FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

EXERCÍCIOS DE PROGRAMAÇÃO EM LÓGICA

LUÍS PAULO REIS
DANIEL CASTRO SILVA

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO

PROGRAMAÇÃO EM LÓGICA - 3º ANO

SETEMBRO DE 2007

Exercício JP1. Cavalo num Tabuleiro de Xadrez

Considere um tabuleiro de xadrez com as coordenadas representadas por pares X/Y com X e Y entre 1 e 8.

a) Defina salto_cavalo (Quad1, Quad2) que relaciona duas posições consecutivas dum cavalo de acordo com os movimentos possíveis deste. Assuma que Quad1 é sempre instanciado com as coordenadas de um quadrado.

Exemplo:

```
?-salto_cavalo(1/1, Q).
Q=3/2;
Q=2/3;
no
```

b) Defina o predicado trajecto_cavalo (Traj) em que Traj é uma lista de pares de coordenadas. Esta lista representa um trajecto possível do cavalo num tabuleiro vazio.

c) Recorrendo ao predicado trajecto_cavalo, coloque como objectivo ao interpretador descobrir para um cavalo um caminho composto por 4 passos com partida do quadrado 2/1 e chegada ao extremo oposto do tabuleiro. O cavalo deve ainda passar pelo quadrado 5/4 a seguir ao seu primeiro movimento. Considere que o tabuleiro não contém mais nenhuma pedra.

Solução JP1:

a)

```
salto_cavalo(X/Y,X1/Y1):-
      (movimento(Distx,Disty); movimento(Disty,Distx)),
      X1 is X+Distx,
      Y1 is Y+Disty,
      dentro_do_tabuleiro(X1),
      dentro_do_tabuleiro(Y1).
movimento(2,1).
movimento(-2,1).
movimento(2,-1).
movimento(-2,-1).
dentro_do_tabuleiro(X):- 1 =< X, X =< 8.
```

b)

```
trajecto_cavalo([Traj]).
```

```
trajecto_cavalo([P1,P2|Traj]):-
      salto_cavalo(P1,P2),
      trajecto_cavalo([P2|Traj]).
```

c)

```
?-trajecto_cavalo([2/1,J1,5/4,J3,J4x/8]).
```

JP 2. Representação e Avaliação do Estado no Xadrez

Considere o algoritmo MINIMAX, aplicado ao jogo do xadrez.

a. Especifique a estrutura de dados que utilizaria em Prolog para a representação de um estado de um tabulerio de Xadrez.

b. Implemente em Prolog o predicado que realiza o cálculo do valor utilidade de um estado. Inclua nessa função o número de peças existentes no tabuleiro, e respectivas posições (centrais e margens do tabuleiro, penalizando estas).

JP 3. Jogo do Nim

Consideremos o jogo Nim que é jogado por duas pessoas, da seguinte forma:

No início existe um número arbitrário de pilhas de fósforos. Cada pilha contém um número arbitrário de fósforos. Em cada jogada um jogador pode apanhar um ou mais fósforos duma pilha. O vencedor é aquele que apanhe os últimos fósforos.

Escreva um programa que indique se existe uma jogada para a qual um dos jogadores pode sempre vencer o jogo (quaisquer que sejam a partir dessa jogada os lances do opositor).

Exemplo:

```
?-vence ([2, 2, 1], J, NovaConfig).
J=1
NovaConfig=[2, 2]
?-vence([2, 2], J, NovaConfig).
no
```

Solução JP3:

```
vence([X],X,[]). % para ganhar apanha todos os fósforos
vence(L,X,L1):- % vence se faz uma jogada a partir da qual o adversario
      joga(L,X,L1), % não vence
      not vence(L1,_,_).
joga(S,X,S1):- % apanha todos os fósforos duma pilha
      delete(X,S,S1).
joga(S,X,S2):- % apanha alguns fósforos
      delete(Y,S,S1),
      entre(X,0,Y), % X entre 0 e Y
      Y1 is Y-X,
      insert(Y1,S1,S2).
```

JP 4. Jogo do Solitário

Escreva um programa que jogue o jogo do SOLITÁRIO (jogo individual). O SOLITÁRIO é jogado com peças que se comportam como as pedras do jogo de damas (uma pedra pode "comer" sse existe uma casa livre a seguir à pedra que vai "comer"). O jogo começa com todas as casas menos uma ocupadas por pedras. O objectivo é terminar só com uma pedra no tabuleiro. As posições iniciais do tabuleiro e o número de peças contidas neste podem ser diversas.

Solução JP4:

```
:- op(300, xfy, :).
solitario(CasaLivre) :-
      casas_do_jogo(Casas),
      remove(CasaLivre, Casas, CasasOcupadas),
      descobre_movimentos(CasasOcupadas, [CasaLivre], Movs),
      imprime(Movs), nl.
solitario(_) :-
      nl, write(´ Não me é possível resolver essa questão´).

descobre_movimentos([X],_,[]). % Só uma casa ocupada.
descobre_movimentos(CasasOcup, CasasLivres, [M | Movs]) :-
      seleciona_mov(M, CasasOcup, NCasasOcup,
      CasasLivres, NCasasLivres),
      descobre_movimentos(NCasasOcup, NCasasLivres, Movs).

seleciona_mov( A:B:C , CO, CL, NCO, NCL) :-
      A:B:C,
      remove(A, CO, NCO1),
      remove(B, NCO1, NCO2),
      remove(C, CL, NCL1),
      adiciona(C, NCO2, NCO),
      adiciona(A, NCL1, NCL2),
      adiciona(B, NCL2, NCL).
      adiciona(X, [], [X]).
adiciona(X, [L|R], [L|V]) :- adiciona(X, R, V).
remove(X, [X|V], V) :- !.
remove(X, [Y|R], [Y|V]) :- remove(X, R, V).
imprime([A:B:C] :-
      write(´De ´), write(A), write(´ para ´), write(C),
      write(´ comendo ´), write(B), nl.
imprime([Mov| Movs]) :-
      imprime([Mov]), imprime(Movs).
a:c:f. a:b:d. b:d:g. b:e:i. c:e:h. c:f:j.
d:g:k. ... ... f:c:a. ... ... ...
casas_do_jogo([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o]).
```

JP 5. Problema dos 2 Baldes

Dois baldes, de capacidades 4 litros e 3 litros, respectivamente, estão inicialmente vazios. Quais as operações a efectuar de modo a que o primeiro balde contenha 2 litros ?

Os baldes não possuem qualquer marcação intermédia. As únicas operações que pode realizar são:

- esvaziar um balde
- encher (completamente) um balde
- despejar um balde para o outro até que o segundo fique cheio
- despejar um balde para o outro até que o primeiro fique vazio

JP 6. Problema dos Missionários e dos Canibais

Três missionários e três canibais, que se encontram na margem esquerda de um rio, querem atravessar para a margem direita. O barco existente transporta, no máximo, duas pessoas. Determine as operações a realizar, sabendo que o número de canibais não pode ser superior ao número de missionários em qualquer margem do rio.

Solução JP6:

```
inicial(estado(3,3,e)).
final(estado(0,0,d)).

seguro(estado(M,C,_)):-
        sobrevive(M,C),
        M1 is 3-M, C1 is 3-C,
        sobrevive(M1,C1).

sobrevive(0,_).
sobrevive(M,C):-M>=0,M>=C.

seguinte(estado(M,C,e),estado(M1,C,d)):-  ( M>=1, M1 is M-1 ) ; ( M>=2, M1 is M-2 ).
seguinte(estado(M,C,e),estado(M,C1,d)):-  ( C>=1, C1 is C-1 ) ; ( C>=2, C1 is C-2 ).
seguinte(estado(M,C,e),estado(M1,C1,d)):-  M>=1, C>=1, M1 is M-1, C1 is C-1.
seguinte(estado(M,C,d),estado(M1,C,e)):-
        Md is 3-M,
        ( ( Md>=1, M1 is M+1 ) ; ( Md>=2, M1 is M+2 ) ).
seguinte(estado(M,C,d),estado(M,C1,e)):-
        Cd is 3-C,
        ( ( Cd>=1, C1 is C+1 ) ; ( Cd>=2, C1 is C+2 ) ).
seguinte(estado(M,C,d),estado(M1,C1,e)):-
        Md is 3-M, Cd is 3-C,
        Md>=1, Cd>=1, M1 is M+1, C1 is C+1.

atravessa(Ef,Ef,[Ef],_).
```

```
atravessa(Ea,Ef,[Ea|R],Eants):-
	seguinte(Ea,Eseg),
	seguro(Eseg),
	not member(Eseg,Eants),
	atravessa(Eseg,Ef,R,[Eseg|Eants]).

miss_can:-
	inicial(Ei), final(Ef),
	atravessa(Ei,Ef,L,[Ei]),
	nl, escrever(L).

escrever([X]).
escrever([E1,E2|T]):- explicacao(E1,E2),  nl,  escrever([E2|T]).

explicacao(estado(M,C,Marg),estado(M1,C,_)):-
	( (Marg=e,Mm is M-M1,Marg1=esquerda,Marg2=direita)
	; (Mm is M1-M, Marg1=direita,Marg2=esquerda) ),
	write('passaram '),write(Mm),
	write(' missionarios da margem '),write(Marg1),
	write(' para a margem '),write(Marg2),nl.
explicacao(estado(M,C,Marg),estado(M,C1,_)):-
	( (Marg=e,Cc is C-C1,Marg1=esquerda,Marg2=direita)
	; (Cc is C1-C, Marg1=direita,Marg2=esquerda) ),
	write('passaram '),write(Cc),
	write(' canibais da margem '),write(Marg1),
	write(' para a margem '),write(Marg2),nl.
explicacao(estado(M,C,Marg),estado(M1,C1,_)):-
	( (Marg=e,Mm is M-M1,Cc is C-C1,Marg1=esquerda,Marg2=direita)
	; (Mm is M1-M, Cc is C1-C, Marg1=direita,Marg2=esquerda) ),
	write('passaram '),write(Mm),write(' missionarios e '),
	write(Cc),write(' canibais da margem '),write(Marg1),
	write(' para a margem '),write(Marg2),nl.
```

JP 7 Problema da Torres de Hanoi

Implemente em Prolog o predicado *hanoi(Num, Pino1, Pino2, Pino3, Movimentos)* que dado um número *Num* e o nome de 3 pinos, devolve em *Movimentos* a lista com a sequência de movimentos para resolver o problema das torres de hanoi com *Num* discos, do pino *Pino1* para o pino *Pino2*. Cada movimento deverá ser da forma *m(de, Para)*, sendo *De* e *Para* o nome de dois pinos distintos.

Exemplos:

?- hanoi(2,a,b,c,L).

L = [m(a,c),m(a,b),m(c,b)]

JP 8 Problema das N-Rainhas

Construa um programa em Prolog que permita resolver o problema das N-Rainhas. Este problema consiste em colocar, num tabuleiro com NxN casa, N rainhas, sem que nenhuma rainha ataque uma outra rainha posicionada no tabuleiro. (Nota: Este exercício é mais adequado para a matéria de Programação em Lógica com Restrições – parte final da disciplina)

JP 9 Problema dos Criptogramas

O Problema dos *CRIPTOGRAMAS* consiste em atribuir dígitos decimais às letras, de modo a que a respectiva soma seja válida. (Nota: Este exercício é mais adequado para a matéria de Programação em Lógica com Restrições – parte final da disciplina)

```
puzzle(1,[D,O,N,A,L,D],[G,E,R,A,L,D],[R,O,B,E,R,T]).
puzzle(2,[0,C,R,O,S,S],[0,0,R,O,A,D],[D,A,N,G,E,R]).
puzzle(2,[0,S,E,N,D],[0,M,O,R,E],[M,O,N,E,Y]).
soma(P1, P2, R, Cant, Cap, Dgts, Ddisp)
      P1, P2, R - 1ª e 2ª parcelas, e resultado da soma
      Cant, Cap - "carry" antes e após efectuar a soma
      Dgts - lista dos digitos que podem ser usados
      Ddisp - digitos disponíveis (ainda não usados)
```

Solução JP8:

```
inicio(X):- puzzle(X, P1, P2, Result),
      soma(P1, P2, Result, 0, 0, [0,1,2,3,4,5,6,7,8,9], _),
      write(P1), write(' + '), write(P2), write(' = '),
      write(Result).
soma([], [], 0, 0, Digts, Digts).
soma([D1|R1], [D2|R2], Cant, Cap, Digts, Ddisp):-
      soma(R1, R2, R, Cant, Cap1, Digts, Ddisp1),
      soma_dig(D1, D2, D, Cap1, Cap, Ddisp1, Ddisp).
soma_dig(D1, D2, D, Cant, Cap, Ddant, Ddap):-
      del(D1, Ddant, Ddaux1),
      del(D2, Ddaux1, Ddaux2), del(D, Ddaux2, Ddap),
      S is D1+D2+Cant, D is S mod 10, Cap is S//10.
del(A,L,L):- nonvar(A), !.
del(A, [A|L], L).
del (A, [H|T], [H|T1]):- del (A, T, T1).
```

JP 10. Os Degraus da Casa do Paulo

À entrada da casa do Paulo há uma escada com 10 degraus. Cada vez que entra em casa, o Paulo avança pelas escadas subindo um ou dois degraus em cada passada. De quantas maneiras diferentes pode o Paulo subir as escadas?"

Faça em Prolog um predicado *casa_degraus(Degraus, N, L)* que dado o número de degraus *Degraus* da escada (que pode ser diferente de 10), devolve em *N* o número de maneiras diferentes de subir a escada, e em *L* a lista das possibilidades (cada possibilidade será também uma lista com uma sequência de 1s e 2s).

Exemplo:

```
?- jogo_escadas(3,N,L).
N = 3
L = [[1,1,1],[1,2],[2,1]]
?- jogo_escadas(10,N,_).
N=89
```

JP 11. Aposta de Dinheiro

Um amigo apostou consigo o dinheiro (em escudos) que estava em nove caixas se adivinhasse o conteúdo dessas caixas. As caixas estão dispostas num quadrado de 3 linhas e 3 colunas, como indicado abaixo.

C1 C2 C3

C4 C5 C6

C7 C8 C9

Para o ajudar a adivinhar, o seu amigo deu-lhe as seguintes pistas:

1) Todas as caixas têm uma e uma só nota;
2) Há uma nota de 1000 em cada linha
3) Nas quatro caixas dos cantos estão três notas de 500.
4) Há duas notas de 2000 na segunda linha;
5) Nas caixas da terceira coluna existem duas notas de 1000;
6) A ùnica nota de 5000 não está na terceira linha.

Atendendo a que, para ganhar a aposta, tem de descobrir o conteúdo de cada caixa;

Escreva um programa Prolog que lhe permita vencer a aposta e , já agora, calcular quanto vai ganhar. O programa deve ser invocado através do predicado aposta/0 que mostrará no ecrã uma lista com os valores das notas existentes nas caixas C1 a C9 (por esta ordem).

JP 12. Agricultores em Competição

No concurso anual de produtores do campo, havia 4 tipos de produtos: cebolas, abóboras, ovos e melancias. 4 competidores: Ana, Bruno, Carolina e Diana, cujos sobrenomes, não necessariamente nesta ordem, são: Almeida, Bernardes, Castro e Damásio, entraram na competição por um dos tipos de produto e ganharam, cada um, um prémio. Um competidor ganhou o primeiro prémio, outro ganhou o segundo, outro o terceiro, e outro ganhou o quarto prémio.

Dados os seguintes dados, escreva um programa Prolog que encontre os nomes completos dos quatro competidores, os seus produtos e a colocação de cada um na competição: "*Bruno Almeida não ganhou competindo com o produto abóbora, que ficou em quarto lugar. Castro ganhou o primeiro prémio, e não se chamava Carolina. Ana ficou em terceiro lugar. Damásio ofereceu a todos da competição o seu produto: ovos. Diana não era a competidora que concorria com cebolas.*"

JP 13. O “Zebra Puzzle”

Este é um puzzle tradicional da programação em lógica. Há cinco casas com cinco cores diferentes. Em cada casa, vive uma pessoa de nacionalidade diferente, tendo uma bebida, uma marca de cigarros e um animal favoritos. A configuração é:

O Inglês vive na casa vermelha
O Espanhol tem um cão
O Norueguês vive na primeira casa a contar da esquerda
Na casa amarela, o dono gosta de Marlboro
O homem que fuma Chesterfields vive na casa ao lado do homem que tem uma raposa
O Norueguês vive ao lado da casa Azul
O homem que fuma Winston tem uma iguana
O fumador de Luky Strike bebe sumo de laranja
O Ucraniano bebe chá
O Português fuma SG Lights
Fuma-se Marlboro na casa ao lado da casa onde há um cavalo
Na casa verde, a bebida preferida é o café
A casa verde é imediatamente à direita (à sua direita) da casa branca
Bebe-se leite na casa do meio

A pergunta é: Onde vive a Zebra, e em que casa se bebe água?

Construir um programa em Prolog que permita resolver o “Zebra Puzzle”. (Nota: Este exercício pode ser resolvido de forma mais adequada com a PLR).

JP 14. A Cronometragem do Ovo

Utilizando apenas duas ampulhetas, uma de 7 minutos e outra de 11, qual o processo mais expedito de cronometrar a cozedura de um ovo que demora 15 minutos? Construa um programa Prolog para resolver este problema.