Matemática e Prolog

Delfim F. M. Torres
delfim@mat.ua.pt

Fevereiro de 2002

Departamento de Matemática
Universidade de Aveiro
3810-193 Aveiro, Portugal

Índice

1 Gananciosa (CeNPL'01, Univ. da Beira Interior) 2

2 Nim (CeNPL'00, Univ. do Minho) 5

3 Sangradura (CeNPL'00, Univ. do Minho) 10

4 Cucamonga (CeNPL'99, Univ. Nova de Lisboa) 13

5 Animais (CeNPL'98, Univ. de Aveiro) 16

Nótula, à guisa de prefácio

Este manuscrito surgiu da necessidade de adestrar equipas de estudantes que representem a Universidade de Aveiro no próximo Concurso/Encontro Nacional de Programação em Lógica, CeNPL'02, que se realizará no Departamento de Informática da Universidade de Coimbra de 11 a 13 de Abril de 2002. Este evento anual surgiu no Departamento de Matemática da Universidade de Aveiro, em 1998, e o leitor interessado poderá encontrar informação detalhada sobre todas as edições (1998 em Aveiro, 1999 em Lisboa, 2000 em Braga, 2001 na Covilhã e 2002 em Coimbra) no endereço `http://www.mat.ua.pt/cnpl`. Dois dos meus três problemas de 1998 ("Os dez degraus do Miguel" e o "Codificador Primo"), com respectivas soluções em Prolog, foram já publicados em [1]. Aqui são apresentados os restantes problemas da minha autoria, que apareceram nos Concursos/Encontros Nacionais de Programação em Lógica de 1998 (problema "Animais"), de 1999 ("Cucamonga"), de 2000 ("Nim" e "Sangradura") e 2001 ("Gananciosa"), assim como possíveis soluções. Divirtam-se com o Prolog, a Informática e a Matemática! Participem nos próximos CeNPL's!

1 Gananciosa

CeNPL'01, Departamento de Matemática e Informática, Universidade da Beira Interior, 4–6 de Abril de 2001. Inspirado no livro [2, pp. 160–162].

Fracções unitárias

As fracções unitárias são fracções cujo numerador é 1. Tais fracções eram muito valorizadas pelos antigos Egípcios. O papiro de Rhind ou de Ahmes, com mais de três milénios e meio, começa com uma tabela que representa $\frac{2}{n}$ como uma soma de fracções unitárias distintas para todos os valores ímpares de n de 5 a 101. Por exemplo:

$$\begin{aligned} \frac{2}{5} &= \frac{1}{3} + \frac{1}{15} \\ \frac{2}{7} &= \frac{1}{4} + \frac{1}{28}. \end{aligned}$$

É trivial representar uma fracção como a soma de fracções unitárias se for possível repetir termos. A fracção $\frac{4}{5}$, por exemplo, torna-se $\frac{1}{5} + \frac{1}{5} + \frac{1}{5} + \frac{1}{5}$. Mas, se for exigido que todos os denominadores sejam distintos, como faziam os Egípcios, será uma representação unitária sempre possível? Foi em 1202 que Leonardo Fibonacci, o maior matemático europeu da Idade Média, demonstrou que a resposta era afirmativa.

A expansão gananciosa de Fibonacci

Vamos ilustrar o método de Fibonacci para a fracção $\frac{3}{7}$. A obtenção do algoritmo para a situação genérica é completamente trivial.

Começamos por tomar a maior fracção unitária menor do que $\frac{3}{7}$ – nomeadamente $\frac{1}{3}$, dado que $\frac{1}{2}$ é um pouco maior do que $\frac{3}{7}$ – e subtraímos $\frac{1}{3}$ de $\frac{3}{7}$ para obter $\frac{2}{21}$. Agora a maior fracção unitária menor do que $\frac{2}{21}$ é $\frac{1}{11}$. Subtraindo $\frac{1}{11}$ de $\frac{2}{21}$ fica $\frac{1}{231}$: uma fracção unitária, pelo que o procedimento termina. A expansão gananciosa de $\frac{3}{7}$ é então:

$$\frac{3}{7} = \frac{1}{3} + \frac{1}{11} + \frac{1}{231}.$$

Fibonacci mostrou que este procedimento ganancioso produz sempre uma soma de fracções que tem um fim.

Tarefa

A sua tarefa consiste em escrever um programa Prolog, que dada uma fracção, definida pelo seu numerador e denominador, devolva a respectiva expansão gananciosa. Implemente para isso o predicado

```
gananciosa/3
```

que recebe no primeiro argumento o numerador; no segundo o denominador; e devolve no terceiro argumento uma lista com os denominadores das fracções unitárias que formam a expansão gananciosa dessa fracção. *Use apenas aritmética inteira e simplifique sempre as fracções.*

Os Dados

Não existem mais dados a considerar neste problema.

Os Resultados

Mostramos de seguida dois exemplos de uso do predicado `gananciosa/3` e os respectivos resultados esperados:

```
?- gananciosa(3,7,EG).
   EG = [3,11,231]

?- gananciosa(3,121,EG).
   EG = [41,2481,12308241]
```

Testes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'Testes' para o problema GANANCIOSA
% por Delfim (delfim@mat.ua.pt)
% 19/Dezembro/2000
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

?- gananciosa(3,7,EG).
   EG = [3,11,231]

?- gananciosa(3,121,EG).
   EG = [41,2481,12308241]

?- gananciosa(5,17,EG).
   EG = [4,23,1564]

?- gananciosa(4,17,EG).
   EG = [5,29,1233,3039345]

?- gananciosa(16,17,EG).
   EG = [2,3,10,128,32640]
```

Uma Solução

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          CeNPL'2001
% Solucao do problema GANANCIOSA
% Delfim (delfim@mat.ua.pt)
% 19/Dezembro/2000
% Ultima alteracao: 4/Fev/2001
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
gananciosa(N,D,L) :-
  simplifica(N,D,NN,ND),
  gan(NN,ND,L), !.

gan(1,D,[D]) :- !.
gan(N,D,[MFU|R]) :-
  mfu(N,D,1,MFU),
  subtrai(N,D,MFU,NN,ND),
  gan(NN,ND,R), !.

mfu(N,D,M,M) :-
  N * M > D.
mfu(N,D,I,M) :-
  I1 is I + 1,
  mfu(N,D,I1,M).

subtrai(N,D,MFU,NN,ND) :-
  N1 is N*MFU-D,
  D1 is D*MFU,
  simplifica(N1,D1,NN,ND).

simplifica(N,D,NN,ND) :-
  decompoe(N,2,LN),
  decompoe(D,2,LD),
  tiracomuns(LN,LD,L1,L2),
  multiplica(L1,NN),
  multiplica(L2,ND).

decompoe(1,_,[]).
decompoe(N,I,[I|R]) :-
  0 is N mod I,
  NN is N // I,
  decompoe(NN,2,R).
decompoe(N,I,L) :-
  I1 is I + 1,
  decompoe(N,I1,L).

tiracomuns([],L,[],L).
tiracomuns([X|R],L,R1,R2) :-
  membro(X,L),
  tira(X,L,LX),
  tiracomuns(R,LX,R1,R2).
tiracomuns([X|R],L,[X|R1],R2) :-
  tiracomuns(R,L,R1,R2).
```

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

tira(X,[X|R],R).
tira(X,[Y|R],[Y|T]) :-
  tira(X,R,T).

multiplica([],1).
multiplica([X|R],M) :-
  multiplica(R,MR),
  M is X * MR.
```

2 Nim

CeNPL'00, Departamento de Informática, Universidade do Minho, 5–7 de Abril de 2000.

O Jogo

O *Nim* joga-se com pilhas de feijões e foi primeiro estudado por Charles Bouton em 1901.[1] Neste jogo, cada jogador, quando lhe toca jogar, pode retirar feijões de uma só pilha, mas quantos quizer, de um mínimo de um a um máximo de toda a pilha. Ganha o jogador que retirar o último punhado de feijões.

Tarefa

A sua tarefa consiste em escrever um programa Prolog que permita jogar o Nim sem nunca perder!

Níneros e o jogo do Nim

"Nímero" é a palavra que resulta de se combinar "Nim" com "número" (*vide* [4]). Os níneros têm uma nova aritmética, para a qual usaremos aqui os símbolos $\oplus$ e $\equiv$. Quais são as posições boas que convém atingir num jogo Nim? Se se estiver a jogar Nim apenas com um monte, a resposta é fácil. É bom passar para 0 e é mau passar para qualquer outro nínero $1, 2, 3, 4, 5, 6, \ldots$ visto que então o oponente poderá contra-atacar, passando para 0. Eis o que é necessário saber sobre a adição de níneros:

- A soma de dois níneros iguais é sempre zero.
- Se o maior de dois níneros for uma potência de 2: 1 ou 2 ou 4 ou 8 ou 16 ou ..., eles somam-se como os correspondentes números ordinários.

Com estas regras e as leis da álgebra, pode fazer-se aritmética Nim. Exemplo:

$$5 \oplus 6 \equiv (4 \oplus 1) \oplus (4 \oplus 2) \equiv (4 \oplus 4) \oplus (1 \oplus 2) \equiv 3.$$

[1] Mais informação sobre este e outros jogos do mesmo tipo, pode ser encontrada em [3].

Qualquer conjunto de montes Nim, a, b, c, ... pode ser substituído por um monte único de tamanho $a \oplus b \oplus c \oplus \cdots$. Assim, a resposta à questão colocada será: deverá tentar-se jogar para uma situação a, b, c, ... para a qual se tenha $a \oplus b \oplus c \oplus \cdots \equiv 0$. Vejamos um exemplo. Se o utilizador passou para 3, 5, 7 (ou tiver escolhido esta como configuração inicial) então, como $3 \oplus 5 \oplus 7 \equiv (2 \oplus 1) \oplus (4 \oplus 1) \oplus 7 \equiv 6 \oplus 7 \equiv \cdots \equiv 1$ o seu programa deverá jogar para uma das possibilidades 2, 5, 7 ou 3, 4, 7 ou 3, 5, 6 que somam zero: $2 \oplus 5 \oplus 7 \equiv 3 \oplus 4 \oplus 7 \equiv 3 \oplus 5 \oplus 6 \equiv 0$. Prova-se que é sempre possível passar de uma situação perdente para uma ganhante. Para transformar uma situação perdente numa ganhante, somamos o valor do tamanho da situação com o número de feijões de uma determinada pilha. Se o resultado for inferior ou igual ao número de feijões da pilha, devemos tirar elementos dessa pilha de modo a deixar lá esse resultado. Se o resultado for superior, devemos tirar feijões de uma outra pilha. Por exemplo, da situação perdente 1, 8, 4, 7 (que tem tamanho $1 \oplus 8 \oplus 4 \oplus 7 \equiv 10$) podemos passar para a situação ganhante 1, 2, 4, 7: $10 \oplus 8 \equiv 2 < 8$ e $1 \oplus 2 \oplus 4 \oplus 7 \equiv 0$.

Os Dados

Os dados a considerar neste problema são o número de pilhas e o número de feijões em cada pilha. Estes dados serão especificados pelo utilizador por uma lista `Pilhas` do tipo `[NumFeijPilha1, ..., NumFeijPilhaN]` aquando da chamada ao predicado `nim(Pilhas)`. Antes do início do jogo propriamente dito, o seu programa deve usar o predicado

```
soma(Pilhas,T)
```

que devolve em `T` a soma dos níneros em `Pilhas`, para decidir se começa ele ou se, pelo contrário, opta por deixar o utilizador fazer a primeira jogada. Todas as jogadas do computador devem ser realizadas à custa do predicado `proximaSituacao(SP,SG)` que, dada uma situacao perdente `SP`, devolve uma situação ganhante `SG` (outras situações ganhantes deverão poder ser obtidas por *backtracking*).

Os Resultados

Seguem-se alguns exemplos:

```
?- soma([1,8,4,7],S).
   S = 10

?- proximaSituacao([1,8,4,7],PS).
   PS = [1,2,4,7];
   no

?- nim([1,1,2,3,1]). % Computador cede a 1a jogada ao utilizador
--> Utilizador joga para:  [1,1,2,0,1]. % por exemplo
--> Computador joga para:  [1,1,1,0,1]
--> Utilizador joga para:  ...........  % etc.
```

Uma Solução

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NIM
%
% Uma solucao baseada em nimeros
% Delfim, 28/01/2000
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%
%     EXEMPLO
%
% ?- soma([3,5,7],S).
%    S = 1
%    yes
%%%%%%%%%%%%%%%%%%%%%%

soma(L,S) :-
  expandePot2(L,LP2),
  ordena(LP2,LO),
  tiraParesIguais(LO,LOS),
  calcula(LOS,S), !.

expandePot2([],[]).
expandePot2([0|R],RE) :-
  expandePot2(R,RE).
expandePot2([X|R],[X|RE]) :-
  potencia2(X),
  expandePot2(R,RE).
expandePot2([X|R],[N1|RE]) :-
  decompoe(X,N1,N2),
  expandePot2([N2|R],RE).

potencia2(1).
potencia2(N) :-
  0 is N mod 2,
  R is N // 2,
  potencia2(R).

decompoe(N1,N1,0) :-
  potencia2(N1).
decompoe(N1,P1,P2) :-
  N is N1 - 1,
  decompoe(N,P1,P),
  P2 is P + 1.
```

```
ordena(L,L) :-
  ordenada(L).
ordena(L,LO) :-
  bubble(L,L1),
  ordena(L1,LO).

ordenada([]).
ordenada([_]).
ordenada([X,Y|R]) :-
  X =< Y, ordenada([Y|R]).

bubble([X],[X]).
bubble([X,Y|R],[X|O]) :-
  X =< Y,
  bubble([Y|R],O).
bubble([X,Y|R],[Y|O]) :-
  bubble([X|R],O).

tiraParesIguais(L,R) :-
  membro2(X,L),
  divideLista(L,L1,[X,X],L3),
  junta(L1,L3,J),
  tiraParesIguais(J,R).
tiraParesIguais(L,L).

membro2(X,[X,X|_]). membro2(X,[_,Y|R]) :- membro2(X,[Y|R]).

divideLista(L,L1,L2,L3) :-
  junta(L1,L2,J),
  junta(J,L3,L).

junta([],L,L). junta([X|R],L,[X|J]) :- junta(R,L,J).

calcula([],0).
calcula([X],X).
calcula([X,Y|R],S) :-
  S1 is X + Y,
  calcula([S1|R],S).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          EXEMPLO
% ?- proximaSituacao([3,5,7],S).
%    S = [2,5,7];
%    S = [3,4,7];
%    S = [3,5,6];
%    yes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
proximaSituacao(S,PS) :-
  soma(S,N),
  proxS(N,S,PS).

proxS(N,[X|R],[V|R]) :-
  soma([N,X],V),
  V =< X.
proxS(N,[X|R1],[X|R2]) :-
  proxS(N,R1,R2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     PREDICADO PRINCIPAL
%
%  nim([NumFeij1,...,NumFeijN])
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nim(SI) :-
  soma(SI,S),
  S =:= 0,
  joga(utilizador,SI),!.
nim(SI) :-
  joga(computador,SI),!.

joga(utilizador,_) :- % nao faco verificacoes de validade
  write('Utilizador joga para: '),
  read(NS),
  joga(computador,NS).
joga(computador,SA) :-
  proximaSituacao(SA,PS),
  write('Computador joga para: '),
  write(PS), nl,
  continua(PS).

continua(PS) :-
  nao_acabou(PS),
  joga(utilizador,PS).
continua(_) :-
  write('Ganhei!'), nl.

nao_acabou(L) :-
  membro(X,L),
  X =\= 0.

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

3 Sangradura

CeNPL'00, Departamento de Informática, Universidade do Minho, 5–7 de Abril de 2000. A fonte de inspiração foi [5].

O Desafio

Os cincos duques de Sangradura (Bóris, o Cruel; Bóris, o Impiedoso; Ferdinando, o Selvagem; Ivan, o Flagelador; e Klaus, o Inqualificável) decidiram declarar guerra a países vizinhos, usando pretextos fúteis. Partindo da seguinte informação:

- O governador de Pacífica provocou a ira de um duque de Sangradura, que não era Klaus, o Inqualificável, aprovando uma lei anti-violência que o duque considerou contrária à ordem natural da criação e, portanto, herética;
- Bóris, o Cruel, levou dez dias para triunfar sobre o seu inimigo;
- A guerra mais longa não foi travada contra Betífica;
- Foi o duque Ferdinando, o Selvagem, quem declarou guerra quando o monarca de um país vizinho se esqueceu de o cumprimentar pelo seu aniversário com um presente apropriado;
- A vítima de Ferdinando, o Selvagem, não foi Misericórdia;
- A guerra mais curta foi a declarada quando alguns pombos de um país vizinho invadiram o espaço aéreo do ducado de Sangradura;
- A guerra de Ivan, o Flagelador, foi contra Inocência, que não era o país onde hastearam a bandeira de Sangradura de cabeça para baixo, ofendendo o duque que estava de visita oficial à capital do país inimigo;
- A guerra com Miserircórdia durou sete dias; Felícia não conseguiu resistir tanto;
- Chamava-se Bóris o duque que reinava quando um cidadão inocente de Sangradura foi injustamente feito prisioneiro por um estado vizinho, por ter assassinado acidentalmente um habitante local a quem devia dinheiro;

determine, para cada duque, o país atacado, o motivo da guerra e quantos dias se passaram em cada caso (2, 4, 7, 10 ou 13 dias), até ao triunfo inevitável das forças de Sangradura.

Tarefa

A sua tarefa consiste em escrever um programa Prolog que solucione o desafio acima colocado.

Os Dados

Não existem mais dados a considerar neste problema.

Os Resultados

O programa deve ser invocado através do predicado

```
sangradura/0
```

que mostrará no ecrã os cinco quádruplos

```
[Duque,País,Pretexto,DuraçãoDaGuerra]
```

que são consequência lógica da informação dada.

Uma Solução

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Sangradura
%
% Uma solucao possivel
% Delfim, 20/12/1999
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% inimigos(ListaPaisesVizinhosDeSangradura)

inimigos([pacifica,beatifica,misericordia,inocencia,felicia]).

% pretextos(ListaMotivosGuerra)

pretextos([ lei_anti_violencia,
            aniversario_esquecido,
            pombos_no_espaco_aereo,
            bandeira_invertida,
            cidadao_aprisionado
          ]).

% diasGuerra(ListaDias)

diasGuerra([2,4,7,10,13]).

sangradura :-
  S = [   [[boris,cruel],I1,P1,10],
          [[boris,impiedoso],I2,P2,DG2],
          [[ferdinando,selvagem],I3,aniversario_esquecido,DG3],
          [[ivan,flagelador],inocencia,P4,DG4],
          [[klaus,inqualificavel],I5,P5,DG5]
      ],
  inimigos(LI),
  pretextos(LP),
  diasGuerra(LDG),!,
```

```
  permutacao(LI,[I1,I2,I3,inocencia,I5]),
  permutacao(LP,[P1,P2,aniversario_esquecido,P4,P5]),
  permutacao(LDG,[10,DG2,DG3,DG4,DG5]),
  valida(S),
  mostra(S).

valida([]).
valida([X|_]) :-
  nao_valida(X), !, fail.
valida([_|R]) :-
  valida(R).

nao_valida([[klaus,inqualificavel],pacifica,_,_]).
nao_valida([_,pacifica,P,_]) :-
  P \= lei_anti_violencia.
nao_valida([[boris,cruel],_,_,N]) :-
  N =\= 10.
nao_valida([_,beatifica,_,M]) :-
  diasGuerra(LDG),
  maior(LDG,M).
nao_valida([[ferdinando,selvagem],_,P,_]) :-
  P \= aniversario_esquecido.
nao_valida([[ferdinando,selvagem],misericordia,_,_]).
nao_valida([_,_,pombos_no_espaco_aereo,DG]) :-
  diasGuerra(LDG),
  menor(LDG,M),
  DG =\= M.
nao_valida([[ivan,flagelador],I,_,_]) :-
  I \= inocencia.
nao_valida([_,inocencia,bandeira_invertida,_]).
nao_valida([_,misericordia,_,DG]) :-
  DG =\= 7.
nao_valida([_,felicia,_,DG]) :-
  DG >= 7.
nao_valida([[D,_],_,cidadao_aprisionado,_]) :-
  D \= boris.

maior([N],N).
maior([M|R],ML) :-
  maior(R,MR),
  ( M >= MR, !, ML = M ;  ML = MR ).

menor([N],N).
menor([M|R],ML) :-
  menor(R,MR),
  ( M =< MR, !, ML = M ;  ML = MR ).
```

```
mostra([]).
mostra([X|R]) :-
  write(X), nl,
  mostra(R).

permutacao([],[]).
permutacao([X|R],P) :-
  permutacao(R,RP),
  tira(X,P,RP).

tira(X,[X|R],R).
tira(X,[Y|R],[Y|RSX]) :-
  tira(X,R,RSX).
```

4 Cucamonga

CeNPL'99, Departamento de Informática, Universidade Nova de Lisboa, 13–15 de Abril de 1999. Inspirado em [6].

Mães e Filhos à Mistura...

No sexto número do jornal *O Integral* do *"Clube de Matemática"*, pode ser encontrado o seguinte problema:

> A população da aldeia de Cucamonga é constituída por seis homens (Abelardo, Béléléu, Cunta Kinté, Didol, Eleutério e Flaneco) e pelas suas mães.
>
> Cada uma das mães é viúva e casada, em segundas núpcias, com um dos outros cinco homens.
>
> Assim, a esposa de Didol faz notar à mãe de Cunta Kinté que ela (mãe de Cunta Kinté) se tornou bisavó (por afinidade) da esposa de Eleutério, que Abelardo se tornou avô de Béléléu e que a esposa de Flaneco se tornou nora da neta da esposa de Cunta Kinté.
>
> *Quem casou com quem?*

Tarefa

Escreva um programa em Prolog que resolva este jogo, i.é, determine a resposta à questão acima colocada.

Os Dados

Neste problema não existem outros dados para além do conhecimento descrito na introdução.

Os Resultados

O seu programa deve ser activado através do predicado

```
solucao/0
```

que mostra no ecrã a solução para este problema:

```
abelardo casou com mae_?
beleleu casou com mae_?
cuntaKinte casou com mae_?
didol casou com mae_?
eleuterio casou com mae_?
flaneco casou com mae_?
```

Uma Solução

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solução para o problema CUCAMONGA
% por Delfim F. Marado Torres, Fev. 1999
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mae(mae_abelardo,abelardo).
mae(mae_beleleu,beleleu).
mae(mae_cuntaKinte,cuntaKinte).
mae(mae_didol,didol).
mae(mae_eleuterio,eleuterio).
mae(mae_flaneco,flaneco).

pertence(X,[X|_]).
pertence(X,[_|R]) :- pertence(X,R).

pai(X,Y,S) :-
  pertence(X/Z,S),
  mae(Z,Y).

filho(X,Y,_) :-
  mae(Y,X).
filho(X,Y,S) :-
  pai(Y,X,S).

progenitor(X,Y,S) :-
  filho(Y,X,S).

avo(W,X,S) :-
  progenitor(Y,X,S),
  progenitor(W,Y,S).
```

```
nora(X,Z,S) :-
  pertence(Y/X,S),
  progenitor(Z,Y,S).

neta(X,A,S) :- avo(A,X,S).

bisavo(C,X,S) :-
  progenitor(C,B,S),
  avo(B,X,S).
bisavo(C,X,S) :-
  progenitor(C,B,S),
  avo(B,X1,S),
  pertence(X1/X,S).

solucao :-
  S = [abelardo/Y1,beleleu/Y2,cuntaKinte/Y3,
       didol/Y4,eleuterio/Y5,flaneco/Y6],
  M = [mae_abelardo,mae_beleleu,mae_cuntaKinte,
       mae_didol,mae_eleuterio,mae_flaneco],
  permutacao(M,[Y1,Y2,Y3,Y4,Y5,Y6]),
  Y1 \= mae_abelardo,
  Y2 \= mae_beleleu,
  Y3 \= mae_cuntaKinte,
  Y4 \= mae_didol, Y4 \= mae_cuntaKinte,
  Y5 \= mae_eleuterio,
  Y6 \= mae_flaneco,
  avo(abelardo,beleleu,S),
  neta(Y,Y3,S),
  nora(Y6,Y,S),
  bisavo(mae_cuntaKinte,Y5,S),
  mostra(S).

permutacao([],[]).
permutacao(S,[X|P]) :-
  pertence(X,S),
  tira(X,S,R),
  permutacao(R,P).

tira(X,[X|R],R).
tira(X,[Y|R],[Y|T]) :-
  tira(X,R,T).

mostra([]).
mostra([X/Y|R]) :-
  write(X), write(' casou com '), write(Y), nl,
  mostra(R).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%?- solucao.
%% abelardo casou com mae_cuntaKinte
%% beleleu casou com mae_eleuterio
%% cuntaKinte casou com mae_beleleu
%% didol casou com mae_abelardo
%% eleuterio casou com mae_flaneco
%% flaneco casou com mae_didol
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5 Animais

CeNPL'98, Departamento de Matemática, Universidade de Aveiro, 27–28 de Abril de 1998. Inspirado em [7].

Caça

Atente ao seguinte enunciado:

> *Cinco amigos de nomes predestinados, Coelho, Pato, Rola, Pombo e Lebre, regressam de uma caçada. Trazem consigo animais correspondentes aos respectivos nomes. Cada um matou um único animal, que não corresponde, porém, ao seu nome. Cada um falhou um animal diferente do que matou e que não corresponde também ao próprio nome. O coelho foi morto pelo caçador cujo nome é o do animal falhado por Rola. O pato foi morto pelo caçador cujo nome é o do animal falhado por Lebre. Pato, que falhou uma lebre, ficou muito aborrecido por ter morto apenas uma rola. Quais os bichos mortos e falhados por cada um dos caçadores?*

Tarefa

Escreva um programa em Prolog que resolva este jogo, i.é, determine a resposta à questão acima colocada.

Os Dados

Neste problema não existem outros dados para além do conhecimento descrito na introdução.

Os Resultados

O seu programa deve ser activado através do predicado

```
jogo/0
```

que mostra no ecrã os 5 triplos

```
Cacador/Animal_Morto/Animal_Falhado
```

Uma Solução

```
%%%
```

```
faz_regra(C/N1/F,N2/M/C,LC,LM,LF,NLC,NLM,NLF) :-
  tirar(C,LC,LCI), tirar(N2,LCI,NLC),
  tirar(N1,LM,LMI), tirar(M,LMI,NLM),
  tirar(F,LF,LFI), tirar(C,LFI,NLF),
  diferentes(C,N1,F), diferentes(N2,M,C).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%     Predicados auxiliares         %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

diferentes(C,M,F) :-
  C \= M, C \= F, M \= F.

tirar(X,[X|R],R).
tirar(X,[Y|L],[Y|R]) :-
  tirar(X,L,R).

mostra([]).
mostra([X|R]) :-
  write(X), nl,
  mostra(R).
```

Referências

[1] Delfim F. M. Torres, *Introdução à Programação em Lógica*, Universidade de Aveiro, 2000 (346 pp, ISBN 972–8021–93–3).

[2] Paul Hoffman, *O Homem que só gostava de números – A história de Paul Erdös e a busca da Verdade Matemática*, Gradiva, 2000.

[3] Jorge Nuno Silva, *Jogórios*, Boletim da SPM (Sociedade Portuguesa de Matemática), n^o40, 1999, pp. 57–68.

[4] John H. Conway e Richard K. Guy, *O Livro dos Números*, Gradiva Universidade de Aveiro, 1999 (Tradução de José Sousa Pinto).

[5] Jornal *"O Integral"* N^o 5, Ano III, *Clube de Matemática*, Departamento de Matemática, Universidade de Aveiro, 1998.

[6] Jornal *"O Integral"* N^o 6, Ano III, *Clube de Matemática*, Departamento de Matemática, Universidade de Aveiro, 1998.

[7] Pierre Berloquin, *100 Jogos Lógicos*, O Prazer da Matemática, n^{o}4, Gradiva, 2^a edição, 1991.