2 Exercícios sobre Prolog

2.1 A Árvore Genealógica da Família Pinheiro

Enunciado:

Pouco se sabe da história passada da família **Pinheiro**. Existem alguns registos antigos que indicam que o casal José e Maria criou dois filhos, o João e a Ana. Que a Ana teve duas filhas, a Helena e a Joana, também parece ser verdade, segundo os mesmos registos. Além disso, o Mário é filho do João, pois muito se orgulha ele disso. Estranho também, foi constatar que o Carlos nasceu da relação entre a Helena, muito formosa, e o Mário.

a) Utilizando o predicado **progenitor(X,Y)** (ou seja, X é progenitor de Y), represente em Prolog todos os progenitores da família Pinheiro.
b) Represente em Prolog as relações: **sexo** (masculino ou feminino), **irmã**, **irmão**, **descendente**, **mãe**, **pai**, **avô**, **tio**, **primo**[2].
c) Formule em Prolog as seguintes questões:
    1. O João é filho do José?
    2. Quem são os filhos da Maria?
    3. Quem são os primos do Mário?
    4. Quantos sobrinhos/sobrinhas com um Tio existem na família Pinheiro?
    5. Quem são os ascendentes do Carlos?
    6. A Helena tem irmãos? E irmãs?

Explicação:

Este exercício envolve **objectos** e **relações entre objectos**, sendo uma adaptação livre do programa *family* da Figura 1.8 do livro **[Brakto, 1990]**. Dado que o enunciado é livre neste aspecto, optou-se por utilizar a notação **sexo(Nome, Sexo)** para representar o sexo de cada pessoa. Em algumas das relações pode existir mais do que uma forma de resolver aquilo que é pedido. As questões da alínea c) podem ser ter diferentes interpretações (por exemplo se a questão deve retornar uma ou todas as soluções), sendo que nestes casos, optou-se por apresentar as diversas alternativas (e.g. **q2a**, **q2b**). Para correr o programa no SWI-Prolog basta executar os seguintes comandos:

```
?- [pinheiro].
Yes
?- q2b(X).
X = [joao, ana]
```

(executar as restantes questões q2, q3, q3b, ...)

[2] Neste caso, por *primo* entende-se primo ou prima.

Resolução:

pinheiro.pl

```
% factos
progenitor(maria,joao).
progenitor(jose,joao).
progenitor(maria,ana).
progenitor(jose,ana).
progenitor(joao,mario).
progenitor(ana,helena).
progenitor(ana,joana).
progenitor(helena,carlos).
progenitor(mario,carlos).

sexo(ana,feminino).
sexo(maria,feminino).
sexo(joana,feminino).
sexo(helena,feminino).

sexo(mario,masculino).
sexo(joao,masculino).
sexo(jose,masculino).
sexo(carlos,masculino).

irma(X,Y):- progenitor(A,X),
            progenitor(A,Y),
            X\==Y,
            sexo(X,feminino).

irmao(X,Y):- progenitor(A,X),
             progenitor(A,Y),
             X\==Y,
             sexo(X,masculino).

descendente(X,Y):- progenitor(X,Y).
descendente(X,Y):- progenitor(X,A),
                   descendente(A,Y).

avo(X,Y):- progenitor(X,A),
           progenitor(A,Y),
           sexo(X,masculino).

mae(X,Y):- progenitor(X,Y),
           sexo(X,feminino).

pai(X,Y):- progenitor(X,Y),
           sexo(X,masculino).

tio(X,Y):- irmao(X,A),
           progenitor(A,Y).

primo(X,Y):-irmao(A,B),
            progenitor(A,X),
            progenitor(B,Y),
            X\==Y.
primo(X,Y):-irma(A,B),
            progenitor(A,X),
            progenitor(B,Y),
            X\==Y.
% questoes:
q1:- progenitor(jose,joao).
```

```
q1b:- pai(jose,joao).

q2(X):- mae(maria,X).
q2b(L):-findall(X,mae(maria,X),L).

q3(X):- primo(mario,X).
q3b(L):- findall(X,primo(mario,X),L).
q3c(L):- findall(X,primo(mario,X),LR),list_to_set(LR,L).

q4(X):- tio(_,X).
q4b(L):- findall(X,tio(_,X),LR),list_to_set(LR,L).

q5(X):- descendente(X,carlos).
q5b(L):- findall(X,descendente(X,carlos),L).

q6a(X):- irmao(helena,X).
q6b(X):- irma(helena,X).
```

2.2 Exercício sobre Listas

Enunciado:

Represente em **Prolog** os seguintes predicados genéricos sobre listas (sem utilizar os correspondentes predicados do módulo lists do SWI-Prolog):

1) **adiciona(X,L1,L2)** – onde L2 é a lista que contém o elemento X e a lista L1. Testar este predicado no interpretador Prolog, executando:
   ?- adiciona(1,[2,3],L).
   ?- adiciona(X,[2,3],[1,2,3]).
2) **apaga(X,L1,L2)** – onde L2 é a lista L1 sem o elemento X. Testar com:
   ?- apaga(a,[a,b,a,c],L).
   ?- apaga(a,L,[b,c]).
3) **membro(X,L)** – que é verdadeiro se X pertencer à lista L. Testar com:
   ?- membro(b,[a,b,c]).
   ?- membro(X,[a,b,c]). % carregar em ;
   ?- findall(X,membro(X,[a,b,c]),L).
4) **concatena(L1,L2,L3)** – onde L3 é resultado da junção das listas L2 e L1. Testar com:
   ?- concatena([1,2],[3,4],L).
   ?- concatena([1,2],L,[1,2,3,4]).
   ?- concatena(L,[3,4],[1,2,3,4]).
5) **comprimento(X,L)** – onde X é o número de elementos da lista L. Testar com:
   ?- comprimento(X,[a,b,c]).
6) **maximo(X,L)** – onde X é o valor máximo da lista L (assumir que L contém somente números). Testar com:
   ?- maximo(X,[3,2,1,7,4]).
7) **media(X,L)** – onde X é o valor médio da lista L (assumir que L contém somente números). Testar com:
   ?- media(X,[1,2,3,4,5]).
8) **nelem(N,L,X)** – onde N é um número e X é o elemento da lista L na posição L. Por exemplo (testar com):
   ?- nelem(2,[1,2,3],2).
   ?- nelem(3,[1,2,3],X).
   ?- nelem(4,[a,b,c,d,e,f,g],X).

Explicação:

Este exercício serve para praticar a manipulação de **listas**, sendo uma adaptação livre do código apresentado no Capítulo 3 do livro **[Brakto, 1990]**. A maior parte destes predicados já se encontra definido no SWI-Prolog em inglês no módulo lists. Por exemplo: membro - **member**, adiciona - **append**, apaga - **delete**, máximo - **max_list**, nelem - **nth1** (ver mais predicados em **[Wielemaker, 2008b]**). De notar que a maioria dos predicados utilizam o mecanismo de recursividade, por forma a se poder *navegar*

ao longo de uma lista. Para correr o programa no SWI-Prolog basta executar os seguintes comandos:

```
?- [listas].
Yes
?- q1a(L).
L = [1, 2, 3]
```

(executar as restantes questões q1b, q2a, q2b, ...)

Resolução:

listas.pl

```
% 1
adiciona(X,L,[X|L]).

% 2
apaga(X,[X|R],R).
apaga(X,[Y|R1],[Y|R2]):-
      apaga(X,R1,R2).

% 3
membro( X, [X|_] ).
membro( X, [_|R] ) :- membro( X, R ).

% 4
concatena([],L,L).
concatena([X|L1],L2,[X|L3]):- concatena(L1,L2,L3).

% 5
comprimento(0,[]).
comprimento(N,[_|R]):-  comprimento(N1,R),
                        N is 1 + N1.

% 6
max(X,[X]).
max(X,[Y|R]):- max(X,R), X > Y, !.
max(Y,[Y|_]).

% 7
somatorio(0,[]).
somatorio(X,[Y|R]):-    somatorio(S,R),
                        X is S+Y.

media(X,L):-      comprimento(N,L),
                  somatorio(S,L),
                  X is S/N.

nelem(N,L,X):-nelem(N,1,L,X).
nelem(N,N,[X|_],X):-!.
nelem(N,I,[_|R],X):-    I1 is I+1,
                        nelem(N,I1,R,X).

% testar os predicados:
q1a(L):-adiciona(1,[2,3],L).
q1b(X):-adiciona(X,[2,3],[1,2,3]).
```

```
q2a(L):-apaga(a,[a,b,a,c],L).
q2b(L):-apaga(a,L,[b,c]).

q3a:-membro(b,[a,b,c]).
q3b(X):-membro(X,[a,b,c]).
q3c(L):-findall(X,membro(X,[a,b,c]),L).

q4a(L):-concatena([1,2],[3,4],L).
q4b(L):-concatena([1,2],L,[1,2,3,4]).
q4c(L):-concatena(L,[3,4],[1,2,3,4]).

q5(X):-comprimento(X,[a,b,c]).

q6(X):-max(X,[3,2,1,7,4]).

q7(X):-media(X,[1,2,3,4,5]).

q8:-nelem(2,[1,2,3],2).
q8b(X):-nelem(3,[1,2,3],X).
q8c(X):-nelem(4,[a,b,c,d,e,f,g],X).
```