Paradigmas de Linguagens de Programação

Linguagens de Programação Lógicas

Cristiano Lehrer, M.Sc.

ybadoo

Introdução

- Uma metodologia de programação diferente.
- Expressar programas na forma de lógica simbólica e a utilização de inferência lógica para produzir resultados.
- Programação lógica é declarativa, não procedural.
- Sintaxe diferente das linguagens imperativas e funcionais.

Visão Geral (1/3)

- Programa consiste de declarações ao invés de atribuições e controle de fluxo.
- Estas declarações são proposições lógicas.
- Semântica declarativa:
  - Existe uma única forma de determinar o significado de cada instrução, e isto não depende de como a instrução pode ser usada para resolver o problema.
- Semântica declarativa é mais simples do que a semântica de linguagens imperativas.

Visão Geral (2/3)

- O significado de uma dada proposição em uma linguagem de programação lógica pode ser determinado pela própria instrução.
- Em linguagens imperativas, uma simples atribuição requer o exame de declarações locais, regras de escopo e determinação do tipo.
- Programação imperativa e funcional são procedurais.
- Programação lógica não é procedural.
- Não descrevem como o resultado deve ser alcançado, mas descreve a forma do resultado.

Visão Geral (3/3)

- O sistema de computação pode, de alguma forma, determinar como o resultado pode ser alcançado.
- Precisa de um meio para fornecer informações relevantes (cálculo dos predicados) e um método de inferência para computar os resultados (método de prova).
- Diferença entre procedural e não procedural:
  - Exemplo: produzir uma lista ordenada
    - Procedural: tem que dizer tudo o que tem que ser feito para produzir a lista ordenada.
    - Não procedural: descrever as características de uma lista ordenada.

Prolog

- PROgrammation en LOGique:
  - Criada em meados de 1972 por Alain Colmerauer e Philippe Roussel.
  - Principais dialetos:
    - University of Aix-Marseille.
    - University of Edinburgh.
  - *Fifth Generation Computing Systems*:
    - Projeto japonês realizado na década de 80.
    - Criação de um computador com desempenho semelhante a um supercomputador e capacidade prática de inteligência artificial.

Termos (1/2)

- Todas as instruções em Prolog são construídas a partir de termos:
  - Um termo é uma constante, uma variável ou uma estrutura:
    - Constante
      - Átomo – sequencia de letras, números e *underscore*, mas sempre iniciando com uma letra minúscula, ou com aspas simples:
        - `fulano`
        - `'ciclano'`
      - Número – sequencia de dígitos, permitindo também os sinais de `.` (para números reais) e `–` (para números negativos).

Termos (2/2)

- Variáveis não tem tipo por declaração:
  - A vinculação é chamada de instanciação:
    - Ocorre durante o processo de resolução.
    - Permanece até que um objetivo seja alcançado, o que implica na prova ou não de uma proposição.
- Estruturas representam as proposições atômicas do cálculo de predicados:
  - As estruturas têm a seguinte forma:
    - função(lista de parâmetros)
      - função – identificador da estrutura (átomo)
      - parâmetros – qualquer lista de átomos, variáveis e outras estruturas
  - São meios de se especificar fatos.

Fatos

- Proposição onde uma nova informação é dada e é assumida ser verdadeira:
  - `mulher(shelley).`
  - `homem(bill).`
  - `mulher(mary).`
  - `homem(jake).`
  - `pai(bill, jake).`
  - `pai(bill, shelley).`
  - `mae(mary, shelley).`
  - `mae(mary, jake).`

Regras

- Uma conclusão pode ser alcançada se um conjunto de condições é satisfeito:
  - Lado direito – antecedente ou if
  - Lado esquerdo – consequente ou then
- Se o antecedente é verdade, o consequente também será:
  - `pais(X, Y) :- mae(X, Y).`
  - `pais(X, Y) :- pai(X,Y).`
  - `avos(X, Z) :- pais(X, Y), pais(Y, Z).`
  - `irmaos(X, Y) :- mae(M, X), mae(M, Y), pai(P, X), pai(P, Y).`

Meta

- As instruções são usadas para descrever fatos e regras para descrever o relacionamento entre fatos.
- Estas instruções são a base do modelo de prova de teoremas.
- O teorema é uma proposição que se quer provar ou não provar:
  - `?- pai(X, jake).`

Aritmética

- Originalmente, operadores eram funções:
  - `+ (7, X).`
- Uma sintaxe mais abreviada – operador is:
  - `A is B * 17 + C.`
  - `Sum is Sum + Number.`
    - Sempre falha.
- Exemplos:
  - `speed(ford, 100).`
  - `speed(chevy, 105).`
  - `time(ford, 20).`
  - `time(chevy, 21).`
  - `distance(X, Y) :- speed(X, S), time(X, T), Y is S * T.`

Listas (1/2)

- Os elementos são separados por vírgulas e a lista inteira é delimitada por colchetes:
  - [maçã, ameixa, uvas, banana]
- Em consultas, a lista pode ser decomposta:
  - [primero elemento | resto dos elementos]
- Denotação de uma lista vazia:
  - []

Listas (2/2)

```
append([], Lista, Lista).

append([Cabeca | Lista1], Lista2,

[Cabeca | Lista3]) :- append(Lista1, Lista2,
Lista3).

?- append([bob, jo], [jake, darcie], Familia),
write(Familia).
```

Família = [bob, jo, jake, darcie]

ybadoo

Deficiências do Prolog (1/3)

- Embora Prolog seja uma ferramenta útil, não deve ser considerada uma linguagem perfeita de programação lógica.
- Controle da ordem de resolução:
  - Prolog sempre começa do início dos fatos e regras.
  - Em linguagens puras, a ordem seria não determinística ou concorrente.
  - Usuário pode moldar para uma solução particular.

Deficiências do Prolog (2/3)

- Controle do backtracking:
  - Operador de corte (cut) representado por uma exclamação.
  - `A, B, !, C, D.`
- A suposição do mundo fechado:
  - Prolog não tem conhecimento além de seu banco de dados (fatos e regras).
  - Quando não tem informações suficientes é assumido ser falso (negação por falha).
  - Pode provar que é verdadeiro, mas nunca que é falso.

Deficiências do Prolog (3/3)

- Problema da negação:
  - `pais(bill, jake).`
  - `pais(bill, shelley).`
  - `irmaos(X, Y) :- pais(P, X), pais(P, Y).`
  - Se fizermos a pergunta `?- irmaos(X, Y).`
  - A resposta seria X = jake e Y = jake.
  - Solução:
    - `irmaos(X, Y) :- pais(P, X), pais(P, Y), not (X = Y).`

Aplicações

- Sistemas Especialistas:
  - Consiste de um banco de dados de fatos, processo de inferência, heurísticas e interfaces.
  - Apropriada para lidar com o problema de inconsistência e incompletude da base de conhecimento.
  - Aprendizado:
    - Pode adicionar fatos e regras e explicar o raciocínio através do princípio da resolução.
- Processamento de Linguagem Natural:
  - Interfaces de linguagem natural para sistemas computacionais como banco de dados inteligentes e sistemas baseados em conhecimento.

Exemplos (1/4)

```
% comentários em Prolog
% write(var) - escrever var na interface de saída
% nl - new line - pular de linha

% Hello world!
saudacao(Texto) :- write(Texto), nl.
?- saudacao("Hello world!").
```

Exemplos (2/4)

```
% Exibir uma caixa de diálogo
dialogo() :-   read(Nome, "Qual o seu nome?", s),
               nl, write("Bom dia "), write(Nome), nl.
?- dialogo().
```

Exemplos (3/4)

```
% Contar o número de elementos de uma lista

quantidade([], 0).

quantidade([CAR|CDR], N) :-   quantidade(CDR, AuxQ),
                              N is AuxQ + 1.

?- quantidade([a, b, c, d], N), write(N), nl.
```

Exemplos (4/4)

```
% Concatenar elementos em uma lista

concatena([], L, L).

concatena([CAR|Lista1], Lista2, [CAR|Lista3]) :-
                        concatena(Lista1, Lista2, Lista3).

?- concatena([a, b, c], [d, e, f], Lista),
   write(Lista), nl.
```