Aula 5 – Prolog (Parte2)

22705/1001336 - Inteligência Artificial
2019/1 - Turma A
Prof. Dr. Murilo Naldi

naldi@dc.ufscar.br

Agradecimentos

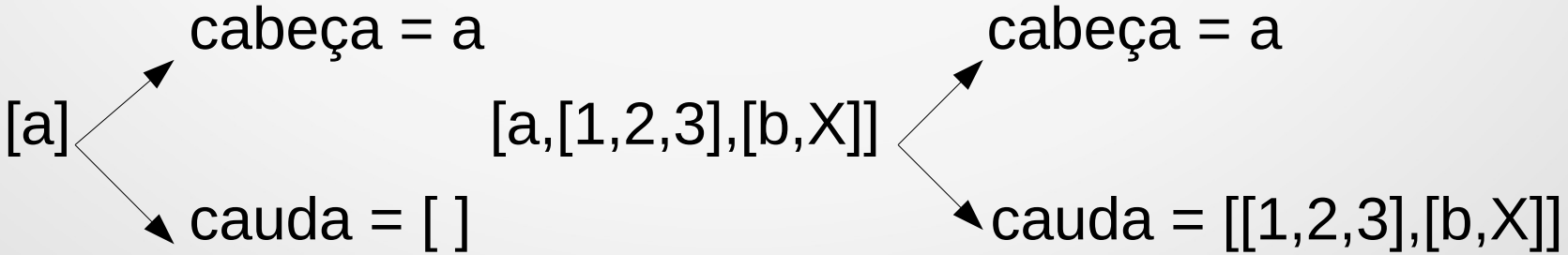
- Agradecimentos pela base do material utilizado nesta aula foi cedido ou adapatado do material dos professores Maria Carmo Nicoletti, Maria Carolina Monard, Solange Rezende, Andréia Bonfante, Heloísa Camargo e Ricardo Cerri.

Listas

- Lista é uma sequência ordenada de elementos e pode ter qualquer comprimento.
  - Pode inclusive ser vazia
- Notação genérica
  - Todos os elementos devem estar entre colchetes
    - Exemplo: [ ], [1], [a], [Variavel]
  - e separados por vírgulas
    - Exemplo:[a,[1,2,3],[b,X]]

Sintaxe para listas

- Cabeça é o primeiro *elemento* (simples ou composto) de uma lista não vazia
- Cauda são os elementos restantes em uma *lista*



Padrão de listas

- A barra vertical separa a cabeça da cauda.
  - [X|Y] representa listas com pelo menos um elemento
- A barra vertical pode separar também vários elementos do início da lista do restante da lista:
  - [X,Y | Z] representa listas com pelo menos dois elementos
- Símbolos antes da barra são ELEMENTOS
- Símbolo após a barra é LISTA

Exemplos

- A cabeça e cauda das listas podem conter elementos simples ou compostos. Essas partes podem ser separadas pelo operador especial “|”.

Lista	Cabeça	Cauda
[gosto,de,vinho]	gosto	[de,vinho]
[[3],5,[2,7]]	[3]	[5,[2,7]]
[X,Y|Z]	X	[Y|Z]

Cauda e lista vazia

- Uma lista *não vazia* possui um ou mais elementos
  - Se *exatamente um elemento*
    - Cauda = [ ]
  - Se *mais de um elemento*
    - Cauda ≠ [ ]
  - Listas vazias não podem ser divididas!

Exemplos

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

Exemplos

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

Exemplos

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]	
[[a, b], c, d]	[X | Y]	
[[ana, Y] | Z]	[[X, foi], ao, cinema]	
[[ana, Y] | Z]	[[X, foi], [ao, cinema]]	
[a, b, c, d]	[X, Y | Z]	
[ana, maria]	[X, Y | Z]	
[ana, maria]	[X, Y, Z]	

Exemplos

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]	
[[ana, Y] | Z]	[[X, foi], ao, cinema]	
[[ana, Y] | Z]	[[X, foi], [ao, cinema]]	
[a, b, c, d]	[X, Y | Z]	
[ana, maria]	[X, Y | Z]	
[ana, maria]	[X, Y, Z]	

Exemplos

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]	
[[ana, Y] | Z]	[[X, foi], [ao, cinema]]	
[a, b, c, d]	[X, Y | Z]	
[ana, maria]	[X, Y | Z]	
[ana, maria]	[X, Y, Z]	

Exemplos

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]]	
[a, b, c, d]	[X, Y | Z]	
[ana, maria]	[X, Y | Z]	
[ana, maria]	[X, Y, Z]	

Exemplos

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]	
[ana, maria]	[X, Y | Z]	
[ana, maria]	[X, Y, Z]	

Exemplos

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]	
[ana, maria]	[X, Y, Z]	

Exemplos

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]	

Exemplos

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

Operações sobre listas

- Listas não possuem acesso direto a todos os seus elementos (por definição)
  - É preciso iterar sobre seus elementos na ordem definida pela própria lista
- Portanto, necessário criar ciclos (laços)
- Prolog é naturalmente recursivo
  - Por meio de regras recursivas
  - Ciclos são feitos recursivamente
  - Portanto, a iteração nas listas é recursiva

Operações em listas

- Vimos na aula passada que regras recursivas (de forma geral) são dividas em dois casos:
  - Caso base: indica o final do processo recursivo
  - Caso recursivo: aplica a recursão e combina resultados
- Depois aplicar na lista
  - Acesso ao elemento da cabeça
  - Aplicar recursão sobre a calda (que é uma lista!).

Cuidado!!!!!


Learn to program
Make recursive function
No exit condition
Learn to program
Make recursive function
No exit condition

Operação 1: Pertence

- Um elemento X pertence ou é membro de lista L se:
  - X for a cabeça de L
  - X pertencer a cauda de L
- Ou seja:
  - X = Cabeça de L e L = [X|Cauda]
  - X ∈ Cauda de L
- Em Prolog
  - regra base ??????
  - regra recursiva ??????

Operação 1: Pertence

- Um elemento X pertence ou é membro de lista L se:
  - X for a cabeça de L
  - X pertencer a cauda de L
- Ou seja:
  - X = Cabeça de L e L = [X|Cauda]
  - X ∈ Cauda de L
- Em Prolog
  - pertence(Elem,[Elem|_]).
  - pertence(Elem,[_| Cauda]) :- pertence(Elem,Cauda).

Exemplo 1: Pertence

- Exemplos:

  ? – pertence(a,[a,b,c,d,t]).

Exemplo 1: Pertence

- Exemplos:

  ? – pertence(a,[a,b,c,d,t]).

  pertence(Elem,[Elem|_]).

Exemplo 1: Pertence

- Exemplos:

  ? – pertence(a,[a,b,c,d,t]).

  pertence(Elem,[Elem|_]).

  pertence(a,[a|[b,c,d,t]]). /*após unificação */

  true

Exemplo 1: Pertence

? - pertence(c,[a,b,c,d,t]).

Exemplo 1: Pertence

```
? - pertence(c,[a,b,c,d,t]).
        pertence(Elem,[Elem|_]). /*primeira regra falha */
        pertence(Elem,[_|Cauda]) :-  pertence(Elem,Cauda).
        pertence(c,[a|[b,c,d,t]]) :- pertence(c,[b,c,d,t]).
```

Exemplo 1: Pertence

```
? - pertence(c,[a,b,c,d,t]).
      pertence(Elem,[Elem|_]). /*primeira regra falha */
      pertence(Elem,[_|Cauda]) :-  pertence(Elem,Cauda).
      pertence(c,[a|[b,c,d,t]]) :- pertence(c,[b,c,d,t]).
        pertence(c,[b,c,d,t]).
        pertence(Elem,[Elem|_]). /*primeira regra falha */
        pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).
        pertence(c,[b|[c,d,t]]) :- pertence(c,[c,d,t]).
```

Exemplo 1: Pertence

```
? - pertence(c,[a,b,c,d,t]).
      pertence(Elem,[Elem|_]). /*primeira regra falha */
      pertence(Elem,[_|Cauda]) :-  pertence(Elem,Cauda).
      pertence(c,[a|[b,c,d,t]]) :- pertence(c,[b,c,d,t]).
        pertence(c,[b,c,d,t]).
        pertence(Elem,[Elem|_]). /*primeira regra falha */
        pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).
        pertence(c,[b|[c,d,t]]) :- pertence(c,[c,d,t]).
          pertence(c,[c,d,t]).
          pertence(Elem,[Elem|_]).
          pertence(c,[c|[d,t]])
          true.
```

Exemplo 1: Pertence

? - pertence(c,[d,b,e]).

Exemplo 1: Pertence

```
? - pertence(c,[d,b,e]).

    pertence(Elem,[Elem|_]). /*primeira regra falha */

    pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).

    pertence(c,[d|[b,e]]) :- pertence(c,[b,e]).
```

Exemplo 1: Pertence

```
? - pertence(c,[d,b,e]).
    pertence(Elem,[Elem|_]). /*primeira regra falha */
    pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).
    pertence(c,[d|[b,e]]) :- pertence(c,[b,e]).
       pertence(c,[b,e]).
       pertence(Elem,[Elem|_]). /*primeira regra falha */
       pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).
       pertence(c,[b|[e]]) :- pertence(c,[e]).
       pertence(c,[e]).
```

Exemplo 1: Pertence

? - pertence(c,[d,b,e]).

pertence(Elem,[Elem|_]). /*primeira regra falha */

pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).

pertence(c,[d|[b,e]]) :- pertence(c,[b,e]).

pertence(c,[b,e]).

pertence(Elem,[Elem|_]). /*primeira regra falha */

pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).

pertence(c,[b|[e]]) :- pertence(c,[e]).

pertence(c,[e]).

pertence(Elem,[Elem|_]). /*primeira regra falha */

pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).

pertence(c,[ ]). /*primeira e segunda regras falham */

false

Exemplo 1: Pertence

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

Exemplo 1: Pertence

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

pertence(Elem,[Elem|_]). /*Unificação*/

pertence(1,[1|[2,3]]).

X = 1;

Exemplo 1: Pertence

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

Não permitidos p/X: 1

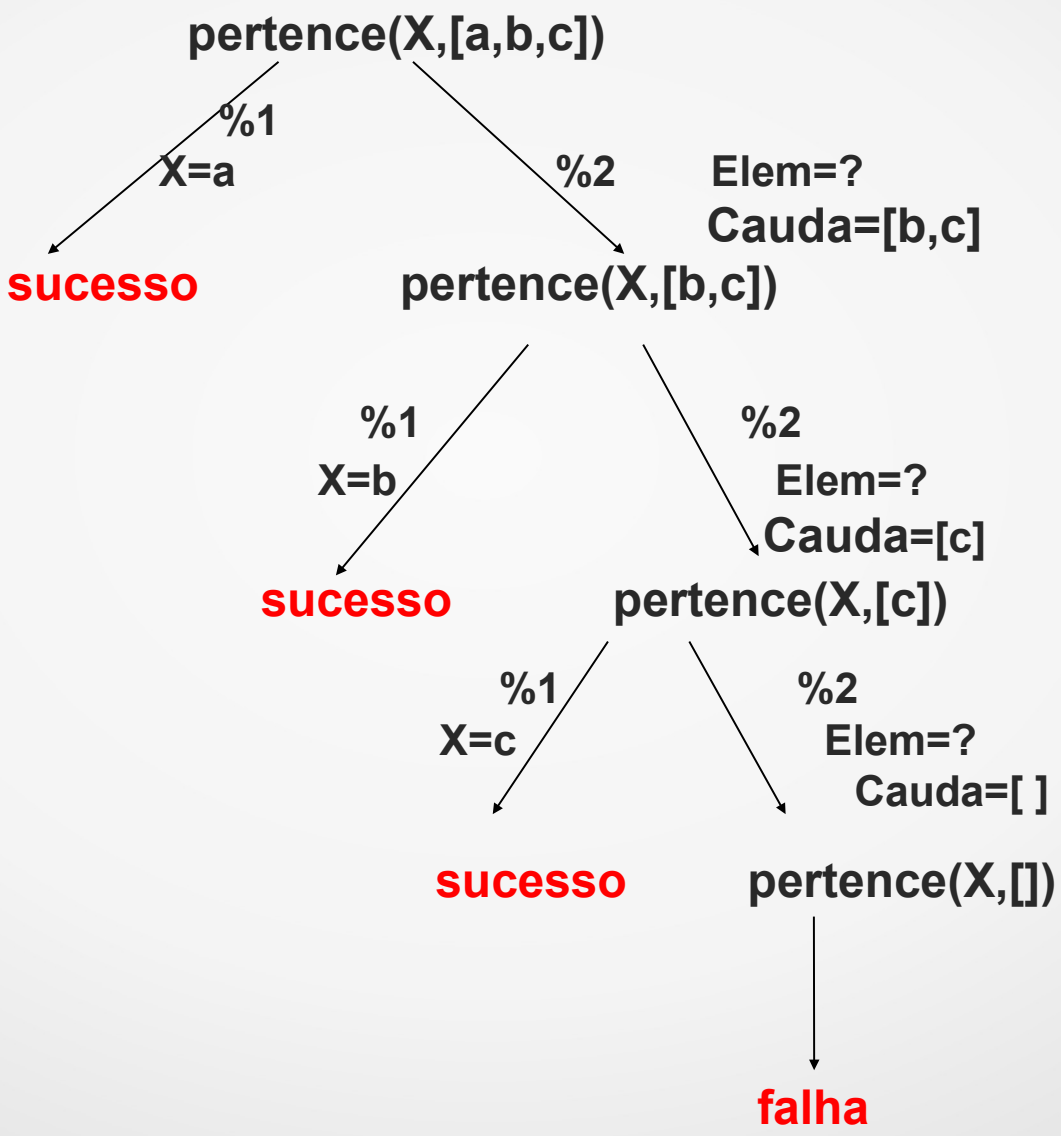
```
pertence(Elem,[Elem|_]). /*Unificação*/
    pertence(1,[1|[2,3]]).
    X = 1;     /* X=1 unificação não permitida e retrocesso*/
        pertence(Elem,[Elem|_]).  /*1ª regra falha com X=1*/
        pertence(Elem,[_|Cauda]) :-pertence(Elem,Cauda).
    pertence(X,[1|[2,3]]) :- pertence(X,[2,3]).
            pertence(X,[2,3]).
            pertence(Elem,[Elem|_]).
            pertence(2,[2|[3]]).
            X = 2;        /*continua...*/
```

Exemplo 1: Pertence

```
pertence(X,[2,3]). /*X=2 não permitido e retrocesso*/
    pertence(Elem,[Elem|_]).  /*X=2, 1ª regra falha */
    pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).
        pertence(X,[2|[3]]) :- pertence(X,[3]).
            pertence(X,[3]).
            pertence(Elem,[Elem|_]).
            pertence(3,[3|[ ]]).
            X = 3;        /*inserir X=3 na lista gera falha */
```

Não permitidos p/X: 1, 2

Exemplo 1: Pertence



Exemplo 1: Pertence

? - pertence(a,L).

Exemplo 1: Pertence

? - pertence(a,L).

pertence(Elem,[Elem|_]).

pertence(a,[a|_]).

L = [a|_];

Exemplo 1: Pertence

```
? - pertence(a,L).
      pertence(Elem,[Elem|_]).
      pertence(a,[a|_]).
 L = [ a|_ ]; /* insere lista não permitidos e retrocesso*/
   pertence(Elem,[Elem|_]). /*1a regra falha*/
   pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).
       pertence(a,[_|a|_]) :- pertence(a,[a|_]).
   L = [ _, a|_ ];
```

Exemplo 1: Pertence

? - pertence(a,L).

pertence(Elem,[Elem|_]).

pertence(a,[a|_]).

L = [a|_]; /* insere lista não permitidos e retrocesso*/

pertence(Elem,[Elem|_]). /*1ª regra falha*/

pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).

pertence(a,[_|a]) :- pertence(a,[a]).

L = [_, a|_];

L = [_, _, a|_]; /*repete processo*/

Exemplo 1: Pertence

```
? - pertence(a,L).
      pertence(Elem,[Elem|_]).
      pertence(a,[a|_]).
 L = [ a|_ ]; /* insere lista não permitidos e retrocesso*/
   pertence(Elem,[Elem|_]). /*1ª regra falha*/
   pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).
        pertence(a,[_|a]) :- pertence(a,[a]).
   L = [ _, a|_ ];
   L = [ _, _, a|_ ]; /*repete processo*/
   L = [ _, _, _, a|_ ]; /*repete processo*/
```

Exemplo 1: Pertence

```
? - pertence(a,L).
      pertence(Elem,[Elem|_]).
      pertence(a,[a|_]).
 L = [ a|_ ]; /* insere lista não permitidos e retrocesso*/
   pertence(Elem,[Elem|_]). /*1ª regra falha*/
   pertence(Elem,[_|Cauda]) :- pertence(Elem,Cauda).
        pertence(a,[_|a]) :- pertence(a,[a]).
   L = [ _, a|_ ];
   L = [ _, _, a|_ ]; /*repete processo*/
   L = [ _, _, _, a|_ ]; /*repete processo*/
   L = [ _, _, _, _, a|_ ]; /*repete processo*/
```

Operação 2: Inserir elemento

- Adiciona na primeira posição da lista:

  inserir_primeiro(X,L,[X|L]).

  - Em que X é um elemento atômico e L é uma lista

  - Exemplo:

    ? - inserir_primeiro(ana,[joao,lucas],R).

    R = [ana,joao,lucas]

Operação 2: Inserir elemento

- Adiciona elemento na última posição

Operação 2: Inserir elemento

- Adiciona elemento na última posição

  add_ultimo(Elem,[ ],[Elem]).

  add_ultimo(Elem, [Cabeça|Cauda],[Cabeça| Cauda_Resultante]) :- add_ultimo(Elem,Cauda, Cauda_Resultante).

Operação 2: Inserir elemento

- Adiciona elemento na última posição

  add_ultimo(Elem,[ ],[Elem]).

  add_ultimo(Elem, [Cabeça|Cauda],[Cabeça| Cauda_Resultante]) :- add_ultimo(Elem,Cauda, Cauda_Resultante).

- ?- add_ultimo(x,[a,b,c],L).

  L=[a,b,c,x];

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

  L = [f,d,i,[g,h]] ;

- ?- add_ultimo(X,Y,[a,b,c]).

  X = c,

  Y = [a,b] ;

Operação 3: Retirar elemento

- Retirar a primeira ocorrência de um elemento

  retirar_elemento(X,L,LR).

  - X é o elemento, L é uma lista, LR é a lista L sem o elemento X

Operação 3: Retirar elemento

- Retirar a primeira ocorrência de um elemento

  retirar_elemento(X,L,LR).

  - X é o elemento, L é uma lista, LR é a lista L sem o elemento X
- Prolog
  - retirar_elemento(Elem,[Elem|Cauda],Cauda).
  - retirar_elemento(Elem,[Cabeça|Cauda], [Cabeça| Resultado]) :- retirar_elemento(Elem,Cauda,Resultado).

Operação 3: Retirar elemento

- Retirar a primeira ocorrência de um elemento

  retirar_elemento(X,L,LR).

  - X é o elemento, L é uma lista, LR é a lista L sem o elemento X
- Prolog
  - retirar_elemento(Elem,[Elem|Cauda],Cauda).
  - retirar_elemento(Elem,[Cabeça|Cauda],[Cabeça|Resultado]) :- retirar_elemento(Elem,Cauda,Resultado).
- Exemplo

  ? - retirar_elemento(ana,[lucas,jose,ana],L).

  L = [lucas, jose] .

Operação 3: Retirar

- Retirar todas as ocorrências de um elemento da lista

Operação 3: Retirar

- Retirar as ocorrências de um elemento da lista
  remover_todos(Elem,[ ],[ ]).

  remover_todos(Elem, [Elem|Cauda1], Resultado) :-
  remover_todos(Elem,Cauda1,Resultado).

  remover_todos(Elem, [Cabeça|Cauda], [Cabeça|Cauda_Resposta]) :- Elem \== Cabeça,
  remover_todos(Elem,Cauda,Cauda_Resposta).
- ?- remover_todos(a, [a,b,a,c],L).

  L=[b,c]

Operação 4: Concatenação

- Concatenação de duas listas (também recursiva):

  concatena(L1,L2,LR).

- L1 e L2 são as listas, LR é lista resultante

Operação 4: Concatenação

- Concatenação de duas listas (também recursiva):

  concatena(L1,L2,LR).

- L1 e L2 são as listas, LR é lista resultante
- Se L1 = [ ] então concatena(L1,L2,L2)
- Senão, L1 = [CabeçaL1|CaudaL1]
  - LR terá
    - cabeça = CabeçaL1
    - cauda = resultado da concatenação de CaudaL1 com L2.

Operação 4: Concatenação

- Em Prolog

```
concatena([ ],L,L).
concatena([Cab|Cauda],L2,[Cab|Resultado]) :-
        concatena(Cauda,L2,Resultado).
```

- Exemplos:

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

Operações aritméticas em lista

- Somar os elementos de uma lista numérica

Operações aritméticas em lista

- Somar os elementos de uma lista numérica

  soma([ ],0).

  soma([Elem| Cauda], S) :- soma(Cauda,S1),

  S is S1 + Elem.

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

  S = 21

Operações aritméticas em listas

- Contar o número de elementos de uma lista

Operações aritméticas em listas

- Contar o número de elementos de uma lista

  conta([ ],0).

  conta([_ | Cauda], N) :- conta(Cauda, N1),

  N is N1 + 1.

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

  C = 6

Número de ocorrências

- Contar o número de ocorrências de um dado elemento de uma lista

Número de ocorrências

- Contar o número de ocorrências de um dado elemento de uma lista

```
conta_ocorr(Elem,[ ],0).

conta_ocorr(Elem,[Elem|Cauda],N) :-
        conta_ocorr(Elem,Cauda,N_Cauda),
                N is N_Cauda + 1.

conta_ocorr(Elem,[Cabeça|Cauda], N) :-
                Elem \== Cabeça,
                conta_ocorr(Elem,Cauda,N).
```

Exercício

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

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

  P = [1,3,0,37,0,19] ,

  N = [-5,-64,-53] ;

  No

Exercício

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

  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).

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

  P = [1,3,0,37,0,19] ,

  N = [-5,-64,-53] ;

  No

Exercício

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

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

  P = [1,3,37,19] ,

  N = [-5,-64,-53] ;

  no

Exercício

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

  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).

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

  P = [1,3,37,19] ,

  N = [-5,-64,-53] ;

  no

Corte do retrocesso

- Retrocesso é ativado pelo Prolog sempre que ele uma união feita previamente não satisfaz uma cláusula futura
  - Mesmo em casos em que não há outra solução
  - Casos que são excludentes:
    - Quando uma regra é bem sucedida, a outra não deve ser
- Nesses casos, o retrocesso é desnecessário
  - Como evitar desperdício de recursos?

Exemplo

- Considere:

$$f(x) \begin{cases} 0 \text{ se } x < 3 \\ 2 \text{ se } x \geq 3 \text{ e } x < 6 \\ 4 \text{ se } x \geq 6 \end{cases}$$

- Em Prolog :

```
f(X,0) :- X < 3.
f(X,2) :- 3 =< X,   X < 6.
f(X,4) :- 6 =< X.
```

Exemplo

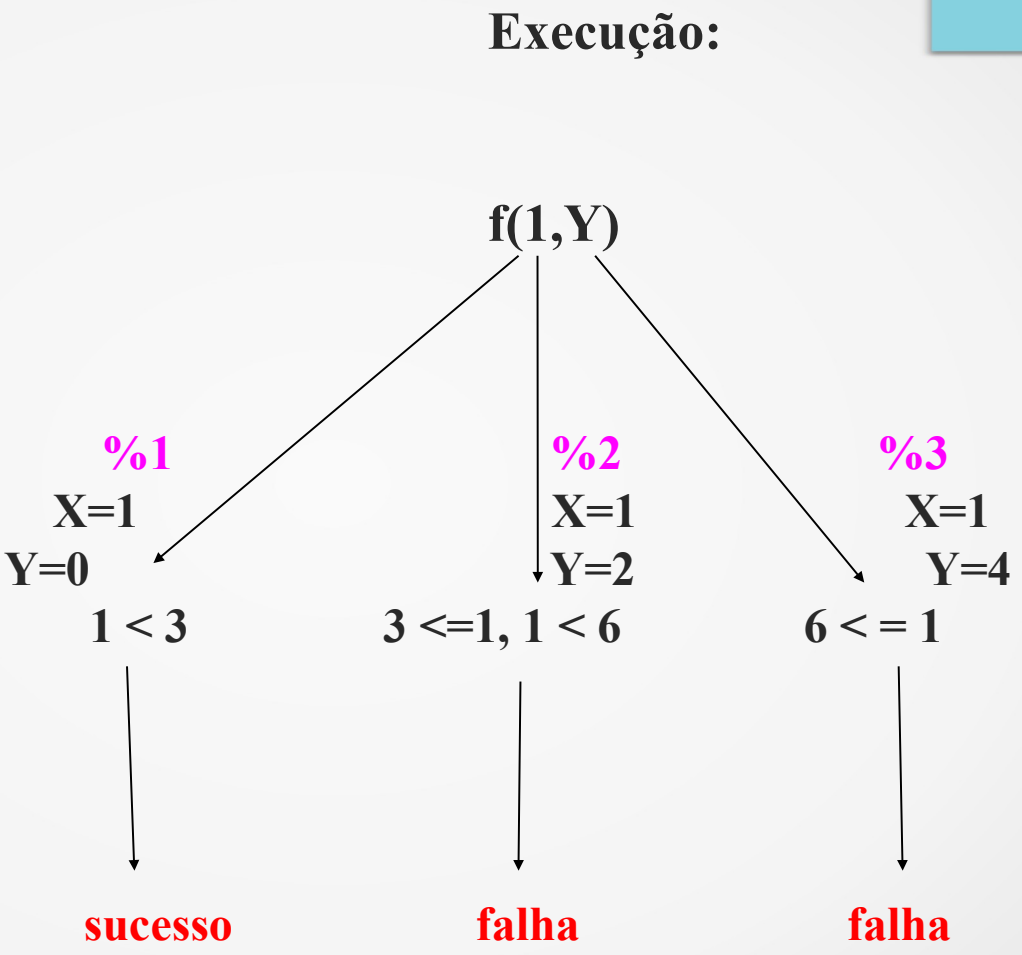
- Ao se interrogar Prolog com:

  ? - f(1,Y), 2<Y.

- Y é instanciado com 0 e a prova 2<0 falha.
- Contudo, antes de responder, Prolog tenta as outras duas alternativas por meio do retrocesso
- Desnecessário pois a própria definição de f(x) faz com que não sejam satisfeitas!

Exemplo

Consulta:
?- f(1,Y).
Y = 0 ;
no



70

Uso de Corte

- Para prevenir retrocesso inútil, o Prolog pode ser direcionado a não realizá-lo:

```
f(X,0) :- X < 3, !.
f(X,2) :- 3 =< X,  X < 6, !.
f(X,4) :- 6 =< X.
```

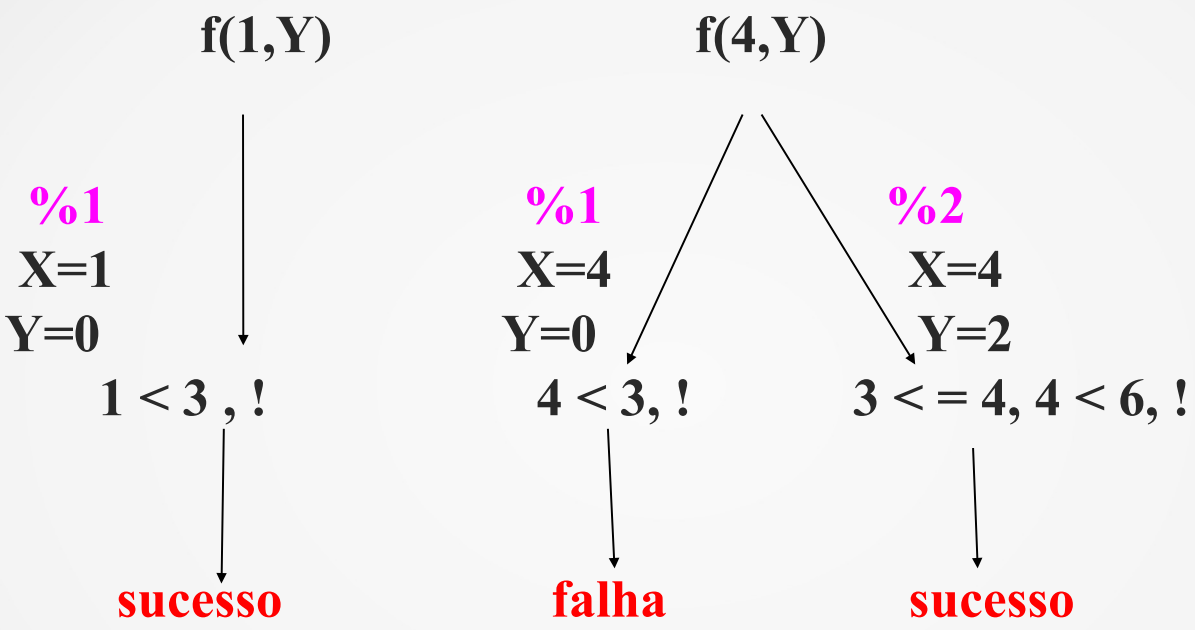
- O Prolog não tentará o retrocesso além do ponto marcado com ! (corte)
- Isso torna a 2ª versão mais eficiente.

Uso de corte

- Existems dois tipos de corte:
  - Verde: não muda o significado das regras se retirado.
  - Vermelho: faz parte integrante da lógica das regras.
- Exemplo corte vermelho:

```
f(X,0) :- X < 3, !.
f(X,2) :- X < 6, !.
f(X,4).
```

Exemplo corte



Este tipo de corte é chamado de corte verde : se for retirado, o programa mantém o mesmo significado. Altera-se apenas a eficiência da execução.

Outros exemplos

- Regras:

```
p(1).
p(2) :- !.
p(3).
```

- Consulta:

```
?- p(X), p(Y).
```

```
X = 1   X = 1   X = 2   X = 2
Y = 1 ; Y = 2 ; Y = 1 ; Y = 2 ;
```

Outros exemplos

- Regras:

```
p(1).
p(2) :- !.
 p(3).
```

- Consulta:

```
?- p(X), !, p(Y).
```

```
X = 1   X = 1
Y = 1 ; Y = 2 ;
```

Número de ocorrências c/ corte

- Versão de contar ocorrências com corte

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

Eliminar ocorrências c/ corte

- Eliminar ocorrências usando corte

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

Separar em listas c/ corte

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

```
separa([ ],[ ],[ ]) :- !.

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

separa([X|Y],Z,[X|W]) :- separa(Y,Z,W).

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

P = [1,3,0,37,0,19] ,

N = [-5,-64,-53] ;

no
```

Fatorial c/ corte

- Fatorial

  fatorial(N,F) :- N>=0, fat(N,F).

  fat(0,1) :- ! .

  fat(N,F) :- N1 is N – 1, fat(N1,F1), F is F1 * N.

- | ?- fatorial(0,F).

  F = 1

  | ?- fatorial(3,F).

  F = 6

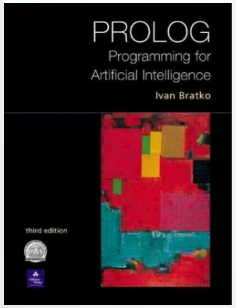
  | ?- fatorial(-4,F).

  no

Exercícios

- Implemente funções que:
  - Retorne o último elemento de uma lista
  - Indique que dois elementos são consecutivos
  - Insira um elemento em qualquer posição da lista
  - Outras??
    - Olhar operações em lista na Cartilha Prolog!

Bibliografia indicada



- NICOLETTI, M. C. A Cartilha Prolog. EDUFSCAR. 2005. ISBN 8576000113
- Bratko – Prolog: Programming for Artificial Intelligence 2001
- Swi-prolog para download: http://www.swi-prolog.org/