Prolog – Recursividade e Listas

PARADIGMAS DE PROGRAMAÇÃO

Recursividade

- Regras recursivas devem ser permitidas a fim de tornar a linguagem útil para muitas aplicações. Um predicado definido por uma regra recursiva deve necessariamente ter, no mínimo uma definição não recursiva. Se isto não acontecer, a definição é logicamente mal-formada e o programa ficaria em laço infinito.

Recursividade

- Considerando o seguinte programa:

- **pai(david, john).**
- **pai(jim, david).**
- **pai(steve, jim).**
- **pai(nathan, steve).**

- **avo(A, B) :- pai(A, X), pai(X, B).**

Recursividade

- Seria interessante poder definir ancestral(X, Y). Uma forma não-recursiva seria impraticável, pois seria necessário incluir todas as possíveis gerações:

- **ancestral(A,B) :- pai(A, B).**
- **ancestral(A,B) :- pai(A, X), pai(X, B).**
- **ancestral(A,B) :- pai(A, X), pai(X, Y), pai(Y, B).**
- **ancestral(A,B) :- pai(A, X), pai(X, Y), pai(Y, Z), pai(Z,B).**
- **% continue escrevendo - e ainda falta incluir as mães!**

Recursividade

- A implementação da regra sobre ancestralidade usando recursão é muito mais simples e elegante:

- **ancestral(X,Y) :- pai(X,Y).**

  ancestral(X,Y) :- pai(X,Z), ancestral(Z,Y).

Recursividade

- Deve-se tomar cuidado com a ordem na qual unificações são procurados para objetivos. Se invertermos a ordem nas regras recursivas do predicado ancestral, isto é, a consulta resultará uma recursão infinita.

- **ancestral(X,Y) :- ancestral(Z,Y), pai(X,Z).**

Recursividade - Exercício

- Considere o predicado sucessor(X, Y), que indica que um número, X, é sucessor de outro número, Y. Escreva os fatos para relacionar os números de 1 a 7 de acordo com o predicado sucessor. Ex.:

- **sucessor(2, 1).**
- **sucessor(3, 2).**
- **...**

- Agora defina regras para os predicados maior_que(X, Y) e menor_que(X, Y), definir o menor_que sem usar a regra maior_que.

Listas

- Uma lista é uma sequência finita de elementos. Ex.:

- **[mia, vincent, jules, yolanda]**
- **[mia, cor(amarelo), X, 2, mia]**
- **[]**
- **[mia, [vincent, jules], [laranja, fruta(laranja)]]**

Listas

- Uma lista **não-vazia** pode ser pensada como tendo duas partes:
  - **cabeça** (*head*): primeiro elemento da lista
  - **cauda** (*tail*): lista que sobra quando retiramos a cabeça

- Uma lista não-vazia pode ser representada de forma a apresentar explicitamente a sua cabeça e a sua cauda, usando a sintaxe [CabeçalCauda]. Exemplo:

  - **% lista [b]**
  - **[b|[]]**

Listas

- Essa sintaxe é útil em consultas quando queremos decompor uma lista em cabeça e cauda.

- [X|Y] = [morango, laranja, uva, goiaba].

- X = morango
- Y = [laranja, uva, goiaba]

Listas

- Predicado
- membro(X, L) determina se o elemento X faz parte da lista L.

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

Listas

- Considere a busca nas seguintes consultas:

- **membro(vincent,[yolanda,trudy,vincent,jules]).**
- **membro(zed,[trudy,vincent,jules]).**

- Outro exemplo de consulta:
- **membro(X,[yolanda,trudy,vincent,jules]).**

Listas

- Predicado a2b(X, Y), retorna “true” somente se:
  - o 1o argumento é uma lista somente de as
  - o 2o argumento é uma lista somente de bs
  - as listas têm o mesmo tamanho

- **Ex.: a2b([a,a,a,a],[b,b,b,b]). é verdadeiro**
- **Ex.: a2b([a,a,a,a],[b,b,b]). é falso**

- **a2b([], []).**
- **a2b([a|A], [b|B]) :- a2b(A, B).**

Listas - Exercício

O primeiro passo é criar uma **base de conhecimento** com algumas características conhecidas de alguns animais. São elas:

- Têm pelo: cavalo e coelho;
- Põe ovos: besouro, salmão, cobra, tucano, borboleta, tainha, canarinho e lagarto;
- Vive na água: salmão e tainha;
- Vertebrado: cavalo, salmão, coelho, cobra, tucano, tainha, canarinho e lagarto;
- Têm penas: tucano e canarinho.

Agora precisamos definir **regras** para as classes de animais, crie uma regra específica classe(X, Y) para cada classe de animais que utilize as características acima (e **não o nome do animal**) como condição. As regras devem se adequar aos seguintes resultados:

- classe(mamifero, X) : que deve dar verdadeiro para cavalo e coelho;
- classe(peixe, X) : que deve dar verdadeiro para salmão e tainha;
- classe(ave, X) : que deve dar verdadeiro para tucano e canarinho;
- classe(inseto, X) : que deve dar verdadeiro para besouro e borboleta;
- classe(reptil, X) : que deve dar verdadeiro para cobra e lagarto;

Note que para mamífero, peixe e ave existem características únicas e óbvias. Para os demais deve ser exercitada a lógica.

Listas - Exercício

Definir uma **regra** classes([H|T], X) que receba como entrada na consulta uma lista([H|T]) e de forma recursiva encontre a classe X de cada elemento.

- Dica: Utilizem o primeiro algoritmo que verifica se um elemento pertence à uma lista.
- Utilizem a seguinte consulta como exemplo: **classes([cavalo, besouro, salmao, coelho, cobra, tucano, borboleta, tainha, canarinho, lagarto], X).** A resposta deve retornar 10 linhas com a classe de cada animal na ordem da lista:

    X = mamífero
    X = inseto
    X = peixe
    X = mamífero
    X = réptil
    X = ave
    X = inseto
    X = peixe
    X = ave
    X = reptil