TÉCNICO LISBOA

Lógica para Programação

Relatório do Projecto

2012/2013

Os suspeitos do costume

Grupo	**Nº:** 72913	Bruno Alexandre Pires Henriques
43	**Nº:** 72960	Tiago Manuel Ferrão dos Santos

1 – Introdução

O presente relatório tem como objectivo demonstrar como desenvolvemos um programa em Prolog que permite ao detective Horácio compreender, de uma forma progressiva e automática, o conjunto de pistas que a Avó Einstein lhe oferece, podendo assim identificar criminosos através de dedução lógica.

2 – Implementação

O nosso programa assenta apenas sobre regras. Durante a produção da solução para o problema apresentado, optámos por usufruir o que o Prolog tinha para oferecer, isto é, optámos por usar regras *built-in* de manipulação listas e criar regras que irão servir de base para outras. Por outro lado, tirámos partido de um programa em *Prolog* depender da ordem com que as regras são apresentadas. Isto é relevante para o correcto e esperado funcionamento do programa desenvolvido, visto que a ordem com que estas estão apresentadas é crucial para uma resolução correcta , contribuindo também para a eficiência desta.

Por exemplo, optou-se por criar a seguinte regra auxiliar: `sublista(X,Y)` que devolve `true` se X for uma sublista de Y e `false` caso contrário. Esta serviu de base para as regras `lado(S1,S2,Suspeitos)` e `entre(S1,S2,S3, Suspeitos)` (ambas descritas no enunciado do presente projecto), pois era intuitivo pesquisar a lista `Suspeitos` por uma sublista `[S1, S2]` ou `[S2, S1]`, para o caso de `S1` e `S2` estarem seguidos, ou `[S1, S2, S3]` ou `[S3, S2, S1]`, para o caso de `S2` estar entre `S1` e `S3`. A regra `entre(S1, S2, S3)` foi resultado da reformulação duma solução inicial não tão evidente mas igualmente simples onde `s1`, `s2` e `s3` eram procurados seguidos na lista, não necessariamente por essa ordem, de uma forma recursiva.

Como, tal como descrito no enunciado, a regra `esquerda(S1, S2, Suspeitos)` não implica que `S1` e `S2` estejam seguidos na lista Suspeitos, optámos por utilizar uma regra recursiva onde o caso de paragem era quando tínhamos `S1` na cabeça da lista e `S2` como membro do resto da lista. Como

chamada à recursão temos como sub-objectivo `esquerda(S1, S2, L)` onde `L` é a lista `Suspeitos` sem o seu primeiro elemento. A regra `direita(S1, S2, Suspeitos)` é uma consequência imediata de `esquerda(S2, S1, Suspeitos)`. Por fim, a regra `ou(S1,S2,S3, Suspeitos)` e a regra `naoEntre(S1, S2, S3, Suspeitos)` foram as mais problemáticas, sendo que a descrição dos problemas encontrados e a solução para os mesmos encontra-se na secção 3.

3 - Problemas encontrados e sua solução

Durante o desenvolvimento do projecto, inicialmente tivémos problemas na elaboração da regra `naoEntre(S1, S2, S3, Suspeitos)`, pois a regra intuitivamente pensada, usando a negação da regra `entre(S1,S2,S3, Suspeitos)` com as variáveis `S1`, `S2` e `S3`, não era capaz de deduzir os suspeitos possíveis visto que o uso do operador *not/1* impedia o comportamento desejado, pois não estávamos a inicializar as variáveis `S1`, `S2` e `S3` como elementos existentes na lista de `Suspeitos`. Deste modo foi necessário alterar a abordagem utilizada para que obtivéssemos o comportamento desejado. A segunda abordagem, passou por ser uma mera correcção da anterior, onde após inicializadas as variáveis do mesmo modo descrito acima, procurou-se o suspeito entre o primeiro e o terceiro e verificava-se se este era diferente do suspeito `S2`. Esta abordagem, apesar de produzir os resultados esperados nos testes fornecidos pelo corpo docente, não passava nos testes por nós realizados - descrito na seccção de exemplos de utlização do programa.

Finalmente, optámos por tentar obter as posições numéricas dos elementos `S1`, `S2`, e `S3` na lista `Suspeitos` e, de seguida, comparar as posições numéricas dos suspeitos `S1`, `S2` e `S3` e garantir que existe apenas um elemento entre `S1` e `S3` e que `S2` não ocupe essa mesma posição. Com esta abordagem, obtivémos os resultados esperados nos testes por nós realizados e nos fornecidos pelo corpo docente.

À semelhança do primeiro problema encontrado na regra `naoEntre(S1, S2,`

`S3, Suspeitos)`, foi necessário na regra `ou(S1, S2, S3, Suspeitos)` inicializar as variáveis através do `existe(S1, Suspeitos)`, unificando-as com o seu elemento respectivo na lista de `Suspeitos`. Deste modo, é possível com as variáveis unificadas verificar o ou exclusivo, i.e. verificar se `S1` é `S2` e não `S2` ou vice-versa.

4 – Limitações

As limitações do nosso programa são aproximadamente as mesmas descritas no enunciado do presente projecto. Sendo que a mais notável encontra-se na regra `naoEntre(S1, S2, S3, Suspeitos)` onde é considerado que entre os suspeitos `S1` e `S3` apenas existe um suspeito. Como tal, caso a avó Einstein refira alguma pista com intenção de referir que `S2` não está entre `S1` e `S3` independente da distância relativa entre eles, o nosso programa não abrange esse caso. Apesar desse facto, é possível no mesmo alterar de modo a abrangir esse caso não perdendo funcionalidade antiga.

Outra limitação do nosso programa é a dependência da lista de suspeitos. Isto é, caso a avó Einstein diga uma pista que involva um suspeito que não existe, quando analisada resultará num resultado falso, mesmo que ela tenha referido um suspeito não existente na lista. Por exemplo um `ou(S1,S2,S3, Suspeitos)` em que `S2` existe e `S3` não devolverá falso apesar de o suspeito `S1` ser o suspeito `S3`.

5 - Exemplos de funcionamento

Os testes pessoais referidos anteriormente estão no ficheiro testespessoais.pl. O Resultado esperado encontra-se em comentário no topo do ficheiro.

Ficheiro testespessoais.pl:

```
%(TiaGracinda,tricot,      botas),(Krillin,palito,      Descalco),
(MeganRaposa, PomPom, Sandalias), (Alladeen, Beard, Barbatana)

testespessoais:-
N=[suspeito(X,ArmaV,CalcadoV),suspeito(NomeB,ArmaB,CalcadoB),susp
eito(NomeMF,ArmaMF,CalcadoMF), suspeito(NomeAl, ArmaAl,
CalcadoAl)],
     lado(suspeito('TiaGracinda',_,_), suspeito('Krillin',_,_),
N),
      naoEntre(suspeito(_,'palito',_),suspeito(_,'Tricot',_),suspe
ito(_,_,'Barbatanas'),N),
     ou(suspeito(_,_,'Descalco'), suspeito('Krillin',_,_),
suspeito(_,'PomPom',_),N),
     entre(suspeito('TiaGracinda',_,_),
suspeito(_,'palito',_),suspeito(_,_,'Sandalias'), N),
     esquerda(suspeito(_,_,'Botas'),suspeito('MeganRaposa',_,_),
N),
     direita(suspeito('Krillin',_,_), suspeito(_,'Tricot',_), N),
     direita(suspeito(_,_,'Sandalias'), suspeito(_,'Tricot',_),
N),
     existe(suspeito('Alladeen',_,_), N),
     esquerda(suspeito(_,_,'Descalco'),suspeito('Alladeen',_,_),
N),
      naoEntre(suspeito(_,_,'Botas'),suspeito(_,'Barba',_),suspeit
o(_,_,'Sandalias'),N),
      lado(suspeito('MeganRaposa',_,_),
suspeito(_,_,'Barbatanas'), N),

ou(suspeito('Alladeen',_,_),suspeito(_,'Barba',_),suspeito(_,_,'D
escalco'), N),
     write('NomeTia-'), write(X),
     write('|ArmaTia-'), write(ArmaV),
     write('|CalcadoTia-'), write(CalcadoV),
     write('--------------'),
     write('|NomemeKrillin-'), write(NomeB),
     write('|ArmaKrillin-'), write(ArmaB),
     write('|CalcadoKrillin-'), write(CalcadoB),
     write('--------------'),
    write('NomeMeganRaposa-'),write(NomeMF),
     write('ArmaMeganRaposa-'), write(ArmaMF),
     write('CalcadoMeganRaposa-'),write(CalcadoMF),
     write('--------------'),
     write('|NomeAlladeen-'),write(NomeAl),
     write('|ArmaAlladeen-'),write(ArmaAl),
     write('|CalcadoAlladeen-'),write(CalcadoAl).
```

```
?- existe(X,[1,2,3]).
X = 1 ;
X = 2 ;
X = 3.

10 ?- lado(X, Y, [1,2,3]).
X = 1,
Y = 2 ;
X = 2,
Y = 3 ;
X = 2,
Y = 1 ;
X = 3,
Y = 2 ;
false.

?- direita(X, Y, [1,2,3]).
X = 2,
Y = 1 ;
X = 3,
Y = 1 ;
X = 3,
Y = 2 ;
false.

?- esquerda(X, Y, [1,2,3]).
X = 1,
Y = 2 ;
X = 1,
Y = 3 ;
X = 2,
Y = 3 ;
false.

?- naoEntre(X, 2, 3, [1,2,3,4,5]).
X = 5 ;
false.

?- naoEntre(1, X, 3, [1,2,3,4,5]).
X = 1 ;
X = 3 ;
X = 4 ;
X = 5 ;
false.
```

```
?- ou(2,X,Y,[1,2,3]).
X = 2,
Y = 1 ;
X = 2,
Y = 3 ;
X = 1,
Y = 2 ;
X = 3,
Y = 2 ;
false.

?- entre(X, Y, Z, [1,2,3,4]).
X = 1,
Y = 2,
Z = 3 ;
X = 2,
Y = 3,
Z = 4 ;
X = 3,
Y = 2,
Z = 1 ;
X = 4,
Y = 3,
Z = 2 ;
false.

?- testespessoais.
NomeTia-TiaGracinda|ArmaTia-Tricot|
CalcadoTia-Botas--------------|
NomemeKrillin-Krillin|ArmaKrillin-
palito|CalcadoKrillin-
Descalco--------------|
NomeMeganRaposa-MeganRaposa|
ArmaMeganRaposa-PomPom|
CalcadoMeganRaposa-
Sandalias--------------|
NomeAlladeen-Alladeen|ArmaAlladeen-
Barba|CalcadoAlladeen-Barbatanas
true ;
false.
```

Tabela 1: Exemplos de utilização.

Como evidenciado no *output* descrito acima com testes mais básicos e mais tarde com a execução dos testes por nós criados, o programa produz os resultados esperados.

6 - Listagem do Código

Ficheiro suspeitosDoCostume.pl:

```
% Projecto - Lógica para Programação
% -------------------------------
%            Grupo 43
%
%    72913 - Bruno Henriques
%    72960 - Tiago Santos
% -------------------------------

% ---------------------------------------------------------------
% -----------  Regras auxiliares de manipulação de listas --------
% ---------------------------------------------------------------

% sublista(P, L) - Devolve true se P for uma sublista da lista L,
% e false caso contrário.
%
% Caso terminal: A lista Sb é prefixo da lista L
% Caso recursivo: Verifica-se se a lista Sb é prefixo da lista L
% sem o seu primeiro elemento.

sublista(Sb,L)          :-   prefix(Sb,L).
sublista(Sb,[_|T]) :-   sublista(Sb,T).

% ---------------------------------------------------------------
% ----------- Regras necessárias para o projecto ----------------
% ---------------------------------------------------------------

% existe(E, L) - Devolve true se E for membro da lista L e false
% caso contrário.
%
% Chamada à função built-in do SWI-Prolog que verifica se
% o elemento E pertence à lista L.

existe(E, L)  :-   member(E,L).

% lado(S1, S2, Suspeitos - Devolve true se S1 e S2 estiverem um ao
% lado do outro na lista de Suspeitos independentemente da ordem
% com que estes se apresentam na mesma. Devolve false caso contrário.
%
% É verdade se e só se for encontrado a sequência [S1, S2] ou [S2,S3]
% na lista Suspeitos.

lado(S1, S2, Suspeitos):-   sublista([S1, S2], Suspeitos).
lado(S1, S2, Suspeitos):-   sublista([S2, S1], Suspeitos).

% entre(S1,S2,S3, Suspeitos) - Devolve true se S2 estiver entre S1
% e S3 (ou entre S3 e S1) e false caso contrario.
%
% É verdade se e só se for encontrado a sequência [S1, S2, S3] ou
% [S3, S2,S1] na lista Suspeitos.

entre(S1,S2,S3, Suspeitos)  :-   sublista([S1,S2,S3], Suspeitos).
entre(S1,S2,S3, Suspeitos)  :-   sublista([S3,S2,S1], Suspeitos).
```

```
% esquerda(S1,S2, Suspeitos) - Retorna true se S1 estiver à esquerda
% de S2 na lista Suspeitos e false caso contrário
%
% Caso de paragem: Se o suspeito S2 existe no resto da lista de
suspeitos
% que tem S1 na cabeça.
% Caso recursivo: Caso o suspeito S1 não corresponder ao primeiro
% elemento da lista de Suspeitos, continua-se a pelo mesmo no resto
% da lista.

esquerda(S1, S2, [S1|R])    :-  existe(S2, R).
esquerda(S1, S2, [_|R])     :-  esquerda(S1, S2, R).

% direita(S1,S2, Suspeitos) - Retorna true se S1 estiver à direita
% de S2 na lista Suspeitos e false caso contrário
%
% S1 está à direita de S2 se e só se S2 estiver à esquerda do
primeiro
% na lista de suspeitos.

direita(E1, E2, Suspeitos)  :-  esquerda(E2, E1, Suspeitos).

% naoEntre(S1,S2,S3, Suspeitos) - Devolve true se S2 não estiver
% entre S1 e S3 (ou entre S3 e S1) e false caso contrario.
%
% O suspeito S1 e S3 têm que ter 1 elemento entre eles e a posição do
suspeito
% S2 na lista de Suspeitos não pode ter a posição do elemento que se
encontra
% entre S1 e S3.

naoEntre(S1, S2, S3, Suspeitos) :-
     nth1(M, Suspeitos, S1),
     nth1(K, Suspeitos, S2),
     nth1(N, Suspeitos, S3),
     ((K =\= N+1, M is N+2) ; ((K =\= M+1), N is M+2)).

% ou(S1,S2,S3, Suspeitos) - Devolve true se o Suspeito S1 for o
Suspeito S2 ou S3
% se e só se os Suspeitos S2 e S3 forem diferentes. Devolve false
caso contrário.
%
% As variáveis S1, S2 e S3 são inicializadas usando a regra existe(S,
% Suspeitos) e de seguida verifica-se se S1 unifica com o suspeito S2
% ou com S3 onde S2 e S3 não são unificáveis.

ou(S1, S2, S3, Suspeitos) :-
     existe(S1, Suspeitos),
     existe(S2, Suspeitos),
     existe(S3, Suspeitos),
     ((S1 = S2, S2 \= S3) ; (S1 = S3, S2 \= S3)).

% -----------              END OF FILE                    --------
```