. Basta escrever:

```
s --> sn, sv.
```

Ou seja, Prolog, ao encontrar o símbolo "`-->`", já coloca os dois parâmetros de controle em cada predicado.

8.12 ENTENDIMENTO DE LÍNGUA NATURAL

As pessoas confundem "língua natural" com "linguagem humana", e os computadores simplesmente não estão adequados ainda para a interação humana. Para ser capaz de entender

a linguagem humana, um computador necessitaria possuir o tipo de conhecimento sobre a linguagem que as pessoas possuem.

Uma gramática para linguagem humana assegura que as partes da sentença concordam em tempo verbal, pessoa etc., isso é referido como informação de "contexto". As GLCs não são capazes de manipular a informação de contexto, mas as GCDs sim. Portanto, quando construídas com GCDs, as línguas naturais podem imitar a linguagem humana.

Além da complexidade, as linguagens humanas são grandes demais para os computadores trabalharem. Para uma conversação na linguagem humana, um computador teria que armazenar as definições de milhares de palavras e todas as formas possíveis nas quais essas palavras podem ser combinadas.

Ainda que o tamanho e a complexidade das linguagens humanas não possam ser manipulados pela tecnologia corrente, sistemas de língua "naturais" úteis, porém limitados, podem ser desenvolvidos. Se o sistema é limitado de tal forma que ele trabalhe apenas com um comando de cada vez, a gramática é menos complexa. Isso torna possível a construção de um sistema usando técnicas bem conhecidas, tais como as GCDs. Se os tópicos sobre os quais o sistema pode conversar são também limitados, a gramática é pequena.

Mesmo com essas limitações, ainda existem problemas. As linguagens humanas permitem anomalias que as línguas naturais não podem permitir. Uma anomalia que as linguagens humanas permitem é a referência ambígua. Por exemplo, na sentença "O sorvete está no *freezer* e ele está gelado", deve-se decidir se *ele* refere ao sorvete ou ao *freezer*. Pode-se usualmente adivinhar ao que *ele*, *isso* e *aquilo* referem simplesmente pelo contexto. Essas regras de adivinhação não são facilmente traduzidas em regras gramaticais.

Ainda que palavras como *ele*, *isso* e *aquilo* alertem para referências ambíguas, outras referências ambíguas são menos óbvias de se ver. Considere a sentença "Eu conheço uma mulher com uma perna de madeira chamada Dalva". Sabe-se que Dalva é o nome da mulher e não da perna de madeira. Entretanto, o sistema de língua natural precisaria saber que as pessoas têm nomes e as pernas de madeira normalmente não.

Na linguagem humana, as partes de uma sentença podem ser arranjadas de diferentes formas e ainda assim produzir o mesmo significado. Por exemplo, "A que horas o trem parte?" e "O trem parte a que horas?" significam a mesma coisa. Entretanto, cada forma requer sua própria definição de gramática.

Um sistema de língua natural que possa manipular todas essas situações seria grande e complexo demais. Os sistemas de língua natural não devem apenas limitar a gramática a sentenças simples, eles devem também fazer assunções sobre os tipos de sentenças que o usuário entrará. Quando se projeta um sistema de língua natural, deve-se decidir quais formas de sentenças o sistema reconhecerá e o que essas sentenças significarão. Frequentemente, dentro de uma área bem definida, pode-se escolher um número limitado de estruturas de sentenças para atender à maioria das necessidades do usuário.

8.13 INTERPRETAÇÃO DA LÍNGUA

Os principais aspectos na habilidade de um computador em entender uma sentença em língua natural são a análise sintática, a adição de características às palavras, a alocação de funções às palavras e a fixação da sentença na sua estrutura de contexto. A adição de características novas às pa-

lavras significa a incorporação ao sistema de entendimento interativo de características como número e gênero. A alocação de funções às palavras significa explicitamente a expressão das funções executadas por palavra na sentença; essas funções incluem o fato de a palavra ser um agente ou um objeto. As pessoas usam essas mesmas técnicas, mas não necessariamente na mesma ordem.

As pessoas e os computadores devem trabalhar com a sintaxe e a semântica cooperativamente, mas existem muitas formas nas quais sintaxe e semântica podem interagir. A análise pode produzir uma representação sintática completa que é então interpretada semanticamente. Usar uma fase de análise separada para representar concisamente os relacionamentos sintáticos é uma vantagem porque muitas regras semânticas diferentes pedem essencialmente a mesma informação sintática.

Análise Sintática. Um analisador sintático aplica regras de estrutura de frase para construir uma árvore de análise. Dados um conjunto de regras e um léxico apropriado, pode-se construir uma árvore de análise, que é uma estrutura sintática que um sistema de entendimento semântico pode usar. O propósito da análise sintática é checar a validade da sentença de entrada e organizar a sentença em unidades sintáticas que determinam seu significado. Veja a Figura 8.4, que mostra a árvore de análise sintática da sentença "O menino esperto estudou muito".

Usando os fatos e as cláusulas de Horn do Prolog, o analisador pode construir a estrutura lista:

```
sentença(sn(det(o),subst(menino),adj(esperto)),sv(verbo(estudou),
adv(muito))).
```

A adição de características às palavras. Para checar a compatibilidade de número e gênero, a consistência de tipos de verbos e certas inconsistências semânticas, deve-se adicionar informação ao analisador sobre palavras definindo suas características. Com essa informação, o analisador pode fazer um cheque de consistência para eliminar estruturas incorretas em sintaxe e em semântica (Geetha e Subramanian, 1990):

- Compatibilidade de número. Considere a sentença "Ele gostam de comida quente". O número do substantivo e do verbo não casa, e então a sentença está sintaticamente incorreta. A incompatibilidade está entre palavras pertencentes a frases sintáticas diferentes, que são os grupos formados por palavras da sentença baseados nas suas categorias sintáticas (como sintagma nominal e sintagma verbal). Considere um outro exemplo: "Maria comeu estas manga". Nesse caso, a incompatibilidade está entre palavras pertencentes à mesma frase sintática. Isso traz a necessidade de adicionar o número como uma característica da palavra.
- Compatibilidade de gênero. Considere a sentença "Moacir e o amigo dela foram ao parque". Quando considerada isolada, essa sentença mostra que os gêneros de *Moacir* e *dela* não casam, então a conclusão é que o amigo não é de Moacir. Para achar a incompatibilidade, é necessário conhecer os gêneros de *Moacir* e *dela*. Então, deve-se adicionar o gênero como uma característica da palavra.
- Consistência de tipo de verbos. Verbos têm muitas características que ditam a estrutura do resto da sentença. Considere a sentença "Henrique correu Tom". "Correr" é um verbo intransitivo, que significa que ele não age num objeto específico, então ele não pode ter um SN como objeto direto. No máximo, sintagmas preposicionais ou adverbiais como "com Tom" ou "demais". Portanto, a analisador deve ter a informação de se o verbo é transitivo ou intransitivo.

- Inconsistências semânticas. Número, gênero e tipo de verbo são usados apenas para checar o aspecto sintático das sentenças. Podem-se usar certas características de palavras para achar as inconsistências semânticas que dependem somente da natureza de determinadas palavras. Considere a sentença "Ivan come pessoas". O analisador checa as características do sujeito (*Ivan*) e objeto (*pessoas*) para ver se eles são semanticamente compatíveis com o verbo (*come*). Nesse exemplo, o verbo prefere um sujeito animado e um objeto não humano, então existe uma inconsistência semântica mesmo que a sentença esteja sintaticamente correta. (Para as exceções, como Ivan ser um canibal, adicionar-se-ia esse fato ao sistema, que daria precedência sobre as regras gerais no léxico do sistema.)

A estrutura lexical. O léxico que o analisador usa para sua análise em língua natural deve ter dois tipos de informação: a informação sintática e os detalhes semânticos. Ele usa essa informação para auxiliar a análise sintática e mais tarde construir a representação contextual e semântica. A forma da entrada lexical é diferente para categorias sintáticas diferentes.

Substantivos, adjetivos, advérbios e outras partes do discurso — exceto verbos — têm a mesma estrutura no léxico:

tipo_sintático da palavra (X, R, N, G, T)

O tipo_sintático da palavra seria um token como é_substantivo, é_pronome, ou é_nome_próprio. X é a palavra do texto de entrada, R é a raiz da palavra, N é o número da palavra (singular, plural ou indeterminado), G é o gênero (masculino, feminino, comum ou neutro) e T é uma lista contendo propriedades semânticas como humano, animado ou entidade.

Para verbos, o léxico também inclui características de verbo como tipo (transitivo ou intransitivo) para análise semântica; essas características podem ser incluídas como regras separadas. Podem-se usar outras características (como número e tempo verbal) diretamente (por casamento de padrões simples) para achar anomalias sintáticas como discordância de sujeito-verbo (por exemplo, "Ele comem"). A estrutura para verbos do léxico é:

é_verbo (X, R, N, TN, T)

em que X é a palavra do texto de entrada, R é a raiz do verbo, N é o número, TN é o tempo verbal (como passado simples, presente, futuro etc.) e T é o tipo (transitivo ou intransitivo). Para a língua portuguesa, haveria também a necessidade da pessoa.

Depois da análise sintática, o analisador agrupa as palavras em frases sintáticas como SN, SV e SP. Cada frase sintática tem uma palavra principal da qual a frase depende, e a frase sintática aceita as características dessa palavra principal. O sistema determina quais funções essas frases executam, aplicando regras apropriadas e casamento de padrões.

A representação semântica. As funções reais, as quais definem as relações que as outras frases sintáticas têm com o verbo, são executadas por frases sintáticas operando como uma unidade. O conhecimento das diferentes funções executadas por essas frases dá ao computador uma compreensão parcial do significado da sentença. O significado da estrutura inteira depende do verbo e das outras unidades que executam funções relacionadas com o verbo principal.

Cada verbo terá um *frame*, o qual será armazenado no léxico. Outros constituintes sintáticos serão alocados aos seus respectivos *slots* se eles satisfizerem os limites semânticos dados.

8.14 LÓGICA E LÍNGUA NATURAL

Lógica é o ramo do conhecimento que trata da verdade e da inferência — ou seja, como determinar as condições sob as quais uma proposição pode ser verdadeira, ou pode ser inferida a partir de outras proposições. Tal conhecimento é essencial para a comunicação, pois a maioria das crenças humanas sobre o universo vem não diretamente do contato com ele, mas do que as pessoas contam às outras.

Várias pesquisas em interações de língua natural com máquinas usam a lógica como base, incluindo análise, interpretação semântica e raciocínio. Entretanto, não apenas lógica de primeira ordem, mas também formas de lógica mais poderosas — ou pelo menos, muito diferentes — estão sendo usadas no entendimento e na geração de língua natural.

A ideia de se usar lógica como um suporte conceitual em sistemas de perguntas e respostas não é nova. O fato de que se pode formalmente tratar com a noção de consequência lógica a torna atrativa para a representação do significado. O cálculo de predicados padrão, entretanto, não parece adequado para representar todas as características semânticas da língua natural, por exemplo, pressuposições e as sutilezas do significado envolvidas nos quantificadores da língua natural. Entretanto, alguns desenvolvimentos indicam que a lógica tem um papel importante no Processamento de Línguas Naturais (Dahl, 1981).

Em primeiro lugar, a pesquisa linguística chegou a resultados interessantes considerando a extensão do cálculo de predicados padrão a fim de prover um melhor modelo de linguagem formal.

Em segundo lugar, a programação em lógica tornou-se possível desde o desenvolvimento da linguagem de programação Prolog. A lógica pode agora ser usada tanto como um formalismo básico quanto como a ferramenta de programação.

Em terceiro lugar, muitas das implementações em Prolog incluem uma versão da gramática de metamorfose (GM), um formalismo baseado em lógica útil para descrever processadores de língua natural em termos de regras de rescrita muito mais poderosas.

Finalmente, a evolução da tecnologia de base de dados tem se voltado cada vez mais na direção do uso da lógica, seja na descrição dos dados, seja para perguntas.

O entendimento e a geração da língua são exercícios do raciocínio. Para o raciocínio, a lógica é a melhor ferramenta.

8.15 O PROBLEMA DA INTEGRAÇÃO

A interpretação da língua natural requer a aplicação cooperativa de muitos sistemas de conhecimento, conhecimento específico da língua sobre o uso da palavra, a ordem da palavra e a estrutura da frase, e o conhecimento do "mundo real" sobre situações estereotípicas, eventos, casos, contextos etc. Não se pode construir um processador de língua natural psicologicamente realista meramente juntando vários módulos de processamento de conhecimento específico serialmente ou hierarquicamente.

Os problemas que surgem para a integração dos vários sistemas de conhecimento para o Processamento de Línguas Naturais são os seguintes:

Ambiguidade. A ambiguidade é o maior problema no Processamento de Línguas Naturais. Existem basicamente duas abordagens para tratamento de sentenças ambíguas: o *backtracking*, como usado nas redes de transição aumentadas (ATNs) de Woods (1970), e o *delay*, usado no analisador *espere e veja*.

Interpretação única. Outro fenômeno interessante na interpretação da língua é que as pessoas podem considerar apenas uma interpretação de uma sentença ambígua de cada vez, mas podem facilmente saltar entre interpretações.

Erros de compreensão. Erros na compreensão, como as sentenças enganosas (*garden path*), têm sido explicados por princípios estruturais puros. Entretanto, existem explicações mais completas e naturais como os efeitos colaterais de processos fortemente interativos. Os efeitos enganosos podem ocorrer em todos os níveis do processamento da língua. Considere a seguinte sentença:

O astrônomo casou-se com a estrela. (Charniak, 1983)

Os leitores usualmente reportam essa sentença como "temporariamente anômala", isto é, uma "sentença semanticamente enganosa". Uma explicação plausível para a dupla interpretação cognitiva da sentença é que o poder primitivo de "astrônomo" no sentido errado de "estrela" é inicialmente mais forte que o poder lógico das restrições de seleção de "estrutura de caso"; no final, as restrições de seleção forçam a interpretação de "estrela" como uma pessoa.

Texto não gramatical. As pessoas são capazes de interpretar linguagem não gramatical se esta ocorrer naturalmente (devido a gramática pobre, estrangeirismos, interferência de barulho externo, interrupções, autocorreções etc.).

8.16 INTEGRANDO FONTES DE CONHECIMENTO

Todos esses fenômenos indicam a necessidade de uma teoria de processamento de linguagem que afirme, em vez de uma simples passagem de resultados incompletos entre componentes processadores, uma forte interação entre esses componentes, de tal forma que todas as decisões sejam interdependentes. Acredita-se que a maioria das teorias de processamento de linguagem existentes é falha, porque está limitada a um conjunto de ideias computacionais que efetivamente não podem tratar com decisões interdependentes e por causa das peculiaridades da língua portuguesa e da história da pesquisa linguística que levou a assunções da autonomia da sintaxe no entendimento da língua natural.

Essas observações, é claro, não são inteiramente novas. No início dos anos 1970, argumentava-se que a semântica, e não a sintaxe, deveria ter o papel central nas teorias programadas de Processamento de Línguas Naturais.

8.17 MARCADORES DE PASSO: UMA TEORIA DE INFLUÊNCIA CONTEXTUAL

A maior parte dos sistemas de compreensão de linguagem em Inteligência Artificial segue o modelo geral mostrado na Figura 8.8 (Charniak, 1983). Um componente inicial usa conhecimento sintático para pegar a estrutura funcional da sentença (por exemplo, em "*Frederico foi morto por José*", é preciso distinguir o matador do morto). O processo sintático é guiado

por uma unidade semântica que também é responsável por transformar a entrada em uma representação semântica. Para a maioria desses modelos, a informação passa entre sintaxe e semântica em alguma forma de árvore sintática.

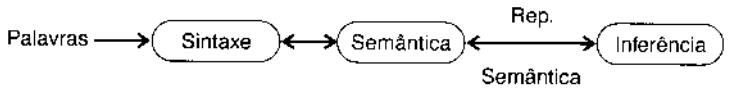


Figura 8.8 A teoria padrão (Charniak, 1983).

Serão tratados aqui os seguintes casos:

1. a determinação do "contexto" na compreensão da história;
2. o papel do contexto na não ambiguidade do sentido da palavra;
3. a relação entre sintaxe e semântica.

Essa teoria está mostrada na Figura 8.9.

Uma teoria de ativação alastradora (ou marcadores de passo) do processamento semântico humano, proposta inicialmente por Quillian (1968), além de ser uma teoria para explicar dados, era uma teoria projetada para mostrar como construir uma estrutura e processamento semânticos humanos em um computador.

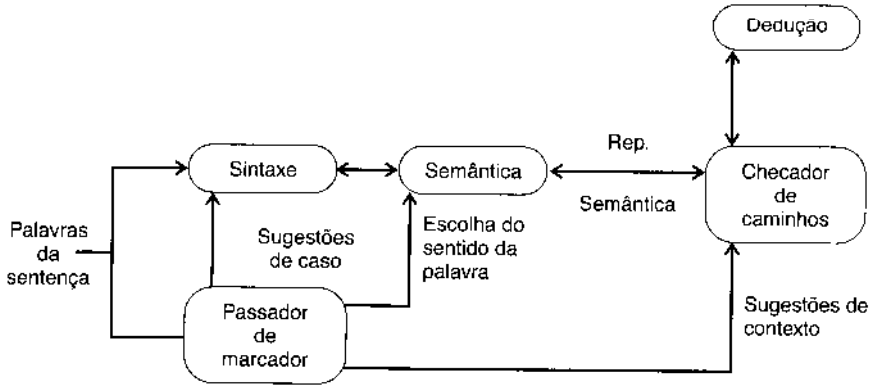


Figura 8.9 A teoria proposta (Charniak, 1983).

8.17.1 A teoria de Quillian da memória semântica

O fato de a teoria de Quillian ser desenvolvida para um computador digital impôs a ela certas restrições.

Os conceitos das pessoas contêm quantidades indefinidamente grandes de informação. Quillian usava o exemplo de uma máquina. Se alguém pedir para uma pessoa contar tudo que sabe sobre máquinas, ela vai começar dando propriedades óbvias, por exemplo, que as máquinas são feitas pelo homem e que têm partes móveis. Mas logo a pessoa vai desprezar os fatos óbvios e, cada vez mais, vai fornecer fatos que são menos relevantes, por exemplo, que uma

furadeira é uma máquina etc. A quantidade de informação que uma pessoa pode gerar dessa forma sobre qualquer conceito parece ilimitada.

Um conceito pode ser representado como um nó numa rede, com as propriedades do conceito representadas como ligações relacionais rotuladas a partir do nó para outros nós de conceito. Essas ligações são ponteiros e usualmente vão a ambas as direções entre dois conceitos. As ligações podem ter *critérios* diferentes, que são números indicando o quão essencial é cada ligação para o significado do conceito. Os critérios em qualquer par de ligações entre dois conceitos podem ser diferentes; por exemplo, deve ser de alto critério para o conceito de uma furadeira ser uma máquina, e de baixo critério para o conceito de um tipo de máquina ser uma furadeira. A partir de cada um dos nós ligados a um nó dado existirão ligações para outros nós de conceito e a partir de cada um desses nós para outros e assim por diante.

8.18 DETERMINAÇÃO CONTEXTUAL DE SENTIDOS DE PALAVRAS

Na presença do contexto certo, as pessoas não estão conscientemente preparadas para as palavras ambíguas. Nunca são consideradas todas as alternativas. As coisas que parecem ser organizadas têm o significado certo considerado primeiro. Porém, no exemplo de Charniak (1983):

"O astrônomo casou-se com a estrela."

a maioria das pessoas tem dificuldade de achar a leitura de "estrela de cinema" para "estrela", a despeito do fato de que a leitura de "objeto astronômico" torna difícil imaginar a cerimônia de casamento. Esse exemplo evidencia uma teoria na qual as pessoas consideram que os sentidos das palavras são modificados por itens contextuais (ou mesmo que apenas alguns sentidos estão "disponíveis" para consideração). De acordo com essa teoria, os problemas com exemplos desse tipo vêm do fato de que o primeiro sentido tentado não está de acordo com os requisitos reais da sentença e portanto deve ser rejeitado.

Através dos experimentos de Swinney (1979), conclui-se que o contexto não pré-seleciona quais sentidos de palavras considerar. Mas se esse é realmente o caso, por que as pessoas têm problemas com o exemplo do "astrônomo/estrela"? Faz sentido quando se assume que o contexto pré-selecionou o sentido incorreto da palavra. Mas foi dito que o contexto não pré-seleciona. Então parece que a semântica tem todos os sentidos disponíveis para fazê-lo e tem todas as informações necessárias para fazer a escolha certa (em particular, ela sabe que "estrela" é o objeto de "casar" e que "casar" requer uma pessoa como objeto). Ainda que a semântica faça a escolha errada. No modelo padrão da Figura 8.10, não há razão para isso acontecer. Por que então as pessoas erram?

8.19 ABORDAGENS AO PROCESSAMENTO SIMBÓLICO DE LÍNGUAS NATURAIS

8.19.1 Introdução

Pereira e Grosz (1993) dividem o Processamento Simbólico de Línguas Naturais em três abordagens: baseada em casos, baseada em princípios e baseada em regras. Mas o que são

casos, princípios e regras? Um *caso* é uma associação entre uma situação prototípica e a informação relevante à tarefa que a segue. Por exemplo, um caso pode representar uma sentença da linguagem natural envolvendo o verbo principal *dar* e alguma outra informação dessa sentença, por exemplo, que depois da ação descrita pelo verbo o agente da ação não tem mais a posse do "paciente" da ação. O raciocínio baseado em casos envolve a descrição de analogias entre situações observadas recentemente e casos relevantes e o uso da informação de tarefa associada para determinar as inferências apropriadas às novas situações.

Um *princípio* é uma restrição aos tipos de situações possíveis: permite que um sistema infira características de situações adicionais a partir de outras características observadas. Por exemplo, um princípio na sintaxe da linguagem natural requer que cada sintagma nominal em uma sentença preencha exatamente uma posição argumental de um item lexical com posição argumental tal como um verbo ou uma preposição. Tal princípio restringe as associações possíveis entre itens lexicais com posição argumental e sintagmas nominais e portanto restringe a faixa de significados que podem ser expressos por uma determinada sentença.

Uma *regra* especifica como certas características de, ou relacionamentos entre, situações seguem de outras. Por exemplo, de novo na sintaxe da linguagem natural, uma regra de algumas línguas estabelece que um sintagma nominal (SN) seguido por um sintagma verbal (SV), havendo concordância em gênero e número, pode formar uma sentença (S), com o SN como sujeito e o SV como predicado.

8.19.2 O relacionamento entre regras e casos

O contraste entre as abordagens baseada em regras e baseada em casos está essencialmente na fonte de generalidade de um sistema. Em sistemas baseados em regras, a generalidade vem da escolha de primitivas descritivas que permitem grandes coleções de situações com resultados similares a serem identificados e trabalhados por regras; em contraste, a generalidade em um sistema baseado em casos vem dos procedimentos de recuperação de caso e unificação (*matching*) que determinam o resultado para uma situação nova a partir de resultados para casos similares armazenados. Permitindo noções de unificação parcial ou aproximada, os sistemas baseados em casos são frequentemente capazes de agir mesmo quando seu conhecimento de caso não unifica totalmente com a situação sob análise. Por outro lado, as regras previamente projetadas podem resumir e identificar eficientemente os itens comuns em grandes conjuntos de casos, tornando então o conhecimento do sistema mais largamente aplicável.

A utilidade de uma abordagem baseada em casos depende crucialmente da eficiência dos mecanismos de aquisição e do uso da informação específica sobre a distribuição das situações de interesse. No PLN, tais situações envolvem objetos linguísticos tais como palavras ou unidades fonéticas em determinados contextos. Enquanto as abordagens baseadas em casos devem ser avaliadas por sua habilidade de aprender casos relevantes, generalizá-los apropriadamente e aplicá-los, nossa falta de seleção de caso e métodos de generalização efetivos força os praticantes atuais a criar a maior parte da informação de caso manualmente. Dado isso, os problemas mais importantes enfrentados por esses sistemas são a escolha dos traços (*features*) de caso relevantes à seleção de caso, o reconhecimento dos casos que se aplicam a uma situação dada e a construção de interpretações para enunciados (*utterances*) complexos a partir de combinações de casos apropriados unificando partes do enunciado.

Enquanto as abordagens ao PLN baseadas em casos se inspiram nas ideias das Ciências Cognitivas, que tratam da organização da memória e inferência do senso comum, as abordagens baseadas em regras derivam fundamentalmente das tradições fortes da Linguística e da Teoria da Linguagem Formal. Essas origens têm levado a arquiteturas de sistema centradas nas noções da descrição estrutural e da transdução estrutura a estrutura. Por exemplo, as regras de estrutura de frase são usadas para descrever a sintaxe da linguagem natural, e regras adicionais em cascata são então usadas para transformar tais descrições estruturais, através de uma sucessão de representações intermediárias, em uma representação do conteúdo das sentenças originais. Várias representações têm sido usadas, incluindo fórmulas lógicas, redes semânticas e quadros (*frames*). Embora as arquiteturas baseadas em regras tenham produzido sistemas de processamento de linguagem muito expressivos, elas têm encontrado sérias dificuldades na área da robustez, isto é, a habilidade de produzir saída útil mesmo diante de regras muito específicas ou ausentes e de tratar com fenômenos não composicionais, ou seja, situações nas quais a saída apropriada numa situação complexa não pode ser derivada por uma regra simples a partir das saídas para suas partes.

8.19.3 O relacionamento entre regras e princípios

Outro conjunto de dificuldades com sistemas baseados em regras no PLN surge da rigidez e especificidade das regras. Por exemplo, com a exceção de alguns sistemas recentes que usam formalismos de regra baseados em restrições declarativas e estratégias de aplicação de regras sofisticadas, as considerações baseadas em regras do mapeamento sintaxe-significado são tipicamente unidirecionais; portanto, evita-se o uso das mesmas regras para interpretação e geração da linguagem. Mais fundamentalmente, os sistemas a regras são específicos da língua e da construção, portanto requerem esforço maior para serem transportados para outras línguas ou mesmo para outras partes da mesma língua ou outros domínios.

Essas dificuldades podem ser vistas como sintomas da restrição da noção usual de regra, que força uma definição gerativa do relacionamento entre análises e interpretações. Por exemplo, um sistema que mapeia análise sintática para fórmulas lógicas que representam significados da sentença teria tipicamente uma regra da gramática estabelecendo que uma sentença como "um estudante fez todo teste" é composta de um sintagma nominal sujeito ("um estudante") seguido por um sintagma verbal predicado ("fez todo teste"). Associada a essa regra da gramática haveria uma regra de interpretação estabelecendo que o significado da sentença é igual ao significado do sujeito aplicado ao significado do predicado. No nosso exemplo, o significado do sujeito poderia ser uma fórmula que pode ser explicada como "verdadeira para qualquer propriedade que tem algum estudante", e o significado do predicado, como uma fórmula que se pode explicar como "propriedade de fazer todo teste". A interpretação resultante para a sentença poderia então ser explicada como "existe um estudante que tem a propriedade de ter feito todo teste". Essa interpretação força o quantificador do sujeito a ter escopo mais largo do que o quantificador do objeto. Mas, para manipular adequadamente linguagem natural, o processo de interpretação precisa considerar escopos alternativos antes de escolher aquele que é contextualmente mais apropriado.

A causa fundamental desse problema é que regras privilegiam conexões gerativas particulares entre evidência e interpretação. Em contraste, a evidência específica que pode ser extraída de uma situação natural tal como um enunciado (isto é, a estrutura sujeito-predicado

no exemplo anterior) é muito indeterminada para ser confiavelmente modelada como uma transdução entre um domínio de descrições estruturais e um domínio de interpretações.

Em contraste às abordagens baseadas em regras, nas abordagens baseadas em princípios os princípios fornecem restrições fundamentais gerais entre tipos de evidências em diferentes níveis de descrição. A análise da linguagem, interpretação ou geração é vista não como um processo de reescrita, mas como uma busca pela hipótese que melhor explica a evidência observada; o espaço de busca é implicitamente definido pelos princípios e por restrições de domínio específico.

Numa visão baseada em princípios, em processos com componente perceptual como o processamento da linguagem, as restrições impostas pelo sistema são uma fonte de princípios. Na linguagem natural, uma visão correspondente (devido à teoria linguística de princípios e parâmetros de Chomsky e seus seguidores) assegura que os princípios, ancorados em critérios determinadores evolucionários tal como aprendizagem, eficiência comunicativa e carga cognitiva, proveem um sistema de regularidades "legais" que definem os espaços de representações possíveis nos vários níveis relevantes de descrição e as restrições entre esses níveis. Na linguagem, esses níveis incluem sintaxe, semântica, discurso e prosódia, ainda que o trabalho de PLN baseado em princípios se tenha concentrado somente nos níveis sintático e lexical. Abstratamente, os princípios não são apenas independentes de modelos de processamento específicos, mas também de línguas particulares ou domínios de discurso. Entretanto, para ser usado na prática, um sistema baseado em princípios deve ser "preenchido" com conhecimento sobre objetos particulares — línguas, palavras, conceitos — nos termos dos princípios. Noções de aprendizagem têm um papel importante na concepção das teorias baseadas em princípios; entretanto, algoritmos efetivos para aprendizagem do conhecimento específico necessário ainda não estão disponíveis. Por ora, o conhecimento específico deve ser descrito manualmente, como também é o caso dos sistemas baseados em regras e baseados em casos, mesmo que a generalidade dos princípios em algumas instâncias permita especificação mais concisa do conhecimento específico.

Enquanto nas abordagens baseadas em regras tem-se uma regra para cada construção, nas abordagens baseadas em princípios consideram-se alguns princípios apenas, que, combinados, atendem a qualquer construção. Os princípios considerados são (Berwick, 1992): teoria X-barra, filtro de caso, critério temático, move-a, teoria de vestígios e teoria de ligação. Apesar de, aparentemente, parecer ineficiente computacionalmente, esse tipo de abordagem é bem interessante, pelas seguintes razões:

1. o mesmo conjunto (pequeno) de princípios pode ser recombinado várias vezes, de diferentes formas, resultando em muitas sentenças de superfície, e variando os parâmetros, diferentes dialetos e línguas;
2. princípios abstratos e heterogêneos, estabelecidos como um conjunto de restrições declarativas, ao contrário de uma representação mais uniforme como um conjunto de regras livres de contexto;
3. ênfase na importância do léxico, fonte, por exemplo, de restrições de papel temático e variação de línguas particulares.

8.19.4 *Parser* baseado em princípios

Segundo Crocker (1991), as abordagens tradicionais ao PLN podem ser baseadas em construção. Isto é, elas empregam regras específicas da linguagem orientadas à superfície, ou na forma de

Redes de Transição Aumentadas (ATN, de Woods, 1970), gramáticas lógicas ou algum outro formalismo de gramática ou *parsing* (Pereira e Warren, 1980). Os problemas de tais abordagens são claros, pois envolvem grandes conjuntos de regras, frequentemente *ad hoc*, e sua adequação com respeito à gramática da língua é difícil de assegurar. Em contraste, esforços na pesquisa linguística têm observado certas regularidades nas línguas naturais. De fato, uma tentativa inicial foi caracterizar esses universais linguísticos, que definiriam a classe das linguagens naturais, resultando na teoria da Gramática Universal (GU). A melhor teoria da GU desenvolvida por Chomsky e outros foi um paradigma de Princípios e Parâmetros, frequentemente referido como a teoria da Regência e Ligação (GB para *Government and Binding*). GB é uma teoria dedutiva e modular da gramática que trabalha com vários níveis de representação relacionados por uma regra transformacional, a *move*-α. A aplicação de *move*-α é restringida pela interação de vários princípios que agem como condições às possíveis representações ou derivações. Associados aos princípios estão os parâmetros que dão conta das variações entre as línguas. Então a gramática para uma determinada língua é especificada pelo estabelecimento de parâmetros apropriados e por um léxico.

As abordagens baseadas em princípios não são apenas independentes de modelos de processamento específico, mas também de línguas ou domínios de discurso particulares.

Crocker (1996), no contexto da teoria da gramática dos princípios e parâmetros assumida, discute a noção de "baseado em princípios": um modelo que usa os princípios da gramática diretamente na recuperação de uma análise sintática. Isto é, ele *não* usa uma gramática compilada, transformada.[2] Ou seja, existem várias teorias de *performance*[3] que podem ser consideradas baseadas em princípios, no sentido de que elas usam os princípios da gramática *on-line*, mas tomam decisões na base de critérios "não linguísticos", tal como eficiência computacional ou complexidade representacional. Tais modelos claramente contrastam com teorias de processamento que operam de acordo com estratégias baseadas em gramática, sugerindo um relacionamento mais próximo entre *parser* e gramática. Uma teoria de *performance* que é baseada em princípios e incorpora estratégias que são baseadas em gramática (no sentido descrito) é descrita como 'fortemente baseada em princípios'.

8.19.5 *Parser* baseado em casos

Para conectar o texto de entrada ao conhecimento e metas prévios do entendedor, deve-se ter um modelo no qual o acesso ao conhecimento e metas prévios seja uma parte integral do processo de *parsing*. O *parsing* baseado em casos atende esse requisito. O objetivo de um *parser* baseado em casos é reconhecer quais estruturas de memória já existentes são mais relevantes à entrada, em que essa "relevância" é determinada pelos planos e metas do entendedor. Essa abordagem difere dos modelos tradicionais de parsing que tenta construir uma análise sintática ou uma estrutura de significado conceptual para um texto. O *parsing* baseado em casos é primariamente um processo de *reconhecimento* (Martin, 1989).

Como o *parsing* baseado em casos objetiva uma meta diferente dos outros *parsers*, o algoritmo para um *parser* baseado em casos também é diferente. Certos aspectos do algoritmo

[2] Um exemplo disso é o *parser* de Marcus (1980), que computa uma estrutura de superfície, incluindo relações antecedente-vestígio. O *parser* faz isso sem tornar explícito o uso dos princípios da gramática, mas no entanto ele os obedece. Esse *parser* é "fracamente baseado em princípios".

[3] *Competência* refere-se ao conhecimento da língua, e *performance*, ao modo como se usa esse conhecimento.

codificam conhecimento sintático ou conceptual, mas o algoritmo tem a principal preocupação de prover acesso às estruturas de memória preexistentes o mais cedo possível no curso do PLN. Esse acesso às estruturas de memória é essencial ao entendimento. Portanto, a organização da memória é fundamental a um *parser* baseado em casos. Este deve ser a ponte entre os itens lexicais primitivos da entrada e as estruturas de memória direcionadas identificadas como a saída do processo de entendimento. O *parsing* baseado em casos depende dos planos e metas idiossincráticos do entendedor.

Um texto pode — e deve — se referir a muitas estruturas de memória, e cada estrutura é uma caracterização diferente da entrada nos termos relacionados à meta. A noção de um único significado para um texto é abandonada no *parsing* baseado em casos.

Os *parsers* convencionais, que constroem uma representação do significado de um texto de entrada, geralmente retornam uma representação como a saída do processo de *parsing*. Um *parser* baseado em casos, entretanto, pode ser chamado a reconhecer múltiplas estruturas de memória no curso do processamento de um texto de entrada. O que é significativo sobre essas estruturas de memória não são suas representações individuais, mas suas conexões com outras estruturas de memória que podem também ser relevantes ao texto. O conjunto de expectativas e referências do *parser* determina quais estruturas de memória serão reconhecidas. É esse conjunto de expectativas e referências que constitui um resultado do processo do *parsing* bem-sucedido.

A saída de um *parser* baseado em casos pode ser caracterizada como *um novo estado de memória*. Algumas estruturas terão sido referenciadas por um texto de entrada ou processo de inferência, e algumas serão esperadas. Ainda que novas estruturas de memória sejam adicionadas, quando a informação específica já não estiver na memória, é o estado de referência e expectativas que constitui a saída real do sistema.

Um *parser* baseado em casos usa itens linguísticos tais como palavras indivíduais para direcionar o processo de busca aos conceitos na memória. A tarefa de busca na memória consiste em conectar essas referências espalhadas, achando as unidades organizacionais para a memória que melhor organizem a entrada.

O conhecimento do processamento de um *parser* baseado em casos está na forma de *expectativas*. As expectativas são baseadas na ideia de senso comum de que as pessoas são capazes de fazer previsões sobre o que deve acontecer no futuro com base no que aconteceu no passado e na sua experiência anterior.

As expectativas são derivadas de exemplos estereotípicos do uso da linguagem, que apontam para as unidades organizacionais da memória. Esses são os índices para um *parser* baseado em casos. Portanto, esses *parsers* usam exemplos específicos de uso da linguagem para indexar estruturas de memória.

A hipótese fundamental do *parsing* baseado em casos é que o acesso ao conhecimento prévio na forma de estruturas de memória dinâmicas, específicas do domínio, é crucial nos estágios mais iniciais do entendimento da linguagem natural. É a ideia do *parsing* através da lembrança. Realizar o *parsing* a partir de casos significa lembrar instâncias passadas do uso da linguagem tal que possam ser reconhecidas de novo e lembrar conceptualizações passadas tal que possam ser chamadas novamente. O *parsing* baseado em casos é muito diferente da análise conceptual. Questões de organização de memória, indexação e busca na memória são de importância central para um *parser* baseado em casos porque este deve operar dentro de um modelo de memória existente.

A tarefa do *parsing* é um problema de busca na memória. Conceitos não são construídos a partir de pedaços derivados da entrada. Ao invés disso, já existem conceitos que preenchem muitas necessidades do entendedor. A tarefa é usar os indícios supridos pela tarefa para localizar os conceitos mais relevantes e modificá-los quando necessário para refletir as diferenças entre o que é visto e o que já é conhecido. Já que um *parser* baseado em casos faz uso da memória, ele pode fazer uso das expectativas derivadas dessa memória. Essas expectativas dirigem o processo de *parsing*.

O *parser* baseado em casos difere de outras abordagens principalmente em relação aos seguintes pontos:

1. Em vez de acessar uma gramática geral da sintaxe da linguagem para determinar os elementos relacionados de um enunciado, um *parser* baseado em casos captura as formas idiossincráticas nas quais a linguagem é usada para referenciar determinados conceitos na memória.
2. Em vez de construir conceptualizações para representar o significado de um texto de entrada, um *parser* baseado em casos faz uso dos elementos de análise conceptual para direcionar o processo de busca à memória aos conceitos que organizam esses elementos.
3. Em vez de depender da memória para resolver ambiguidades de possíveis interpretações depois do *parsing* realizado, um *parser* baseado em casos usa as metas e expectativas na memória para resolver ambiguidades da entrada durante o processo de *parsing*.

A saída de um *parsing* baseado em casos inclui mudanças nas estruturas de memória e no contexto de expectativas na memória. A moderna teoria linguística busca "capturar as generalizações significativas" no uso da linguagem. Isso tem levado quase que exclusivamente à busca dos padrões sintáticos. O *parsing* baseado em casos, entretanto, se preocupa mais com a caracterização de como o texto se refere a conceitos.

8.19.6 Conclusão

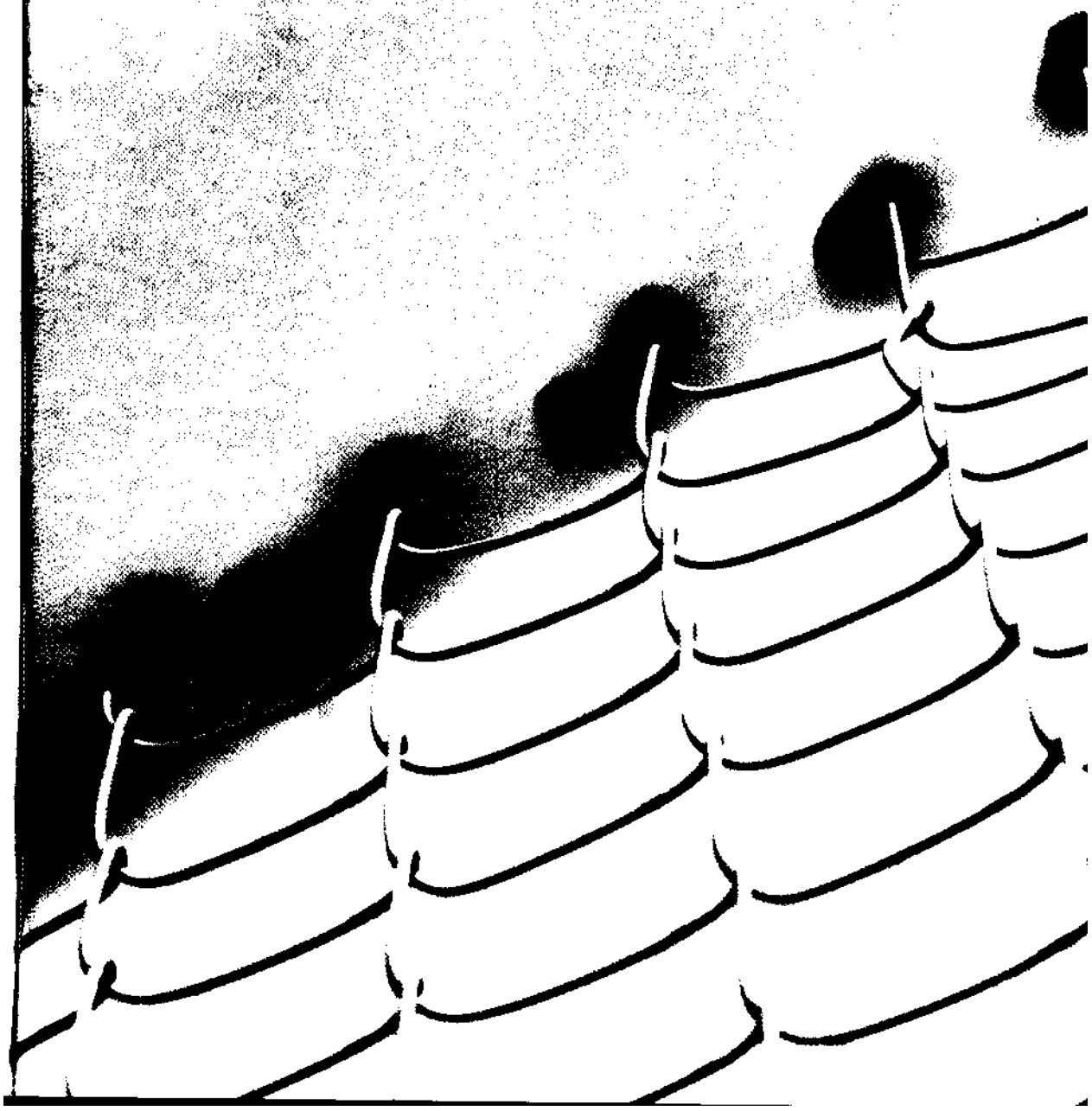
A faixa das organizações de sistemas constituída por princípios, casos e regras forma um *continuum* multifacetado no qual muitas opções diferentes podem ser consideradas (Pereira e Grosz, 1993). Em uma faceta, os princípios podem ser vistos como fornecedores do conhecimento inicial crucial na especificação do espaço de casos possíveis e representações apropriadas, adquiridas ou recuperadas. Os mecanismos baseados em casos podem ser usados como uma reserva, que entram em ação quando os princípios conhecidos são insuficientes para derivar a interpretação de uma situação particular ou para decidir entre interpretações alternativas compatíveis com os princípios. Em uma outra faceta, a informação derivada de caso pode ser altamente abstraída pelos projetistas de sistemas para as restrições específicas da linguagem tal como a ordem da palavra e sistemas flexionais ou representações e restrições específicas do domínio tais como aquelas que especificam as propriedades sintáticas, semânticas e de domínio de determinadas entradas lexicais. Em ainda uma outra faceta, as regras podem ser vistas como codificações orientadas computacionalmente de determinadas instâncias de princípios apropriados às tarefas ou situações particulares; como a computação direta a partir dos princípios é em geral muito difícil, as regras podem ser preferidas por motivos computacionais.

8.20 SISTEMAS TUTORES INTELIGENTES

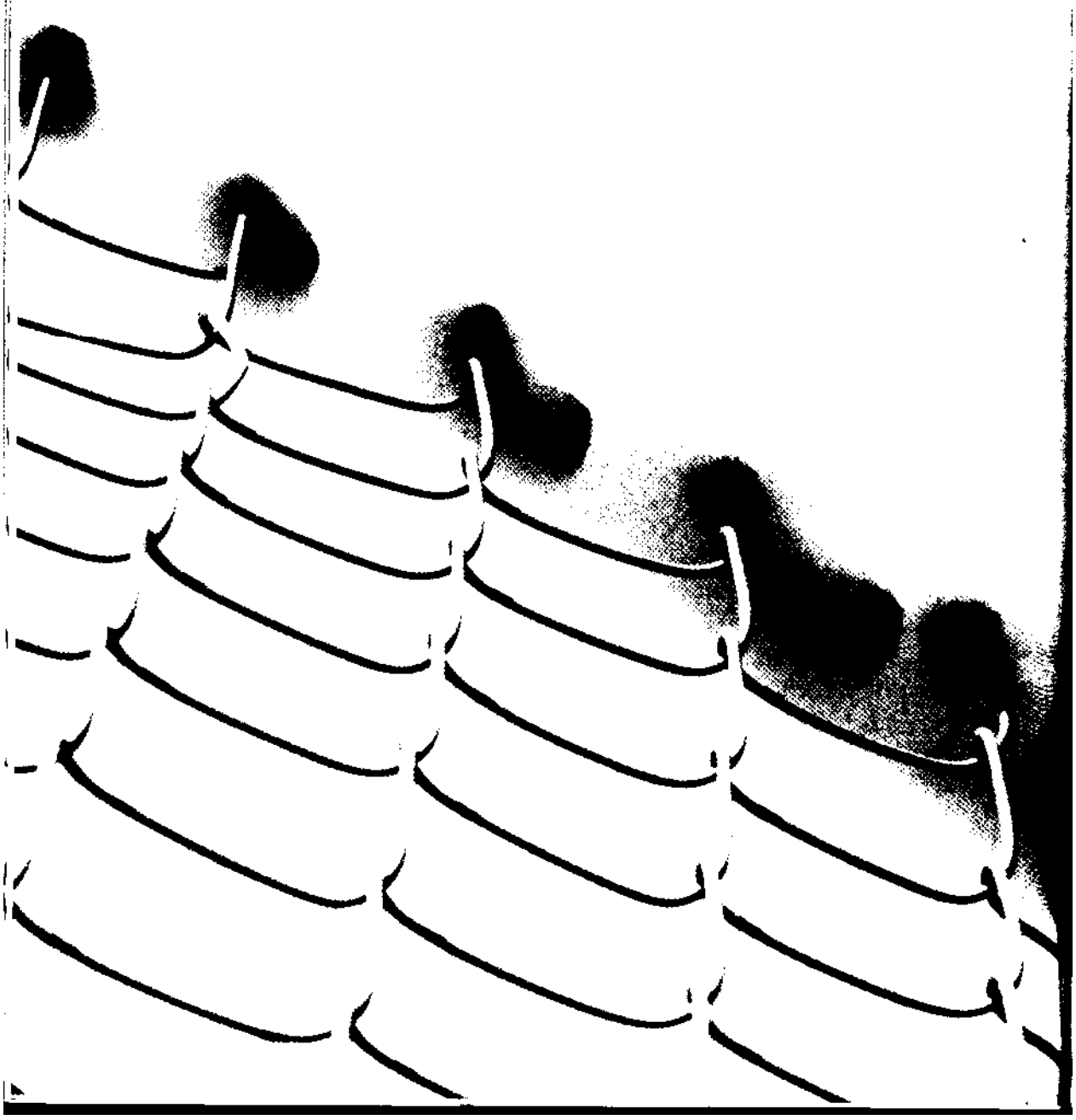
Sistemas Tutores Inteligentes (STI) (Brusilovsky, 1999; Costa, 2000; Freedman *et al.*, 2000; McArthur *et al.*, 1993; Shute e Psotka, 1996) são sistemas computacionais que unem técnicas da Inteligência Artificial (IA) a métodos educacionais, com a finalidade de propiciar o desenvolvimento de sistemas tutores adaptativos, seja para educação tradicional presencial, seja através de ambientes de educação a distância.

A nova geração dos STIs envolve o uso de um módulo de Processamento de Línguas Naturais (PLN) dentro do ambiente proposto, os chamados Sistemas Tutores Inteligentes Baseados em Linguagem Natural (STIBLN), uma nova abordagem aos STI clássicos. Espera-se com isso alcançar uma eficiência maior no processo de aprendizagem. A ideia desse módulo baseia-se na experiência bem-sucedida de sistemas como o AutoTutor (Universidade de Memphis) (Freedman *et al.*, 1998; Graesser *et al.*, 1999; Person e Graesser, 2000; Person *et al.*, 2001a; Person *et al.*, 2001b; Person *et al.*, 1999; Wiemer-Hastings *et al.*, 1998), o CIRCSIM (Instituto de Tecnologia de Illinois) (Freedman *et al.*, 1998; Glass, 2000; Zhou *et al.*, 1999a; Zhou *et al.*, 1999b) e o Atlas (Universidade de Pittsburgh) (Freedman *et al.*, 2000b; Rosé, 2000). Todos esses sistemas são STIBLNs. Esses tutores baseados em diálogo frequentemente se referem à comunicação escrita, ou seja, há a necessidade da elaboração de um parser para a língua usada. Esse *parser* deve fazer uso de uma base de conhecimento sobre o assunto objeto da disciplina que se deseja ensinar, que deve estar sempre atualizada em relação ao progresso da aprendizagem. Isso significa que o sistema deve ter a capacidade de adaptação ao usuário. Ferramentas da IA, com capacidade de aprendizagem e generalização, como as redes neurais artificiais, podem ser usadas (Olde *et al.*, 1999). Métodos estatísticos para extração de conhecimento linguístico também podem ser usados, como a Análise Semântica Latente (*Latent Semantic Analysis* — LSA) (Wiemer-Hastings, 1999; Wiemer-Hastings *et al.*, 1999). Conhecimento da língua e de teorias linguísticas é útil na implementação de parsers conexionistas psicolinguisticamente plausíveis (Miikkulainen, 1996; Miikkulainen, 1993; Rosa e Françozo, 2000; Rosa e Françozo, 1999).

CAPÍTULO 9



Redes Neurais Artificiais



9.1 INTRODUÇÃO

A evolução natural deu ao cérebro humano muitas características desejáveis que não estão presentes na máquina de von Neumann (os computadores atuais) tais como (Jain e Mao, 1996):

- Paralelismo massivo
- Representação e computação distribuídas
- Habilidade de aprendizado
- Habilidade de generalização
- Adaptabilidade
- Processamento de informação contextual inerente
- Tolerância a falhas
- Baixo consumo de energia

É desejável que os dispositivos baseados nas redes neurais biológicas possuam algumas dessas características. Cérebros e computadores digitais realizam tarefas bem diferentes e têm propriedades diferentes. Apesar disso, veja no quadro a seguir (de Jain e Mao, 1996) a comparação entre o computador de von Neumann e o sistema neural biológico.

	Computador de von Neumann	**Sistema neural biológico**
Processador	Complexo Alta velocidade Um ou poucos	Simples Baixa velocidade Um grande número
Memória	Separado do processador Localizado Não endereçável pelo conteúdo	Integrada ao processador Distribuída Endereçável pelo conteúdo
Computação	Centralizada Sequencial Programas armazenados	Distribuída Paralela Autoaprendizado
Confiabilidade	Muito vulnerável	Robusta
Especialidade	Manipulações numéricas e simbólicas	Problemas perceptuais
Ambiente operacional	Bem definido, bem restrito	Pobremente definido, irrestrito

A Inteligência Artificial trabalha basicamente com duas abordagens: a abordagem *simbólica*, baseada na lógica, e a abordagem *conexionista*, baseada nas redes neurais artificiais, ou seja, simulação do cérebro humano. Fazem parte da abordagem simbólica os sistemas baseados em regras, apresentados nos capítulos anteriores. É a abordagem conexionista que será tratada neste capítulo.

9.2 O NEURÔNIO BIOLÓGICO

O neurônio "clássico" (Figura 9.1A) tem muitos dendritos, usualmente ramificados, que recebem informação de outros neurônios, e um único axônio, que fornece como saída a informação processada, usualmente através da propagação de um *spike* ou "potencial de ação".[1] O axônio se divide em vários ramos, que fazem sinapses[2] com os dendritos e corpos celulares de outros neurônios.

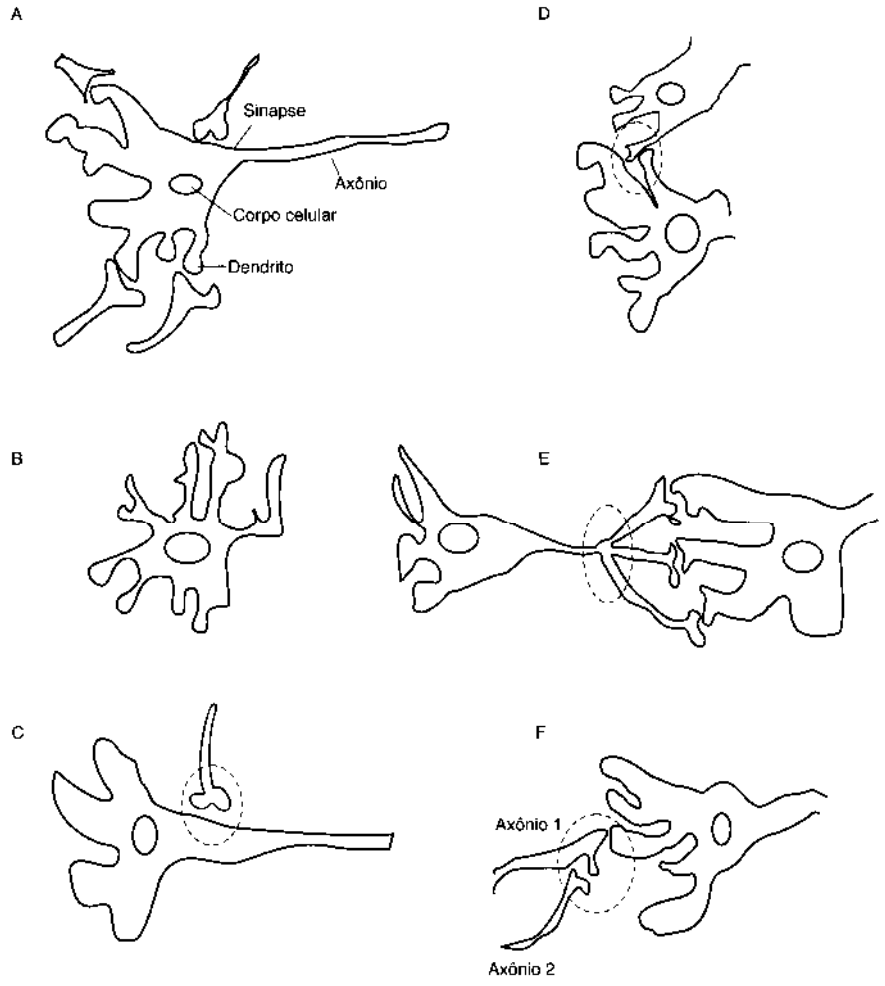


Figura 9.1 Diagramas esquematizados do neurônio clássico (A) e algumas de suas variantes (B-F) (Crick e Asanuma, 1986).

9.2.1 Variantes do neurônio "clássico"

Esse quadro simples se torna complicado nas seguintes situações:

[1] O potencial de ação é um impulso numa fibra nervosa que se move rapidamente ao longo do nervo.
[2] As junções entre as células nervosas são chamadas de sinapses. São os locais onde as células transferem sinais.

- um neurônio pode não ter axônios, mas apenas "processos" que servem tanto para receber como para transmitir informação (Figura 9.1B);
- axônios podem formar sinapses em outros axônios (Figura 9.1C);
- dendritos podem formar sinapses em outros dendritos (Figura 9.1D);
- um axônio pode não propagar um *spike*, mas produzir um potencial de gradiente (*graded potential*). Por causa da atenuação, deve-se esperar que essa forma de sinalização da informação não ocorra através de distâncias longas (Figura 9.1E). Esses potenciais de gradiente podem ocorrer em outro nível. Por exemplo, um terminal de axônio formando uma sinapse em uma dada célula pode receber uma sinapse (Figura 9.1F). A sinapse pré-sináptica pode exercer apenas uma mudança de potencial local que é portanto restrito àquele terminal de axônio.

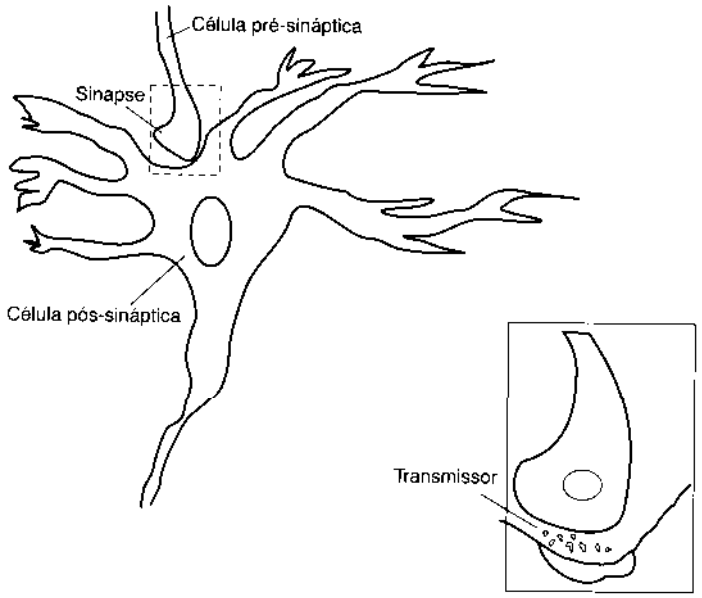


Figura 9.2 Na maior parte das sinapses, o terminal pré-sináptico libera uma substância química, o transmissor, em resposta a uma despolarização (Kuffler e Nicholls, 1984).

9.2.2 Sinapses: junções entre células nervosas

O tipo predominante de sinapse no cérebro do mamífero é a sinapse química, que opera através de liberação de uma substância transmissora do terminal pré-sináptico para o terminal pós-sináptico (Figura 9.2).

As vesículas usadas fundem-se com a membrana do terminal pré-sináptico, e novas vesículas são formadas da membrana nas margens do terminal.

9.2.2.1 As sinapses são químicas e não elétricas

Como já foi mencionado, a maior parte das sinapses que ocorrem no córtex cerebral é química e não elétrica. Os contatos sinápticos podem ser classificados morfologicamente em dois tipos básicos:

- Tipo I (Figura 9.3A): essas sinapses têm especializações de membrana assimétricas (a espessura da membrana é maior no lado pós-sináptico), e o processo pré-sináptico contém vesículas sinápticas redondas bastante grandes (50 nm), onde se acredita que existam pacotes de neurotransmissores.
- Tipo II (Figura 9.3B): essas sinapses têm especializações de membrana simétricas. As vesículas sinápticas são menores, e, com os fixativos usuais usados pela microscopia eletrônica, são frequentemente elipsoidais ou achatadas. (A forma das vesículas depende dos detalhes de fixação e nem sempre é um critério completamente confiável quando se comparam resultados relatados por diferentes pessoas.) A zona de contato é usualmente menor que da sinapse tipo I.

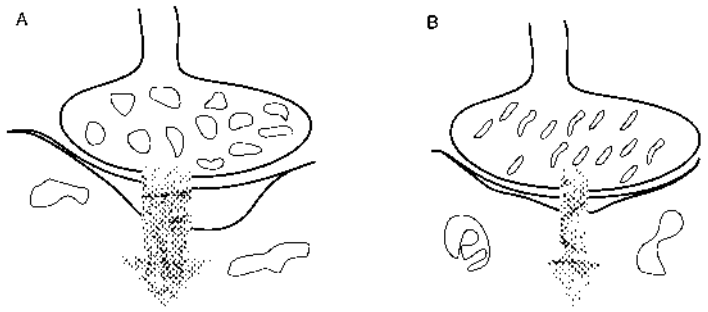


Figura 9.3 Diagramas idealizados das sinapses tipo I (A) e tipo II (B). Veja o texto para esclarecimentos (Crick e Asanuma, 1986).

9.2.2.2 As sinapses podem excitar ou inibir

A importância da classificação nos dois tipos morfológicos é que as sinapses do tipo I parecem ser excitatórias, ao passo que as sinapses do tipo II parecem ser inibitórias.[3]

Existe outro critério possível para determinar o caráter das sinapses: o transmissor que elas usam. Em geral, assume-se que um dado transmissor fará usualmente a mesma coisa em lugares diferentes, apesar de haver exceções, dependendo da natureza dos receptores pós-sinápticos.

9.2.2.3 Generalizações sobre sinapses

Vários métodos têm sido usados para identificar os neurotransmissores, mas cada técnica tem limitações. No momento, é difícil identificar os transmissores envolvidos e seus efeitos pós-sinápticos em muitas sinapses do sistema nervoso central. Pode-se fazer uma lista de tentativas de possíveis generalizações sobre sinapses:

- nenhum axônio faz sinapses tipo I em alguns locais enquanto faz tipo II em outros;
- nenhum axônio no cérebro de mamífero mostrou liberação de dois neurotransmissores diferentes não peptídeos (mas parece que muitos neurônios, incluindo neurônios corti-

[3] As células nervosas influenciam outras por (a) excitação, ou seja, elas produzem impulsos em outras células, e (b) inibição, ou seja, elas previnem a liberação de impulsos em outras células.

cais, podem liberar um transmissor "convencional" e um neuropeptídeo, ou, em alguns casos, dois ou mais neuropeptídeos);
- não existe evidência no cérebro de mamífero de que um mesmo axônio possa causar excitação e inibição em sinapses diferentes, mas isso é certamente possível, já que o efeito de um dado transmissor depende dos tipos dos receptores presentes e de seus canais de íon associados.

9.2.2.4 Peptídeos: moduladores da função sináptica

Ao longo dos últimos 10 anos, descobriu-se que existem muitos peptídeos distintos, de vários tipos e tamanhos, que podem agir como neurotransmissores. Há, no entanto, razões para suspeitar que os peptídeos são diferentes de muitos transmissores convencionais:
- peptídeos aparecem para *modular* a função sináptica ao invés de ativá-la;
- a ação de peptídeos, em poucos casos estudados, geralmente consiste em avançar vagarosamente e persistir por algum tempo, isto é, por segundos ou mesmo minutos, ao passo que os transmissores convencionais duram poucos milissegundos;
- em alguns casos foi mostrado que os peptídeos não agem onde foram liberados, mas a alguma distância. A difusão leva tempo. O tempo demorado de persistência seria compatível com os possíveis atrasos de tempo produzidos pela difusão;
- existem muitos exemplos agora conhecidos de um neurônio único produzindo, e presumivelmente liberando, mais de um neuropeptídeo.

9.2.2.5 Peptídeo: transmissor lento ou neuromodulador?

Foi mostrado que os peptídeos formam um segundo, mais lento, meio de comunicação entre neurônios, mais econômico do que usar neurônios extras para esse propósito.

Os peptídeos têm papel de modulação principalmente em sistemas neurais, nos quais o modo de comunicação é o endereçamento químico.

Como transmissores, os peptídeos agem em locais bem restritos, mesmo assim como um meio de condução muito lento, não sustentando as altas frequências dos impulsos. Como neuromoduladores da função sináptica, sua atividade é mais intensa. Os efeitos excitatórios da substância P (um peptídeo) são muito lentos no início e prolongados na duração (mais de um minuto) e por si sós não podem causar a despolarização[4] suficiente para excitar as células. O efeito, entretanto, é tornar os neurônios mais prontamente excitáveis por outras entradas excitatórias — um claro exemplo de *neuromodulação*.

9.3 O CÉREBRO COMO MODELO

A ideia de simular o cérebro já era o objetivo de muitos trabalhos iniciais em Inteligência Artificial. O cérebro era visto como uma *rede neural*, ou seja, um conjunto de nós, ou neurô-

[4] Despolarização é uma redução do potencial da membrana celular para zero mV, e o interior do neurônio torna-se mais positivo. A despolarização para um nível de potencial crítico, o limiar, causa o início de um impulso. No seu pico, o interior da célula torna-se positivo em relação ao seu exterior. Na maioria das sinapses, o terminal pré-sináptico libera uma substância química, o transmissor, em resposta a uma despolarização. Numa sinapse excitatória, o transmissor liberado pelo terminal pré-sináptico despolariza a célula pós-sináptica, fazendo com que o potencial de sua membrana atinja o limiar. Numa sinapse inibitória, o transmissor tende a manter o potencial da membrana da célula pós-sináptica abaixo do limiar.

nios, conectados por linhas de comunicação. Atualmente tem havido um crescente interesse no uso de modelos de redes neurais ou conexionistas. Modelos conexionistas são aplicáveis a vários problemas de ciência cognitiva, incluindo processamento de línguas naturais, processamento de fala e visão.

Num nível mais simples, o cérebro funciona da seguinte forma: neurônios ativam ou inibem o disparo de outros neurônios. Se um determinado neurônio dispara ou não depende das entradas inibitórias ou excitatórias de todos os neurônios conectados a ele.

9.3.1 O perceptron

O primeiro modelo matemático do neurônio foi o modelo proposto por McCulloch e Pitts, em 1943. Mais tarde, Rosenblatt (1957) criou o modelo do perceptron. Um perceptron modela um neurônio tomando uma soma ponderada de suas entradas e enviando a saída 1 se essa soma é maior que um determinado limiar (senão, envia 0). Veja a Figura 9.4.

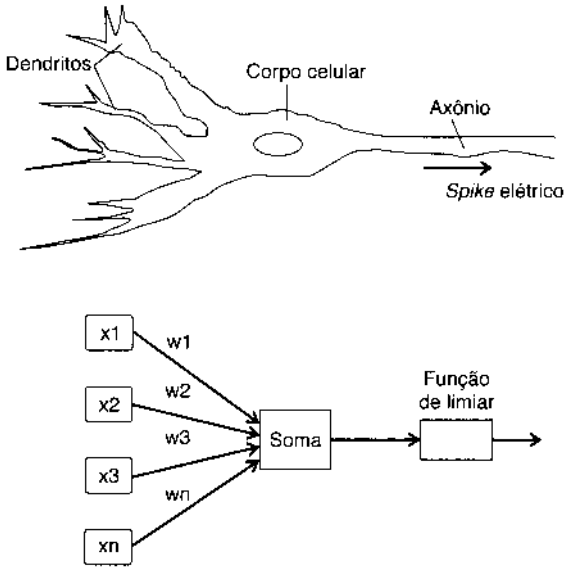


Figura 9.4 Um neurônio e um perceptron (Rich e Knight, 1994).

9.3.2 Paralelismo

Outra razão para se estudar modelos parecidos com o cérebro é seu paralelismo. Os *circuitos* do cérebro são mais lentos do que os de um computador. Para que o cérebro trabalhe o mais rápido possível — os psicólogos mostraram que se pode reconhecer objetos num segundo — muitos neurônios devem trabalhar em paralelo. Em contraste, muitos programas de Inteligência Artificial rodam muito lentamente, pois são simulados em um sistema uniprocessador.

Nos últimos anos, a computação paralela tem sido bastante explorada em ciência da computação. As redes neurais representam apenas uma linha de pesquisa em computação paralela. Basicamente, deve-se responder a duas questões fundamentais no projeto de um sistema de

computador paralelo: como conectar os processadores para propósito de comunicação e quanto de potência computacional e memória cada processador deve ter.

Os pesquisadores de redes neurais acreditam que seus modelos, por serem os mais fiéis ao cérebro conhecido, terão sucesso. Infelizmente, as redes neurais raramente têm sido construídas em hardware; normalmente elas são simuladas por software. Essas simulações são geralmente muito lentas, pois um processador tem que fazer o trabalho de muitos. Até que se construa hardware de processamento paralelo efetivo, os modelos conexionistas alcançarão soluções não muito eficientes para problemas de Inteligência Artificial.

9.3.3 Variedades de redes neurais

Muitos modelos de redes neurais devem alguma coisa aos perceptrons, mas são mais gerais. O modelo típico de rede neural consiste em um conjunto de nós, ou neurônios, e conexões. Cada nó tem a sua ativação, que geralmente é um número binário (1 significa presença do impulso de entrada e 0, a ausência). Cada conexão contém um número real, seu peso. Algumas unidade são conectadas à entrada e à saída. Os pesos representam a força de conexão entre dois neurônios (força sináptica).

Geralmente, a rede neural é um sistema dinâmico, movendo de um estado para o próximo. Como tal, ela tem uma regra matemática que rege esse movimento. Um número infinito de tais regras é possível. Entretanto, usualmente quer-se limitar os modelos a influenciar a ativação de um dado nó baseado apenas nas ativações dos nós conectados a ele e nos pesos das conexões a esses nós. Muitas críticas ao modelo conexionista vêm do fato de que essa abordagem é pobre biologicamente. Para viabilizar a implementação computacional, simplifica-se o modelo. Mas modelos alternativos estão sendo estudados (Rosa, 2001; Silva e Rosa, 2008).

As redes neurais não são explicitamente programadas como um computador convencional. Por melhor dizer, elas obedecem a leis, ou regras, como um sistema físico. Deve-se programar um computador convencional, mas uma rede neural simplesmente se conduz. Os projetistas de redes neurais veem isso como uma vantagem, pois provê um mecanismo por meio do qual a inteligência pode surgir da lei física.

Uma das mais simples dessas regras é a regra linear. Computa-se a ativação de um dado nó como a soma dos produtos do peso de cada nó ao qual está conectado e a força dessa conexão. Essa regra é frequentemente limitada: valores que passam de um certo limiar são cortados, para evitar os valores de ativação grandes. Existem muitas variantes das regras lineares.

Outra regra, sugerida por D. O. Hebb (1949), reforça a conexão entre dois nós que são altamente ativados ao mesmo tempo. Algumas versões da regra de aprendizado hebbiana permitem entradas, que ensinam, para influenciar a mudança de peso. Este tipo de regra é uma formalização da psicologia associacionista, que assegura que associações são acumuladas entre coisas que ocorrem juntas.

9.3.4 Aprendizado competitivo

O aprendizado é, talvez, o fenômeno mais importante em psicologia. Os primeiros pesquisadores em redes neurais eram ansiosos para mostrar como as redes podiam aprender padrões

de entrada apresentados a elas — ou seja, como elas podiam vir a perceber esses padrões por si mesmas.

Um dos métodos que vários pesquisadores vêm planejando através dos anos é o aprendizado competitivo. Esse método tem um nível abaixo, de unidades de entrada que contêm o padrão a ser entrado ao sistema. O nível acima das unidades de entrada consiste em *grupos* (*clusters*) de unidades. Cada unidade num grupo compete com as outras unidades no grupo pelo direito de reconhecer um padrão de entrada. Depois de um período de aprendizado, cada unidade num grupo reconhece um subconjunto dos padrões apresentados a ela. Portanto, cada grupo representa uma classificação, ou grupo, de padrões de entrada.

No aprendizado competitivo, cada unidade em cada grupo é conectada a todas as unidades de entrada. Os pesos das conexões são inicialmente colocados em valores aleatórios. Os pesos aleatórios fazem com que certas unidades nos grupos comecem a responder mais a determinados padrões de entrada, pois os pesos das conexões a essas unidades de entrada são mais fortes para alguns do que para outros.

No decorrer do aprendizado, os pesos mudam. Como determinadas unidades no grupo se tornam sensíveis a determinadas unidades no padrão de entrada, os pesos conectando os pares associados de unidades aumenta, à custa de pares não associados de unidades. Unidades diferentes no mesmo grupo se inibem, de tal forma que apenas uma unidade num grupo "ganha" o direito de reconhecer um dado padrão.

Assim, com o tempo, unidades diferentes num grupo "reconhecem" propriedades diferentes de padrões de entrada. Por exemplo, um grupo de duas unidades pode separar todos os padrões de entrada naqueles que têm a maioria das suas unidades altamente ativadas e naqueles que têm a maioria desligada. Os grupos maiores fariam mais classificações discriminatórias.

9.3.5 Representações distribuídas

Uma importante característica de muitos modelos de redes neurais é sua natureza distribuída. Uma rede semântica padrão, como aquelas usadas nos primeiros esquemas de representação do conhecimento, consiste em um conjunto de nós conectados de alguma forma. Cada nó representa uma única palavra ou conceito. Se a rede estiver "pensando" na palavra *gato*, o nó para *gato* é ativado, e todos os outros nós não. Essa é uma representação local.

Em contraste, numa rede distribuída, os nós não têm um único significado; ou seja, um conceito individual é representado por um padrão por todos os nós. Por exemplo (Zeidenberg, 1987), se há 10 nós, ativando os nós 1, 3, 4 e 7 pode-se representar o conceito *gorila*, enquanto ativando os nós 2, 4, 5 e 7 pode-se representar o conceito próximo *chimpanzé*. Conceitos que são próximos têm representações similares.

Uma rede de processamento paralelo distribuído, uma rede neural que usa representação distribuída, oferece a vantagem de generalização automática. Se se quer representar o conceito "gorilas são cabeludos", reforça-se a conexão entre todos os nós que compõem o conceito *gorila* e todos os nós que compõem o conceito *cabeludo*. Como resultado, desde que a maioria dos nós em *gorila* é também usada em *chimpanzé*, uma associação é também feita entre *chimpanzé* e *cabeludo*. É assim que a generalização automática trabalha. Numa representação local, em que *gorila* e *chimpanzé* são representados por nós separados, uma conexão entre *gorila* e *cabeludo* não implicaria uma conexão entre *chimpanzé* e *cabeludo*.

Outra vantagem de uma representação distribuída é sua insensibilidade a danos. Numa representação local, se o sistema perde o nó que representa *avó*, ele perde seu conceito de avó.

Em uma representação distribuída, para perder um conceito devem-se perder todos os nós que o representa. Se se perde apenas um ou dois nós, o conceito pode se degradar, mas ainda está lá. Isso é mais próximo ao tipo de memória perdida observada em idosos.

9.3.6 Máquinas de Boltzmann

Uma importante classe de redes neurais simula o comportamento de sistemas físicos. Os sistemas físicos têm uma tendência a se mover para dentro de estados de energia potencial mínima. Um exemplo simples é uma bola rolando num vale entre duas colinas. No alto da colina, a energia potencial é alta; no vale, é baixa.

Esse processo é chamado de relaxação. John Hopfield (1982) mostrou que uma certa regra evolucionária simples para uma rede neural levará à relaxação. Sistemas como os de Hopfield, que remontam aos sistemas termodinâmicos, são chamados de máquinas de Boltzmann. As máquinas de Boltzmann são muito usadas em várias aplicações de redes neurais.

9.3.7 Esquemas

Uma crítica aos modelos de redes neurais é que eles não são tão flexíveis na representação do conhecimento quanto os métodos padrões. Os métodos padrões incluem a rede semântica local.

Os psicólogos cognitivos, notadamente Jean Piaget, usam o conceito de esquema (*schema*). Um esquema é um espelho — na mente — de uma situação real. Como crianças, e como adultos, aprendem-se novas associações e relações entre objetos que são integrados ao esquema de cada um.

Não é imediatamente claro como um modelo de rede neural pode considerar o conhecimento representado em um esquema; entretanto, Rumelhart, Paul Smolensky, McClelland e Geoffrey Hinton mostraram que é possível (McClelland e Rumelhart, 1986).

9.3.8 Hierarquias cognitivas

Frequentemente, modelos de redes neurais são ordenados em hierarquias. Muitos níveis existem numa hierarquia, cada um composto de um conjunto de unidades. Tipicamente, as unidades que recebem a entrada estão no fundo do sistema, e as unidades que dão saída estão no alto. Num sistema *bottom-up*, as unidades em cada nível são conectadas a outras unidades no seu próprio nível e influenciam as unidades em níveis acima deles. Num sistema *top-down*, as unidades novamente se conectam a unidades no seu próprio nível, mas influenciam as unidades em níveis abaixo.

Top-down e *bottom-up* são conceitos familiares em ciência cognitiva. Por exemplo, na percepção de sentença, esses termos referem a como elementos linguísticos de tamanhos diferentes, o fonema (som), morfema (elemento palavra), palavra, sintagma e sentença, interagem uns com os outros.

9.3.9 Uma rede de leitura paralela

Um problema na criação de uma rede de leitura é que as pessoas tendem a ler mais de uma palavra de cada vez. Como uma rede simples lê apenas uma palavra, não funciona. Como solução, McClelland (McClelland e Rumelhart, 1986) propõe cópias duplicadas de redes. Redes de reconhecimento de palavras individuais duplicadas teriam conexões programáveis ao invés de conexões fixas por hardware (*hardwired*) entre letras e palavras.

9.3.10 Processamento de sentenças

Um importante aspecto do entendimento de sentença envolve determinar os vários casos que as partes diferentes de uma sentença têm. Por exemplo, considere as seguintes sentenças:

O macaco morreu.
O macaco quebrou.

Na primeira sentença, *macaco* é um animal, pois *morrer* é uma característica dos seres vivos; na segunda, *macaco* é uma ferramenta de trocar pneus, pois um animal não pode "quebrar". De alguma forma, o modelo deve discernir seus casos diferentes.

McClelland e Kawamoto (1986) desenvolveram um sistema conexionista para fazer essa atribuição de casos. Palavras são descritas por *microcaracterísticas semânticas* — dimensões básicas que descrevem muitos objetos e ações. Por exemplo, duas das microcaracterísticas que descrevem substantivos são "humano" e "leveza", que têm os valores "humano, não humano", e "leve, pesado", respectivamente. As palavras não são representadas diretamente nas redes do sistema, mas em termos das ativações de unidades representando microcaracterísticas. Versões desse programa para a língua portuguesa foram desenvolvidas por Rosa e Françozo (1999).

O modelo tem um grupo de unidades para cada um dos casos principais que substantivos diferentes podem ter em uma ação. Esses casos são Agente (ator), Paciente (agido sobre), Instrumento (coisa usada) e Modificador (palavra adverbial ou cláusula). Por exemplo, a sentença "O homem comeu o sanduíche" ativaria as microcaracterísticas de "comeu" e "homem" no conjunto das unidades que correspondem ao Agente; isso representa o fato de que o Agente para o verbo "comeu" é "homem".

O sistema é treinado em uma série de sentenças. As atribuições do caso correto para as sentenças de treinamento são mostradas ao sistema. Essas atribuições correspondem às ativações de nós particulares. O sistema ajusta as conexões entre esses nós de tal forma que eles se reforcem mutuamente.

Depois de ser treinado com um número suficiente de sentenças, o sistema pode fazer atribuições de caso correto para novas sentenças. Ele ainda pode fazer atribuições de caso correto para sentenças com alguma ambiguidade sintática. Por exemplo, na sentença "O homem abateu o garoto com a maleta", o sistema considera que "maleta" é o Instrumento de "abateu" em vez de pertencer ao "garoto", desde que "maleta" tenha microcaracterística que indique que ela é um instrumento.

O sistema também manipula bem vários outros problemas, e geralmente faz um bom trabalho em atribuição de casos.

9.3.11 O futuro

As redes neurais são boas para várias tarefas de processamento de linguagem natural, incluindo reconhecimento de letra, leitura e entendimento de sentença. Elas também são úteis no armazenamento de conhecimento em esquemas e em recuperar itens da memória. Elas não são milagrosas, mas trazem uma direção para a Inteligência Artificial e a Psicologia Cognitiva, forte e biologicamente plausível, para muitos problemas importantes.

No futuro, um modelo conexionista será provavelmente construído do processo de entendimento de língua natural, desde que, como os psicólogos têm mostrado, envolva conhecimento integrado de muitos domínios, incluindo fonética, morfologia, sintaxe e semântica. Modelos conexionistas são particularmente bons na integração desses tipos de conhecimento.

9.4 ALGORITMOS CONEXIONISTAS

Uma vez identificado o problema que se queira solucionar através da abordagem conexionista, deve-se construir a rede neural. Ou seja, montar a arquitetura da rede: para uma rede de três camadas, quantos neurônios eu devo ter na entrada da rede (que corresponde, normalmente, ao número de bits que representa o meu padrão), quantos eu devo ter na saída (que corresponde, normalmente, à quantidade de bits do meu padrão de saída) e, o mais difícil, o número de neurônios na camada escondida. Os neurônios da camada escondida normalmente não são "calculados" e devem ser tentados empiricamente.

Depois de construída a rede neural artificial, deve-se escolher um *algoritmo conexionista* para "treinar" a rede (fase de aprendizado). O treinamento da rede normalmente é demorado, pois requer muitos "ciclos", ou seja, deve-se mostrar à rede várias vezes tudo que se deseja que ela aprenda. Depois do treinamento, a rede neural deve ser capaz de, numa única propagação (único ciclo), reconhecer o padrão no qual ela foi ensinada (fase de reconhecimento).

Os algoritmos de redes neurais em geral se dividem em dois tipos básicos: os algoritmos *supervisionados*, ou seja, quando a saída desejada da rede durante o treinamento é fornecida para comparação, e os *não supervisionados*, quando a rede se conduz por si só, ou seja, não há um supervisor que verifique as suas saídas.

Entre os algoritmos supervisionados mais conhecidos está o algoritmo *backpropagation* (McClelland e Rumelhart, 1986). Nesse algoritmo, a cada ciclo o padrão de entrada é propagado pela rede, e na saída ele é comparado com a saída desejada (supervisor). Caso haja erros, estes são corrigidos gradativamente, através das mudanças de pesos dos neurônios que se conectam à saída errada (propagação de volta dos erros).

Entre os algoritmos não supervisionados mais representativos está o *competitive learning*, ou aprendizado competitivo, já discutido anteriormente.

9.4.1 Redes *backpropagation*

A habilidade para treinar redes com várias camadas é um passo importante na direção da construção de máquinas inteligentes a partir de componentes parecidos com os neurônios. A meta é pegar uma massa de elementos processadores, que simulam a célula nervosa, e ensiná-la a realizar tarefas úteis. É desejável que ela seja rápida e resistente a danos. É desejável que generalize a partir das entradas que vê.

O que uma rede multicamadas pode calcular? A resposta é: qualquer coisa. Dado um conjunto de entradas, pode-se usar unidades de limiar como simples portas AND, OR e NOT, arranjando apropriadamente o limiar e os pesos de conexão. Sabe-se que é possível construir qualquer circuito combinacional a partir dessas unidades lógicas básicas.

O maior problema é o aprendizado. O sistema de representação de conhecimento empregado pelas redes neurais é um tanto obscuro: as redes devem aprender suas próprias representações porque programá-las à mão é impossível. Uma propriedade das redes neurais diz que tudo que elas podem calcular elas podem aprender a calcular.

É útil tratar primeiro com uma subclasse de redes multicamadas, chamadas de redes *totalmente conectadas*, divididas em *camadas* e alimentadas *para a frente*. Um exemplo de tal rede é mostrado na Figura 9.5. Nessa figura, x_i (1 <= i <= A), h_i (1 <= i <= B) e o_i (1 <= i <= C) representam os níveis de unidade de ativação das unidades de entrada, escondida e saída, respectivamente. Os pesos das conexões entre as camadas de entrada e escondida são denotados por $w1_{ij}$, enquanto os pesos das conexões entre as camadas escondida e de saída são denotados por $w2_{ij}$. Essa rede tem três camadas, ainda que seja possível, e algumas vezes útil, ter mais de três. Cada unidade numa camada é conectada a toda unidade da próxima camada na direção para a frente, ou seja, cada unidade da camada de entrada é conectada a todas as unidades da camada escondida, nessa direção. As ativações fluem a partir da camada de entrada através da camada escondida para a camada de saída. O conhecimento da rede é codificado nos pesos das conexões entre as unidades. Os níveis de ativação das unidades da camada de saída determinam a saída da rede.

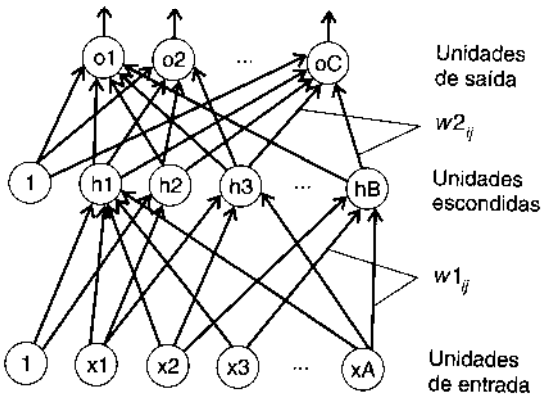


Figura 9.5 Uma rede multicamada (Rich e Knight, 1994).

A existência da camada escondida permite que a rede desenvolva representações internas. O comportamento dessas unidades escondidas é automaticamente aprendido, não é pré-programado.

A maior propriedade dos sistemas conexionistas é que a rede neural não aprende apenas a classificar as entradas nas quais ela é treinada, mas também a *generalizar* e ser capaz de classificar entradas nunca vistas.

Tudo que as redes neurais parecem capazes de fazer é classificar. Os graves problemas da Inteligência Artificial, como planejamento, análise de linguagem natural e prova de teorema, não são simplesmente tarefas de classificação, então como as redes neurais resolvem esses

problemas? Resolver os problemas de classificação é, no presente, o que as redes neurais fazem melhor. Mas pesquisa-se a aplicação a outros problemas, como processamento de línguas naturais, por exemplo.

Uma limitação das redes atuais é como elas lidam com fenômenos que envolvem o tempo. Essa limitação é resolvida, de certa forma, pelas redes recorrentes, mas os problemas ainda são muitos.

A unidade na rede *backpropagation* requer uma função de ativação baseada numa sigmoide (ou forma de *S*) que é contínua e diferenciável. Uma unidade soma suas entradas ponderadas e produz como saída um valor real entre 0 e 1. Seja *soma* a soma ponderada das entradas de uma unidade. A equação para a saída da unidade é dada por:

$$\text{saída} = \frac{1}{1 + e^{-\text{soma}}}$$

Uma rede *backpropagation* tipicamente se inicia com um conjunto de pesos aleatórios. A rede ajusta seus pesos cada vez que vê um par entrada-saída. Cada par requer dois estágios: um passo para a frente e um passo para trás. O passo para a frente envolve a apresentação de uma amostra de entrada à rede, e as ativações propagam-se até alcançarem a camada de saída. Durante o passo para trás, a saída real da rede (do passo para a frente) é comparada com a saída desejada, e as estimativas de erro são calculadas para as unidades de saída. Os pesos conectados às unidades de saída podem ser ajustados a fim de reduzir esses erros. Podem-se usar as estimativas de erro das unidades de saída para derivar as estimativas de erro para as unidades das camadas escondidas. Finalmente, os erros são propagados de volta às conexões que tiveram origem nas unidades de entrada.

O algoritmo *backpropagation* geralmente atualiza seus pesos depois de ver cada par entrada-saída. Depois de vistos todos os pares entrada-saída (e muitas vezes ajustados seus pesos), diz-se que uma *época* se completou. O treinamento de rede *backpropagation* usualmente requer muitas épocas.

9.4.1.1 O algoritmo *Backpropagation*

O algoritmo seguinte é baseado na estrutura básica da Figura 9.4 (Rich e Knight, 1994).

Algoritmo 9.1 *Backpropagation*

Dado: Um conjunto de pares de vetores de entrada-saída.

Calcular: Um conjunto de pesos para uma rede de três camadas que mapeia entradas nas saídas correspondentes.

1. Seja A o número de unidades na camada de entrada, como determinado pelo comprimento dos vetores de treinamento de entrada. Seja C o número de unidades na camada de saída. Agora escolher B, o número de unidades na camada escondida. Como mostrado na Figura 9.5, as camadas de entrada e escondida têm uma unidade extra usada para limiar; portanto, as unidades nessas camadas serão indexadas pela faixa $(0, ..., A)$ e $(0, ..., B)$. Denotam-se os níveis de ativação das unidades na camada de entrada por x_i, na camada escondida por h_j e na camada de saída por o_j. Os pesos

conectando a camada de entrada à camada escondida são denotados por $w1_{ij}$, em que o índice i indexa as unidades de entrada e o índice j indexa as unidades escondidas. Da mesma forma, os pesos conectando a camada escondida à camada de saída são denotados por $w2_{ij}$, com i indexando as unidades escondidas e j indexando as unidades de saída.

2. Iniciar os pesos da rede. A cada peso deve ser atribuído um valor aleatório entre 0.1 e 0.1.
   $w1_{ij}$ = random(-0.1,0.1) para todo $i = 0, ...A, j = 1, ..., B$
   $w2_{ij}$ = random(-0.1,0.1) para todo $i = 0, ...B, j = 1, ..., C$

3. Iniciar as ativações para as unidades de limiar. Os valores dessas unidades nunca devem mudar.
   $x_0 = 1.0$
   $h_0 = 1.0$

4. Escolher um par entrada-saída. Suponha que o vetor de entrada seja x_i e o vetor de saída desejada seja y_j. Atribuir níveis de ativação às unidades de entrada.

5. Propagar as ativações a partir das unidades na camada de entrada para as unidades na camada escondida usando a função de ativação sigmoide:

$$h_j = \frac{1}{1+e^{-soma}} \quad \text{para todo } j = 1, ..., B$$

em que $soma = \Sigma^{A}_{i=0} w1_{ij} x_i$. Note que i varia de 0 a A. $w1_{0j}$ é o peso do limiar para a unidade escondida j (sua propensão a *disparar*,[5] a despeito de suas entradas). x_0 é sempre 1.0.

6. Propagar as ativações a partir das unidades na camada escondida para as unidades na camada de saída.

$$o_j = \frac{1}{1+e^{-soma}} \quad \text{para todo } j = 1, ..., C$$

em que $soma = \Sigma^{B}_{i=0} w2_{ij} h_i$. Novamente, o peso de limiar $w2_{0j}$ para a unidade de saída j traz uma contribuição à soma ponderada. h_0 é sempre 1.0.

7. Calcular os erros[6] das unidades na camada de saída, denotado por $\delta 2_j$. Os erros são baseados na saída real da rede (o_j) e na saída desejada (y_j).
   $\delta 2_j = o_j(1 - o_j)(y_j - o_j)$ para todo $j = 1, ..., C$

8. Calcular os erros das unidades na camada escondida, denotados por $\delta 1_j$.
   $\delta 1_j = h_j(1 - h_j) \Sigma^{C}_{i=1} \delta 2_j \times w2_{ij}$ para todo $j = 1, ..., B$

[5] Disparar é tornar-se igual a 1.0.
[6] A fórmula do erro é relacionada à derivada da função de ativação.

9. Ajustar os pesos entre a camada escondida e a camada de saída. A taxa de aprendizado é denotada por η. Um valor razoável para η é 0.35.
   $\Delta w2_{ij} = \eta \times \delta 2_j \times h_i$ para todo $i = 0, \ldots B, j = 1, \ldots, C$

10. Ajustar os pesos entre a camada de entrada e a camada escondida.
    $\Delta w1_{ij} = \eta \times \delta 1_j \times x_i$ para todo $i = 0, \ldots A, j = 1, \ldots, B$

11. Ir para o passo 4 e repetir. Quando todos os pares de entrada-saída tiverem sido apresentados à rede, uma época se completou. Repetir os passos 4 a 10 para quantas épocas desejar.

O algoritmo pode ser generalizado para redes com mais de três camadas.[7] A velocidade do aprendizado pode ser aumentada alterando os passos de modificação de pesos 9 e 10, com a inclusão de um termo α. As fórmulas de atualização de pesos ficam:

$$\Delta w2_{ij}(t+1) = \eta \times \delta 2_j \times h_i + \alpha \Delta w2_{ij}(t)$$

$$\Delta w1_{ij}(t+1) = \eta \times \delta 1_j \times x_i + \alpha \Delta w1_{ij}(t)$$

em que h_i, x_i, $\delta 1_j$ e $\delta 2_j$ são medidos no tempo t + 1. $\Delta w_{ij}(t)$ é a mudança que o peso experimenta durante o passo para a frente-para trás anterior. Se α é colocado em 0.9, a velocidade de aprendizado aumenta.[8]

Como a função de ativação tem a forma sigmoide, pesos infinitos seriam necessários para as saídas reais da rede alcançarem 0.0 e 1.0; portanto, as saídas desejadas (os y_j dos passos 4 e 7 anteriormente) são usualmente dadas como 0.1 e 0.9. A sigmoide é útil para a rede *backpropagation*, pois a derivação da regra de atualização do peso requer que a função de ativação seja contínua e diferenciável.

9.4.1.2 Generalização

Se todas as entradas e saídas possíveis são mostradas a uma rede *backpropagation*, a rede achará (provavelmente de modo acidental) um conjunto de pesos que mapeia as entradas nas saídas. Para muitos problemas de Inteligência Artificial, entretanto, é impossível fornecer todas as entradas possíveis. Para resolver esse problema, a rede *backpropagation* é boa no mecanismo de generalização. Se se trabalha num domínio em que entradas similares são mapeadas em saídas similares, a rede *backpropagation* irá interpolar quando forem fornecidas entradas que a rede nunca viu antes.

9.4.2 Redes recorrentes

Uma deficiência clara nos modelos de redes neurais comparados aos modelos simbólicos é a dificuldade que eles têm em lidar com tarefas temporais em Inteligência Artificial tais como

[7] Uma rede com três camadas (uma única camada escondida) pode calcular qualquer função que uma rede com muitas camadas escondidas pode calcular. Entretanto, o aprendizado é, às vezes, mais rápido com múltiplas camadas escondidas (Rich e Knight, 1994).

[8] Empiricamente, os melhores resultados acontecem quando α é zero para os primeiros passos de treinamento, aumentando seu valor gradativamente até 0.9 durante o treinamento, segundo Rich e Knight (1994).

planejamento e análise de língua natural. As redes recorrentes, ou redes com ciclos, são uma tentativa de corrigir essa situação.

Considere a tentativa de ensinar uma rede a arremessar uma bola de basquete à cesta (Rich e Knight, 1994). Pode-se apresentar a rede como uma situação de entrada (distância e altura da cesta, posição inicial dos músculos), mas necessita-se de mais do que um simples vetor de saída. Necessita-se de uma série de vetores de saída: primeiro mova os músculos dessa forma, depois dessa forma etc. A *rede de Jordan* faz algo parecido com isso. É mostrada na Figura 9.6. As *unidades de plano* da rede permanecem constantes. Elas correspondem a uma instrução como "arremessar uma bola à cesta". As *unidades de estado* codificam o estado corrente da rede. As *unidades de saída* simultaneamente dão comandos (por exemplo, movimente o braço x para a posição y) e atualiza as unidades de estado. A rede nunca se estabiliza, ou seja, nunca alcança um estado estável; ao invés disso, ela muda a cada passo de tempo.

As redes recorrentes podem ser treinadas com o algoritmo *backpropagation*. A cada passo, comparam-se as ativações das unidades de saída com as ativações desejadas e os erros são propagados de volta através da rede. Quando o treinamento está completo, a rede ainda é capaz de realizar uma sequência de ações. Características de *backpropagation*, tal como a generalização automática, também ocorrem nas redes recorrentes. Entretanto, poucas modificações são úteis. Primeiro, deseja-se que as unidades de estados mudem suavemente. A suavidade pode ser implementada como uma mudança na regra de atualização de peso; essencialmente, o *erro* de uma saída torna-se uma combinação do erro real e da magnitude da mudança nas unidades de estado. O reforço da restrição da suavidade torna-se muito importante no aprendizado rápido, já que ele remove muitas das opções de manipulação de peso disponíveis no *backpropagation*.

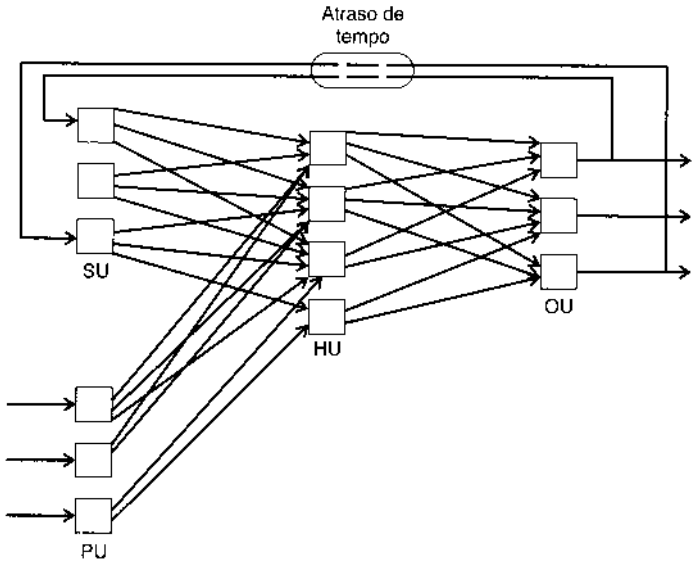


Figura 9.6 Uma rede de Jordan, em que SU = unidades de estado, PU = unidades de plano, HU = unidades escondidas e OU = unidades de saída.

Um problema maior nos sistemas de aprendizado supervisionado ocorre na correção do comportamento da rede. Se dados de treinamento suficientes podem ser coletados, en-

tão saídas-alvo podem ser providas para muitos vetores de entrada. Entretanto, as redes recorrentes têm problemas de treinamento especiais, por causa da dificuldade de especificar completamente uma série de saídas-alvo. No arremesso de bolas de basquete, por exemplo, a retroalimentação vem do mundo externo (isto é, onde a bola cai), não de um professor mostrando como mover cada músculo. Para contornar essa dificuldade, pode-se aprender um *modelo mental*, um mapeamento que relaciona as saídas da rede aos eventos no mundo. Com tal modelo, uma vez conhecido, o sistema proposto pode aprender tarefas sequenciais pela propagação de volta (*backpropagation*) dos erros que ele vê no mundo real. Então isso é necessário para aprender duas coisas diferentes: o relacionamento entre o plano e a saída da rede e entre a saída da rede e o mundo real.

As redes desse tipo são essencialmente iguais à da Figura 9.6, exceto pela adição de mais duas camadas: uma outra camada escondida e uma camada representando os resultados como visto no mundo. Primeiro, a última porção mencionada da rede é treinada (usando *backpropagation*) em vários pares de saídas e alvos até que a rede consiga saber como suas saídas afetam o mundo real. Depois os pesos brutos são estabelecidos, e a rede inteira é treinada usando retroalimentação do mundo real até que ela seja capaz de funcionar bem.

Outro tipo de rede recorrente é descrito por Elman (1990). Nesse modelo, os níveis de ativação são explicitamente copiados das unidades escondidas para as unidades de estado. As redes desse tipo têm sido usadas em várias aplicações, incluindo análise de língua natural.

9.4.2.1 A representação do tempo

A questão de como representar o tempo em modelos conexionistas é muito importante. Uma abordagem é representar o tempo implicitamente pelos seus efeitos no processamento ao invés de explicitamente (como numa representação espacial).

O tempo é muito importante em cognição. Está intrinsecamente ligado a muitos comportamentos (como na língua natural) que se expressam como sequências temporais. A questão de como representar o tempo parece ser um problema unicamente dos modelos de processamento paralelo, mas mesmo em sistemas tradicionais (seriais) a representação da ordem serial e a interação de uma entrada serial ou saída com níveis mais altos de representação apresentam desafios. Os linguistas teóricos têm tentado se preocupar menos com a representação de processamento de aspectos temporais de discursos (assumindo, por exemplo, que toda a informação num discurso é tida simultaneamente numa árvore sintática); mas a pesquisa na análise de língua natural sugere que o problema não é trivialmente resolvido. Portanto, o que é um dos fatos mais elementares sobre a atividade humana — que tem extensão temporal — é algumas vezes ignorado e é frequentemente problemático.

Nos modelos de processamento paralelo distribuído, o processamento de entradas sequenciais é completado de muitas formas. A solução mais comum é "paralelizar o tempo", dando a ele uma representação espacial. Entretanto, existem problemas com essa abordagem, e ela não é mais considerada uma boa solução. Uma abordagem mais interessante seria representar o tempo implicitamente, isto é, representa-se o tempo pelo efeito que ele tem no processamento e não como uma dimensão adicional da entrada (Elman). Isso significa dar ao sistema de processamento propriedades dinâmicas que são respostas às sequências temporais. Em resumo, à rede deve ser dada memória.

Essa abordagem pode ser modificada da seguinte forma. Suponha uma rede (mostrada na Figura 9.7.) aumentada no nível de entrada por unidades adicionais chamadas de Unidades de Contexto. Essas unidades também estão "escondidas", no sentido de que elas interagem exclusivamente com outros nós internos da rede e não com o mundo externo.

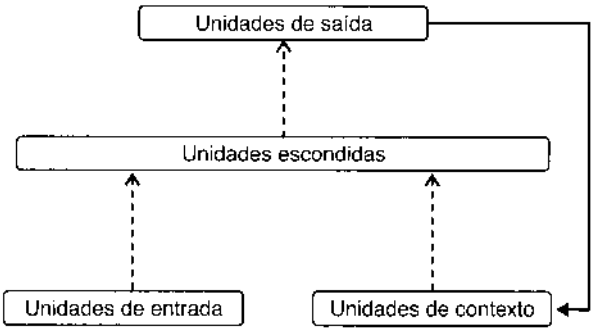


Figura 9.7 Uma rede recorrente simples na qual as ativações são copiadas da camada de saída para a camada de contexto na base um por um, com pesos fixos em 1.0. As linhas pontilhadas representam conexões de treinamento.

Imagine que existam uma entrada sequencial a ser processada e algum relógio que controle a apresentação da entrada à rede. O processamento então consistiria na seguinte sequência de eventos. No tempo t, as unidades de entrada recebem a primeira entrada da sequência. Cada unidade deve ter um valor escalar simples ou um vetor, dependendo da natureza do problema. As unidades de contexto são inicialmente colocadas em 0.5.[9] As unidades de entrada e de contexto ativam, ambas, as unidades escondidas; as unidades escondidas, então, alimentam para frente para ativar as unidades de saída. As unidades de saída também retroalimentam para ativar as unidades de contexto. Isso constitui a ativação para a frente. Dependendo da tarefa, pode existir ou não uma fase de aprendizado nesse ciclo de tempo. Se existir, a saída é comparada à entrada mestre e a propagação para trás do erro é usada para ajustar os pesos de conexão. As conexões recorrentes são fixas em 1.0 e não são sujeitas ao ajuste.[10] No próximo passo de tempo, t+1, essa sequência anterior é repetida. Dessa vez, as unidades de contexto contêm valores que são exatamente os valores das unidades de saída no tempo t, então essas unidades de contexto proveem a rede de memória.

9.4.2.2 Conclusões sobre tarefas temporais

Muitos comportamentos humanos desenvolvem-se através do tempo. Seria tolice tentar entender esses comportamentos sem levar em consideração sua natureza temporal. O conjunto corrente de simulações explora as consequências de desenvolver representações do tempo que são distribuídas, dependentes de tarefas e nas quais o tempo é representado implicitamente na dinâmica da rede.

[9] No caso de a função de ativação usada ter valores entre 0.0 e 1.0.

[10] Na maioria das redes usadas, existem conexões um para um entre cada unidade de saída e cada unidade de contexto. Isso implica que existe um número igual de unidades de contexto e unidades de saída. As conexões para cima entre as unidades de contexto e as unidades escondidas são totalmente distribuídas, de tal forma que cada unidade de contexto ativa todas as unidades escondidas.

9.5 REDES NEURAIS BASEADAS EM CONHECIMENTO

A rede neural trabalha muito bem ao resolver certos tipos de problemas. A maior crítica que se faz aos sistemas conexionistas é o fato de que se sabe que funcionam, mas não se sabe como. A representação interna dos pesos de uma rede neural é uma incógnita para os pesquisadores. Mas esse tipo de crítica está sendo respondido gradativamente. Já há alguns anos começaram as pesquisas em Redes Neurais Baseadas em Conhecimento, ou seja, redes neurais em que um conhecimento inicial é representado. A rede não começa mais com pesos sinápticos aleatórios e sim com um conjunto de pesos que refletem regras de produção. Na verdade, trata-se de uma mistura das abordagens simbólica e conexionista (a chamada abordagem *híbrida*).

Numa rede baseada em conhecimento, dada uma teoria simbólica, cria-se, a partir das regras dessa teoria, uma arquitetura de rede neural. Ou seja, de uma regra $A \rightarrow B$ cria-se uma conexão entre um neurônio que representa o conceito A e outro neurônio que representa o conceito B. Para uma regra $(A \wedge B) \rightarrow C$, têm-se dois neurônios de entrada A e B: um neurônio na camada escondida faz o papel da conjunção e um neurônio na camada de saída representa o conceito C. Basta ligá-los através de pesos de conexão grandes e tem-se uma rede que representa essa regra. A função da rede neural é revisar essa teoria. A teoria inicial (regras) está representada na rede. A rede começa a aprender. No final do aprendizado pode-se extrair de volta as regras da rede. A teoria representada pelas regras foi revisada pelo aprendizado. Fu (1991a, 1991b, 1993), Towell e Shavlik (1993), Setiono e Liu (1996) e outros apresentam sistemas neurais baseados em conhecimento.

Numa rede neural baseada em conhecimento, através do conhecimento inicial simbólico baseado em regras de produção, constroem-se conexões fortes entre neurônios que representam esses conceitos. No restante da rede, atribuem-se pesos pequenos, mas não nulos, de tal forma que uma regra ainda inexistente possa por fim se estabelecer. Através do treinamento a que a rede neural é exposta, vários padrões diferentes são apresentados à rede. Se um determinado padrão ocorre várias vezes, supondo que o treinamento é coerente com o que acontece no mundo, significa que deveria haver uma regra relacionando os conceitos envolvidos por esse padrão. Isto é, a teoria simbólica inicial deveria prever esse tipo de comportamento. Se a teoria realmente já contava com esse acontecimento, sua regra já foi implementada, e o treinamento se encarregará de fortalecer ainda mais os pesos sinápticos que relacionam esses conceitos. Caso contrário, é desejável que a teoria seja revista, ou seja, que essa "nova" regra seja adicionada ao conjunto de regras da teoria simbólica. Isso é conseguido através da extração de regras da rede neural.

Segundo Fu (1993), o procedimento completo é o seguinte: primeiro atribuem-se à rede os pesos altos, relativos às regras da teoria inicial. Aos demais pesos são atribuídos valores pequenos. A seguir a rede passa pelo processo de treinamento, através do algoritmo *backpropagation* ou de um outro algoritmo conexionista qualquer. Depois, a rede sofre um processo de anulação de seus pesos pequenos, pois estima-se que pesos muito pequenos não vão contribuir para o conhecimento da rede. Com a anulação, a rede é simplificada. Outra simplificação ocorre quando a rede é "clusterizada", ou seja, formam-se grupos de neurônios com vetores de pesos próximos entre si. Depois, novamente a rede é submetida a um treinamento, e no final desse processo ocorre a extração das regras. Fu propõe um algoritmo de extração de regras chamado KT, que trabalha com subconjuntos de pesos das entradas de um determinado neurônio, que, alcançado o limiar de ativação, forma uma regra com o neurônio alvo representando o conceito de saída. O grande problema do algoritmo KT é a grande quantidade de regras geradas.

Já Towell e Shavlik (1993) propõem um outro algoritmo de extração de regras um pouco diferente do algoritmo KT, chamado MofN. Esse algoritmo reduz bastante o número de regras obtidas. Basicamente, o algoritmo MofN consiste em seis passos:

1. Agrupar: criação de classes equivalentes, para agrupar as conexões da rede em grupos;
2. Tirar a média: depois de agrupados, faz com que os pesos de todas as conexões dentro de um grupo tenham o valor médio de todas as conexões desse grupo;
3. Eliminar: eliminação dos grupos com valores insignificantes, que não contribuem para o cálculo;
4. Otimizar: com os grupos sem importância eliminados no passo 3, otimizam-se os limiares da unidade;
5. Extrair: formam-se regras que expressam a rede, tal que uma regra é verdadeira se a soma dos seus antecedentes ponderados exceder o limiar;
6. Simplificar: as regras são simplificadas quando possível para eliminar pesos e limiares.

9.6 OUTRAS REDES

9.6.1 Rede de Hopfield – 1.ª versão

Utilização de uma Rede Neural para reconhecimento de imagem (padrões numéricos). Uma rede de 42 nós é usada como memória associativa. O algoritmo utilizado é o da Rede de Hopfield, que consiste nos seguintes passos (Lippmann, 1987):

1. Atribuição dos pesos de conexão:
   É feita a entrada dos padrões que se deseja que a rede "aprenda", no caso dígitos, na forma de matriz de 6 × 7 pontos (no vídeo são caracteres "0" ou "."). Com esses padrões, faz-se o seguinte cálculo:

$$T_{ij} = \sum_{s=0}^{M-1} x_i^s x_j^s, i \neq j$$

$$0, i = j, 0 \leq i, j \leq M - 1$$

   Ou seja, a matriz T (42 × 42) é obtida como a soma dos produtos do nó i pelo nó j ($i \neq j$) para todos os padrões fornecidos (s).
2. Iniciação com o padrão desconhecido:
   Entra-se com um padrão distorcido (com ruído) e faz-se:

$$\mu_i(0) = x_i, \ 0 \leq i \leq N - 1$$

3. Iteração até a convergência:

$$\mu_j(t+1) = f_h\left[\sum_{i=0}^{N-1} T_{ij}\, \mu_i(t)\right], 0 \leq j \leq M - 1$$

   Faz-se a soma dos produtos $T_{ij}\mu_i$ e obtém-se o valor da função f_h para essa soma. A função f_h é a função degrau (*hard limiter*), descrita a seguir (Figura 9.8).

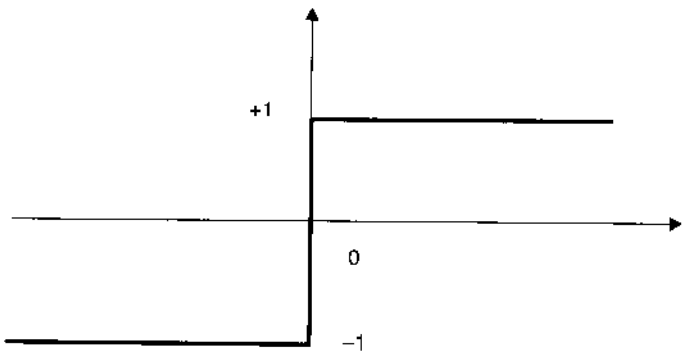


Figura 9.8 Função *hard limiter*.

A cada iteração, μ_j deve convergir para um dos padrões entrados no início.

Notas:

1. Os valores de saída dos nós da Rede de Hopfield são +1 e -1 (portanto, trata-se de uma rede binária). Para facilitar a visualização, cada vetor x de 42 posições foi "quebrado" numa matriz 6 × 7. Os valores entrados pelo teclado são "0", para +1 e espaço, ou qualquer outro caractere, para -1. A cada iteração, é fornecida a saída da aproximação obtida (escolheram-se seis iterações, por se considerar suficientes).
2. Quando se entra com os números 7-9-0-4-9-1, por exemplo (veja que existem dois "noves"), nota-se uma maior tendência para o padrão 9, mesmo que se deseje obter outro padrão. Quando o dígito distorcido, mas próximo, é entrado, obtém-se o padrão inicial. Mas quando a distorção é muito grande, pode-se não obter padrão nenhum.
3. Antes de se chegar a esse tamanho de vetor para x (42 posições) foram testados outros tamanhos de vetores, maiores e menores. Por uma questão de conveniência, escolheu-se o vetor de 42 posições, por resultar numa matriz de 6 × 7 e nesse caso facilitar a visualização dos dígitos, além de não ser necessária a entrada de muitos caracteres.

9.6.2 Rede de Hopfield – 2.ª versão

Esta segunda versão do algoritmo de Hopfield utiliza um cálculo diferente para a matriz de conexões. A matriz T é calculada pelo método da projeção. Esse método consiste no seguinte:

Hipótese: os padrões ensinados à rede (v^s) são linearmente independentes.

Como os padrões ensinados à rede devem ser pontos de equilíbrio, e assumindo o vetor de limiares nulo, então:

$$Tv^s = v^s$$

Uma solução dessa equação é:

$$T = P\,(P^T\,P)^{-1}\,P^T$$

em que:

P (N × M) é a matriz formada por colunas que são os padrões ensinados v^s;

P^T (M × N) é a transposta de P;

$(P^T\,P)^{-1}$ (M × M) é a inversa do produto de P^T por P.

A inversa foi calculada pelo método da eliminação de Gauss. Como se trata de uma matriz relativamente pequena (6 × 6), o cálculo torna-se simples.

Todos os outros passos seguem o algoritmo de Hopfield (1.ª versão).

Como o método da projeção considera que os padrões ensinados são pontos de equilíbrio, haverá nesse caso uma maior capacidade da rede em reconhecer os padrões ensinados e distinguir um do outro, fato que, muitas vezes, não acontecia na versão inicial.

Para uma melhor comparação entre as execuções do algoritmo de Hopfield padrão (versão inicial) e esta 2.ª versão, foram feitas simulações de ambos com os mesmos padrões ensinados. Nesse caso, para que ficasse clara a capacidade de reconhecimento de padrões entre ambos, foram ensinados dígitos bem semelhantes entre si.

Notas:

1. Os valores de saída dos nós da Rede de Hopfield são +1 e −1 (portanto, trata-se de uma rede binária). Para facilitar a visualização, cada vetor x de 42 posições foi "quebrado" numa matriz 6 × 7. Os valores entrados pelo teclado são "0", para +1 e espaço, ou qualquer outro caractere, para −1. A cada iteração, é fornecida a saída da aproximação obtida (escolheram-se seis iterações, por se considerar suficientes).
2. Observou-se que na versão inicial a rede não era capaz de distinguir o oito do nove, alguns padrões ensinados não eram aprendidos, pois convergiam para o oito: o zero quadrado e o zero redondo. Algumas aproximações para outros padrões acabavam convergindo para o oito: nove, zero, quatro e zero invertido. Mesmo quando não se entrou com padrão nenhum houve uma convergência para o oito. Em alguns casos, houve uma convergência para o inverso (complemento) do oito. Deve-se contudo observar que a versão inicial reconheceu alguns padrões ensinados.

   Na 2.ª versão, há um perfeito aprendizado de todos os padrões ensinados. Há convergência para o padrão mais próximo. Há também a convergência para o inverso mais próximo e, finalmente, há a convergência para um padrão não ensinado.
3. Antes de se chegar a esse tamanho de vetor para x (42 posições), foram testados outros tamanhos de vetores, maiores e menores. Por uma questão de conveniência, escolheu-se o vetor de 42 posições, por resultar numa matriz de 6 × 7 e nesse caso facilitar a visualização dos dígitos, além de não ser necessária a entrada de muitos caracteres.

9.6.3 Rede de Hamming

Utilização de uma Rede Neural para reconhecimento de imagem. O algoritmo utilizado é o da Rede de Hamming. Uma Rede de Hamming implementa um algoritmo que calcula a distância de Hamming ao exemplar para cada classe e seleciona a classe com a distância mínima. A operação dessa rede é descrita através dos seguintes passos (Lippmann, 1987):

1. Atribuição dos pesos de conexão

É feita a entrada dos padrões que se deseja que a rede "aprenda", no caso dígitos, na forma de matriz de 6 × 7 pontos (no vídeo são caracteres "0" ou "."). A Rede de Hamming tem dois níveis (duas sub-redes). Calculam-se os pesos de conexão da seguinte forma:

Na sub-rede inferior:

$$w_{ij} = \frac{x_i^j}{2}, \quad \theta_j = \frac{N}{2}$$

$$0 \leq i \leq N-1 \qquad 0 \leq j \leq M-1$$

Ou seja, a matriz w (42 × 6) é obtida como a metade dos valores do nó i do padrão j. O vetor θ (seis posições) contém o limiar em cada nó para o disparo do neurônio.
Na sub-rede superior:

$$t_{kl} = \begin{cases} 1, & k = l \\ -\varepsilon, & k \neq l, \ \varepsilon < \dfrac{1}{M} \end{cases}$$

$$0 \leq k, l \leq M - 1$$

A matriz t (6 × 6) contém o peso de conexão do nó k ao nó l, e os limiares nessa sub-rede são zero.

2. Iniciação com o padrão desconhecido

Entra-se com um padrão distorcido (com ruído) e faz-se:

$$\mu_j(0) = f_t \left[\sum_{i=0}^{N-1} w_{ij} x_i - \theta_j \right]$$

$$0 \leq j \leq M-1$$

Nessa equação, $\mu_j(t)$ é a saída do nó j na sub-rede superior no tempo t, x_i é o elemento i da entrada e f_t é a função do limiar (*threshold logic*):

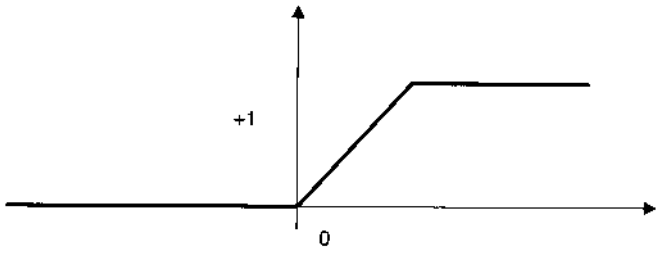


Figura 9.9 Função *threshold logic*.

3. Iteração até a convergência

$$\mu_j(t+1) = f_t \left[\mu_j(t) - \varepsilon \sum_{k \neq j} \mu_k(t) \right]$$

$$0 \leq j, k \leq M-1$$

Esse processo é repetido até a convergência, que deve acontecer quando a saída de um nó apenas permanece positiva.

Notas:

1. Os valores de entrada dos nós da Rede de Hamming são +1 e −1 (trata-se portanto de uma rede binária). Os valores de saída, devido à função f_t, são +1 e 0. Para facilitar a visualização, cada vetor x de 42 posições foi "quebrado" numa matriz 6 × 7. Os valores entrados pelo teclado são "0", para +1 e espaço, ou qualquer outro caractere, para −1. Depois da convergência do padrão de entrada distorcido para um dos padrões ensinados à rede, que deve ocorrer quando a saída de um único nó for positiva (a cada 10 iterações), é fornecido o padrão de saída.
2. Tentou-se obter o padrão ensinado através do padrão de entrada distorcido. Em alguns casos é possível obter padrões diferentes daqueles pretendidos, devido à proximidade de distâncias de Hamming. Em outros casos ainda consegue-se obter um padrão ensinado quando se entra com algo em nada parecido com um padrão. Esse fato é devido à característica de Rede de Hamming, que trabalha com a menor distância de Hamming entre padrões para localizar o padrão ensinado, não resultando, em hipótese nenhuma, em um padrão "estranho", como acontecia na Rede de Hopfield.

9.6.4 Rede de Carpenter/Grossberg

Utilização de uma Rede Neural para reconhecimento de imagem. O algoritmo utilizado é o da Rede de Carpenter/Grossberg. Essa rede trabalha com grupos (*clusters*) e é treinada sem supervisão. O primeiro padrão apresentado à rede é o primeiro grupo. O próximo padrão é comparado ao primeiro exemplar do grupo. Ele "segue o líder" e é "clusterizado" com o primeiro se a distância ao primeiro é menor que um limiar. Em caso contrário, ele é um exemplar para um novo grupo. Esse processo é repetido para todas as entradas seguintes. A operação dessa rede é descrita através dos seguintes passos (Lippmann, 1987).

1. Iniciação

$$t_{ij}(0) = 1$$

$$b_{ij}(0) = \frac{1}{1+N}$$

$$0 \leq i \leq N\text{-}1$$
$$0 \leq j \leq M\text{-}1$$

Iniciar ρ, em que $0 \leq \rho \leq 1$

Nessas equações, $b_{ij}(t)$ é o peso de conexão de baixo para cima e $t_{ij}(t)$ é o peso de conexão de cima para baixo entre o nó de entrada i e o nó de saída j no tempo t. A fração r é o limiar de vigilância que indica o quão perto uma entrada deve estar de um exemplar armazenado, para casar (*match*).

2. Entrada de uma nova entrada

É feita a entrada dos padrões que se deseja que a rede "aprenda", no caso dígitos, na forma de matriz de 5 × 7 pontos (no vídeo são caracteres "0" ou ".").

3. Cálculo dos *matching scores*

Calculam-se os *matching scores* da seguinte forma:

$$\mu_j = \left[\sum_{i=0}^{N-1} b_{ij}(t) x_i\right]$$

$$0 \leq j \leq M-1$$

Nessa equação, μ_j é a saída do nó j, x_i é o elemento i da entrada, que pode ser 0 ou 1.

4. Seleciona o melhor *matching exemplar*

Calcula-se o índice j do máximo valor dos μ_j.

5. Teste de vigilância

$$||X|| = \sum_{i=0}^{N-1} x_i$$

$$||T \cdot X|| = \sum_{i=0}^{N-1} t_{ji\_max} x_i$$

$$\frac{||T \cdot X||}{||X||} > p\ ?$$

sim => vá ao passo 7
não => vá ao passo 6

6. Desabilita o melhor *matching exemplar*

A saída selecionada no passo 4 é temporariamente setado em zero. Então vá ao passo 3.

7. Adapte o melhor *matching exemplar*

$$t_{ij\_max}(t+1) = t_{ij\_max}(t) * x_i$$

$$b_{ij\_max}(t+1) = \frac{t_{j\_max}(t)\, x_i}{.5 + \sum_{i=0}^{N-1} t_{j\_max}(t) x_i}$$

8. Repita indo ao passo 2.

Notas:

1. Os valores de entrada dos nós da Rede de Carpenter/Grossberg são +1 e 0 (trata-se portanto de uma rede binária). Para facilitar a visualização, cada vetor x de 35 posições foi

"quebrado" numa matriz 5 × 7. Os valores entrados pelo teclado são "0", para +1 e espaço, ou qualquer outro caractere, para 0.

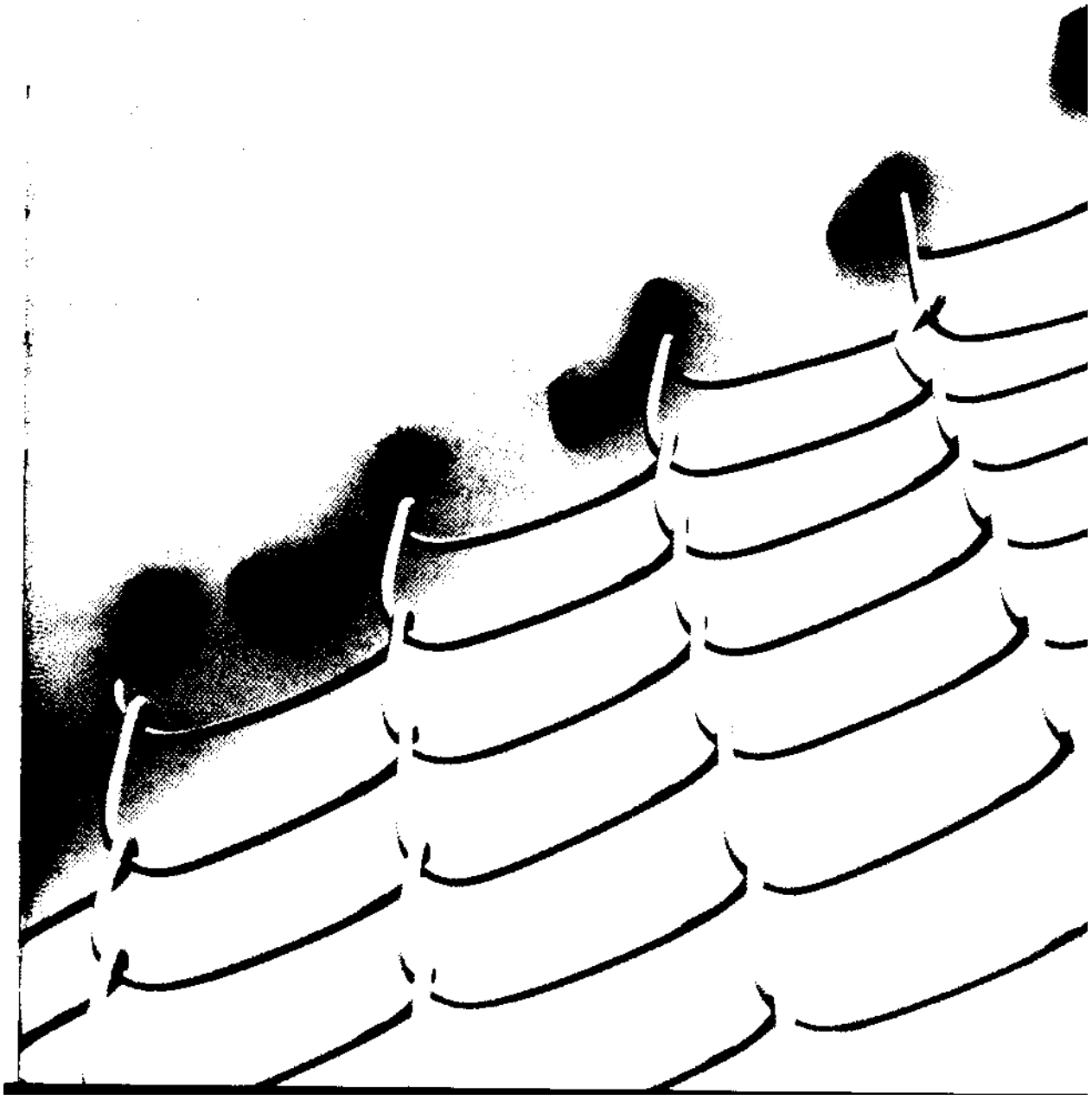
2. Foram feitas várias execuções da rede, para vários valores de limiares de vigilância. Foram ensinados sempre os mesmos padrões: um sete, um sete pouco distorcido, um nove quadrado, um nove redondo, um zero, um quatro, um quatro caído e uma letra "U" (um quatro mais caído ainda). Para o valor de limiar 0.1, observou-se que a rede reconhece poucos padrões, apenas o primeiro (que é o "grupo" inicial) e o sexto (o quatro). Já para o limiar igual a 0.3, apesar de a rede reconhecer os mesmos padrões que no caso anterior, há uma maior aproximação entre os padrões. Para o limiar 0.5, a rede é capaz de reconhecer quatro padrões (o primeiro, o nove redondo, o zero e o quatro). Para o limiar 0.7, a rede reconhece os mesmos padrões do limiar 0.5, só que há uma correspondência entre o quatro e o nove redondo e o reconhecimento do quatro caído. Para o limiar 0.95, houve o reconhecimento de cinco padrões e uma aproximação para os três restantes.

   Note que, à medida que se aumenta o valor do limiar de vigilância, aumenta-se a capacidade da rede em distinguir um padrão do outro já ensinado. Observe que os padrões nem sempre são encarados como um novo padrão. Isso se deve ao fato de que os padrões entrados são considerados "parecidos" com os já ensinados à rede.

9.7 CONCLUSÃO

O cérebro humano é o modelo natural para a construção de máquinas inteligentes. Uma ideia óbvia para a Inteligência Artificial é simular o cérebro em um computador (Rich e Knight, 1994). O aprendizado em representações de redes complexas é um dos tópicos mais atuais em ciência (Russell e Norvig, 1995), que promete grandes aplicações em ciência da computação, neurobiologia, psicologia e física. Apresentaram-se neste capítulo algumas das ideias e técnicas básicas das redes neurais artificiais. Uma rede neural é um modelo computacional que compartilha algumas das propriedades dos cérebros: ela consiste em muitas unidades simples trabalhando em paralelo sem nenhum controle central. As conexões entre as unidades têm pesos numéricos que podem ser modificados pelo aprendizado.

BIBLIOGRAFIA



BELLMAN, R. E. *An introduction to artificial intelligence: can computers think?* San Francisco: Boyd and Fraser Publishing, 1978.

BERWICK, R. C. Principles of principle-based parsing. In: BERWICK, R. C.; ABNEY, S. P.; TENNY, C. (Ed.) *Principle-based parsing: computation and psycholinguistics.* Dordrecht, Holland: Kluwer Academic Publishers, 1-37, 1992.

BRATKO, I. *Prolog programming for artificial intelligence.* 3. ed. Reading, MA: Addison-Wesley, 2000.

BRUSILOVSKY, P. Adaptive and intelligent technologies for web-based education. In: ROLLINGER, C.; PEYLO, C. (Ed.) *Künstliche intelligenz, special issue on intelligent systems and teleteaching.* 4, 1999, p. 19-25.

CASANOVA, M. A.; GIORNO, F. A. C.; FURTADO, A. L. *Programação em lógica e a linguagem prolog.* São Paulo: Edgard Blücher, 1987.

CHARNIAK, E. Passing markers: a theory of contextual influence in language comprehension. *Cognitive Science,* 7, 1983, p. 171-190.

_____; MCDERMOTT, D. *Introduction to artificial intelligence.* Reading, MA: Addison-Wesley, 1985.

CHOMSKY, N. On certain formal properties of grammars. *Information and Control* (2), 1959, p. 137-167.

_____. *Syntactic structures.* The Hague: Mouton, 1972.

COSTA, E. B. Inteligência artificial em educação presencial e a distância: modelos, tecnologias, aplicações e tendências. *Tutorial 4. International Joint Conference 7th. Iberoamerican Conference on Artificial Intelligence - 15th. Brazilian Symposium on Artificial Intelligence - IBERAMIA-SBIA 2000.* Nov. 2000, p. 19-22, Atibaia, SP.

CRICK, F.; ASANUMA, C. Certain aspects of the anatomy and physiology of the cerebral cortex. In: MCCLELLAND, J. L.; RUMELHART, D. E. (Ed.) *Parallel distributed processing.* Cambridge, MA; London, UK: The MIT Press, 1986, v. 2.

CROCKER, M. W. A principle-based system for syntactic analysis, *Canadian Journal of Linguistics,* 36 (1): 1-26, 1996.

_____. *Computational psycholinguistics: an interdisciplinary approach to the study of language.* Dordrecht, Holland: Kluwer Academic Publishers, 1996.

DAHL, V. Translating Spanish into Logic through Logic, *American Journal of Computational Linguistics,* v. 7, n. 3, jul./sept. 1981, p. 149-164.

DUBOIS-CHARLIER, F. *Bases de análise linguística.* Coimbra: Livraria Almedina, 1976.

ELMAN, J. L. Finding structure in time. *Cognitive Science,* 14, 1990, p. 179-211.

FELDMAN, J. A. Connections. *Byte,* apr. 1990, p. 277-285.

FREEDMAN, R.; ALI, S. S.; MCROY, S. What is an intelligent tutoring system? *ACM Intelligence,* 2000, p. 15-16.

_____. et al. Its tools for natural language dialogue: a domain-independent parser and planner.

Fifth International Conference on Intelligent Tutoring Systems (ITS'00). Montreal. Lecture Notes in Computer Science: Springer Verlag, 2000.

_____. et al. Using rule induction to assist in rule construction for a natural-language based intelligent tutoring system. *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*. Madison, WI. 1998.

FU, L. M. Rule learning by searching on adapted nets. In: *Proceedings of the AAAI-91*. Anaheim, CA, 1991, p. 590-595.

_____. Knowledge base refinement by back-propagation, *Data and Knowledge Engineering*, 7, 1991, p. 35-46.

_____. Knowledge based connectionism for revising domain theories. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 23, n. 1, jan./feb. 1993, p. 173-182.

GEETHA, T. V.; SUBRAMANIAN, R. K. Representing natural language with prolog. *IEEE Software*, v. 7, n. 2, 1993, p. 85-92.

GLASS, M. Processing language input in the CIRCSIM-Tutor intelligent Tutoring system. *AAAI 2000 Fall Symposium on Building Dialogue Systems for Tutorial Applications*, 2000.

GRAESSER, A. C. et al. Autotutor: a simulation of a human tutor. *Journal of Cognitive Systems Research*, 1, 1999, p. 35-51.

HAUGELAND, J. (Ed.) *Artificial intelligence: the very idea*. Cambridge, MA: MIT Press, 1985.

HAYKIN, S. *Neural networks: a comprehensive foundation*. 2. ed. Upper Saddle River, NJ: Prentice Hall, 1999.

HEBB, D. O. *The organization of behavior*. New York: Wiley, 1949.

HOPCROFT, J. E.; ULLMAN, J. D. *Formal languages and their relation to automata*. Reading, MA: Addison-Wesley, 1969.

HOPFIELD, J. J. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences (USA)*, 79:2554-2558, 1982.

IVERSEN, L. L. Amino acids and peptides: fast and slow chemical signals in the nervous system? *Proceedings of the Royal Society of London*, B, 221 (1224), 22 may 1984, p. 245-260.

JAIN, A. K.; MAO, J. Artificial neural Networks: a tutorial. *Computer*, v. 29, n. 3, mar. 1996, p. 31-44.

JURAFSKY, D.; MARTIN, J. H. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, NJ: Prentice-Hall, 2000.

KANDEL, E. R.; SCHWARTZ, J. H.; JESSELL, T. M. (Ed.) *Essentials of Neural Science and Behavior*. Stamford, Connecticut: Appleton & Lange, 1995.

KLIR, G. A.; FOLGER, T. A. *Fuzzy sets, uncertainty, and information*. Upper Saddle River, NJ: Prentice-Hall, 1988.

KOLATA, G. How can computers get common sense. *Science*, v. 217, sept. 1982, p. 1237-1238.

KOSKO, B. *Neural networks and fuzzy systems: a dynamical approach to machine intelligence*. Upper Saddle River, NJ: Prentice-Hall, 1992.

KUFFLER, S. W.; NICHOLLS, J. G. *From neuron to brain: a cellular approach to the function of the nervous system*. 2. ed. Sunderland, MA: Sinauer, 1984.

KURZWEIL, R. *The age of intelligent machines*. Cambridge, MA: The MIT Press, 1990.

LAURIÈRE, J.-L. *Problem solving and artificial intelligence*. Upper Saddle River, NJ: Prentice-Hall, 1990.

LIPPMANN, R. P. An introduction to computing with neural nets, *IEEE ASSP Magazine*, apr. 1987, p. 4-22.

LUGER, G. F.; STUBBLEFIELD, W. A. *Artificial intelligence: structures and strategies for complex problem solving*. 2. ed. Redwood City, CA: Benjamin/Cummings, 1993.

MARCUS, C. *Prolog programming: applications for database systems, expert systems, and natural language systems*. Reading, MA: Addison-Wesley, 1986.

MARCUS, M. P. *A theory of syntactic recognition for natural language*. Cambridge, MA: The MIT Press, 1980.

MARTIN, C. E. Case-based parsing. In: RIEBECK, C. K.; SCHANK, R. C. (Ed.) *Inside case-based reasoning*. Hillsdale, NJ: Lawrence Erlbaum, 1989, p. 319-372.

MCARTHUR, D.; LEWIS, M.; BISHAW, M. *The roles of artificial intelligence in education: current progress and future prospects*. RAND DRU-472-NSF, Santa Monica, CA, 1993.

MCCLELLAND, J. L.; KAWAMOTO, A. H. Mechanisms of sentence processing: assigning roles to constituents of sentences. In: MCCLELLAND, J. L.; RUMELHART, D. E. *Parallel distributed processing: explorations in the microstructure of cognition – volume 2: psychological and biological models*. Cambridge, MA: A Bradford Book; The MIT Press, 1986.

MCCLELLAND, J. L.; RUMELHART, D. E.; The PDP Research Group. *Parallel distributed processing: explorations in the microstructure of cognition – volume 2: psychological and biological models*. Cambridge, MA: A Bradford Book; The MIT Press, 1986.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115-137, 1943.

MIIKKULAINEN, R. *Subsymbolic natural language processing: an integrated model of scripts, lexicon, and memory*. Cambridge, MA: A Bradford Book; The MIT Press, 1993.

_____. Subsymbolic case-role analysis of sentences with embedded clauses, *Cognitive Science*, 20, 1996, p. 47-73.

MOLL, R. N.; ARBIB, M. A.; KFOURY, A. J. *An introduction to formal language theory*. New York: Springer-Verlag, 1988.

NILSSON, N. J. *Principles of artificial intelligence*. New York: Springer-Verlag, 1982.

_____. *Artificial intelligence: a new synthesis*. San Francisco, CA: Morgan Kaufmann Publishers, 1998.

OBERMEIER, K. K. Natural-language processing. *Byte*, dec. 1987, p. 225-232.

OLDE, B. A. et al. A connectionist model for part of speech tagging. *Proceedings of the American Association for Artificial Intelligence*, Menlo Park, CA: AAAI Press, 1987, p. 172-176.

PEREIRA, F. C. N.; GROSZ, B. J. Introduction (to the special volume on natural language processing), *Artificial Intelligence*, 63, 1993, p. 1-15.

_____; SHIEBER, S. M. *Prolog and natural language analysis*. CSLI – Center for the Study of Language and Information, 1987.

_____; WARREN, D. H. D. Definite clause grammar for language analysis: a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13, 1980, p. 231-278.

PERSON, N. K. et al. Incorporating human-like conversational behaviors into AutoTutor. *Submitted to the workshop on achieving human-like behavior in interactive animated agents*, held in conjunction with *The Fourth International Conference on Autonomous Agents*. Barcelona, Spain, 2001.

_____; GRAESSER, A. C. Designing AutoTutor to be an affective conversational partner. *Proceedings of the Fourth International Conference of the Learning Sciences*, 2000.

_____; _____; HARTER, D. The dialog advancer network: a mechanism for improving AutoTutor's conversational skills. *Proceedings of the 10th Annual Meeting of the Society for Text and Discourse*. Lyon, France, 2001.

_____, et al. The integration of affective responses into AutoTutor. *Proceedings of the International Workshop on Affect in Interactions*. Siena, Italy, 1999, p. 167-178.

QUILLIAN, M. R. Semantic memory. In: MINSKY, M. *Semantic information processing*. Cambridge: The MIT Press, 1968.

RICH, E.; KNIGHT, K. *Inteligência artificial*. 2. ed. São Paulo: Makron Books, 1994.

ROCHA, A. F. *Neural nets: a theory for brains and machines*. Berlin: Springer-Verlag, 1992.

ROSA, J. L. G. *Linguagens formais e autômatos*. Rio de Janeiro: LTC, 2010.

_____. An artificial neural network model based on neuroscience: looking closely at the brain. In: KURKOVÁ, V. N. C. et al. *Artificial neural nets and genetic algorithms: proceedings of the International Conference in Prague*, Czech Republic, 2001 – ICANNGA-2001. April 22-25, Springer-Verlag, 2001, p. 138-141. ISBN: 3-211-83651-9.

_____. *Redes neurais e lógica formal em processamento de linguagem natural*. 1993. Dissertação (Mestrado em Engenharia Elétrica) – Departamento de Engenharia de Computação e Automação Industrial – Faculdade de Engenharia Elétrica e de Computação – Universidade Estadual de Campinas – DCA-FEEC-Unicamp.

_____. *Um sistema híbrido simbólico-conexionista para o processamento de papéis temáticos*. 1999. Tese (Doutorado em Linguística) – Departamento de Linguística. Instituto de Estudos da Linguagem. Universidade Estadual de Campinas. DL-IEL-Unicamp.

_____; FRANÇOZO, E. Hybrid thematic role processor: symbolic linguistic relations revised by connectionist learning. In: *Proceedings of IJCAI'99 – Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, july 31–august 6, 1999, v. 2, Morgan Kaufmann, 852-857.

_____; _____. Linguistic relations encoding in a symbolic-connectionist hybrid natural language processor. In: Monard, M. C.; SICHMAN, J. S. (Ed.) *Lecture Notes in Artificial Intelligence 1952, Advances in Artificial Intelligence, Proceedings of the International Joint Conference 7th. Ibero-American Conference on AI – 15th. Brazilian Symposium on AI – IBERAMIA-SBIA 2000*, Atibaia, São Paulo, Brazil, november 19-22, 259-268, Berlin: Springer-Verlag, 2000.

ROSÉ, C. P. Facilitating the rapid development of language understanding interfaces for tutoring systems. *Proceedings of the AAAI Fall Symposium on Building Tutorial Dialogue Systems*, 2000.

ROSENBLATT, F. *The perceptron: a perceiving and recognizing automaton*. Report 85-460-1, Project PARA. Ithaca, NY: Cornell Aeronautical Laboratory, 1957.

ROWE, N. C. *Artificial Intelligence Through Prolog*. Upper Saddle River, NJ: Prentice Hall, 1988.

RUSSELL, S.; NORVIG, P. *Artificial intelligence: a modern approach*. Upper Saddle River, NJ: Prentice-Hall, 1995.

SAVADOVSKY, P. *Introdução ao projeto de interfaces em linguagem natural*. São Paulo: SID Informática, 1988.

SCHALKOFF, R. J. *Artificial intelligence: an engineering approach*. New York: McGraw-Hill, 1990.

SCHWIND, C. B. Logic based natural language processing. In: DAHL, V.; SAINT-DIZIER, P (Ed.) *Natural language understanding and logic programming*. North-Holland: Elsevier Science Publishers, 1985, p. 207-219.

SETIONO, R.; LIU, H. Symbolic representation of neural networks. *Computer*, v. 29, n. 3, mar. 1996, p. 71-77.

SHUTE, V. J.; PSOTKA, J. Intelligent tutoring systems: past, present, and future. In: Jonassen, D. H. (Ed.) *Handbook of research for educational communication and technology*. New York: Macmillan, 1996.

SILVA, A. B.; ROSA, J. L. G. Biological plausibility in artificial neural networks: an improvement on earlier models. *Proceedings of The Seventh International Conference on Machine Learning and Applications* (ICMLA'08), 11-13 dec. 2008, San Diego, California, USA. IEEE Computer Society Press, p. 829-834. DOI 10.1109/ICMLA.2008.73.

SOWA, J. F. *Conceptual structures: information processing in mind and machine*. Reading, MA: Addison-Wesley, 1984.

STABLER JR., E. P. Parsing as non-horn deduction. *Artificial Intelligence*, 63, 1996, p. 225-264.

SWINNEY, D. A. Lexical access during sentence comprehension: (re)consideration of context effects. *Journal of Verbal Learning and Verbal Behavior* 18, 1979, p. 645-659.

TOWELL, G. G.; SHAVLIK, J. W. Extracting refined rules from knowledge-based neural networks, *Machine Learning*, 13, 1993, p. 71-101.

TURING, A. Computing machinery and intelligence. *Mind*, 59, 1950, p. 433-460.

WALTZ, D. L.; POLLACK, J. B. Massively parallel parsing: a strongly interactive model of natural language interpretations. *Cognitive Science*, 9, 1985, p. 51-74.

WARNER, A. J. Natural language processing in information retrieval. *Bulletin of the American Society for Information Science*, aug./sept. 1988, p. 18-19.

WIEMER-HASTINGS, P.; GRAESSER, A. C.; HARTER, D. The foundations and architecture of AutoTutor. *Proceedings of the 4th International Conference on Intelligent Tutoring Systems*. Berlin: Springer, 1998, p. 334-343.

_____. How latent is latent semantic analysis? In: *Proceedings of IJCAI'99 – Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, july 31-august 6, 1999, v. 2, Morgan Kaufmann, p. 932-937.

_____; WIEMER-HASTINGS, K.; GRAESSER, A. C. Approximate Natural Language Understanding for an Intelligent Tutor. *Proceedings of the 12th. International Florida Artificial Intelligence Research Society Conference*. Menlo Park, CA: AAAI Press, 1999, p. 192-196.

WINSTON, P. H. *Artificial intelligence*. 3. ed. Reading, MA: Addison-Wesley, 1992.

WOODS, W. A. Transition network grammar for natural language analysis. *Communications of the ACM*, v. 13, n. 10, oct. 1970, p. 591-606.

_____. What's important about knowledge representation? *IEEE Computer*, oct. 1983, p. 22-27.

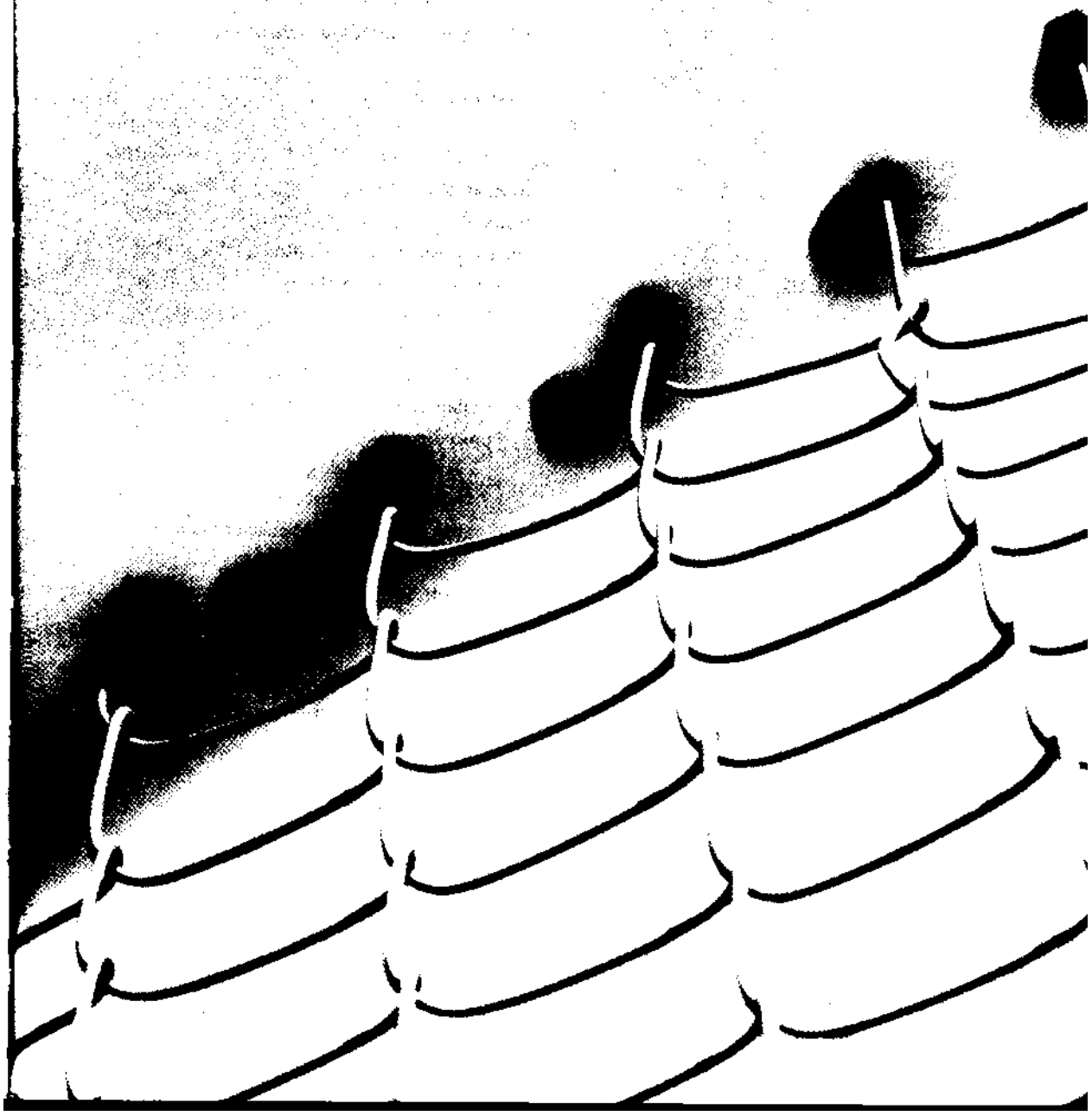
ZADEH, L. Fuzzy Sets. *Information and Control*, 8, 1965, p. 338-353.

ZEIDENBERG, M. Modeling the brain. *Byte*, dec. 1987, p. 237-246.

ZHOU, Y. et al. Delivering hints in a dialogue-based intelligent tutoring system. *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*. Orlando, FL, 1999.

_____. et al. What should the tutor do when the student cannot answer a question? *Proceedings of the Twelfth Florida Artificial Intelligence Symposium (FLAIRS'99)*. Orlando, FL, 1999.

ÍNDICE



R

S

Serviços de impressão e acabamento
executados, a partir de arquivos digitais fornecidos,
nas oficinas gráficas da EDITORA SANTUÁRIO
Fone: (0XX12) 3104-2000 - Fax (0XX12) 3104-2016
http://www.editorasantuario.com.br - Aparecida-SP

João Luís Garcia Rosa é professor doutor do Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP), em São Carlos, onde leciona disciplinas relacionadas à Inteligência Artificial na graduação e na pós-graduação. Graduado em Engenharia Elétrica pela Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas – Unicamp, recebeu os títulos de mestre em Engenharia de Computação pela mesma faculdade e de doutor em Linguística (Computacional) pelo Instituto de Estudos da Linguagem da Unicamp. Foi professor titular do Centro de Ciências Exatas, Ambientais e de Tecnologias (CEATec) da Pontifícia Universidade Católica de Campinas (PUC-Campinas), onde lecionou em cursos de graduação e no mestrado.

LTC

www.grupogen.com.br
http://gen-io.grupogen.com.br

De caráter multidisciplinar, a Inteligência Artificial desperta cada vez mais a curiosidade e o interesse de diversos segmentos acadêmicos, das ciências exatas às humanas. Muito além das conhecidas aplicações em jogos, há inúmeras tarefas – das corriqueiras às mais sofisticadas – que merecem a atenção de especialistas para o desenvolvimento de máquinas inteligentes.

Atento à lacuna significativa nas publicações sobre o tema, João Luís Garcia Rosa produziu amplo material com os fundamentos básicos e as principais ferramentas para aplicação da teoria à prática, incluindo as redes neurais artificiais.

FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL traz todos os conhecimentos necessários à compreensão da disciplina. Apresentada de forma didática e bastante acessível, a obra pode ser utilizada por estudantes de variados cursos de graduação e pós-graduação voltados ao embasamento teórico dos sistemas inteligentes, além dos demais leitores interessados nesta área de conhecimento.

gen | LTC

www.grupogen.com.br
http://gen-io.grupogen.com.br

ISBN 978-85-216-0593-5