Listas em Prolog

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

Inteligência Artificial

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]

Listas

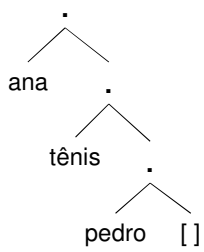
- O uso de colchetes é apenas uma melhoria da notação, pois internamente listas são representadas como **árvores**, assim como todos os objetos estruturados em Prolog
- Para entender a representação Prolog de listas, é necessário considerar dois casos
  - A lista é vazia, escrita como **[ ]** em Prolog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

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

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



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]]

Listas

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

Cabeça e Cauda - Exemplo 1

- [maria, vicente, julia, yolanda]

  Cabeça:
  Cauda:

Cabeça e Cauda - Exemplo 1

- [maria, vicente, julia, yolanda]

  Cabeça: maria
  Cauda:

Cabeça e Cauda - Exemplo 1

- [maria, vicente, julia, yolanda]

  Cabeça: maria
  Cauda: [vicente, julia, yolanda]

Cabeça e Cauda - Exemplo 2

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

  Cabeça:
  Cauda:

Cabeça e Cauda - Exemplo 2

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

  Cabeça: [ ]
  Cauda:

Cabeça e Cauda - Exemplo 2

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

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

Cabeça e Cauda - Exemplo 3

- [dead(z)]

  Cabeça:
  Cauda:

Cabeça e Cauda - Exemplo 3

- [dead(z)]

  Cabeça: dead(z)
  Cauda:

Cabeça e Cauda - Exemplo 3

- [dead(z)]

  Cabeça: dead(z)
  Cauda: [ ]

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

O operador built-in |

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

Cabeça = maria
Cauda = [vicente,julia,yolanda]
yes

?-
```

O operador built-in |

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

X = maria
Y = [vicente,julia,yolanda]
yes

?-
```


O operador built-in |

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

no

?-
```

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

?-
```


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

Variável Anônima

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

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

?-
```


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

?-
```

Operações em Listas

- Freqüentemente, 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?

Operações em Listas

- Freqüentemente, 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

Predicado de Pertinência

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

Predicado de Pertinência

- Sempre que um procedimento recursivo é definido, deve-se procurar pelas condições limites (ou condições de parada) e pelo caso recursivo:

```
pertence(Cabeca, [Cabeca|Cauda]).
pertence(Cabeca, [OutraCabeca|Cauda]) :-
    pertence(Cabeca,Cauda).
```

- Após a definição do procedimento pertence/2 , é possível interrogá-lo.

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

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**

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]).
```


Exemplo pertence/2

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

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

Exemplo pertence/2

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

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


Exemplo pertence/2

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

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

Exemplo pertence/2

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

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


Exemplo pertence/2

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

```
?- pertence(zeus,[yolanda,tadeu,vicente,julia]).
no
?-
```

Exemplo pertence/2

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

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


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
```

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).
```


Exercício

- Inserção de um elemento na primeira posição de uma lista

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]
```


Exercício

- Converter valores de uma lista em seus valores absolutos

Exemplo: Converte

- Converte valores de uma lista em seus valores absolutos.
- Exemplo:

  ?- converte([5,-3, 1, -4], L).

  L= [5, 3, 1, 4]

```
converte([ ], [ ]).
converte([X|L1], [Y|L2]):-
    abs(X,Y),
    converte(L1, L2).
```


converte/2 (arvore de busca)

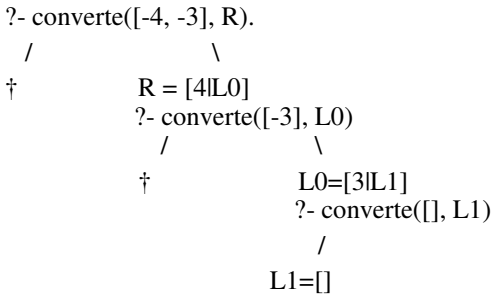
```
?- converte([-4], R).
  /                    \
†              R = [4|L0]
               ?- converte([ ],L0)
                 /

          L0=[ ]
```

L0=[ ]
R=[4|L0]=[4]

```
converte([ ], [ ]).
converte([X|L1], [Y|L2]):-
    abs(X,Y),
    converte(L1, L2).
```

converte/2 (arvore de busca)

```
converte([ ], [ ]).
converte([X|L1], [Y|L2]):-
    abs(X,Y),
    converte(L1, L2).
```



L1=[ ]
L0=[3|L1]=[3]
R=[4|L0]=[4,3]

Exercício

- Definir o procedimento **último(Elem,Lista)** que encontra o último elemento **Elem** de uma lista **Lista**
  - Qual a lógica?

Exercício

- Definir o procedimento **último(Elem,Lista)** que encontra o último elemento **Elem** de uma lista **Lista**
  - O último elemento de uma lista que tem somente um elemento é o próprio elemento
  - O último elemento de uma lista que tem mais de um elemento é o ultimo elemento da cauda

Último Elemento de uma Lista

- O último elemento de uma lista que tem somente um elemento é o próprio elemento

```
ultimo(Elemento, [Elemento]).
```

- O último elemento de uma lista que tem mais de um elemento é o ultimo elemento da cauda

```
ultimo(Elemento, [Cabeca|Cauda]) :-
        ultimo(Elemento,Cauda).
```

- Procedimento completo:

```
% ultimo(?Elemento, +Lista)
ultimo(Elemento, [Elemento]).
ultimo(Elemento, [Cabeca|Cauda]) :-
        ultimo(Elemento,Cauda).
```

Exercício

- Concatenar duas listas, formando uma terceira
  - Qual a lógica?

Exercício

- Concatenar duas listas, formando uma terceira
  - Se o primeiro argumento é a lista vazia, então o segundo e terceiro argumentos devem ser o mesmo
  - Se o primeiro argumento é a lista não-vazia, então ela tem uma cabeça e uma cauda da forma [X|L1]; concatenar [X|L1] com uma segunda lista L2 resulta na lista [X|L3], onde L3 é a concatenação de L1 e L2

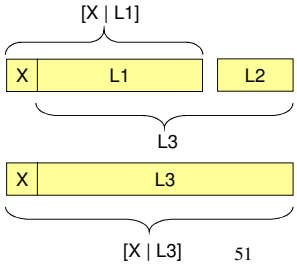
Concatenar duas Listas

- Se o primeiro argumento é a lista vazia, então o segundo e terceiro argumentos devem ser o mesmo
  concatenar([ ],L,L).
- Se o primeiro argumento é a lista não-vazia, então ela tem uma cabeça e uma cauda da forma [X|L1]; concatenar [X|L1] com uma segunda lista L2 resulta na lista [X|L3], onde L3 é a concatenação de L1 e L2

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

- Procedimento completo:

```
% concatenar(?/+L1,?/?L2,+/?L)
concatenar([ ],L,L).
concatenar([X|L1],L2,[X|L3]) :-
     concatenar(L1,L2,L3).
```



concatenar/3 (arvore de busca)

?- concatenar([a,b,c],[1,2,3], R).

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

concatenar/3 (arvore de busca)

```
?- concatenar([a,b,c],[1,2,3], R).
  /                        \
```

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


concatenar/3 (arvore de busca)

```
?- concatenar([a,b,c],[1,2,3], R).
  /                        \
†                R = [a|L0]
                 ?- concatenar([b,c],[1,2,3],L0)
```

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

concatenar/3 (arvore de busca)

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

?- concatenar([a,b,c],[1,2,3], R).
/ \
† R = [a|L0]
?- concatenar([b,c],[1,2,3],L0)
/ \

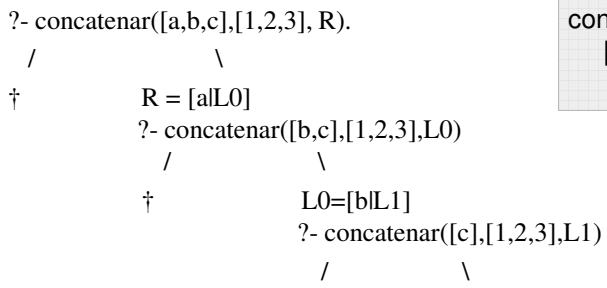
concatenar/3 (arvore de busca)

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

?- concatenar([a,b,c],[1,2,3], R).
/ \
† R = [a|L0]
?- concatenar([b,c],[1,2,3],L0)
/ \
† L0=[b|L1]
?- concatenar([c],[1,2,3],L1)

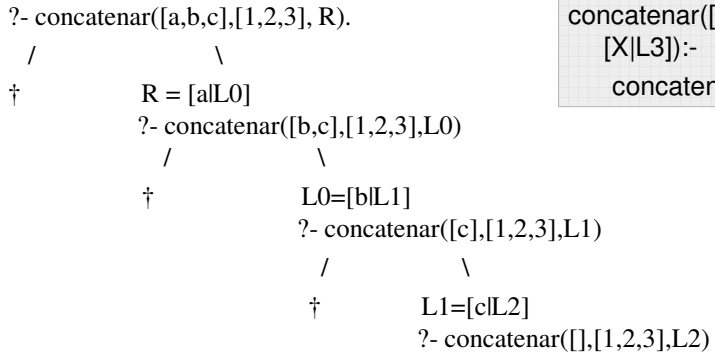
concatenar/3 (arvore de busca)

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



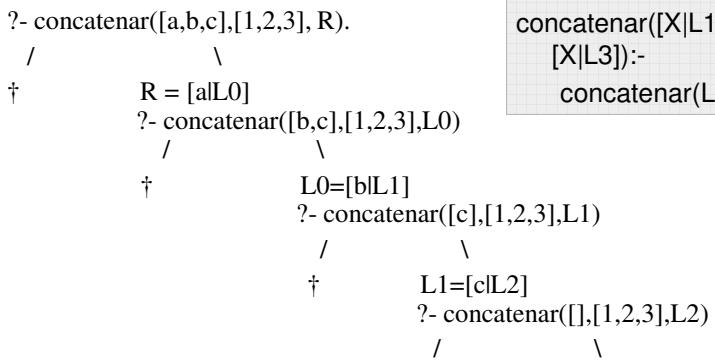
concatenar/3 (arvore de busca)

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



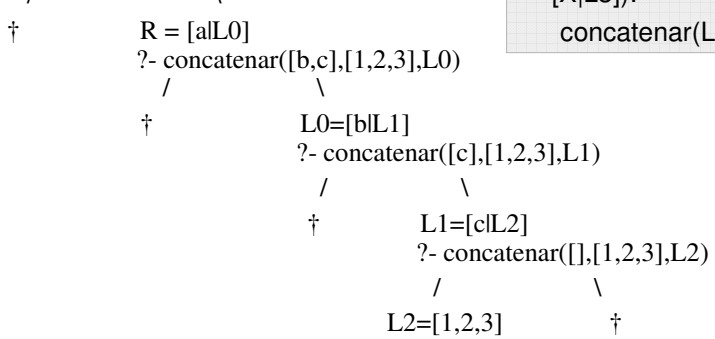
concatenar/3 (arvore de busca)

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



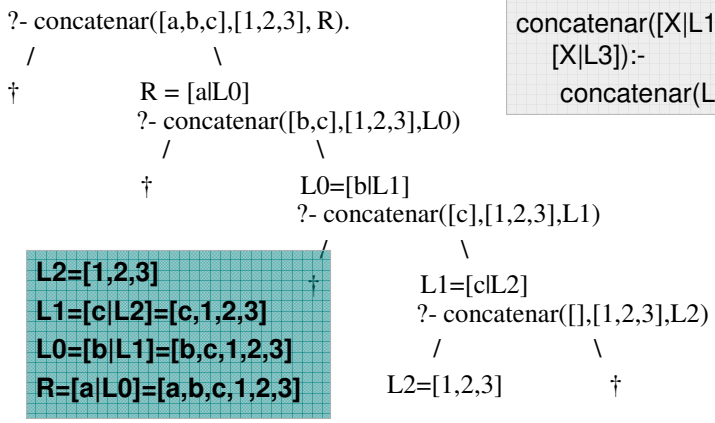
concatenar/3 (arvore de busca)

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



concatenar/3 (arvore de busca)

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



L2=[1,2,3]
L1=[c|L2]=[c,1,2,3]
L0=[b|L1]=[b,c,1,2,3]
R=[a|L0]=[a,b,c,1,2,3]

Exemplo concatenar/3

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

```
?- concatenar(X, Y, [1,2,3,4]).
X = [ ]          Y = [1,2,3,4];
X = [1]          Y = [2,3,4];
X = [1,2]        Y = [3,4];
X = [1,2,3]      Y = [4];
X = [1,2,3,4]    Y = [ ];
no
```

Exercícios

- Encontrar o maior valor de uma lista de valores numéricos.

Exercícios

- Verificar se dois elementos são consecutivos em uma lista.

Exercícios

- Somar elementos de uma lista numérica.

Exercícios

- Encontrar n-ésimo elemento de uma lista.

Exercícios

- Determinar o número de elementos de uma lista.

Exercícios

- Retirar uma ocorrência de um elemento de uma lista.

Exercícios

- Substituir elemento de uma lista por um outro elemento.

Exercícios

- Dividir uma lista numérica em 2 sublistas que contenham os elementos iguais ou menores e os maiores que um dado elemento.