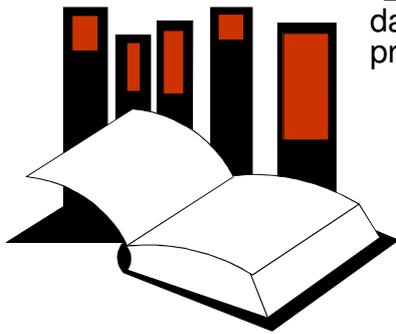


Listas em Prolog



Inteligência Artificial

- Esta aula trata da estrutura de dados lista e programas Prolog para processamento de listas

1

Listas

- Lista é **uma das estruturas mais simples** em Prolog, muito comum em programação não numérica
 - Ela é uma **seqüência ordenada de elementos**
 - Uma lista pode ter **qualquer comprimento**
 - Por exemplo uma lista de elementos tais como **ana, tênis, pedro** pode ser escrita em Prolog como:
 - [ana, tênis, pedro]

2

Listas

- O uso de colchetes é apenas uma melhoria da notação, pois internamente listas são representadas como **árvores**, assim como todos os objetos estruturados em Prolog
- Para entender a representação Prolog de listas, é necessário considerar **dois casos**
 - A lista é vazia, escrita como **[]** em Prolog
 - Uma lista (não vazia) consiste:
 - no primeiro item, chamado **cabeça** (*head*) da lista
 - na parte restante da lista, chamada **cauda** (*tail*)

3

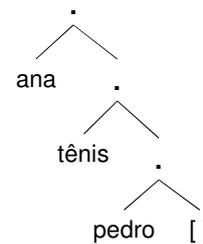
Listas

- No exemplo [ana, tênis, pedro]
 - **ana** é a Cabeça da lista
 - **[tênis, pedro]** é a Cauda da lista
- A **cabeça** de uma lista pode ser **qualquer objeto** (inclusive uma lista); a **cauda tem que ser uma lista**
- A Cabeça e a Cauda são então combinadas em uma estrutura pelo **functor especial .**
 - **.(Cabeça, Cauda)**
- Como a Cauda é uma lista, ela é vazia ou ela tem sua própria cabeça e sua cauda

4

Listas

- Assim, para representar listas de qualquer comprimento, nenhum princípio adicional é necessário
- O exemplo [ana, tênis, pedro] é representando como o termo:
– `.(ana, .(tênis, .(pedro, [])))`
- O programador pode escolher ambas notações



5

Listas

- ?- Lista1 = [a,b,c],
Lista2 = .(a,.(b,.(c,[]))).
Lista1 = [a, b, c]
Lista2 = [a, b, c]
- ?- Hobbies1 = .(tênis, .(música,[])),
Hobbies2 = [esqui, comida],
L = [ana,Hobbies1,pedro,Hobbies2].
Hobbies1 = [tênis,música]
Hobbies2 = [esqui,comida]
L = [ana, [tênis,música], pedro, [esqui,comida]]

6

Listas

- Em geral, é comum tratar a **cauda como um objeto simples**
- Por exemplo, $L = [a,b,c]$ pode ser escrito como
 - $\text{Cauda} = [b,c]$
 - $L = .(a,\text{Cauda})$
- Para expressar isso, Prolog fornece uma **notação alternativa, a barra vertical**, que separa a cabeça da cauda
 - $L = [a \mid \text{Cauda}]$
- A notação é geral por permitir que qualquer número de elementos seja seguido por '|' e o restante da lista:
 - $[a,b,c] = [a \mid [b,c]] = [a,b \mid [c]] = [a,b,c \mid []]$

7

Cabeça e Cauda - Exemplo 1

- $[maria, vicente, julia, yolanda]$

Cabeça:

Cauda:

8

Cabeça e Cauda - Exemplo 1

- [maria, vicente, julia, yolanda]

Cabeça: maria

Cauda:

9

Cabeça e Cauda - Exemplo 1

- [maria, vicente, julia, yolanda]

Cabeça: maria

Cauda: [vicente, julia, yolanda]

10

Cabeça e Cauda - Exemplo 2

- $[[], \text{dead}(z), [2, [b,c]], [], Z, [2, [b,c]]]$

Cabeça:

Cauda:

11

Cabeça e Cauda - Exemplo 2

- $[[], \text{dead}(z), [2, [b,c]], [], Z, [2, [b,c]]]$

Cabeça: $[]$

Cauda:

12

Cabeça e Cauda - Exemplo 2

- $[[], \text{dead}(z), [2, [b,c]], [], Z, [2, [b,c]]]$

Cabeça: $[]$

Cauda: $[\text{dead}(z), [2, [b,c]], [], Z, [2, [b,c]]]$

13

Cabeça e Cauda - Exemplo 3

- $[\text{dead}(z)]$

Cabeça:

Cauda:

14

Cabeça e Cauda - Exemplo 3

- [dead(z)]

Cabeça: dead(z)

Cauda:

15

Cabeça e Cauda - Exemplo 3

- [dead(z)]

Cabeça: dead(z)

Cauda: []

16

O operador built-in |

- Prolog tem um built-in **operador especial** | o qual pode ser usado para decompor uma lista em sua Cabeça e Cauda

17

O operador built-in |

```
?- [Cabeça|Cauda] = [maria, vicente, julia, yolanda].
```

```
Cabeça = maria
```

```
Cauda = [vicente,julia,yolanda]
```

```
yes
```

```
?-
```

18

O operador built-in |

```
?- [X|Y] = [maria, vicente, julia, yolanda].
```

```
X = maria
```

```
Y = [vicente,julia,yolanda]
```

```
yes
```

```
?-
```

19

O operador built-in |

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

```
no
```

```
?-
```

20

O operador built-in |

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

```
X = [ ]
```

```
Y = dead(z)
```

```
Cauda = [[2, [b,c]], [ ], [2, [b,c]]]
```

```
yes
```

```
?-
```

21

Unificação em Listas

Lista1	Lista2	Lista1 = Lista2
[mesa]	[X Y]	X=mesa Y=[]
[a,b,c,d]	[X,Y Z]	X=a Y=b Z=[c,d]
[[ana,Y] Z]	[[X,foi],[ao,cinema]]	X=ana Y=foi Z=[[ao,cinema]]
[ano,bissexto]	[X,Y Z]	X=ano Y=bissexto Z=[]
[ano,bissexto]	[X,Y,Z]	Não unifica

22

Variável Anônima

- Suponha que estamos interessados no segundo e quarto elemento de uma lista

```
?- [X1,X2,X3,X4|Cada] = [maria, vicente, marcelo, josy, yolanda].
```

```
X1 = maria
```

```
X2 = vicente
```

```
X3 = marcelo
```

```
X4 = josy
```

```
Cada = [yolanda]
```

```
yes
```

```
?-
```

23

Variável Anônima

- Há uma maneira mais simples de obter somente a informação que queremos:

```
?- [_,X2,_,X4_] = [maria, vicente, marcelo, josy, yolanda].
```

```
X2 = vicente
```

```
X4 = josy
```

```
yes
```

```
?-
```

24

Operações em Listas

- Frequentemente, é necessário realizar **operações em listas**, por exemplo, buscar um elemento que faz parte de uma lista
 - Para isso, a **recursão** é o recurso mais amplamente **empregado**
 - **Questão:** como verificar se um elemento está em uma lista?

25

Operações em Listas

- Frequentemente, é necessário realizar **operações em listas**, por exemplo, buscar um elemento que faz parte de uma lista
 - Para isso, a **recursão** é o recurso mais amplamente **empregado**
 - Para verificar se um elemento está na lista, é preciso verificar se ele está na cabeça ou se ele está na cauda da lista
 - Se o final da lista for atingido, o elemento não está na lista

26

Predicado de Pertinência

- Inicialmente, é necessário definir o nome do predicado que verifica se um elemento pertence ou não a uma lista, por exemplo, **ptence(X,Y)**
 - A primeira condição especifica que **um elemento X pertence à lista se ele está na cabeça** dela. Isso é indicado como:
 - **ptence(X,[X|Z]).**
 - A segunda condição especifica que **um elemento X pertence à lista se ele pertencer à sua cauda**. Isso pode ser indicado como:
 - **ptence(X,[W|Z]) :-
ptence(X,Z).**

27

Predicado de Pertinência

- Sempre que um procedimento recursivo é definido, deve-se procurar pelas **condições limites** (ou **condições de parada**) e pelo caso recursivo:
 - ptence(Cabeca, [Cabeca|Cauda]).**
 - ptence(Cabeca, [OutraCabeca|Cauda]) :-
ptence(Cabeca,Cauda).**
- Após a definição do procedimento **ptence/2**, é possível interrogá-lo.
 - ?- ptence(a,[a,b,c]).**
 - yes**

28

Predicado de Pertinência

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

no

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

X = a ;

X = b ;

X = c ;

no

- Entretanto, se as interrogações forem:
 - ?- pertence(a,X).
 - ?- pertence(X,Y).
- deve-se observar que cada uma delas tem infinitas respostas, pois existem infinitas listas que validam essas interrogações para o procedimento **pertence/2**₂₉

Exemplo pertence/2

```
pertence(X,[X|T]).
```

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

?-

Exemplo pertence/2

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

```
?- pertence(yolanda,[yolanda,tadeu,vicente,julia]).
```

31

Exemplo pertence/2

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

```
?- pertence(yolanda,[yolanda,tadeu,vicente,julia]).
```

yes

```
?-
```

32

Exemplo pertence/2

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

```
?- pertence(vicente,[yolanda,tadeu,vicente,julia]).
```

33

Exemplo pertence/2

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

```
?- pertence(vicente,[yolanda,tadeu,vicente,julia]).
```

```
yes
```

```
?-
```

34

Exemplo pertence/2

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

```
?- pertence(zeus,[yolanda,tadeu,vicente,julia]).
```

35

Exemplo pertence/2

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

```
?- pertence(zeus,[yolanda,tadeu,vicente,julia]).
```

no

```
?-
```

36

Exemplo pertence/2

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

```
?- pertence(X,[yolanda,tadeu,vicente,julia]).
```

37

Exemplo pertence/2

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

```
?- pertence(X,[yolanda,tadeu,vicente,julia]).
```

```
X = yolanda;
```

```
X = tadeu;
```

```
X = vicente;
```

```
X = julia;
```

```
no
```

38

Modos de Ativação de Procedimentos

- Para **documentar as possíveis instanciações** de variáveis para as quais o procedimento é correto, utiliza-se a seguinte notação (como comentário no programa):
 - + o argumento é de entrada (deve estar instanciado)
 - - o argumento é de saída (não deve estar instanciado)
 - ? o argumento é de entrada e saída (pode ou não estar instanciado)
- O procedimento **pertence/2** documentado com o modo de chamada para atingir a condição de parada é:

```
% pertence(?Elemento, +Lista)
pertence(E, [E|_]).
pertence(E, [_|Cauda]) :-
    pertence(E,Cauda).
```

39

Exercício

- Inserção de um elemento na primeira posição de uma lista

40

Exemplo: Inserção

- Inserção na primeira posição:

```
insere(X, L, [X|L]).
```

```
?- insere(a, [b,c,d], Y).
```

```
Y=[a,b,c,d];
```

```
no
```

```
X = a
```

```
L = [b, c, d]
```

```
Y = [X|L] = [a, b, c, d]
```