Listas

Prof. Paulo H R Gabriel
Programação Lógica
UFU - 2019

1

Créditos

- O material a seguir consiste de adaptações e extensões dos originais gentilmente cedidos pelo Prof. Alexsandro Soares e pela Prof. Elaine Feria
- Agradecimento especial ao Prof. Gabriel Coutinho que auxiliou na confecção do material

2

Lista

- É uma seqüência finita de elementos
  - Ex: [brasil, uruguai, argentina, paraguai]
    - Os itens são separados por vírgula e delimitados por colchetes
    - Cabeça: brasil
    - Cauda: [uruguai, argentina,paraguai]
    - Comprimento: nro de elementos → 4
  - Ex: [maria, [vicente, julio], [beto, amigo(beto)]]
  - Lista vazia: []

3

Lista – cabeça e cauda

- Uma lista não vazia consiste de duas partes:
  - A cabeça
  - A cauda
- A cabeça é o primeiro elemento
- O corpo é o resto
  - A cauda é a lista que sobra quando retiramos o primeiro elemento
  - A cauda de uma lista sempre é uma lista

4

Lista – cabeça e cauda

- [maria, vicente, julio, iolanda]

  Cabeça:
  Cauda:

5

Lista – cabeça e cauda

- [maria, vicente, julio, iolanda]

  Cabeça: maria
  Cauda:

6

Lista – cabeça e cauda

- [maria, vicente, julio, iolanda]

  Cabeça: maria
  Cauda: [vicente, julio, iolanda]

7

Lista – cabeça e cauda

- [[ ], morto(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Cabeça:
  Cauda:

8

Lista – cabeça e cauda

- [[ ], morto(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Cabeça: [ ]
  Cauda:

9

Lista – cabeça e cauda

- [[ ], morto(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Cabeça: [ ]
  Cauda: [morto(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

10

Lista – cabeça e cauda

- [morto(z)]

  Cabeça:
  Cauda:

11

Lista – cabeça e cauda

- [morto(z)]

  Cabeça: morto(z)
  Cauda:

12

Lista – cabeça e cauda

- [morto(z)]

  Cabeça: morto(z)
  Cauda: []

13

Lista – cabeça e cauda

- A lista vazia não possui cabeça nem cauda
- Para o Prolog, [ ] é uma lista simples especial sem qualquer estrutura interna
- A lista vazia desempenha um importante papel nos predicados recursivos para processamento de listas em Prolog

14

Operador |

- Prolog possui um operador especial **|** que pode ser usado para decompor uma lista em sua cabeça e cauda
- O operador | é uma peça chave na escrita em Prolog de predicados para a manipulação de listas

15

Operador |

```
?- [Cab|Cauda] = [maria, vicente, julio, iolanda].

Cab = maria
Cauda = [vicente, julio, iolanda]
true

?-
```

16

Operador |

```
?- [X|Y] = [maria, vicente, julio, iolanda].

X = maria
Y = [vicente,julio,iolanda]
true

?-
```

17

Operador |

```
?- [X|Y] = [ ].

false

?-
```

18

Operador |

```
?- [X,Y|Cauda] = [[ ], morto(Z), [2, [b,c]], [], Z, [2, [b,c]]] .

X = [ ]
Y = morto(z)
Z = _4543
Cauda = [[2, [b,c]], [ ], Z, [2, [b,c]]]


?-
```

19

Lista

- Notação para lista
  - [H | T] : H cabeça e T o corpo
  - Ex: [X|Y] unifica com [a,b,c,d]
  - Ex: [X | [Y | Z]] unifica com [a,b,c,d]
  - Ex: [X,Y,Z] não unifica com [a,b,c,d]

20

Operações sobre listas

- Predicado constroi: Construção de listas
  constroi(X,Y, [X|Y])

  - Ex: constroi (a, b, Z).
    Z = [a | b]
  - Ex: constroi (a, [b,c], Z).
    Z = [a,b,c]
  - Ex: constroi(a, [], Z).
    Z = [a]
  - Ex: constroi(a, X, [a,b,c]).
    X = [b,c]
  - Ex: constroi([a,b,c], [d,e],Z).
    Z = [[a,b,c],d,e]

21

Operações sobre listas

- Ocorrência de elementos em uma lista
  - membro (X, L)
    - X é um objeto e L é uma lista
    - Retorna verdadeiro se X ocorre em L
    - Ex: membro(b, [a,b,c])
    - Ex: membro([b,c], [a, [b,c], d])

    - Ex: membro(b,[a,[b,c]]) é falsa

22

Operações sobre listas

- Ocorrência de elementos em uma lista
  - X é membro de L se
    - X é cabeça de L, ou
    - X é membro do corpo de L
  - Em Prolog

```
membro(X,[X | T]).
membro(X,[H|C]):-
        membro(X,C).
```

23

Variáveis anônimas

- Assuma que estejamos interessados no segundo e quarto elementos de uma lista

```
?- [X1,X2,X3,X4|Cauda] = [maria, vicente, marcelo, jonas, iolanda].
X1 = maria
X2 = vicente
X3 = marcelo
X4 = jonas
Cauda = [iolanda]

?-
```

24

Variáveis anônimas

- Existe uma forma mais simples de obter somente a informação que desejamos:

```
?- [ _,X2, _,X4|_ ] = [maria, vicente, marcelo, jonas, iolanda].
X2 = vicente
X4 = jonas

?-
```

- O sublinhado é a variável anônima

25

Variáveis anônimas

- É usada quando se necessita utilizar uma variável, mas não se está interessado no valor que o Prolog instanciou para ela
- Cada ocorrência da variável anônima é independente, i.e., pode ser ligada a algo diferente

26

Reescrevendo membro/2

```
membro(X,[X|_]).
membro(X,[_|T]):- membro(X,T).
```

27

Reescrevendo membro/2

```
membro(X,[X|_]).
membro(X,[_|T]):- membro(X,T).
```

```
?- membro(iolanda,[iolanda,patricia,vicente,julio]).
true
```

28

Reescrevendo membro/2

```
membro(X,[X|_]).
membro(X,[_|T]):- membro(X,T).
```

```
?- membro(vicente,[iolanda,patricia,vicente,julio]).
```

29

Reescrevendo membro/2

```
membro(X,[X|_]).
membro(X,[_|T]):- membro(X,T).
```

```
?- membro(vicente,[iolanda,patricia,vicente,julio]).
true
```

30

Reescrevendo membro/2

```
membro(X,[X|_]).
membro(X,[_|T]):- membro(X,T).
```

```
?- membro(zeca,[iolanda,patricia,vicente,julio]).
```

31

Reescrevendo membro/2

```
membro(X,[X|_]).
membro(X,[_|T]):- membro(X,T).
```

```
?- membro(zeca,[iolanda,patricia,vicente,julio]).
false
```

32

Reescrevendo membro/2

```
membro(X,[X|_]).
membro(X,[_|T]):- membro(X,T).
```

```
?- membro(X,[iolanda,patricia,vicente,julio]).
```

33

Reescrevendo membro/2

```
membro(X,[X|_]).
membro(X,[_|T]):- membro(X,T).
```

```
?- membro(X,[iolanda,patricia,vicente,julio]).
X = iolanda;
X = patricia;
X = vicente;
X = julio.
```

34

Exemplo a_para_b/2

- O predicado `a_para_b/2` recebe duas listas como argumentos e é verdadeiro se:
  - O primeiro argumento é uma lista de `a`'s, e
  - O segundo argumento é uma lista de `b`'s, com exatamente o mesmo comprimento do primeiro

```
?- a_para_b([a,a,a,a],[b,b,b,b]).
true
?- a_para_b([a,a,a,a],[b,b,b]).
false
?- a_para_b([a,c,a,a],[b,b,b,t]).
false
```

35

Exemplo a_para_b/2

```
a_para_b([],[]).
```

- Muitas vezes o melhor modo de resolver tais problemas é pensar no caso mais simples possível.
- Aqui isto significa: a lista vazia

36

Exemplo a_para_b/2

```
a_para_b([],[]).
a_para_b([a|L1],[b|L2]):- a_para_b(L1,L2).
```

- Agora, pense recursivamente!
- Quando a_para_b/2 deveria decidir se duas listas não vazias são uma lista de `a`'s e uma lista de `b`'s com exatamente o mesmo tamanho?

37

Exemplo a_para_b/2

```
a_para_b([],[]).
a_para_b([a|L1],[b|L2]):- a_para_b(L1,L2).
```

```
?- a_para_b([a,a,a],[b,b,b]).
true
?-
```

38

Exemplo a_para_b/2

```
a_para_b([],[]).
a_para_b([a|L1],[b|L2]):- a_para_b(L1,L2).
```

```
?- a_para_b([a,a,a,a],[b,b,b]).
false
?-
```

39

Exemplo a_para_b/2

```
a_para_b([],[]).
a_para_b([a|L1],[b|L2]):- a_para_b(L1,L2).
```

```
?- a_para_b([a,t,a,a],[b,b,b,c]).
false
?-
```

40

Exemplo a_para_b/2

```
a_para_b([],[]).
a_para_b([a|L1],[b|L2]):- a_para_b(L1,L2).
```

```
?- a_para_b([a,a,a,a,a], X).
X = [b,b,b,b,b]
true
?-
```

41

Exemplo a_para_b/2

```
a_para_b([],[]).
a_para_b([a|L1],[b|L2]):- a_para_b(L1,L2).
```

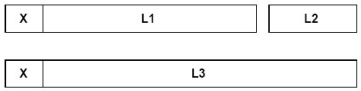
```
?- a_para_b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
true
?-
```

42

Concatenação de Listas

- Relação concatenar

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



Luis, A. M. Palazzo, Introdução à Programação Prolog, Educat, 1997.

43

Concatenação de Listas

- Execução

```
?-conc([a, b, c], [1, 2, 3], L).
    L=[a, b, c, 1, 2, 3]

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

?-conc([a, b], [c | R], L).
    L=[a, b, c | R]

?- 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
```

44

Concatenação de Listas

```
?-M=[jan,fev,mar,abr,mai,jun,jul,ago,set,out,nov,dez],
  conc(Antes, [mai | Depois], M).
        Antes=[jan,fev,mar,abr]
       Depois=[jun,jul,ago,set,out,nov, dez]

?-conc(_, [X, g, Y | _], [a, b, c, d, e, f, g, h]).
        X=f Y=h

?-conc(Trab, [sex | _], [seg,ter,qua,qui,sex,sab,dom]).
        Trab=[seg,ter,qua,qui]
```

45

Remoção de elementos da lista

remover(X, L, L1)

L1 é a mesma lista L com o elemento X removido

- A relação remover/3 pode ser definida
  - Se X é a cabeça da lista L, então L1 será o seu corpo;
  - Se X está no corpo de L, então L1 é obtida removendo X desse corpo

46

Remoção de elementos da lista

- Relação remover

```
remover(X, [X | C], C).
remover(X, [Y | C], [Y | D]) :-
    remover(X, C, D).
```

- Exemplo

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

47

Remoção de elementos da lista

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

48

O predicado findall/3

- Pode-se gerar, através de *backtracking*, todos os objetos, um a um, que satisfazem algum objetivo
- Cada vez que uma nova solução é gerada, a anterior desaparece e não é mais acessível

- O que fazer quando deseja-se dispor de todos os objetos gerados?

49

O predicado findall/3

findall(X, P, L)

produz a lista L de todos os objetos X que satisfazem P

Exemplo:

```
classe(a, vog).
classe(b, con).
classe(c, con).
classe(d, con).
classe(e, vog).

?-findall(Letra, classe(Letra, Classe), Letras).
      Letras=[a, b, c, ..., z]
```

50

Referências

- Luis, A. M. Palazzo, Introdução à Programação Prolog, Educat, 1997.

51