Programação Lógica
Parte 2

PLP-2019
Profa. Heloisa

Programação Lógica

- Aritmética em prolog
- Listas
- Unificação de listas
- Operações sobre listas

Programação Lógica

- Convenção de notação
  - A definição de predicados adota um conjunto de meta símbolos, para facilitar o entendimento da forma de uso de cada parâmetro do predicado
  - Meta símbolos são símbolos que NÃO fazem parte da sintaxe do predicado mas são usados na documentação para auxiliar a definição da forma de uso dos argumentos.
  - Utilizaremos os símbolos +, - e ? Para indicar quando o argumento de um predicado deve estar instanciado ou não no momento da execução do predicado.

Programação Lógica

- Convenção de notação
  - Definição de Predicados

    pred(+Arg1, ?Arg2, -Arg3)
  - Modo de Declaração

- \+ Argumento de entrada. Deve estar instanciado quando o predicado é chamado
- \- Argumento de saída. Deve ser uma variável não instanciada quando o predicado é chamado. Se o predicado der sucesso, será instanciada ao valor retornado
- ? Argumento de entrada ou de saída. Pode estar instanciado ou não.

Programação Lógica

- Aritmética em Prolog
- Operadores aritméticos são considerados funtores
  - 2+5 é representado internamente como +(2,5)
  - Para ativar as operações é necessário usar o predicado IS:
  - Sintaxe: X is <expressão>

    onde X (variável)

    <expressão> (expressão aritmética)
  - calcula a expressão e instancia o resultado com a variável X

Programação Lógica

- Aritmética em Prolog
- Exemplos:

```
?- X is 1+2.
X = 3
?- Y is 1+5*(4-2).
Y=11
?- X is 4/2.
X = 2
?- X is 1+2.
X = 3
```

OBS: O predicado IS **NÃO** faz a operação de ATRIBUIÇÃO!

Programação Lógica

- Aritmética em Prolog
- Alguns operadores aritméticos que podem ser usados com is:

X+Y	abs(X)
X-Y	exp(X)
X*Y	ln(X)
X/Y	log(X)
X//Y (divisão inteira)	sin(X)
X^Y (exponenciação)	cos(X)
-X	sqrt(X)
X mod Y	

Programação lógica

- Operadores relacionais
  - E1 e E2 devem ser expressões aritméticas, que são calculadas antes da aplicação do operador

E1 > E2

E1 < E2

E1 >= E2

E1 =< E2

X is E1 calcula E1 e unifica o resultado com X

E1 =:= E2 calcula E1 e E2 e testa igualdade

E1 =\= E2 calcula E1 e E2 e testa desigualdade

Programação lógica

- Operadores relacionais
  - Exemplos

```
?- 2+1 < 6-2.
true.

?- 2+1 > (8/4)+5.
false.

?- 1+2 =:= 2+1.
true.

?- 1+2 =:= X.
ERROR: =:=/2: Arguments are not sufficiently instantiated
```


Programação lógica

- Operadores relacionais
  - Comparação entre termos
  - Predicado = (unifica termos)
  - Sintaxe: Termo1 = Termo2
    onde ?Termo1
    ?Termo2
  - Retorna sucesso se os termos Termo1 e Termo2 unificam.
  - Retorna os valores das variáveis instanciadas, quando elas aparecem em um dos termos.

Programação lógica

- Operadores relacionais
  - Comparação entre termos

```
?- 5 = 5.
true
?- fred = fred.
true.
?- X = Y.
X = Y.
?- pai_de(joao,paulo) = pai_de(X,Y).
X = joao,
Y = paulo.
```

```
?- X = 2+5.
X = 2+5
?- X is 2+5.
X = 7
?- 1+2 = 2+1.
false.
?- 1+2 =:= 2+1.
true.
```


Programação lógica

- Operadores relacionais
  - Comparação entre termos
  - Predicado = = (verifica se dois termos são idênticos)
  - Sintaxe: Termo1 == Termo2
    onde ?Termo1
    ?Termo2
  - Retorna sucesso se Termo 1 e Termo2 são idênticos.
  - As variáveis NÃO são instanciadas.
  - As expressões NÃO são calculadas.

Programação lógica

- Operadores relacionais
  - Comparação entre termos

```
| ?- nome == nome.
true.

?- X == X.
true.

?- X == 5.
false.

?- X = 5.
X = 5

?- X == Y.
False.
```

```
?- pred(1) == pred(X).
false.

?- pred(1) = pred(X).
X = 1.

?- X = 2+1.
X = 2 + 1.

?- X is 2+1.
X = 3.

?- X == 2+1.
false.

?- X =:= 2+1.
ERROR: =:=/2: Arguments are not
sufficiently instantiated
```


Programação lógica

- Operadores relacionais
  - Comparação entre termos
    - Predicado \= = (verifica se dois termos não são idênticos)
    - Sintaxe: Termo1 \= = Termo2
    - Retorna sucesso se Termo 1 e Termo2 NÃO são idênticos.
    - As variáveis NÃO são instanciadas.
    - As expressões NÃO são calculadas.

Programação lógica

- Operadores relacionais
  - Comparação entre termos

```
| ?- X \= = Y.
true.

?- X \= = 5.
true.

?- X \= = X.
false.
```


Programação Lógica

- Listas
  - Principal estrutura da linguagem Prolog
  - É uma sequência ordenada de elementos
  - Pode ter qualquer comprimento
  - Elementos de listas podem ser simples ou estruturados (inclusive listas)
- No Prolog, geralmente são denotadas por colchetes e elementos separados por vírgulas

[ ] (lista vazia)
[a, b, c]
[maria, joao, pedro, carlos]
[1, 329, -15, par(a,b), X, [2, c, Y], 2000]

Programação Lógica

- Listas
  - Listas são divididas em:
  - cabeça - primeiro elemento
  - cauda - o que resta tirando o primeiro elemento
- Exemplos:

[a, b, c]
cabeça: a
cauda: [b,c]

[X, Y, 234, abc]
cabeça: X
cauda: [Y, 234, abc]

Programação Lógica

- Listas
  - As partes da lista são combinadas pelo funtor • (ponto):
    • (Cabeça, Cauda)
  - Essa forma é usada como representação interna, por motivo de padronização da linguagem.
  - Para a programação usamos a forma sintática abreviada, que é equivalente:

[a, b, c] equivale a • (a, • (b, • (c, [ ])))

Programação Lógica

- Padrão de listas
  - Padrão de lista é uma representação genérica em que a barra vertical separa a cabeça da cauda da lista
  - [X|Y] - lista com cabeça X e cauda Y
    - Representa listas com pelo menos um elemento
  - A barra vertical pode separar também mais de um elemento no início da lista do restante da lista
  - [X,Y | Z] - lista com elementos X e Y e cauda Z
    - Representa listas com pelo menos dois elementos

Programação Lógica

- Padrão de listas
  - Símbolos antes da barra são ELEMENTOS
  - Símbolo após a barra é LISTA

Programação Lógica

- Unificação de listas
  - Os padrões de listas são muito utilizados nas operações de unificação
  - Lista 1: [a1, a2, a3, a4]
  - Lista 2: [X | Y]
  - Resultados da unificação:

```
?- [a1,a2,a3,a4] = [X|Y].
X = a1,
Y = [a2, a3, a4].
```


Lista 1	Lista 2	Resultado
[a1, a2, a3, a4]	[X | Y]	X = a1 Y = [a2, a3, a4]
[a1]	[X | Y]	X = a1 Y = [ ]
[ ]	[X | Y]	não unifica
[[a, b], c, d]	[X | Y]	X = [a, b] Y = [c, d]
[[ana, Y] | Z]	[[X, foi], ao, cinema]	X = ana Y = foi Z = [ao, cinema]
[[ana, Y] | Z]	[[X, foi], [ao, cinema]]	X = ana Y = foi Z = [[ao, cinema]]
[a, b, c, d]	[X, Y | Z]	X = a Y = b Z = [c, d]
[ana, maria]	[X, Y | Z]	X = ana Y = maria Z = [ ]
[ana, maria]	[X, Y, Z]	não unifica

Programação Lógica

- Operações sobre listas
  - Operações sobre listas frequentemente usam busca recursiva.
  - São o mecanismo principal para programação em Prolog.
  - O programa é construído com base nas duas partes da lista: *cabeça* e *cauda*.

Programação Lógica

- Operações sobre listas
  - Como construir programas que realizam operações sobre listas?
  - PROBLEMA: Verificar se um elemento é membro de uma lista.
  - O raciocínio para construção de um programa em Prolog começa com a identificação de parâmetros envolvidos (listas, estruturas, elementos, etc)
  - Todos os elementos envolvidos serão argumentos de uma relação que define a operação principal
  - A operação principal é, portanto, definida como uma relação entre os parâmetros envolvidos.

Programação Lógica

- Operações sobre listas
  - Como construir programas que realizam operações sobre listas?
  - PROBLEMA: Verificar se um elemento é membro de uma lista.
    - No exemplo colocado, temos dois parâmetros (objetos):
      - Lista
      - Elemento
    - A operação principal (predicado) é a verificação de pertinência ou não do elemento à lista

Programação Lógica

- Operações sobre listas
  - Como construir programas que realizam operações sobre listas?
  - PROBLEMA: Verificar se um elemento é membro de uma lista.
    - É necessário definir nomes para a relação e para os parâmetros:
      - Relação: pertence
      - Parâmetros: X, L
    - Depois de definidos os elementos envolvidos, estruturamos a solução como um processo recursivo, explorando o recurso de acessar diretamente a cabeça da lista e a cauda da lista

Programação Lógica

- Operações sobre listas
  - Como construir programas que realizam operações sobre listas?
  - PROBLEMA: Verificar se um elemento é membro de uma lista.
    - Estruturação da solução:
      - X é membro de L se:
    - X é a cabeça de L, ou
    - X é membro da cauda de L

Programação Lógica

- Operações sobre listas
- Programa "pertence"

```
pertence(X, [X | Y ]).          % cláusula 1

pertence(X, [ Z | Y]) :- pertence(X,Y).     % cláusula 2
```

Programação Lógica

- Programa "pertence"
  - Após definir o programa é possível consultá-lo:

```
pertence(X, [X | Y ]).   % cláusula 1

pertence(X, [ Z | Y]) :- pertence(X,Y).      % cláusula 2
```

```
?- pertence(a, [1,2,a,c,b]).
true.

?- pertence(a, [1,2,3]).
false.

?- pertence(a, [1,2,3,[a, b, c], 4]).
false.
```

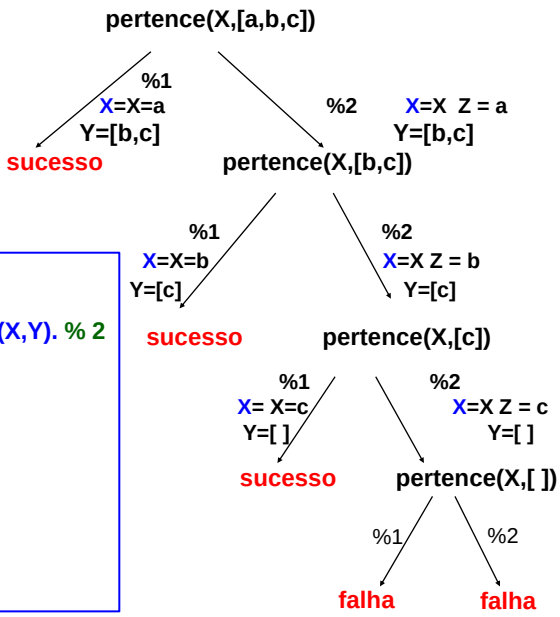

Programação Lógica

- Programa "pertence"
  - Um programa Prolog pode ser consultado de várias formas
  - Esse programa, escrito para verificar se um elemento pertence a uma lista, pode ser usado para recuperar todos os elementos da lista

```
?- pertence (X, [a,b,c]).
X = a ;
X = b ;
X = c ;
false.
```

```
pertence(X, [X | Y ]).   % cláusula 1

pertence(X, [ Z | Y]) :- pertence(X,Y).      % cláusula 2
```




```
pertence(X, [X | Y ]).      % 1

pertence(X, [ Z | Y]) :- pertence(X,Y). % 2
```

Consulta:

```
?- pertence(X,[a,b,c]).
X = a;
X = b;
X = c;
false.
```


Programação Lógica

- Programa "pertence"
  - Variável anônima
- A variável anônima pode ser representada em Prolog pelo caracter _ (undescore)
- Essa variável pode ser usada sempre que o valor instanciado em algum ponto da execução do programa não será utilizado futuramente. Sua função é a eficiência, por reduzir uso de memória e processamento.
- O programa que usa variável anônima gera um resultado equivalente ao que não usa.

Programação Lógica

- Programa "pertence"
  - Variável anônima

```
pertence(X, [X| _ ]).              % cláusula 1

pertence(X, [ _ | Y]) :- pertence(X,Y).      % cláusula 2
```


Programação Lógica

- Concatenação

  Conc(L1, L2, L3)

  Estruturação do problema:

- Se L1 é lista vazia, o resultado da concatenação é igual a L2
- Se L1 não é vazia, é da forma [X|L]. O resultado da concatenação é [X|LR] onde LR é a concatenação de L com L2.

Programação Lógica

- Concatenação
  - Programa:

```
conc([ ], L, L).

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


Programação Lógica

- Concatenação
  - Consultas:

```
?- conc([a,b,c], [1,2], L).
L = [a,b,c,1,2].

?- conc([a,b,c], [1, [X,2]], L).
L = [a,b,c,1,[X,2]].

?- conc([a,b,c,20], [3, z| Z], L).
L = [a, b, c, 20, 3, z|Z].
```

Programação Lógica

- Concatenação
  - É possível consultar esse programa na forma inversa: decompor uma dada lista em duas sublistas

```
?- conc(L1, L2, [a, b, c]).

L1=[]
L2=[a,b,c] ;
L1=[a]
L2=[b,c] ;
L1=[a,b]
L2=[c] ;
L1=[a,b,c]
L2=[] ;
false.
```

```
?- conc([a,b],L1,[a,b,c,d]).
L1 = [c, d].
```

```
?- conc([X,Y],L1,[a,b,c,d]).
X = a,
Y = b,
L1 = [c, d].
```


Programação Lógica

- Adicionar um elemento como último elemento de uma lista:

- add_ultimo(X,[ ],[X]).

```
add_ultimo(X, [X1|Y],[X1|L]) :-
                    add_ultimo(X,Y,L).
```

Programação Lógica

- Adicionar um elemento como último elemento de uma lista.
  - Consultas:

```
?- add_ultimo(x,[a,b,c],L).
L=[a,b,c,x];
false.

?- add_ultimo([g,h],[f,d,i],L).
L = [f,d,i,[g,h]] ;
false.

?- add_ultimo(X,Y,[a,b,c]).
X = c ,
Y = [a,b] ;
false.
```

```
add_ultimo(X,[ ],[X]).

add_ultimo(X, [X1|Y],[X1|L]) :-
                    add_ultimo(X,Y,L).
```


Programação Lógica

- Eliminar um elemento de uma lista:

```
del(X,[X|Y],Y).

del(X,[Y|Cauda],[Y|Cauda1]) :-
                    del(X,Cauda, Cauda1).
```

Programação Lógica

- Eliminar um elemento de uma lista:
  - Consultas:

```
?- del(a,[a,b,a,c],L).
L=[b,a,c] ;
L=[a,b,c] ;
false.

?- del(a,[x,sf,fe,[d,a,c]],L).
false.
```

```
del(X,[X|Y],Y).

del(X,[Y|Cauda],[Y|Cauda1]) :-
                    del(X,Cauda, Cauda1).
```


Programação Lógica

- Somar os elementos de uma lista numérica:
  - Programa:

```
soma([ ],0).
soma([Elem| Cauda], S) :- soma (Cauda,S1),
                          S is S1 + Elem.
```

  - Consulta:

```
?- soma([1,2,3,4,5,6], S).
S = 21.
```

Programação Lógica

- Contar os elementos de uma lista:
  - Programa:

```
conta([ ],0).
conta([ _ | Cauda], N) :- conta(Cauda, N1),
                          N is N1 + 1.
```

  - Consulta:

```
?- conta([1,2,3,4,5,6],C).
C = 6.
```


Programação Lógica

- Eliminar todas as ocorrências de um elemento de uma lista:
  - Programa:

```
del_todas(Elem,[ ],[ ]).
del_todas(Elem, [Elem|Y], Z) :-   del_todas(Elem,Y,Z).
del_todas(Elem,[Elem1|Y], [Elem1|Z]) :- Elem \== Elem1,
                                        del_todas(Elem,Y,Z).
```

  - Consulta:

```
?- del_todas(a, [a,b,a,c],L).
L=[b,c];
false.
```

Programação Lógica

- Retirar todas a repetições de uma lista:
  - Programa:

```
retirar_rep([ ],[ ]).
retirar_rep([Elem|Cauda],[Elem|Cauda1]) :-
        del_todas(Elem,Cauda,Lista),
        retirar_rep(Lista,Cauda1).
```

  - Consulta:

```
?- retirar_rep([a,b,[a],c,b,x,p1,b,a,[a]],Resultado).
Resultado=[a,b,[a],c,x,p1];
false.
```


Programação Lógica

- Contar o número de ocorrências de um dado elemento no primeiro nível de uma lista:
  - Programa:

```
conta_occor(Elem,[ ],0).
conta_occor(Elem,[Elem|Y],N) :-
                              conta_ocorr(Elem,Y,N1),
                              N is N1 + 1.
conta_occor(Elem,[Elem1|Y], N) :-
                               Elem \== Elem1,
                               conta_ocorr(Elem,Y,N).
```

Programação Lógica

- Dada uma lista de números, separar em duas sendo uma com os positivos e o zero, e outra com os negativos

  - Programa:

```
separa([ ],[ ],[ ]).
separa([X|Y],[X|Z],W) :-X >= 0, separa(Y,Z,W).
separa([X|Y],Z,[X|W]) :- X < 0, separa(Y,Z,W).
```

  - Consulta:

```
?- separa([1,3,-5,0,-64,37,0,19,-53],P,N).
P = [1,3,0,37,0,19] ,
N = [-5,-64,-53] ;
false.
```


Programação Lógica

- Dada uma lista de números, separar em duas sendo uma com os positivos e o zero, e outra com os negativos, descartando o zero

  - Programa:

```
separa_sz([ ],[ ],[ ]).
separa_sz([X|Y],[X|Z],W) :-X > 0,  separa_sz(Y,Z,W).
separa_sz([X|Y],Z,[X|W]) :-X < 0,  separa_sz(Y,Z,W).
separa_sz([X|Y],Z,W) :-   X = 0, separa_sz(Y,Z,W).
```

Programação Lógica

- Dada uma lista de números, separar em duas sendo uma com os positivos e o zero, e outra com os negativos, descartando o zero
  - Consulta:

```
?- separa_sz([1,3,-5,0,-64,37,0,19,-53],P,N).
P = [1,3,37,19] ,
N = [-5,-64,-53] ;
false.
```