Linguagem de Programação
Introdução à Linguagem Prolog

TURMA A
PROF. MARCELO LADEIRA CIC/UNB

SLIDES MONTADOS A PARTIR DE SLIDES PREPARADOS PARA A JAI96 E DE SLIDES OBTIDOS NA INTERNET – PARTE 2 DE 2

Linguagem Prolog

Programação em lógica

- definições lógicas são vistas como programas.
  - Em Prolog definições lógicas são cláusulas de Horn.

É praxe representar apenas o conhecimento positivo

- asserções afirmativas

Hipótese do Mundo Fechado

- as declarações relevantes e verdadeiras estão contidas na BC ou podem ser derivadas a partir de fatos, regras e a BC.

Negação

- ausência da declaração.
  - Não é uma negação lógica.

Resolução por refutação

P é consistente se falhar a prova de ¬P.

Linguagem Prolog
Implicação

Representação da implicação

Tabela verdade

A	B	A→B
V	V	V
V	F	F
F	V	V
F	F	V

¬A	B	¬A∨B
F	V	V
F	F	F
V	V	V
V	F	V

A→B ⇔ ¬A∨B

Linguagem Prolog
Cláusulas de Horn

Cláusula que tem, no máximo, um literal positivo

"Well Formed Formula" em forma conjuntiva normal e sem conectivo $\wedge$

- prefixo quantificadores universais ($\forall$) aplicado a predicados conectados por $\vee$

Lógica decidível

Notação Lógica	**Cláusulas de Horn**
$\forall$**x(estimação(x)** $\wedge$ **pequeno(x)** $\rightarrow$ **bichoapt(x))**	$\neg$ **estimação(x)** $\vee$ $\neg$**pequeno(x)** $\vee$ **bichoapt(x)**
$\forall$**x(gato(x)** $\vee$ **cachorro(x)** $\rightarrow$ **estimação(x))**	$\neg$ **gato(x)** $\vee$ **estimação(x)** $\neg$**cachorro(x)** $\vee$ **estimação(x)**
$\forall$**x(poodle(x)** $\rightarrow$ **cachorro(x)** $\wedge$ **pequeno(x)**	$\neg$ **poodle(x)** $\vee$ **cachorro(x)** $\neg$ **poodle(x)** $\vee$ **pequeno(x)**

poodle(fido)

poodle(fido)

Linguagem Prolog
Características

Quantificadores universais não explicitados.

Conectivos ‘,’ e ‘;’ (respectivamente $\wedge$ e $\vee$)

A regra p $\rightarrow$ q é representada como q :- p.

cabeça :- lista de predicados.

Cláusulas de Horn	**Notação Prolog**
$\neg$ **estimação(x)** $\vee$ $\neg$**pequeno(x)** $\vee$ **bichoapt(x)**	**bichoapt(X):- estimação(X), pequeno(X).**
$\neg$ **gato(x)** $\vee$ **estimação(x)**	**estimação(X):-gato(X).**
$\neg$**cachorro(x)** $\vee$ **estimação(x)**	**estimação(X):-cachorro(X).**
$\neg$ **poodle(x)** $\vee$ **cachorro(x)**	**cachorro(X):-poodle(X).**
$\neg$ **poodle(x)** $\vee$ **pequeno(x)**	**pequeno(X):-poodle(X).**
poodle(fido)	**poodle(fido).**

Linguagem Prolog
Cláusulas de Horn

Fatos Prolog

- cláusulas de Horn com premissa única True implícita

  C. $\Leftrightarrow$ True $\rightarrow$ C

Regras Prolog

- cláusulas de Horn

  C :- P1, ... ,Pn. $\Leftrightarrow$ P1 & ... & Pn $\rightarrow$ C

Premissas de cláusulas com a mesma conclusão são implicitamente disjuntivas:

C :- P1, ... ,Pn.

C :- Q1, ... ,Qm.

$\Leftrightarrow$ (P1& ... & Pn) v (Q1 & ... & Qm) $\rightarrow$ C

Linguagem Prolog
Máquina de Inferências

Inferência

- resolução por refutação

Seleção das regras

- raciocínio para trás com indexação das regras pelo predicado e casamento (*matching*) precedido pelas instanciações das variáveis das regras (unificação)

Resolução de conflitos

- preferência baseada em regras (ordem)

Interpretador Prolog
Controle e busca

Aplica regra de resolução

- com estratégia *linear* (sempre tenta unificar *ultimo fato* a provar com a *conclusão de uma cláusula do programa*),
- na *ordem de escrita* das cláusulas *no programa*,
- *com encadeamento* de regras *para trás*,
- *busca em profundidade* e
- *da esquerda para direita* das premissas das cláusulas,
- e com *backtracking sistemático* e *linear* quando a unificação falha,
- e *sem occur-check* na unificação.

Estratégia eficiente mas incompleta.

Interpretador Prolog
Verificação de ocorrência

unificação de Prolog é sem *occur-check*, quando chamado com uma variável X e um literal l, instancia X com l, *sem verificar* antes se X ocorre em l.

Junto com a busca em profundidade:

- faz com que Prolog possa entrar em loop com regras recursivas,
  - c(X) :- c(p(X)). gera lista infinita de objetivos:
    - c(p(U)), c(p(p(U))), c(p(p(p(U)))), ...
- cabe ao programador não escrever tais regras,
- torna a unificação linear no lugar de quadrática no tamanho dos termos a unificar

Vantagens

Ampla expressividade

Representação de associações empíricas (heurísticas) em domínios não estruturados

Codificação da experiência de especialistas na resolução de problemas

Possui sintaxe e semântica simples

Aplicação

- sistemas especialistas, em especial, de diagnóstico

Prototipação

- crescimento incremental da BC.

Desvantagens

Falta de estruturação da BC dificulta

- introduzir modificações na BC.
- localizar informações desejadas.
- representar estruturas inerentes ao domínio tais como:
  - taxonomia de classes
  - relações temporais
  - relações estruturais
  - herança de atributos

⇨Regras podem ser modularizadas de forma a se obter subconjuntos independentes e complementares, o que facilita o processo de resolução de conflitos

Não facilita a distinção semântica entre propriedades essenciais e propriedades complementares dos objetos

Estrutura de um Programa Prolog: termos

Usados para construir programas e estruturas de dados

Tipos

- constantes
  - inteiros
  - reais
  - átomos
    - não são variáveis
      - podem ser vistos como strings constantes
- variáveis
- termos compostos

Estrutura de um Programa Prolog: constantes

Atomos e números são definidos sobre os caracteres:

A,B,..,Z a,b,...,z 0,1,...,9
caracteres especiais como + - * / < > = : . & _ ~

Atomos

- strings de letras, dígitos e o underscore, iniciando com MINÚSCULA
  anna x_25 nil
- string de caracteres especiais
  * . = @ # $
- Átomos especiais
  ::== .:. [] !
- strings de caracteres entre aspas simples
  'Tom' 'x_>:'

Números

reais: 3.14 -0.57 1.23 1.23^4
inteiros: 23 5753 -42

Estrutura de um Programa Prolog: variáveis

Iniciam com MAIÚSCULA ou underscore

- seguido de qualquer número de letras, digitos ou "_"
  - X_25 _chica X Barao Fred
- variável anônima (pense como "não importa!")
  - um simples underscore

    haschild(X) :- parent(X,_).
  - comportamento igual ou diferente ?

    likes(mary,_), likes(_,mary).

    likes(mary,X), likes(X,mary).

Estrutura de um Programa Prolog
Termo Composto

Átomo seguido por uma seqüência de um ou mais termos, entre parenteses e separados por vírgula

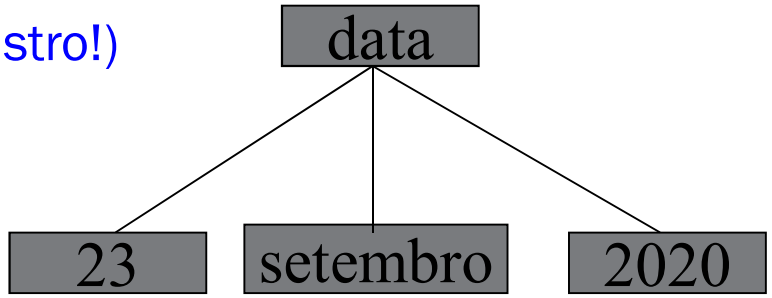
parent(pam,bob).

deseja(cruzeiro,voltar(serie,a)).

- Não é uma função
  - é uma estrutura de dados (tipo um registro!)
- Exemplo
  - entradas via modo interpretador

    assert(data(23, setembro, 2020)).

    data(Dia, setembro, 2020).
    - Dia = 23



Estrutura de um Programa Prolog

Cláusulas Prolog

- fatos, regras ou consultas

Base de conhecimentos (BC)

- fatos
- regras
  - arquivo texto
    - carregado via “consult(arquivo-texto).”

Prompt interpretador

- consultas
  - BC é modificada via

    assert /1 (insere fato ou regra)

    retract /1 (elimina fato ou regra)

Construções Básicas

Fatos Universais

- Fatos com variáveis universalmente quantificadas.
- Functor e átomo iniciam com letra minúscula.

Programa Prolog: fatos e regras

```
mais(0,X,X).                    % fato
mais(A,B,C) :- C is A + B.      % regra
vezes(1,X,X).                   % fato
vezes(A,B,C) :- C is A * B.     % regra
```

Variável

- Está associada a um indivíduo não especificado, é uma incógnita de valor único. É local a uma sentença.

Usando o Prolog

Para usar o Prolog você precisa saber:

- Invocar o Prolog
- Sair do Prolog:

  ^Z

  halt. (retorna ao chamador)
- Usar um editor de arquivo texto para editar um programa
- Carregar um programa

```
load ou load_files(arquivo).      % fonte ou objeto
consult(arquivo).                 % fonte
consult(user).                    % carrega via teclado. Fim ^Z
reconsult(arquivo).               % recarrega arquivo
```

Usando o Prolog

Capacidade especial (de algumas implementações)

- Armazenar a atual BC em arquivo
  - save(arquivo). % salva a BC no arquivo, objeto
  - save_predicates(predicados, arquivo).
    % salva os predicados em arquivo, código
- Recuperar o arquivo salvo
  - restore(arquivo). % recupera o código objeto
  - load ou load_files(arquivo). % objeto

Fato

Relação verdadeira entre termos, via predicado.

- Ex.: gosta(marcelo, leda).
  gosta(marcelo, cruzeiro).

Fato é uma cláusula sem nenhuma condição.

Sintaxe de predicado

- <functor> (t_1,t_2,...,t_n).
  Predicado expressa uma relação entre termos.

Aridade

- Quantidade de termos no predicado. Expressa por <functor>/aridade.
  - Ex.: gosta/2 ou gosta/3
- Predicados com mesmo functor podem ter diferentes aridades:
  gosta(joao, ler, livros).
  gosta(joao,maria).

Regra

cabeça :- corpo.

- o operador ":-" é denominado pescoço (neck)

meta :- sm_1, sm_2,..., sm_k

- meta só é verdadeira se suas submetas também o forem.
- para provar que meta é verdadeira, deve-se provar antes que suas submetas também são.
- meta e submetas são relações predicativas entre termos.
- termos nomeiam objetos do discurso.