Lógica Computacional

2^o Ano LMCC (2005/06)

Exercícios de Prolog

Grupo I

1. Considere o seguinte programa Prolog:

```
% pertence(X,L) indica que X é um elemento da lista L
pertence(X,[X|_]).
pertence(X,[_|T]) :- pertence(X,T).

% prefixo(L1,L2) indica que a lista L1 é prefixo da lista L2
prefixo([],_).
prefixo([X|Xs],[X|Ys]) :- prefixo(Xs,Ys).

% sufixo(L1,L2) indica que a lista L1 é sufixo da lista L2
sufixo(L,L).
sufixo(Xs,[Y|Ys]) :- sufixo(Xs,Ys).

% concatena(L1,L2,L3) indica qua a lista L1 concatenada com a lista L2
% é a lista L3
concatena([],L,L).
concatena([H|T],L1,[H|L2]) :- concatena(T,L1,L2).
```

   Carregue estas definições no interpretador e interrogue a base de conhecimento. Por exemplo:

```
? pertence(2,[1,2,3]).
? pertence(X,[1,2,3]).
? pertence(3,L).
? pertence(X,L).
```

2. Considere a representação dos números naturais baseada nos construtores 0 e sucessor: 0, $suc(0)$, $suc(suc(0))$, $suc(suc(suc(0)))$, ...

   O predicado nat que testa se um termo é um número natural.

```
nat(0).
nat(suc(X)) :- nat(X).
```

   Defina, usando o functor `suc`, predicados que implementem as seguintes relações:

   (a) menor ou igual
   (b) mínimo
   (c) soma
   (d) multiplicação
   (e) factorial
   (f) exponenciação

3. Defina o predicado `last(X,L)` que testa se `X` é o último elemento da lista `L`.

4. Defina a relação `divide(L,L1,L2)` que divide a lista `L` em duas listas `L1` e `L2` com aproximadamente o mesmo tamanho.

Grupo II

1. Defina os seguintes predicados sobre listas:
   (a) `minimo/2` que produz o menor elemento presente numa lista.
   (b) `somatorio/2` que cálcula o somatório de uma lista.
   (c) `nesimo/3` que dá o elemento da lista na n-ésima posição
2. Defina um procedimento que ordene de uma lista segundo o algoritmo *quicksort*.
3. Considere o seguinte procedimento para o cálculo do factorial

   ```
   fact(0,1).
   fact(N,F) :- N>0, N1 is N-1, fact(N1,F1), F is N*F1.
   ```

   Defina uma outra versão de factorial que utilize um parâmetro de acumulação.
4. Defina uma versão do predicado somatório que utilize um acumulador.

Grupo III

1. Defina a relação `flatten/2` (que lineariza uma lista) de forma a que, por exemplo:

   ```
   | ?- flatten([a,b,[c,d],[],[[[e,f]],g],h],X).
   X = [a,b,c,d,e,f,g,h] ?
   yes
   ```

2. Escreva um programa para reconhecer se uma fórmula da lógica proposicional está na forma normal conjuntiva, ou seja, é uma conjunção de disjunções de literais. Um literal é um símbolo proposicional ou a sua negação.

   Considere a declaração das seguintes conectivas lógicas:

   ```
   :- op(500,yfx,/\).
   :- op(500,yfx,\/).
   :- op(300,fy,~).
   ```

3. Defina o predicado `conta_ocorr/3` para contar quantas vezes uma constante ocorre numa lista. (Sugestão: usar `atomic/1`).
4. Suponha que tem factos da forma `quadrado(Lado)`. Defina o predicado `zoom(+X,?Y,+F)` tal que `Y` é o quadrado que resulta de multiplicar pelo factor `F` os lados do quadrado `X`. (Sugestão: usar `=..`).

Grupo IV

1. Considere os seguintes predicados Prolog:

```
% testa um termo representa uma árvore binária válida
arv_bin(vazia).
arv_bin(nodo(X,Esq,Dir)) :- arv_bin(Esq), arv_bin(Dir).

% verifica se um termo pertencea uma árvore binária
na_arv(X,nodo(X,_,_)).
na_arv(X,nodo(Y,Esq,_)) :- na_arv(X,Esq).
na_arv(X,nodo(Y,_,Dir)) :- na_arv(X,Dir).
```

2. Defina predicados que permitam fazer as travessias *preorder*, *inorder* e *postorder*.

3. Defina um predicado `search_tree/1` que teste se uma dada árvore é uma árvore binária de procura.

4. Defina a relação `insert_tree(+X,+T1,?T2)` que sucede se `T2` é uma árvore binária de procura resultado da inserção de `X` na árvore binária de procura `T1`.

5. Defina a relação `path(+X,+Tree,?Path)` que sucede se `Path` é o caminho da raiz da árvore binária de procura `Tree` até `X`.

6. Defina um predicado `no_dupl/2` que remova os duplicados de uma lista.

7. Defina um predicado `enumerar(+N,+M,+P,?L)` que gera a lista `L` de números entre `N` e `M`, a passo `P`. Por exemplo:

```
| ?- enumerar(3,10,2,L).
L = [3,5,7,9] ?
yes
```

8. Defina um programa que faça a ordenação de uma lista pelo algoritmo *merge sort*. Use o *cut* para implementar o predicado merge de forma mais eficiente.

Grupo V

1. Defina o predicado `subconj(-S,+C)` onde `S` e `C` são duas listas que representam dois conjuntos. Este predicado deve gerar, por backtracking, todos os subconjuntos possíveis de `C`.

2. Defina o predicado `partes(+C,-P)`, que dado um conjunto `C` (implementado como lista) dá em `P` o conjunto de todos os subconjuntos de `C`.

3. Considere a linguagem proposicional gerada pelos simbolos proposicionais (átomos) e as conectivas: `falso, verdade, ~, /\, \/, =>`. Relembre que um modelo é um conjunto de simbolos proposicionais.

   ```
   :- op(600,xfy,=>).
   :- op(500,yfx,/\).
   :- op(500,yfx,\/).
   :- op(300,fy,~).
   ```

   Defina o predicado `atrib(+M,+P,?V)` que sucede se `V` é o valor de verdade da proposição `P` no modelo `M`. (Relembre a função μ da aulas teóricas). Pode utilizar o predicado `simp(+E,?V)` que faz o cálculo do valor de uma expressão na álgebra booleana Z_2.

   ```
   simp(X+X,0).
   simp(X*X,X).
   simp(X+0,X).
   simp(0+X,X).
   simp(X*0,0).
   simp(0*X,0).

   simp(X+Y,Z) :- simp(X,X1), simp(Y,Y1), simp(X1+Y1,Z).
   simp(X*Y,Z) :- simp(X,X1), simp(Y,Y1), simp(X1*Y1,Z).
   ```

4. Defina os predicados `formula_valida(+M,+P)` e `teoria_valida(+M,+T)` que sucedem se a proposição `P` e teoria `T` é válida no modelo `M`.

   ```
   | ?- formula_valida([p,q], p/\q => q\/r => ~r /\ ~ ~p).
   yes
   ```

5. Defina o predicado `consequencia(+T,+P)` sucede se a proposição P é uma consequência (semântica) da teoria `T`. (Sugestão: gere primeiro todos os modelos possíveis com os simbolos proposicionais das fórmulas envolvidas.)

6. Defina `tautologia/1` que testa se uma fórmula é uma tautologia.

7. Defina `inconsistente/1` que testa se uma teoria é inconsistente.

8. Defina os predicados `soma/3` e `produto/3` que implementam as operações de soma e produto de conjuntos de conjuntos de literais (ver apontamento da aulas teóricas).

9. Defina o predicado `fnn(+P,-FNN)` que dada uma proposição (da linguagem da alínea 3) gera a forma normal negativa que lhe é semanticamente equivalente.

10. Defina o predicado `fnc(+FNN,-L)` que dada uma forma normal negativa, produz em `L` a forma normal conjuntiva equivalente, representada por conjuntos de conjuntos de literais.

11. Defina o predicado `constroi_fnc(+L,-P)` que sucede se `P` é a proposição que o conjunto de conjuntos de literarais `L` representa.

12. Defina o predicado `gera_fnc(+P,-FNC)` que dada uma proposição `P` produz `FNC` uma proposição na forma normal conjuntiva, semanticamente equivalente a `P`.