# Sintaxe e Semântica de Programas Prolog

Esta aula trata da sintaxe e semântica de conceitos básicos em Prolog relacionados à manipulação de listas

**Inteligência Artificial**

**José Augusto Baranauskas**
**Departamento de Física e Matemática – FFCLRP-USP**
**Sala 228 – Bloco P2 – Fone (16) 602-4439**

**E-mail: augusto@ffclrp.usp.br**
**URL: http://www.fmrp.usp.br/augusto**

# Listas

- **Lista** é uma das estruturas mais simples em Prolog, sendo muito comum em programação simbólica – não numérica
- Uma lista é:
  - uma sequência ordenada de elementos (objetos)
  - 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 objetos estruturados em Prolog

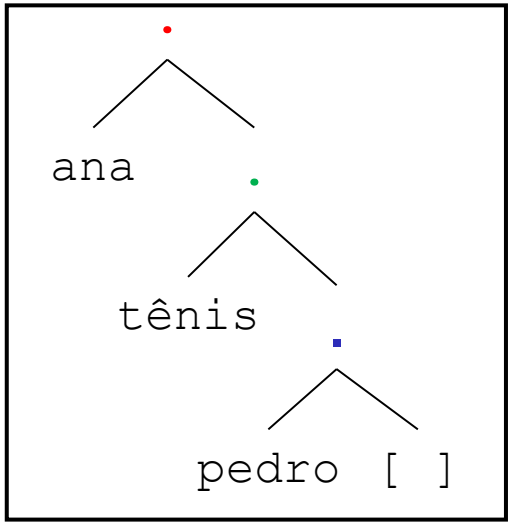
- Para entender a representação Prolog de listas, é necessário considerar dois casos:
  - A lista é vazia, escrita como **[ ]** em Prolog
  - A lista não é vazia, contendo:
    - o primeiro elemento: um objeto chamado **Cabeça** (*head*)
    - a parte restante da lista: uma lista chamada **Cauda** (*tail*)

# 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 precisa ser uma lista
- Como a Cauda é uma lista, ela é vazia ou ela tem sua própria cabeça e sua cauda; ou seja, a lista tem definição recursiva

# Listas

- Assim, para representar listas de qualquer comprimento, nenhum princípio adicional é necessário
- O exemplo [ana, tênis, pedro] também pode ser representado graficamente como indicado na figura a seguir, em forma de árvore



# Listas

```
?- Lista1 = [a,b,c].
Lista1 = [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]]

?- Cauda = [b,c], L = [a,Cauda].
Cauda = [b,c]
L = [a,[b,c]]
```

# Listas

- É possível também criar uma nova lista, na qual a Cauda surge como uma sequência de elementos

- Para expressar isso, Prolog fornece uma notação alternativa, a **barra vertical**, que separa a cabeça da lista Cauda; assim, podemos formar L = [a,b,c]

```
Cauda = [b,c]
L = [a | Cauda]
```

# Listas

- A notação é geral por permitir qualquer número de elementos seja seguido por | e uma lista contendo os demais elementos:

```
[a,b,c]  = [a | [b,c]]
         = [a,b | [c]]
         = [a,b,c | [ ]]
```

# Unificação em Listas

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

# Unificação em Listas

| Lista1 | Lista2 | Lista1 = Lista2 |
|---|---|---|
| [mesa] | [X\|Y] | X=mesa<br>Y=[ ] |
| [a,b,c,d] | [X,Y\|Z] | |
| [[ana,Y]\|Z] | [[X,foi],[ao,cinema]] | |
| [ano,bissexto] | [X,Y\|Z] | |
| [ano,bissexto] | [X,Y,Z] | |

# Unificação em Listas

| Lista1 | Lista2 | Lista1 = Lista2 |
|---|---|---|
| [mesa] | [X\|Y] | X=mesa<br>Y=[ ] |
| [a,b,c,d] | [X,Y\|Z] | X=a<br>Y=b<br>Z=[c,d] |
| [[ana,Y]\|Z] | [[X,foi],[ao,cinema]] | |
| [ano,bissexto] | [X,Y\|Z] | |
| [ano,bissexto] | [X,Y,Z] | |

# Unificação em Listas

| Lista1 | Lista2 | Lista1 = Lista2 |
|---|---|---|
| [mesa] | [X\|Y] | X=mesa<br>Y=[ ] |
| [a,b,c,d] | [X,Y\|Z] | X=a<br>Y=b<br>Z=[c,d] |
| [[ana,Y]\|Z] | [[X,foi],[ao,cinema]] | X=ana<br>Y=foi<br>Z=[[ao,cinema]] |
| [ano,bissexto] | [X,Y\|Z] | |
| [ano,bissexto] | [X,Y,Z] | |

# Unificação em Listas

| Lista1 | Lista2 | Lista1 = Lista2 |
|---|---|---|
| [mesa] | [X\|Y] | X=mesa<br>Y=[ ] |
| [a,b,c,d] | [X,Y\|Z] | X=a<br>Y=b<br>Z=[c,d] |
| [[ana,Y]\|Z] | [[X,foi],[ao,cinema]] | X=ana<br>Y=foi<br>Z=[[ao,cinema]] |
| [ano,bissexto] | [X,Y\|Z] | X=ano<br>Y=bissexto<br>Z=[ ] |
| [ano,bissexto] | [X,Y,Z] | |

# Unificação em Listas

| Lista1 | Lista2 | Lista1 = Lista2 |
|---|---|---|
| [mesa] | [X\|Y] | X=mesa<br>Y=[ ] |
| [a,b,c,d] | [X,Y\|Z] | X=a<br>Y=b<br>Z=[c,d] |
| [[ana,Y]\|Z] | [[X,foi],[ao,cinema]] | X=ana<br>Y=foi<br>Z=[[ao,cinema]] |
| [ano,bissexto] | [X,Y\|Z] | X=ano<br>Y=bissexto<br>Z=[ ] |
| [ano,bissexto] | [X,Y,Z] | Não unifica |

# 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

SUGESTÕES??

- 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 **X** pertence ou não a uma lista **Y**, por exemplo, **pertence(X,Y)**

- A primeira condição especifica que um elemento **X** pertence à lista se ele está na **Cabeça** dela:

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

- A segunda condição especifica que um elemento **X** pertence à lista se ele pertencer à sua **Cauda**:

```
pertence(X,[W|Z]) :-
        pertence(X,Z).
```

# Predicado de Pertinência

- Ao comparar **`pertence/2`** com a função recursiva C dada a seguir, nota-se que Prolog pode levar a soluções mais compactas para problemas baseados em lista

```
/*exemplo de função recursiva para uma lista (vetor)
em que o retorno...
1 indica verdadeiro, i.e., o elemento em questão está
na cabeça da lista
0 indica falso, i.e., a lista esvaziou e o elemento
não foi encontrado OBS.: rotina recursiva considera
tipos de dados, índices e um vetor com tamanho (tam)
<=capacidade máxima (TAM).
OBS.: primeira chamada deve indicar índice (ind) 0*/

//int pertence(int X, int *Y, int tam, int ind);
```

# Predicado de Pertinência

- Ao comparar **`pertence/2`** com a função recursiva C dada a seguir, nota-se que Prolog pode levar a soluções mais compactas para problemas baseados em lista

```
int pertence(int X, int *Y, int tam, int ind){
   if (ind==tam)
    return 0; //fim da lista; elemento não encontrado
   else if (X==Y[ind])
    return 1; //pertence(X,[X|Z]).
   else
    return pertence(X,Y,tam,ind+1);
   //pertence(X,[W|Z]) :-pertence(X,Z).
}
```

# Predicado de Pertinência

- Ao comparar **`pertence/2`** com a função recursiva C dada a seguir, nota-se que Prolog pode levar a soluções mais compactas para problemas baseados em lista

```
/*protótipo da função recursiva C
   int pertence(int X, int *Y, int tam, int ind);

Para ilustrar a chamada da função C dentro de outra
rotina, considera-se vi como um vetor de 5 inteiros já
preenchido e 3 como o elemento que será buscado em vi
usando a função pertence. O índice para começar a
varredura de vetor é 0*/

   pertence(3,vi,5,0);
```

# Predicado de Pertinência

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

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

- Após a definição do programa, é possível interrogá-lo. Por exemplo:

```
?- pertence(a,[a,b,c]).
Yes
?- pertence(d,[a,b,c]).
no
```

# Predicado de Pertinência

```
?- pertence(X,[a,b,c]).
  X = a ;
  X = b ;
  X = c ;
  no
```

- Entretanto, se as perguntas forem:

```
?- pertence(a,X).
?- pertence(X,Y).
```

- Deve-se observar que cada uma delas tem infinitas respostas, pois existem infinitas listas que validam essas perguntas para o programa pertence/2

# Modo de Chamada de Predicados

- Para documentar como um predicado deve ser chamado, utiliza-se a 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 predicado **pertence/2** documentado com o modo de chamada é:

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

# Predicado – Último Elemento de uma Lista

SUGESTÕES??

1. Se a lista tem apenas um elemento, este elemento é seu último elemento;
2. O último elemento de uma lista com mais de um elemento é o último da cauda da lista.

```
% ultimo(L,E): E é último elemento de L
% ultimo(+L,?E)
ultimo([E],E).
ultimo([_|Cauda],E):-
    ultimo(Cauda,E).
```

# Predicado – Inserção

- Inserção na primeira posição. O objetivo deste predicado consiste em concatenar um elemento **E** com uma lista **L** gerando **L1**

```
% insere(E,L,L1): insere E na cabeça de L gerando L1
% insere(?/+E,?/+L,+/?L1)
  insere(E,L,[E|L]).
```

- Inserção na última posição. O objetivo deste predicado consiste em **concatenar** uma lista **L** com um elemento **E** gerando **L1**

```
% insere(E,L,L1): insere E na última posição de L gerando L1
% insere(?/+E,?/+L,+/?L1)
  insere(E,[],[E]).
  insere(E,[Cab|Cauda],[Cab|Cauda1]) :-
      insere(E,Cauda,Cauda1).
```

# Predicado – Elementos Consecutivos

SUGESTÕES??

1. Dois elementos E1 e E2 são consecutivos em uma lista, se forem o 1º e o 2º elementos ou
2. Se forem consecutivos na cauda da lista

```
% consecutivos(E1,E2,L): E1 consecutivo a E2 em L
% consecutivos(?E1,?E2,+L)
  consecutivos(E1,E2,[E1,E2|_]).
  consecutivos(E1,E2,[_|Cauda]):-
        consecutivos(E1,E2,Cauda).

?- consecutivos(E1,E2,[a,b,c,d]).
```

# Predicado – Soma de Elementos Numéricos

SUGESTÕES??

1. Se a lista for vazia, a soma é zero ou
2. Se a lista for [Elem|Cauda], a soma é a soma dos elementos da Cauda mais Elem

```
% soma(L,S): S contém a soma de todos os elementos em L
% soma(+L,?S) (+L,-S)
  soma([],0).
  soma([Elem|Cauda],S):-
       soma(Cauda,S1),
       S is S1+Elem.
```

# Predicado – Soma de Elementos Numéricos

- O que acontece se considerarmos o predicado abaixo?

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

- Prolog exige que S1 esteja instanciado e por isso ocorrerá um erro.

# Predicado – N-ésimo Elemento de uma Lista

SUGESTÕES??

1. O primeiro elemento de uma lista é a cabeça ou
2. O n-ésimo elemento de uma lista é o (n-1)-ésimo elemento de sua cauda

```
% n_esimo(N,E,L): E é N-nésimo elemento de L
% n_esimo(?N,?E,+L)
  n_esimo(1,Elem,[Elem|_]).
  n_esimo(N,Elem,[_|Cauda]):-
          n_esimo(M,Elem,Cauda),
          N is M+1.
```

# Predicado – Pegar elementos de uma Lista, dada a Lista de posições

SUGESTÕES??

1. Pegar nenhum elemento de uma lista é obter uma lista vazia ou
2. Seja a lista de posições [M|N] e a lista de elementos L; pegar os elementos de L especificados na lista de posições significa pegar o M-ésimo elemento de L e os elementos especificados na cauda N da lista de posições.

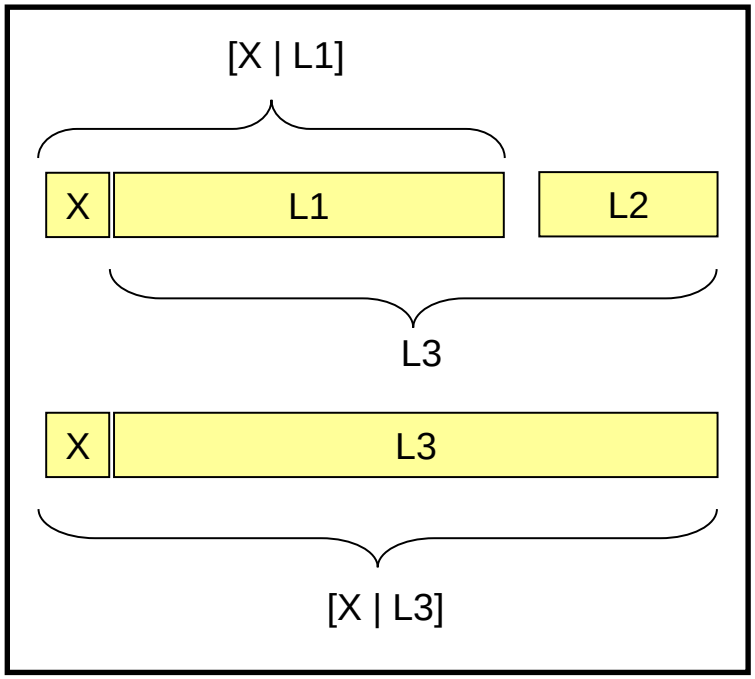
```
% pegar(LP,LE,L): Inserir em L todos os elementos de
    LE nas posições em LP
% pegar(+LP,+LE,?L)
   pegar([],_,[]).
   pegar([M|N],L,[X|Y]):-
       n_esimo(M,X,L),
       pegar(N,L,Y).
```

# Predicado de Concatenação (Exercício)

Concatenar duas listas, formando uma terceira:

1. Se o primeiro argumento é a lista vazia, então o segundo e o terceiro argumentos devem ser o mesmo
2. 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], se L3 é a concatenação de L1 e L2

# Predicado de Concatenação



# Exercícios

1. **Definir** uma nova versão do predicado **último**,que encontra o último elemento de uma lista, utilizando a **concatenação** de listas
2. **Definir** predicado **penúltimo**
3. **Encontrar o comprimento** de uma lista (Sugestão: A is A1+1)
4. **Retirar** elementos de uma lista

Slides baseados em:

Bratko, I.;
*Prolog Programming for Artificial Intelligence*,
3rd Edition, Pearson Education, 2001.

Clocksin, W.F.; Mellish, C.S.;
*Programming in Prolog*,
5th Edition, Springer-Verlag, 2003.

Programas Prolog para o
Processamento de Listas e Aplicações,
Monard, M.C & Nicoletti, M.C., ICMC-USP, 1993

Material elaborado por José Augusto Baranauskas
Adaptado por Huei Diana Lee

Slides baseados em:

Mohomed, I.;
*CSC 324 lecture notes*,
University of Toronto, 2005.

Frade, M. J.;
Lógica Computacional – Prolog, Disponível em: www4.di.uminho.pt/~mjf/pub/LC-Prolog.pdf. Acesso em: Setembro/2019

SWI-PROLOG;
*How to use the search box*, Disponível em: www.swi-prolog.org/search. Acesso em: Setembro/2019.

Slides baseados em:

Wilson, B.;
*The Prolog Dictionary,*
Disponível em: tiny.cc/psixbz. Acesso em: Agosto/2019,
The University of New South Wales, 2015.

SWI-PROLOG;
*Representation and printing of floating point numbers*, Disponível em: www.swi-prolog.org/FAQ/floats.html. Acesso em: Setembro/2019.

Material elaborado por

José Augusto Baranauskas

Adaptado por Huei Diana Lee e Newton Spolaôr