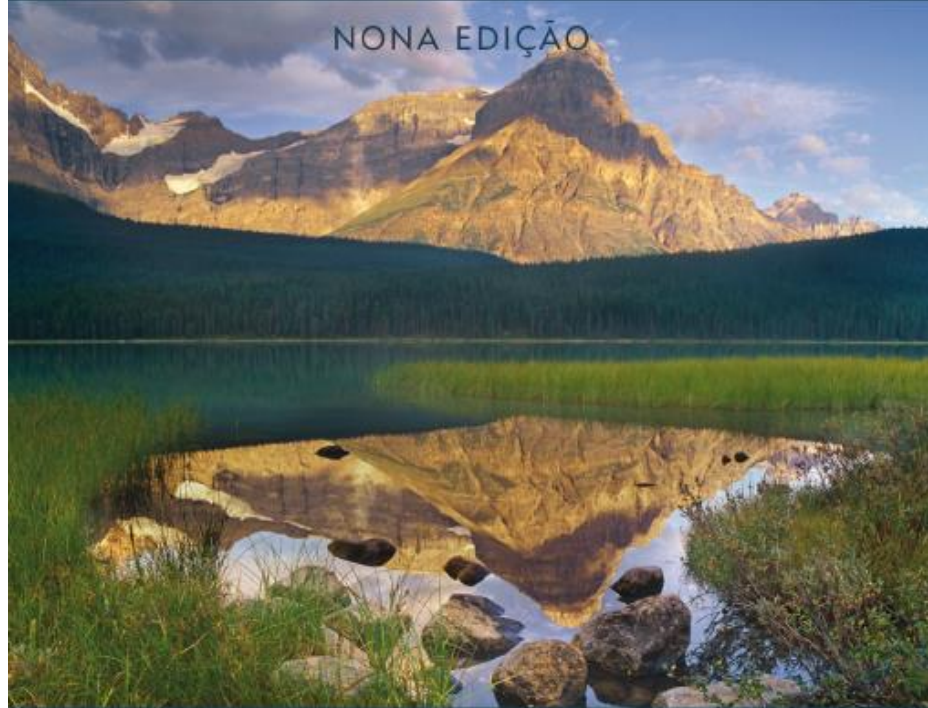


CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

NONA EDIÇÃO

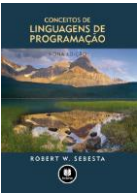


ROBERT W. SEBESTA



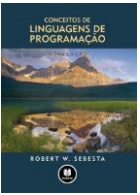
Capítulo 16

Linguagens de Programação Lógica



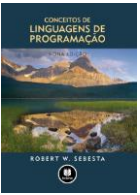
Tópicos do Capítulo 16

- Introdução
- Uma breve introdução ao cálculo de predicados
- Cálculo de predicados e prova de teoremas
- Uma visão geral da programação lógica
- As origens do Prolog
- Os elementos básicos do Prolog
- Deficiências do Prolog
- Aplicações de programação lógica



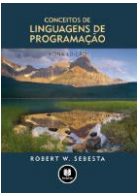
Introdução

- Linguagens de programação *lógicas*, também chamadas de linguagens de programação *declarativas*
- Expressa programas em uma forma de lógica simbólica
- Usa um processo de inferência lógico para produzir resultados
- *Declarativa* em vez de *procedural*:
 - Apenas as especificações dos resultados são expressas (em vez dos procedimentos detalhados para produzi-los)



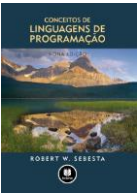
Proposição

- Sentença lógica que pode ou não ser verdadeira
 - Consiste em objetos e relacionamentos de objetos uns com os outros



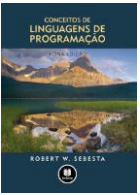
Lógica simbólica

- Pode ser usada para três necessidades básicas da lógica formal:
 - Expressar proposições
 - Expressar os relacionamentos entre proposições
 - Descrever como novas proposições podem ser inferidas a partir de outras que se assume verdadeiras
- A forma particular de lógica simbólica usada para programação lógica é chamada de *cálculo de predicados de primeira ordem*



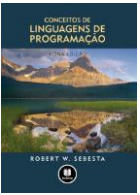
Representação de objetos

- Os objetos em proposições de programação lógica são representados por termos simples: constantes ou variáveis
- *Constante*: um símbolo que representa um objeto
- *Variável*: um símbolo que pode representar objetos diferentes em momentos diferentes
 - Diferente das variáveis em linguagens imperativas



Termos compostos

- *Proposições atômicas* consistem em termos compostos
- *Termos compostos*: um elemento de uma relação matemática, escrito como uma notação de função matemática
 - Função matemática é um mapeamento
 - Pode ser representado como uma tabela



Partes de um termo composto

- Um termo composto tem duas partes
 - Functor: símbolo da função que nomeia a relação
 - Lista ordenada de parâmetros (tupla)

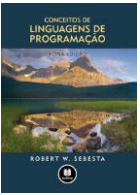
- Exemplos:

`estudante(jonh)`

`gosta(bruna, OSX)`

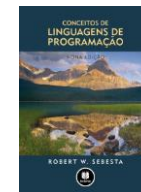
`gosta(andre, windows)`

`gosta(lucilia, linux)`



Formas de uma proposição

- Proposições podem ser definidas de duas formas:
 - *Fato*: proposição definida como verdadeira
 - *Consulta*: a verdade da proposição é algo que deve ser determinado
- Proposições compostas:
 - Têm duas ou mais proposições atômicas
 - Conectadas por operadores



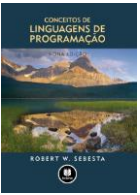
Operadores lógicos

<i>Nome</i>	<i>Símbolo</i>	<i>Exemplo</i>	<i>Significado</i>
negação	$\neg$	$\neg a$	não a
conjunção	$\cap$	$a \cap b$	a e b
disjunção	$\cup$	$a \cup b$	a ou b
equivalência	$\equiv$	$a \equiv b$	a é equivalente a b
implicação	$\supset$	$a \supset b$	a implica em b
	$\subset$	$a \subset b$	b implica em a



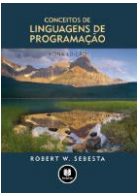
Quantificadores

<i>Nome</i>	<i>Exemplo</i>	<i>Significado</i>
universal	$\forall X.P$	Para todo X , P é verdadeiro.
existencial	$\exists X.P$	Existe um valor de X tal que P é verdadeiro.



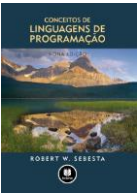
Forma clausal

- Muitas maneiras de definir proposições com o mesmo significado
- Use uma forma padrão para proposições
- *Forma clausal*:
 - $B_1 \cup B_2 \cup \dots \cup B_n \subseteq A_1 \cap A_2 \cap \dots \cap A_m$
 - Significa que todos os A s são verdadeiros, então ao menos um B é verdadeiro
- *Antecedente*: lado direito
- *Consequente*: lado esquerdo



Uma visão geral da programação lógica

- Semântica declarativa
 - Existe uma maneira simples de determinar o significado de cada sentença
 - Mais simples do que a semântica das linguagens imperativas
- A programação é não procedural
 - Os programas em tais linguagens não descrevem exatamente como um resultado será computado, mas a forma do resultado



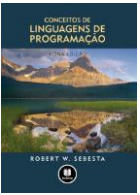
As origens do Prolog

- Universidade de Aix-Marseille
 - Processamento de linguagem natural
- Universidade de Edimburgo
 - Prova automatizada de teoremas



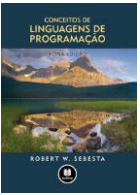
Termos

- Sintaxe de Edimburgo
- *Termo*: constante, variável ou estrutura
- *Constante*: átomo ou inteiro
- *Átomo*: valor simbólico de Prolog
- Átomos consistem em:
 - uma cadeia de letras, dígitos e sublinhados que iniciam com uma letra minúscula
 - uma cadeia de quaisquer caracteres ASCII delimitados por apóstrofes



Termos: variáveis e estruturas

- *Variável*: qualquer cadeia de letras, dígitos e sublinhados que iniciam com uma letra maiúscula
- *Instanciação*: ligação de uma variável a um valor
 - Dura apenas o tempo que for preciso para satisfazer um objetivo completo
- *Estrutura*: representa proposição atômica
`functor (lista de parâmetros)`



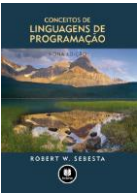
Sentenças de fatos

- Usadas para hipóteses
- Cláusulas de Horn sem cabeça

```
femea(shelley) .
```

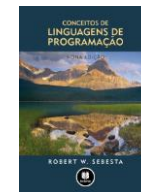
```
macho(bill) .
```

```
pai(bill, jake) .
```



Sentenças de regras

- Usadas para hipóteses
- Cláusula de Horn com cabeça
- Lado direito: *antecedente* (parte **se**)
 - Pode ser um termo simples ou uma conjunção
- Lado esquerdo: *consequente* (parte **então**)
 - É um termo simples
- *Conjunção*: contém múltiplos termos que são separados por operações E lógicas (implicada)



Exemplos de regras

```
ancestral(mary,shelley):- mae(mary,shelley) .
```

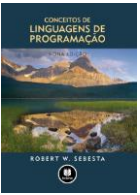
- Pode usar variáveis (*objetos universais*) para generalizar seu significado:

```
parente(X,Y):- mae(X,Y) .
```

```
parente(X,Y):- pai(X,Y) .
```

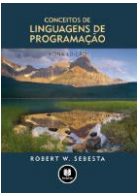
```
avos(X,Z):- parente(X,Y) , parente(Y,Z) .
```

```
irmaos(X,Y):- mae(M,X) , mae(M,Y) ,  
               pai(F,X) , pai(F,Y) .
```



Sentenças de objetivos

- O teorema é na forma de uma proposição que queremos que o sistema prove sua veracidade ou sua falsidade – *sentença de objetivo*
- Mesmo formato das cláusulas de Horn sem cabeça
`homem(fred)`
- Proposições conjuntivas e proposições com variáveis também são objetivos legais
`pai(X,mike)`



O processo de inferência do Prolog

- As consultas são chamadas de **objetivos**
- Quando um objetivo é uma proposição composta, cada um dos fatos (estruturas) é chamado de **subobjetivo**
- Para provar que um objetivo é verdadeiro, deve encontrar uma cadeia de regras de inferência e/ou fatos. Para o objetivo Q:

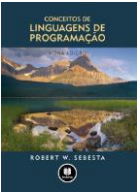
B :- A

C :- B

...

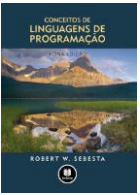
Q :- P

- O processo de fornecer um subobjetivo é chamado de casamento, resolução ou satisfazer o subobjetivo



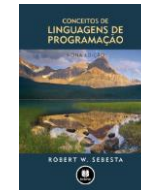
Abordagens

- *Resolução ascendente, encadeamento para frente*
 - Começa com os fatos e as regras da base de dados e tenta encontrar uma sequência de casamentos que levem ao objetivo
 - Funciona bem com um grande conjunto de possíveis respostas corretas
- *Resolução descendente, encadeamento para trás*
 - Inicia o objetivo e tenta encontrar uma sequência de proposições que casem com o objetivo que levem a algum conjunto de fatos originais na base de dados
 - Funciona bem quando existe um conjunto razoavelmente pequeno de respostas candidatas
- As implementações de Prolog usam encadeamento para trás



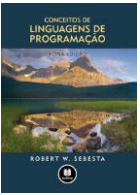
Subobjetivos

- Quando o objetivo tem mais de um subobjetivo, pode usar
 - Buscar primeiro em profundidade: encontrar uma sequência completa de proposições – uma prova – para o primeiro subobjetivo antes de trabalhar com os outros
 - Buscar primeiro em largura: funciona em todos os subobjetivos em paralelo
- Prolog usa busca primeiro em profundidade



Encadeamento para trás

- Quando um objetivo com múltiplos subobjetivos está sendo processado e o sistema falha em mostrar a verdade de um, reconsidera o subobjetivo anterior para encontrar uma solução alternativa:
encadeamento para trás
- Começa a busca onde a anterior para tal subobjetivo parou
- Requer uma boa dose de tempo e espaço porque pode ter de encontrar todas as provas possíveis para cada um dos subobjetivos

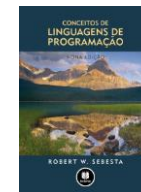


Aritmética simples

- O Prolog suporta variáveis inteiras e aritmética de inteiros
- Operador `is`: recebe uma expressão aritmética como seu operando direito e uma variável como seu operando esquerdo

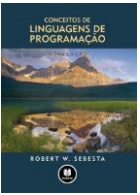
`A is B / 17 + C`

- Diferente da sentença de atribuição



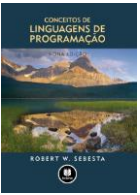
Exemplo

```
speed(ford,100) .
speed(chevy,105) .
speed(dodge,95) .
speed(volvo,80) .
time(ford,20) .
time(chevy,21) .
time(dodge,24) .
time(volvo,24) .
distance(X,Y) :-    speed(X,Speed) ,
                    time(X,Time) ,
                    Y is Speed * Time.
```



Aplicações de programação lógica

- Sistemas de gerenciamento de bases de dados relacionais
- Sistemas especialistas
- Processamento de linguagem natural



Resumo

- A lógica simbólica fornece a base para a programação lógica
- Programas lógicos devem ser não procedurais
- Sentenças Prolog são fatos, regras ou objetivos
- Resolução é a atividade primária de um interpretador Prolog
- A programação lógica tem sido usada em algumas áreas diferentes