

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Universidade do Porto
Faculdade de Engenharia

FEUP

EXERCÍCIOS DE PROGRAMAÇÃO EM LÓGICA

LUÍS PAULO REIS

LICENCIATURA EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO

PROGRAMAÇÃO EM LÓGICA - 3º ANO

SETEMBRO DE 2005



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2004/2005
LEIC
(3º Ano)
1º Sem

Exercícios RC – Representação de Conhecimento em Prolog

Exercício RC 1. Representação de Conhecimento – Autores de Livros

Escreva frases em Prolog que representem o seguinte conhecimento:

Os Maias, livro, Eça de Queiroz, português, inglês, romance, escreveu, autor, nacionalidade, tipo, ficção

Escreva as seguintes questões em Prolog:

- a) Quem escreveu “Os Maias”?
- b) Que autores portugueses escrevem romances?
- c) Quais os autores de livros de ficção que escreveram livros de outro tipo também?

Exercício RC 2. Representação de Conhecimento – Comidas e Bebidas

Escreva frases em Prolog que representem o seguinte conhecimento:

peru, frango, salmão, solha, cerveja, vinho verde, vinho maduro, Ana, António, Barbara, Bruno, gosta, casado, combina

Escreva as seguintes questões em Prolog:

- a) Ana e Bruno são casados e gostam de vinho verde?
- b) Que bebida combina com salmão?
- c) Que comidas combinam com vinho verde?

Exercício RC 3. Representação de Conhecimento – Desportos e Jogos

Escreva frases em Prolog que representem o seguinte conhecimento:

João, Maria, Ana, casa, cão, xadrez, damas, ténis, natação, apartamento, gato, tigre, homem, mulher, animal, mora_em, gosta_de, jogo, desporto

Escreva as seguintes questões em Prolog:

- a) Quem mora num apartamento e gosta de animais?
- b) Será que o João e a Maria moram numa casa e gostam de desportos?
- c) Quem gosta de jogos e de desportos?
- d) Existe alguma mulher que gosta de ténis e gosta de tigres?

Exercício RC 4. Representação de Conhecimento – Tweety e Silverster

Traduza as seguintes frases para Prolog:

“Tweety é um pássaro. Goldie é um peixe. Molie é uma minhoca. Pássaros gostam de minhocas. Gatos gostam de peixes. Gatos gostam de pássaros. Amigos gostam uns dos outros. O meu gato é meu amigo. O meu gato come tudo o que gosta. O meu gato chama-se Silverster.”

- a) Use Prolog para determinar tudo o que come o Silverster?
- b) A resposta é razoável? Se não for, verifique se o problema está na especificação original ou na sua tradução para Prolog, corrija e execute novamente.

Exercício RC 5. Representação de Conhecimento – Programação e Erros

Um estudante acostumado a usar linguagens procedimentais está a desenvolver um compilador em Prolog. Uma das tarefas consiste em traduzir um código de erro para uma pseudo-descrição em português. O código por ele usado é:

```

traduza(Codigo, Significado) :-
    Codigo = 1,
    Significado = integer_overflow.
traduza(Codigo, Significado) :-
    Codigo = 2,
    Significado = divisao_por_zero.
traduza(Codigo, Significado) :-
    Codigo = 3,
    Significado = id_desconhecido.

```

Com sabe, esta não é uma forma apropriada de programar em Prolog. Melhore este código.

Exercício RC 6. Representação de Conhecimento – Cargos e Chefes

Suponha a seguinte Base de Factos Prolog:

```

cargo(tecnico, rogerio).
cargo(tecnico, ivone).
cargo(engenheiro, daniel).
cargo(engenheiro, isabel).
cargo(engenheiro, oscar).
cargo(engenheiro, tomas).
cargo(engenheiro, ana).
cargo(supervisor, luis).
cargo(supervisor_chefe, sonia).
cargo(secretaria_exec, laura).
cargo(diretor, santiago).
chefiado_por(tecnico, engenheiro).
chefiado_por(engenheiro, supervisor).
chefiado_por(analista, supervisor).
chefiado_por(supervisor, supervisor_chefe).
chefiado_por(supervisor_chefe, director).
chefiado_por(secretaria_exec, director).

```

Onde os predicados cargo/2 e chefiado_por/2 são autoexplicativos. Escreva em linguagem natural as seguintes interrogações Prolog:

- ?- chefiado_por(tecnico, X), chefiado_por(X,Y).
- ?- chefiado_por(tecnico, X), cargo(X,ivone), cargo(Y,Z).
- ?- cargo(supervisor, X); cargo(supervisor, X).
- ?- cargo(J,P), (chefiado_por(J, supervisor_chefe); chefiado_por(J, supervisor)).
- ?- chefiado_por(P, director), not(cargo(P, carolina)).

Sem utilizar o computador responda qual seria a primeira resposta encontrada pelo Prolog para cada uma destas interrogações.

Exercício RC 7. Representação de Conhecimento – Alunos e Professores

Considere a seguinte base de factos exemplo:

```

aluno(joao, paradigmas).
aluno(maria, paradigmas).
aluno(joel, lab2).
aluno(joel, estruturas).
frequenta(joao, feup).

```

```
frequenta(maria, feup).  
frequenta(joel, ist).  
professor(carlos, paradigmas).  
professor(ana_paula, estruturas).  
professor(pedro, lab2).  
funcionario(pedro, ist).  
funcionario(ana_paula, feup).  
funcionario(carlos, feup).
```

Escreva as seguintes regras em prolog:

- a) Quem são os alunos do professor X?
- b) Quem são as pessoas da universidade X? (alunos ou docentes)
- c) Quem é colega de quem? Se aluno: é colega se for colega de disciplina ou colega de curso ou colega de universidade. Se professor: se for professor da mesma universidade.

Exercício RC 8. Representação de Conhecimento – Carros e Valores

Considere a seguinte base de factos exemplo:

```
comprou(joao, honda).  
ano(honda, 1997).  
comprou(joao, uno).  
ano(uno, 1998).  
valor(honda, 20000).  
valor(uno, 7000).
```

- a) Crie uma regra `pode_vender` onde o primeiro argumento é a pessoa, o segundo o carro e o terceiro é o ano actual (não especificar “homem” ou “carro” nas regras), onde a pessoa só pode vender o carro se o carro for comprado por ela nos últimos 10 anos e se seu valor for menor do que 10000 Euros.



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação

Programação em Lógica

2004/2005
LEIC
(3º Ano)
1º Sem

Exercícios IP– Introdução ao Prolog

Exercício IP 1. Objectos e Atributos

Suponha a seguinte representação do conhecimento relativo a objectos e relações de posição:

objecto(obj1).	tipo(obj1, mesa).	fragil(vidro).
objecto(obj2).	tipo(obj2, cadeira).	forte(ferro).
objecto(obj3).	tipo(obj3, mesa).	sobre(obj4, obj1).
objecto(obj4).	tipo(obj4, jarra).	sobre(obj3, obj1).
objecto(obj5).	tipo(obj5, matraca).	sobre(obj5, obj3).
material(obj1, madeira).	peso(obj1, 10.5).	
material(obj2, madeira).	peso(obj2, 1.5).	
material(obj3, vidro).	peso(obj3, 1.6).	
material(obj4, vidro).	peso(obj4, 0.5).	
material(obj5, ferro).	peso(obj5, 1.8).	

- Escreva o predicado `debaixo(O1,O2)` que sucede se o objecto O1 está debaixo (directamente) do objecto O2.
- Escreva o predicado `objectos(Lista)` que calcula a Lista de objectos existentes no mundo ($List = [obj1, obj2, obj3, obj4, obj5]$).
- Escreva o predicado `objectos_descricao(Lista)`, que retorne uma lista com a descrição de todos os objectos, incluindo o seu nome, material, tipo e peso ($[obj1-madeira-mesa-10.5, obj2-madeira-cadeira-1.5, \dots, obj5-ferro-matraca-1.8]$).
- Escreva o predicado `tipos_objectos(Lista)`, que retorne uma lista com todos os tipos de objecto ($[mesa, cadeira, jarra, matraca]$), sem elementos repetidos.
- Escreva o predicado `media_resistencia(Obj)` que sucede se o objecto não é frágil nem é forte.
- Escreva o predicado `mais_leve(O1,O2,Dif)` que recebe como argumentos dois objectos e sucede se o primeiro for mais leve do que o segundo, retornando a diferença de peso entre eles.
- Escreva o predicado `um_deles_fragil(O1,O2,Mat)` que recebe como argumentos dois objectos e sucede unicamente se um dos objectos for composto por um material frágil e o outro não, retornando qual é esse material frágil.
- Escreva o predicado `por_cima_indirecto(O1,O2)` que sucede se o objecto O1 está sobre um objecto que por sua vez está sobre O2.
- Escreva o predicado `sobre_mesmo(O1,O2,Obase)` que sucede se dois objectos se encontram directamente sobre o mesmo objecto e retorne o objecto em causa.
- Escreva o predicado `objectos_tipo(Tipo,Lista)` que determine a lista de todos os objectos de um dado tipo.
- Escreva o predicado `nao_toca_material(Obj)` que sucede se um dado objecto não toca em nenhum objecto composto do mesmo material.
- Escreva o predicado `eu_sou_original(Obj)` que sucede se um dado objecto é o único de um dado tipo ou é o único de um dado material.



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2004/2005
LEIC
(3º Ano)
1º Sem

Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira

Exercícios IP – Introdução ao Prolog

Exercício IP 2. Países e Fronteira

Suponha a seguinte representação do conhecimento relativo a diversos países.

continente(asia).	pais(alemanha, europa, 82).	fronteira(franca, espanha).
continente(america).	pais(holanda, europa, 15).	fronteira(franca, belgica).
continente(europa).	pais(eua, america, 235).	fronteira(belgica, alemanha).
pais(portugal, europa, 10).	pais(brasil, america, 155).	fronteira(belgica, holanda).
pais(espanha, europa, 48).	pais(china, asia, 1100).	fronteira(alemanha, holanda).
pais(franca, europa, 52).	pais(mongolia, asia, 3).	fronteira(alemanha, franca).
pais(belgica, europa, 9).	fronteira(portugal, espanha).	fronteira(china, mongolia).

- Escreva o predicado junto(P1,P2) que sucede se o país P1 faz fronteira com P2.
- Escreva o predicado paises_continente(Lista, Cont) que calcula a Lista de países existentes num dado continente.
- Escreva o predicado paises_grandes(Lista, Cont) que calcula a Lista de países com mais de 100 milhões de habitantes de um dado continente.
- Escreva o predicado dois_mais_pop(P1, P2), que calcula quais os dois países com mais habitantes.
- Escreva o predicado descricao(Lista), que retorne uma lista com a descrição de todos os países, incluindo o seu nome, continente e população ([portugal*europa*10, ..., china*asia*1000]).
- Escreva o predicado nao_sou_o_maior_do_continente(P1) que sucede se P1 não é o maior país do seu continente.
- Escreva o predicado chego_la_facil(P1, P2) que sucede se é possível chegar de P1 a P2, directamente ou atravessando unicamente um outro país.
- Escreva o predicado num_paises_atravesados(P1, P2, Num) que calcula o número de países que é necessário atravessar para chegar de P1 a P2.
- Escreva o predicado conta_paises(Num) que calcula quantos países diferentes existem na base de dados.
- Escreva o predicado introduz_novo_pais que pergunta ao utilizador qual o nome do país, continente e população e introduz estes dados na base de dados.
- Escreva o predicado elimina_pais(P1) que dado um país P1, se ele existir, elimina-o da base de dados. Deve ainda eliminar a especificação do continente e da população desse país.
- Escreva o predicado corta_fronteiras(P1) que elimina todas as fronteiras do país P1 da base de dados.
- Escreva o predicado soma_populacao_vizinhos(P1, Soma) que calcula a soma da população dos vizinhos de P1.
- Escreva o predicado o_meu_vizinho_e_maior_que_o_teu(P1,P2) que sucede se o maior vizinho de P1 é maior (em população) que o maior vizinho de P2.



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2003/2004
LEIC
(3º Ano)
1º Sem

Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira

Exercícios IP– Introdução ao Prolog

Exercício IP 3. Relações de Parentesco

Suponha a seguinte representação do conhecimento relativo a diversas pessoas.

homem(americo).	mulher(carla).	idade(joana, 17).	pai(daniel, americo).
homem(daniel).	mulher(barbara).	idade(carla, 26).	pai(daniel, paulo).
homem(paulo).	mulher(maria).	idade(barbara, 51).	pai(joaquim, daniel).
homem(carlos).	idade(americo, 18).	idade(daniel, 60).	mae(maria, daniel).
homem(joaquim).	idade(paulo, 25).	idade(joaquim, 80).	mae(barbara, joana).
homem(filipe).	idade(carlos, 37).	idade(maria, 79).	casados(filipe, carla).
mulher(teresa).	idade(filipe, 32).	irmaos(americo, paulo).	casados(americo,teresa).
mulher(sonia).	idade(teresa, 18).	irmaos(carlos, sonia).	casados(joaquim, maria).
mulher(ana).	idade(sonia, 28).	pai(carlos, teresa).	

- a) Escreva o predicado pessoas(Lista) que calcula a Lista de todas as pessoas existentes na base de factos.
- b) Escreva o predicado avof(Mul, Pess) em que Mul seja avó de Pess.
- c) Escreva o predicado avom(Hom, Pess) em que Hom seja avô de Pess.
- d) Escreva o predicado bisavom(Hom, Pess) que suceda se Hom for bisavô de Pess.
- e) Escreva os predicados tio(Hom, Pess) e tia(Hom, Pess) que sucedem se Hom (Mul) for tio/tia de Pess.
- f) Escreva os predicados neto(Hom, Pess) e neta(Hom, Pess) que sucedem se Hom (Mul) for neto/neta de Pess.
- g) Escreva o predicado primo_1(P1, P2) que sucede se P1 e P2 forem primos no primeiro grau.
- h) Escreva o predicado primo(P1, P2) que sucede se P1 e P2 forem primos no primeiro em qualquer grau.
- i) Escreva o predicado maior_de_idade(Pess) que sucede se Pess for maior de idade.
- j) Escreva o predicado mais_velho(Pess) que retorna a pessoa mais velha que consta na base de factos.
- k) Escreva o predicado adequados(Hom, Mul) que sucede se Hom for um homem, Mul for uma mulher e o homem for (no máximo) mais novo 2 anos do que a mulher ou mais velho 10 anos do que ela e se ambos não tiverem nenhuma relação de parentesco nem nenhum deles for casado!
- l) Escreva o predicado lista_pessoas(Lista,Sexo) que retorne uma lista de todas as pessoas do Sexo indicado (m/f), incluindo as suas respectivas idades. Por exemplo lista_pessoas(Lista, m) deveria retornar Lista=[americo-18, paulo-25, carlos-37, filipe-32].
- m) Escreva o predicado numero_irmaos(Pess, N) que sucede se N for o número de irmãos de Pess.
- n) Escreva o predicado lista_dos_mais_velhos_que_eu(Pess, Lista) que retorne uma lista com todas as pessoas mais velhas que Pess.
- o) Escreva o predicado avos_e_netos(Lista) que retorne uma lista com todos os avos (homens ou mulheres e respectivos netos (Lista = [joaquim-daniel, maria-daniel])).

Exercício IP 4. Ordens de Serviço

Suponha que pretende guardar informação sobre quando e para onde foram enviadas ordens de serviço dentro de uma empresa. Qual é o melhor formato, em PROLOG, para os factos que representam essa informação, e porquê?

- (i) <data> (<nome>, <data>, <autor>, <distribuição>).
- (ii) ordem (<nome>, <data>, <autor>, <distribuição>).
- (iii) facto (ordem, <nome>, <data>, <autor>, <distribuição>).

Exercício IP 5. Electrodomésticos e Componentes - Fogão

a) Escreva os seguintes factos em PROLOG:

Um fogão é composto por uma estrutura e um cordão eléctrico.
Uma das componentes da estrutura é uma resistência de aquecimento.
A resistência de aquecimento é em metal.
Outra parte da estrutura é o painel do fogão.
O painel tem um botão.
Os botões são sempre feitos em plástico.
O cordão eléctrico é composto de fio metálico.
Parte do cordão é um isolador.
O isolador é feito de fibra.

b) Coloque as seguintes questões ao interpretador de PROLOG:

- Que objectos têm metal ?
- Que objectos fazem parte da estrutura do fogão ?

c) Defina regras de transitividade para parte_de e herança ascendente de material com respeito a parte_de.

d) Agora questione o interpretador sobre:

- Que objectos contém plástico ?
- Que objectos não contém plástico ?
- Que objectos contém metal e fibra ?

Solução:

a)
`parte_de(estrutura,fogao).
parte_de(cordao_electrico,fogao).
parte_de(resistencia,estrutura).
contem(resistencia,metal).
parte_de(painel,estrutura).
parte_de(botao,painel).
contem(botao,plastico).
parte_de(fio,cordao_electrico).
contem(fio,metal).
parte_de(isolador,cordao_electrico).
contem(isolador,fibra).`

b)
`?- contem(Obj, metal).
?- parte_de(Obj, estrutura).`

Exercício IP 6. Mundo dos Blocos

Considere três pilhas de blocos sobre uma mesa:

```

      Bloco D
      Bloco C      Bloco G
      Bloco B      Bloco F      Bloco I
      Bloco A      Bloco E      Bloco H
--Mesa-----Mesa--
```

a) Represente em PROLOG os factos que descrevem as relações entre blocos. Use dois predicados, que representem um bloco apoiado noutro e um bloco imediatamente à esquerda de outro. Só escreva factos do 2º tipo para blocos que estejam no topo da pilha.

b) Insira os factos num ficheiro.

c) Agora questione o interpretador sobre:

- Que bloco está apoiado no bloco B?
- Sobre que bloco está o bloco A?
- Que blocos estão apoiados noutros blocos?
- Que blocos apoiam blocos que estão imediatamente à esquerda de outros blocos?

d) Defina um novo predicado *acima_de* que é verdadeiro quando um bloco está algures na pilha sobre outro bloco. Defina um novo predicado *empilhado_a_esq* que é verdadeiro quando um bloco está na pilha imediatamente à esquerda da pilha de outro bloco. Insira estes novos predicados no ficheiro e carregue toda a base de dados.

e) Agora questione o interpretador sobre:

- Que blocos estão acima de outros blocos?
- Que blocos estão sobre o bloco F ou na pilha imediatamente à esquerda da pilha F?
- Que blocos estão sobre outros blocos, mas não estão à esquerda de nenhum bloco?

Exercício IP 7. Países e Fronteiras (Exame Normal 2003/2004)

Suponha que possui um predicado Prolog descrevendo informação sobre Países. Os factos têm o formato *país(Nome, Continente, População, Fronteiras)*, onde *Nome* é o nome do País, *Continente* é o continente a que o país pertence (*africa, america, asia, europa* ou *oceania*), *População* é um inteiro que representa o número de habitantes (em milhões) do país e *Fronteiras* é uma lista contendo os nomes dos países com que o país faz fronteira.

Exemplo:

```
país(alemanha, europa, 82, [frança, belgica, holanda, suica]).    país(holanda, europa, 15, [belgica, alemanha]).
país(australia, oceania, 19, []).                                país(indonesia, oceania, 210, []).
país(belgica, europa, 10, [frança, holanda, alemanha]).          país(italia, europa, 57, [frança, suica]).
país(espanha, europa, 40, [portugal, frança]).                  país(madagascar, africa, 17, []).
país(frança, europa, 59, [espanha, suica, belgica, alemanha,    país(portugal, europa, 10, [espanha]).
italia]).                                                         país(suica, europa, 7, [frança, alemanha, italia]).
```

a) Construa um predicado *pop_elevada(+Continente, -Lista)* que calcule a lista de todos os países com mais de 15 milhões de habitantes de um dado continente, ordenada por ordem crescente de população, no formato indicado.

Exemplos:

```
?- pop_elevada(europa, Lista).
Lista = [40-espanha, 57-italia, 59-frança, 82-alemanha]
?- pop_elevada(africa, Lista).
Lista = [madagascar-17]
```

b) Construa um predicado ***isolados_grandes(-Lista)*** que calcule a lista, ordenada por ordem alfabética, de todos os continentes que possuem pelo menos dois países que tenham simultaneamente uma população superior a 15 milhões e duas ou menos fronteiras terrestres (com países conhecidos). Sugestão: Utilize o predicado *sort* para ordenar a lista final.

Exemplo:

```
?- isolados_grandes(Lista).
```

```
Lista = [europa, oceania] % europa pois possui a espanha e a italia e a oceania devido à austrália e indonésia
```

Solução:

```
%1.1 a) pop_elevada(+Continente, -Lista).
pop_elevada(Continente, Lista):-
    findall(Pop-Pais, (país(Pais, Continente, Pop, _), Pop>15), ListaIni),
    keysort(ListaIni, Lista).

% 1.1 b) isolados_grandes(Lista).
isolados_grandes(Lista):-
    findall(Cont,
        (isolado_grande(Pais1, Cont), isolado_grande(Pais2, Cont) , Pais1\=Pais2),
        Listal),
    sort(Listal, Lista).
isolado_grande(Pais, Cont):- país(Pais, Cont, Pop, Front), Pop>15, length(Front, L), L<3.

% Outra versão
isolados_grandes2(Lista):-
    findall(Cont,
        (país(_,Cont,_,_),findall(Pais1, isolado_grande2(Pais1, Cont), L),
        length(L,N), N>=2),
        Listal),
    sort(Listal, Lista).
isolado_grande2(Pais, Cont):- país(Pais, Cont, Pop, Front), Pop>15, length(Front, L), L<3.
```

Exercício IP 8. Objectos e Pesos (Exame Recurso 2003/2004)

Suponha a seguinte representação do conhecimento relativo a objectos

<i>objecto(mesa1, vidro).</i>	<i>peso(mesa1, 10).</i>	<i>material(vidro, fragil).</i>	<i>sobre(cadeira, mesa1).</i>
<i>objecto(cadeira, madeira).</i>	<i>peso(cadeira, 2).</i>	<i>material(madeira, forte).</i>	<i>sobre(mesa2, mesa1). %mesa2 em cima</i>
<i>objecto(mesa2, madeira).</i>	<i>peso(mesa2, 3).</i>		<i>sobre(bigorna, mesa2).</i>
<i>objecto(bigorna, ferro).</i>	<i>peso(bigorna, 20).</i>	<i>material(ferro, forte).</i>	<i>sobre(jarra, bigorna).</i>
<i>objecto(jarra, vidro).</i>	<i>peso(jarra, 0.5).</i>		<i>sobre(caixa, jarra). % caixa em cima</i>
<i>objecto(caixa, madeira).</i>	<i>peso(caixa, 2.5).</i>		

a) Escreva o predicado ***sobre_tipo(+Tipo,-Lista)*** que retorne uma lista (com o formato indicado abaixo) com todos os objectos que estão colocados directamente sobre objectos de um dado tipo.

Exemplo:

```
?- sobre_tipo(madeira, L).
```

```
L = [bigorna*sobre*mesa2]
```

```
?- sobre_tipo(vidro, L).
```

```
L = [cadeira*sobre*mesa1, mesa2*sobre*mesa1, caixa*sobre*jarra]
```

b) Escreva o predicado ***partem(-Objectos)***, que retorne a lista de objectos que vão partir dado o conhecimento apresentado. Um objecto pode partir se for feito de um material frágil e se estiver debaixo de um conjunto de objectos que tenham um peso que seja mais do dobro do peso desse objecto. Sugestão implemente uma função auxiliar ***pesototal(+Obj, -PesoTot)***, que calcule o peso total suportado por um dado objecto.

Exemplo:

```
?- partem(L).
```

```
L = [mesa1-2.8., jarra-5] %Partem a mesa pois tem 2.8 vezes o seu peso em cima (2+3+20+0.5+2.5=28kg) e a jarra que tem 5 vezes o seu peso em cima (pesa 0.5kg e tem a caixa com 2.5kg em cima).
```



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2003/2004
LEIC
(3º Ano)
1º Sem

Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira

Exercícios REC – Controlo e Recursividade em Prolog

Exercício CR 1. Funcionamento do Backtracking

Considere a seguinte questão em PROLOG:

```
?- r(X,Y), s(Y,Z), not(r(Y,X)), not(s(Y,Y)).
```

tendo a seguinte base de dados:

```
r(a,b).      r(a,c).      r(b,a).      r(a,d).  
s(b,c).      s(b,d).      s(c,c).      s(d,e).
```

a) Sem usar o computador, diga qual é a primeira resposta encontrada para a questão posta. (Tenha em atenção que para encontrar a resposta não necessita de simular os passos dados pelo interpretador de PROLOG).

b) Sem usar o computador, diga quantas vezes o interpretador retrocede do 3º para o 2º objectivo, antes de obter a 1ª resposta.

Exercício CR 2. Funcionamento do Backtracking 2

Considere a seguinte base de factos Prolog:

```
a(a1,1).  
a(A,2).  
a(a3,N).  
b(1,b1).  
b(2,B).  
b(N,b3).  
c(X,Y) :- a(X,N), b(N,Y).  
d(X,Y) :- a(X,N), b(Y,N).  
d(X,Y) :- a(N,X), b(N,Y).
```

Efectue uma previsão de quais serão as respostas às seguintes perguntas e verifique-as utilizando-as o rastreio do Prolog

```
?- a(X,2).  
?- b(X,kalamazoo).  
?- c(X,b3).  
?- c(X,Y).  
?- d(X,Y).
```

Exercício CR 3. Funcionamento do Backtracking 3

3) Considere o seguinte programa:

```
exec(X,Y) :- p(X,Y).
exec(X,X) :- s(X).
p(X,Y) :- q(X), r(Y).
p(X,Y) :- s(X), r(Y).
q(a)
q(b).
r(c).
r(d).
s(e).
```

Faça um esboço da árvore de execução para a consulta `?- exec(X,Y)` com as respectivas soluções para cada ramo da árvore.

Exercício CR 4. Calculo de Factorial e Fibonacci

a) Construa um predicado para calcular o factorial de um número N: *factorial(N, Valor)*.

b) Construir um predicado para calcular o fibonacci de um número N.

Série de Fibonacci

N	0	1	2	3	4	5	6
Fib.	1	1	2	3	5	8	13

Solução:

b)

```
fibonacci(0,1).
```

```
fibonacci(1,1).
```

```
fibonacci(N,F):-
```

```
    N > 1,
```

```
    N1 is N - 1, fibonacci(N1,F1),
```

```
    N2 is N - 2, fibonacci(N2,F2),
```

```
    F is F1 + F2.
```

Exercício CR 5. Números Primos

Escreva um predicado `e_primo(N)` que determine se um dado número é primo

Solução:

```
e_primo(2).
```

```
e_primo(3).
```

```
e_primo(P) :- integer(P), P > 3, P mod 2 \= 0, \+tem_factor(P,3).
```

```
tem_factor(N,L) :- N mod L =:= 0.
```

```
tem_factor(N,L) :- L * L < N, L2 is L + 2, tem_factor(N,L2).
```



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2003/2004
LEIC
(3º Ano)
1º Sem

Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira

Exercícios LIST – Utilização de Listas em Prolog

Exercício LIST 1. Funcionamento das Listas [H|T]

Preveja os resultados das seguintes questões em Prolog:

- a) ?- [a|[b,c,d]] = [a,b,c,d].
- b) ?- [a|b,c,d] = [a,b,c,d].
- c) ?- [H|T] = [apple, broccoli, refrigerator].
- d) ?- [H|T] = [a, b, c, d, e].
- e) ?- [H|T] = [apples, bananas].
- f) ?- [H|T] = [a, [b,c,d]].
- g) ?- [H|T] = [apples].
- h) ?- [H|T] = [].
- i) ?- [One, Two | T] = [apple, sprouts, fridge, milk].
- j) ?- [X,Y|T] = [a|Z].
- k) ?- [H|T] = [apple, Z].
- l) ?- [a|[b|[c|[d|[]]]]] = [a,b,c,d].

Solução:

- a) yes
- b) no
- c) H = apple
T = [broccoli, refrigerator]
- d) H = a
T = [b, c, d, e]
- e) H = apples
T = [bananas]
- f) H = a
T = [[b, c, d]]
- g) H = apples
T = []
- h) no
- i) One = apple
Two = sprouts
T = [fridge, milk]

j) $X = a$
 $Y = _01$
 $T = _03$
 $Z = [_01 \mid _03]$
k) $H = \text{apple}$
 $T = [_01]$
 $Z = _01$
l) yes

Exercício LIST 2. Funcionamento das Listas [H|T]

Resolva as igualdades, dizendo quais os valores finais das variáveis.

- a) $?- \text{lista}([a,[b],c,[d]]) = \text{lista}([_|[X|X]])$.
- b) $?- \text{lista}([a],[b],C) = \text{lista}([C,B,[a]])$.
- c) $?- \text{lista}([c,c,c]) = \text{lista}([X|[X|_]])$.
- d) $?- \text{lista}([a,[b,c]]) = \text{lista}([A,B,C])$.
- e) $?- [\text{joao},\text{gosta},\text{peixe}] = [X,Y,Z]$.
- f) $?- [\text{gato}] = \text{lista}([X|Y])$.
- g) $?- [\text{vale},\text{dos},\text{sinos}] = [\text{sinos},X,Y]$.
- h) $?- [\text{branco},Q] = [P,\text{cavalo}]$.
- i) $?- [1,2,3,4,5,6,7] = [X,Y,Z|D]$.

Exercício LIST 3. Concatenação de Listas - Predicado Append

Implemente o predicado $\text{append}(L1,L2,L)$ em que L é constituída pela concatenação das listas $L1$ e $L2$.

Solução:

```
append([ ],L,L).
append([X|L1],L2,[X|L3]):- append(L1,L2,L3).
```

Exercício LIST 4. Inversão de Listas

Construa um predicado $\text{inverter}(L1, L2)$ que calcule a lista invertida de uma dada lista.

Solução:

```
inverter(Lista,InvLista):-
    rev(Lista,[],InvLista).
rev([H|T],S,R):-
    rev(T,[H|S],R).
rev([],R,R).
```

Exercício LIST 5. Membros de uma Lista - Predicado Member e Last

- a) Implemente o predicado $\text{membro}(X,L)$ que sucede se X for um membro da lista L . (member)
- b) Utilizando unicamente o predicado append uma só vez, implemente o predicado $\text{membro}(X,L)$. (member)

- c) Utilizando unicamente o predicado `append` uma só vez, implemente o predicado `last(L,X)` que retorna o último elemento de uma lista.
- d) Implemente um predicado que determine o *n*-ésimo membro de uma lista.

Solução:

b) `membro(X,L):- append(_, [X|_],L).`

c) `last(L,X):- append(_, [X],L).`

d)

`nth_membro(1, [M|_],M).`

`nth_membro(N, [_|T],M):-`

`N>1,`

`N1 is N-1,`

`nth_membro(N1,T,M).`

Exercício LIST 6. Remover Elementos de Listas - Predicados Delete

- a) Utilizando unicamente o predicado `append` duas vezes, implemente o predicado `delete_one(X,L1,L2)` que remove uma ocorrência de um item numa lista.
- b) Implemente um predicado `delete_all(X,L1,L2)` que remova todas as ocorrências de um elemento numa lista
- c) Implemente um predicado que remove as ocorrências dos elementos que se encontram numa lista *LX*, numa outra lista *L1*, `delete_all_list(LX, L1, L2)`.

Solução:

a) `delete_one(X,L,L1):-`

`append(La, [X|Lb],L),`

`append(La,Lb,L1).`

Exercício LIST 7. Elementos de Listas - Predicado Before

- a) Utilizando unicamente o predicado `append` duas vezes, implemente o predicado `before`. O predicado `before` tem três argumentos e sucede se os dois primeiros argumentos forem membros da lista que constitui o terceiro argumento, e o primeiro argumento ocorrer na lista antes do segundo.

Solução:

`before(A,B,L):-`

`append(_, [A|L1],L),`

`append(_, [B|_],L1).`

Exercício LIST 8. Contagem de Elementos de Listas

- a) Defina o predicado `conta(Lista, N)` que sucede se a Lista tem N elementos.
- b) Defina o predicado `conta_elem(X, Lista, N)` que sucede se a Lista tem N elementos com o valor X.

Exercício LIST 9. Substituição e Eliminação de Elementos de Listas

- a) Defina o predicado *substitui*(*X*,*Y*,*Lista1*,*Lista2*) que substitui todas as ocorrências de *X* em *Lista1* por *Y*, resultando *Lista2*.
- b) Defina o predicado *elimina_duplicados*(*Lista1*,*Lista2*) que elimina os duplicados em *Lista1*, resultando *Lista2*.

Exercício LIST 10. Ordenação de Listas

- a) Defina o predicado *ordenada*(*Lista*) que é verdadeiro se *Lista* é uma lista de inteiros ordenada.
- b) Defina o predicado *ordena*(*L1*, *L2*) que ordena a lista *L1*, tendo como resultado *L2*.

Solução:

```
a)
ordenada([N]).
ordenada([N1,N2]):- N1 =< N2.
ordenada([N1,N2|Resto]):-
    N1 =< N2,
    ordenada([N2|Resto]).
```

Exercício LIST 11. Achatar Listas

Defina a relação *achata_lista*(*Lista*,*ElemListas*) em que *Lista* é uma lista eventualmente de listas e *ElemListas* é uma lista com todos os elementos de *Lista* ao mesmo nível.

Exemplo:

```
?- achata_lista([a, [1,2,3], b],L).
L=[a,1,2,3,b]
```

Solução:

```
achata_lista([],[]).
achata_lista(X,[X]):- atomic(X).
achata_lista([Cab|Rest],L):-
    achata_lista(Cab,L1),
    achata_lista(Rest,L2),
    append(L1,L2,L).
```

Exercício LIST 12. Calcular Permutações

Construa um predicado *permutacao*(*L1*,*L2*) que resulte se *L2* for uma permutação de *L1*.

Exercício LIST 13. Listas de Números.

- a) Construa um predicado *lista_ate*(*N*,*L*) que devolva a lista *L* de todos os números inteiros entre 1 e *N*.
- b) Construa um predicado *lista_entre*(*N1*,*N2*,*L*) que devolva a lista *L* de todos os números inteiros entre *N1* e *N2* (ambos incluídos).

- c) Construa um predicado *soma_lista(L, Soma)*, que some todos os elementos da lista L, obtendo como resultado Soma.
- d) Escreva um predicado *par(N)* que dado um número inteiro N, determine se ele é ou não um número par.
- e) Escreva um predicado *lista_pares(N, Lista)* que aceite um número inteiro e que determine a lista de todos os números pares iguais ou inferiores a esse número.
- f) Escreva um predicado *lista_impares(N, Lista)* que aceite um número inteiro e que determine a lista de todos os números ímpares iguais ou inferiores a esse número.

Exercício LIST 14. Números Primos

- a) Escreva um predicado *primo(N)* que dado um número inteiro N, determine se ele é ou não um número primo (um número primo é aquele que só é divisível por 1 e por ele próprio).
- b) Escreva um predicado *lista_primos(N, Lista)* que aceite um número inteiro e que determine a lista de todos os números primos iguais ou inferiores a esse número.

Exercício LIST 15. Produto Interno de Vectors

Construa um predicado *produto_interno (L1, L2, N)* que calcule o produto interno das listas L1 e L2.

Exercício LIST 16. Predicado Mistério

Considere a seguinte definição do predicado mistério:

```
misterio([], []).
misterio([X],[X]).
misterio([X,Y|L],[X,censurado|M]):- misterio(L,M).
```

- a) Descreva o que o predicado faz sobre uma lista constituída por palavras.
- b) É comum as definições recursivas de manipulação de listas terem unicamente uma condição de base. Porque necessita a definição acima de duas condições de base ?

Exercício LIST 17. Listas Palindromas

- a) Recorrendo ao predicado anterior, implemente um predicado que determine se uma lista é um palindroma. Nota: um palindroma pode ser lido da mesma forma para a frente ou para trás (exemplo: [x,a,m,a,x]).
- b) Implemente o mesmo predicado sem recorrer ao predicado inverter.

Solução:

```
a) palindroma(L):- inverter(L,L).
```

Exercício LIST 18. Duplicação de Elementos de uma Lista

- a) Construa um predicado que duplique os elementos de uma lista
- b) Construa um predicado que copie os elementos de uma lista N vezes para a lista resultado

Exemplo a):

```
?- duplicar([a,b,c,c,d],X).
X = [a,a,b,b,c,c,c,c,d,d]
```

Exemplo b):

```
?- duplicarN([a,b,c],3,X).
X = [a,a,a,b,b,b,c,c,c]
```

Exercício LIST 19. Run-Legth Encoding

- Crie um predicado que faça a compressão run-length encoding de uma lista.
- Modifique o programa de forma a que se um elemento não tiver duplicados, é simplesmente copiado para a lista resultado.
- Construa as versões de descompressão de listas, codificadas em run-lenght nas duas alíneas anteriores.

Exemplo a):

```
?- runlength([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).
X = [[4,a],[1,b],[2,c],[2,a],[1,d],[4,e]]
```

Exemplo b):

```
?- run_lenght_modificado([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).
X = [[4,a],b,[2,c],[2,a],d,[4,e]]
```

Exercício LIST 20. Eliminar Elementos de N em N numa lista

Crie um predicado dropN que elimine elementos de N em N numa lista

Exemplo:

```
?- dropN([a,b,c,d,e,f,g,h,i,k],3,X).
X = [a,b,d,e,g,h,k]
```

Solução:

```
dropN(L1,N,L2) :- drop(L1,N,L2,N).

drop([],_,[],_).
drop([_|Xs],N,Ys,1) :- drop(Xs,N,Ys,N).
drop([X|Xs],N,[X|Ys],K) :- K > 1, K1 is K - 1, drop(Xs,N,Ys,K1).
```

Exercício LIST 21. Extrair uma fatia de uma Lista

Crie um predicado slice(Lista, Ind1, Ind2, Result) que extraia uma fatia de uma lista desde o índice Ind1 até ao índice Ind2 (ambos os extremos incluídos).

Exemplo:

```
?- slice([a,b,c,d,e,f,g,h,i,k],3,7,L).
X = [c,d,e,f,g]
```

Exercício LIST 22. Rotação de uma Lista

Construa um predicado que rode de N elementos para a esquerda uma lista.

Exemplos:

```
?- rodar([a,b,c,d,e,f,g,h],3,X).  
X = [d,e,f,g,h,a,b,c]  
?- rodar([a,b,c,d,e,f,g,h],-2,X).  
X = [g,h,a,b,c,d,e,f]
```

Exercício LIST 23. Sorteios Aleatórios e Listas

- a) Extraia um determinado número de elementos seleccionados aleatoriamente de uma lista. E construa uma nova lista com esses elementos
- b) Sorteie N elementos entre 1 e M aleatoriamente e coloque-os numa lista.
- c) Gere uma permutação aleatória dos elementos de uma lista

Exemplo a):

```
?- rnd_selectN([a,b,c,d,e,f,g,h],3,L).  
L = [e,d,a]
```

Exemplo b):

```
?- rnd_select(6,49,L).  
L = [23,1,17,33,21,37]
```

Exemplo c):

```
?- rnd_permutation([a,b,c,d,e,f],L).  
L = [b,a,d,c,e,f]
```

Exercício LIST 24. Bubble Sort de Listas

Implemente o conhecido método de ordenação bubble sort para listas em Prolog.


Solução:

```
bubble_sort(List,Sorted):-  
    b_sort(List,[],Sorted).  
  
b_sort([],Acc,Acc).  
b_sort([H|T],Acc,Sorted):-  
    bubble(H,T,NT,Max),  
    b_sort(NT,[Max|Acc],Sorted).  
  
bubble(X,[],[],X).  
bubble(X,[Y|T],[Y|NT],Max):-  
    X > Y,  
    bubble(X,T,NT,Max).  
bubble(X,[Y|T],[X|NT],Max):-  
    X <= Y,  
    bubble(Y,T,NT,Max).
```

Exercício LIST 25. Triângulo de Pascal

Defina o predicado $\text{pascal}(N, L)$, onde L é a N ésima linha do triângulo de Pascal:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

 Universidade do Porto Faculdade de Engenharia FEUP	Faculdade de Engenharia da Universidade do Porto Licenciatura em Engenharia Informática e Computação Programação em Lógica	2003/2004 LEIC (3º Ano) 1º Sem
Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira		
Exercícios OPA – Operadores e Aritmética		

Exercício OPA 1. Utilização de Operadores

Suponha que temos definidos os seguintes operadores:

```
:- op(500,xfx,na).
:- op(500,xfy,ad).
:- op(500,yfx,ae).
```

Mostre como seriam representadas em PROLOG as seguintes expressões se não tivéssemos as directivas acima (indique os casos em que o PROLOG assinalaria um erro sintáctico):

- a na b ae c.
- a na b ad c.
- a ad b na c.
- a na b na c.
- a ad b ad c.
- a ae b ae c.
- a ad b ad c na d ae e ae f.

Solução:

- ae(na(a,b),c).
- Erro.
- ad(a,na(b,c)).
- Erro.
- ad(a,ad(b,c)).
- ae(ae(a,b),c).
- ad(a,ad(b,ae(ae(na(c,d),e),f))).

Exercício OPA 2. Definição de Operadores Diversos

Crie as directivas que tornam termos abaixo sintacticamente válidos:

- se X entao Y senao Z.
- Y gostaria_de X se X fosse bom e X fosse inteligente.

Solução:

- :-op(500, xfx, entao).
- :-op(400, fx, se).
- :-op(400, xfx, senao).

```

b) :-op(800, xfx, se).
:-op(600, xfx, gostaria_de).
:-op(500, xfy, e).
:-op(400, xfx, fosse).

```

Exercício OPA 3. Definição de Operadores para Voos

Suponha que temos definidos os seguintes operadores:

```

:-op (700, xfx, \\\).
:-op (600, xfx, //).
:-op (600, xfy, ':' :').
:-op (400, yfx, para).
:-op (400, xfx, de).

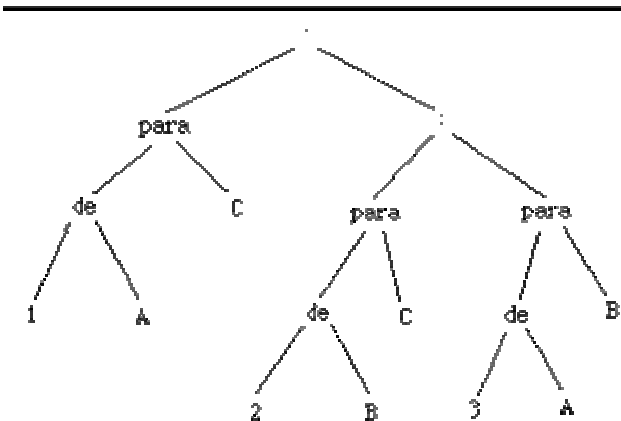
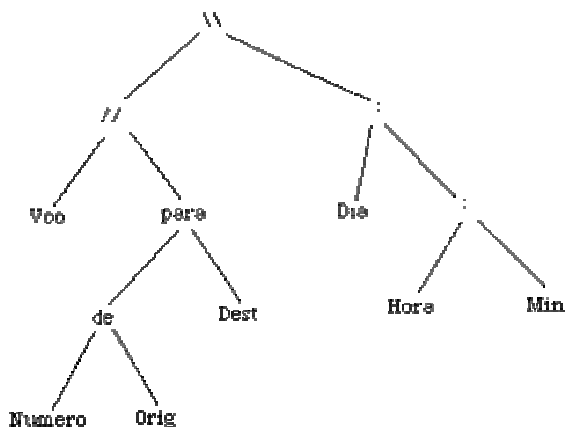
```

Construa uma representação gráfica para os termos:

a) Voo // Número de Orig para Dest \\\ Dia: Hora: min.

b) 1 de A para C: 2 de B para C: 3 de A para B.

Solução:



a) _____ b) _____

Exercício OPA 4. Definição de Operadores para Operações com Listas

Algumas das relações que envolvem listas foram anteriormente escritas no seguinte formato:

```

member(Elemento,Lista),
concatena(Lista1,Lista2,Lista),
delete(Elemento,Lista,NovaLista),
...

```

Suponha que preferíamos escrever estas relações no seguinte formato:

```

Elemento existe_em Lista ,
concatena Lista1 e Lista2 da Lista

```

apaga Elemento a Lista da NovaLista.

Declare existe_em, concatenar, e, etc. como operadores de modo a tornar este formato possível. Redefina as correspondentes relações de acordo com as alterações realizadas.

Solução:

```
:- op(200,xfx,existe_em).
X existe_em [X|_].
X existe_em [_|L]:-
X existe_em L.
:- op(200,fx,concatena).
:- op(150,xfx,da).
:- op(100,xfx,e).
concatena [] e L da L.
concatena [X|L1] e L2 da [X|L3] :-
concatena L1 e L2 da L3.
:- op(200,fx,apaga).
:- op(100,xfx,a).
apaga X a [X|L] da L.
apaga X a [Y|L] da [Y|L1] :-
apaga X a L da L1.
```

Exercício OPA 5. Definição de Operadores – Joga e E

Assumindo as seguintes definições de operadores:

```
:- op(300,xfx,joga).
:- op(200,xfy,e).
```

então os dois termos seguintes possuem sintaxe válida:

```
T1 = marcelo joga futebol e squash.
T2 = renata joga tenis e basquete e volei.
```

Como estes termos são interpretados pelo Prolog? Qual é o functor principal de cada termo e qual a sua estrutura?

Exercício OPA 6. Definição de Operadores – Era e Do

Sugira uma apropriada definição dos operadores "era" e "do" para que seja possível a escrita de cláusulas como:

```
vera era secretária do departamento.
e
paulo era professor do curso.
```

Exercício OPA 7. Definição de Operadores – Operador +

Considere o seguinte programa Prolog:

```
t(0+1, 1+0).
t(X+0+1, X+1+0).
```

```
t(X+1+1, Z) :-
    t(X+1, X1),
    t(X1+1, Z).
```

Como irá este programa responder as seguintes questões, considerando ser + um operador infixo do tipo yfx (como usual).

- a) $?-t(0+1, A).$
- b) $?-t(0+1+1, B).$
- c) $?-t(1+0+1+1+1, C).$
- d) $?-t(D, 1+1+1+0).$

Exercício OPA 8. Definição de Operadores – Se, Então e Senão

Defina os operadores "se", "então", "senão" e "==" de modo que seja válido o termo:

```
se X>Y então Z := X senão Z := Y
```

Escolha a precedência dos operadores de modo que "se" venha a ser o functor principal. Depois defina a relação "se" como um mini-interpretador para um tipo de comando se-então da forma:

```
se V1>V2 então Var:=V3 senão Var:=V4
```

onde V1, V2, V3 e V4 são números (ou variáveis instanciadas com números) e Var é uma variável. O significado da relação "se" deve ser: "se o valor de V1 é maior que o valor de V2, então Var é instanciada com V3, senão Var é instanciada com V4. Um exemplo do uso do mini-interpretador seria:

```
?-X=2, Y=3, V2 is 2*X, V4 is 4*X,
    se Y > V2 então Z:=Y senão Z:=V4,
    se Z > 5 então W=1 senão W=0.
X=2 Y=3 Z=8 W=1
```

Exercício OPA 9. Definição de Operadores – Entre

Defina o procedimento

```
entre(N1, N2, X)
```

que, para dois inteiros dados, N1 e N2, produz através de backtracking todos os inteiros X que satisfazem a restrição

```
N1 >= X >= N2
```

Exercício OPA 10. Definição de Operadores – Polígonos

Estude a definição de um "mundo de polígonos" onde os objectos são definidos em função das coordenadas de seus vértices no plano. Indivíduos desse universo seriam triângulos, rectângulos, quadrados, etc. Por exemplo o termo:

```
triângulo((1,1), (1,2), (2,2))
```


definiria um triângulo cujos vértices seriam os pontos (1,1), (1,2) e (2, 2) em um sistema de coordenadas cartesianas.

Formule as propriedades básicas de cada objecto através de relações unárias, tais como:

`isósceles(X)`

Formule relações entre diferentes indivíduos, representando assertivas tais como:

`"Uma casa é um quadrado com um triângulo em cima".`

ou

`"D é distância entre os centros geométricos de A e B".`

Pense numa versão deste programa para gerar trajectórias de figuras planas ao longo de curvas de equações dadas.



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2003/2004
LEIC
(3º Ano)
1º Sem

Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira

Exercícios – Unificação e Predicados CUT e NOT

Exercício CN 1. Efeito do Cut

1) Considere a seguinte base de dados:

A :- B, C, !, D, E.

A :- F, G.

Supondo que as letras maiúsculas acima representam termos em PROLOG, qual o efeito do "cut" no conjunto de cláusulas acima ?

Exercício CN 2. Efeito do Cut 2

Considere a seguinte base de dados:

p(1).

p(2):-!.
p(3).

Indique todas as respostas dadas pelo interpretador às questões:

a) ?- p(X).

b) ?- p(X), p(Y).

c) ?- p(X), !, p(Y).

Solução:

a) X=1 ; X=2

b) X=1 Y=1 ; X=1 Y=2 ; X=2 Y=1 ; X=2 Y=2

c) X=1 Y=1 ; X=1 Y=2

Exercício CN 3. Efeito do Cut 3

Suponha a seguinte base de factos em Prolog

dados(um).

dados(dois).

dados(tres).

a) Qual o resultado da seguinte pergunta?

```
cut_teste_a(X) :-
```

```
    dados(X).
```

```
cut_teste_a('ultima_clausula').
```

```
?- cut_teste_a(X), write(X), nl, fail.
```

a) Qual o resultado do seguinte programa com um Cut no final da primeira clausula?

```
cut_teste_b(X):-  
    dados(X), !.  
cut_teste_b('ultima_clausula').  
?- cut_teste_b(X), write(X), nl, fail.
```

c) Qual o resultado do seguinte programa com um Cut no meio dos dois objectivos?

```
cut_teste_c(X,Y) :-  
    dados(X),  
    !,  
    dados(Y).  
cut_teste_c('ultima_clausula').  
?- cut_teste_c(X,Y), write(X-Y), nl, fail.
```

Solução:

```
a)    um  
      dois  
      tres  
      ultima_clausula  
      no  
b)    um  
      no  
c)    um - um  
      um - dois  
      um - tres  
      no
```

Exercício CN 4. Maior de Três Números

Estude o seguinte programa que calcula o maior de entre três números:

```
max(X, Y, Z, X):- X>Y, X>Z, !.  
max(X, Y, Z, Y):- Y>X, Y>Z, !.  
max(_, _, Z, Z).
```

a) Diga em que situações o programa não funciona correctamente.

b) Corrija o programa.

Solução:

a) Quando X é igual a Y e X e Y são maiores que Z, Z é indicado como sendo o maior número.

```
b) max2(X,Y,Z,X):-X>=Y,X>=Z,!.  
   max2(X,Y,Z,Y):-Y>=X,Y>=Z,!.  
   max2(_,_,Z,Z).
```

Exercício CN 5. Unificação

Defina o predicado *unificavel(L1, Termo, L2)* em que *L2* é uma lista com todos os elementos de *L1* que são unificáveis com *Termo*. Os elementos de *L2* não são no entanto unificados com *Termo*. Exemplo:

```
?- unificavel([X,b,t(Y)],t(a),L).  
L=[X,t(Y)]
```

Note que se *Termo1=Termo2* resulta então *not(Termo1=Termo2)* falha e a instanciação resultante de *Termo1=Termo2* é anulada.

Solução:

```
unificavel([],_,[]).  
unificavel([T|Resto],T1,Resto1):-  
    not T=T1, !,  
    unificavel(Resto,T1,Resto1).  
unificavel([T|Resto],T1,[T|Resto1]):- unificavel(Resto,T1,Resto1).
```

Exercício CN 6. Cuts Verdes e Vermelhos

Explique a função dos ‘cuts’ incluídos no programa abaixo.

```
imaturo(X):- adulto(X), !, fail.  
imaturo(X).  
adulto(X):- pessoa(X), !, idade(X, N), N>=18.  
adulto(X):- tartaruga(X), !, idade(X, N), N>=50.
```

Solução:

O Cut em *Imaturo* funciona como no predicado *Not*. Se *X* for adulto então *(!, fail)* não é imaturo. Senão é imaturo. É um Cut Vermelho pois é essencial para o funcionamento do programa.

O Cut em *adulto* evita explorar espaço de pesquisa em que é impossível estar a solução. Se *X* for uma pessoa, então só será adulto se tiver uma idade maior ou igual a 18. Só se *X* não for uma pessoa é que se vai verificar se *X* é uma Tartaruga, ...

É um Cut verde pois não altera as soluções obtidas mas sim a eficiência do programa.



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2003/2004
LEIC
(3º Ano)
1º Sem

Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira

Exercícios – Exercícios sobre Lógica

Exercício LOG 1: Tabela de Verdade

Defina os predicados and/2, or/2, nand/2, nor/2, xor/2, impl/2 e equ/2 (equivalência lógica) que sucedem ou falham de acordo com o resultado das respectivas operações, ou seja, and(A,B) sucede se e só se ambos A e B sucedem. Note que A e B podem ser cláusulas Prolog e não só constantes true e fail.

Uma expressão lógica com duas variáveis pode então ser escrita em notação prefixa como: and(or(A,B), nand(A,B)).

Escreva um predicado tabela/3 que escreva uma tabela de verdade para uma expressão lógica com duas variáveis.

Exemplo:

```
?- tabela(A,B,and(A,or(A,B))).  
true true true  
true fail true  
fail true fail  
fail fail fail
```

Solução:

```
and(A,B) :- A, B.  
or(A,_) :- A.  
or(_,B) :- B.  
equ(A,B) :- or(and(A,B), and(not A,not B)).  
xor(A,B) :- not(equ(A,B)).  
nor(A,B) :- not or(A,B).  
nand(A,B) :- not and(A,B).  
impl(A,B) :- or(not A,B).  
% bind(X) :- instancia X a ser verdadeiro e falso sucessivamente  
bind(true).  
bind(fail).  
table(A,B,Expr) :- bind(A), bind(B), do(A, B, Expr), fail.  
do(A,B,_) :- write(A), write(' '), write(B), write(' '), fail.  
do(_,_,Expr) :- Expr, !, write(true), nl.  
do(_,_,_) :- write(fail), nl.
```

Exercício LOG 2: Tabela de Verdade com Operadores

Continue o problema anterior e defina and/2, or/2, etc. como sendo operadores. Isto permite escrever expressões lógicas de forma muito mais natural como por exemplo: A and (A or not B). Defina a precedência dos operadores da forma correcta.

Exemplo:

```
?- table(A,B, A and (A or not B)).  
true true true  
true fail true  
fail true fail  
fail fail fail
```

Exercício LOG 3: Tabela de Verdade com N Variáveis

Generalize as expressões anteriores de forma a que as expressões lógicas possam ter qualquer número de variáveis. Defina tabela/2 de forma a que tabela(Lista, Expr) escreva a tabela de verdade para a expressão Expr que contém a lista de variáveis lógicas enumeradas em Lista.

Exemplo:

```
?- table([A,B,C], A and (B or C) equ A and B or A and C).  
true true true true  
true true fail true  
true fail true true  
true fail fail true  
fail true true true  
fail true fail true  
fail fail true true  
fail fail fail true
```



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2003/2004
LEIC
(3º Ano)
1º Sem

Exercícios – Meta-Programação e Meta-Interpretadores

Exercício MP1. Utilização do Operador =..

É frequente desejarmos realizar uma dada transformação em todos os elementos de uma lista. Para o efeito vamos recorrer a um predicado de aridade 2. A esta transformação chama-se também mapeamento numa lista. Construa um predicado de mapeamento utilizando o operador =.. na sua definição.

Exemplo1:

Tendo

```
f(X,Y):-Y is X*X.
```

vem

```
?-map([2,4,8],f,L).
```

```
L=[4,16,64]
```

Exemplo2:

Tendo

```
duplica(X,Y) :- Y is 2*X.
```

vem

```
?-map([1,2,3],duplica,L).
```

```
L=[2,4,6]
```

Solução:

```
map([],_,[]).
```

```
map([C|R],Transfor,[TC|CR]):-  
    aplica(Transfor, [C,TC]),  
    map(R,Transfor,CR).
```

```
aplica(P,LArgs) :- G =.. [P|LArgs], G.
```

Exercício MP2. Lista de Elementos que tornam Predicado Verdadeiro

Implemente o predicado separa(+L,+Pred,-Lista) que dada uma lista L e um nome de um predicado de aridade 1, devolve a lista com exactamente os mesmos elementos mas em que primeiro aparecem todos aqueles que tornam verdadeiro o predicado.

```
separa(L,P,Res) :- sepDL(L,P,Res-Nots,Nots-[]).
```

```
sepDL([],_,P-P,N-N).
```

```
sepDL([V|L],P,[V|Y]-DY,N) :- aplica(P,[V]),!, sepDL(L,P,Y-DY,N).
sepDL([V|L],P,Y,[V|N]-DN) :- sepDL(L,P,Y,N-DN).
```

Exercício MP3. Idades Mais Próximas

Implemente utilizando o setof/3, o predicado mais_proximos(+Idade,-ListaProximos) que, assumindo a existência de factos idade(Nome,Idade) para representar que um dado indivíduo chamado Nome tem idade Idade, devolve em ListaProximos o nome dos indivíduos cuja idade é mais próxima de Idade.

Solução:

```
mais_proximos(I,[N1|Prox]) :-
    setof(Dif-Nome,prox(I,Dif,Nome),[D1-N1|L]),
    primeiros(D1,L,Prox).

prox(I,Dif,Nome) :- idade(Nome,Id), dif(I,Id,Dif).

dif(A,B,D) :- A > B,!, D is A - B.
dif(A,B,D) :- D is B - A.

primeiros(_,[],[]).
primeiros(D1,[D-_|_],[]) :- D > D1,!.
primeiros(D1,[_N|L],[N|NL]) :- primeiros(D1,L,NL).

%Dados para teste:
idade(maria,30).
idade(pedro,25).
idade(jose,25).
idade(rita,18).
```

Exercício MP4. Definição de functor(Term,F,N) e arg(N,Term,Arg) em termos do operador =..

a) Defina o predicado functor2(Term,F,Arity) que é verdadeiro se Term é um termo cujo functor principal tem o nome F e a aridade Arity.

Solução:

```
functor_(Term,F,N) :- Term=..[F|Args], length(Args,N).
```

b) Defina o predicado arg(N,Term,Arg) que é verdadeiro se Arg é o N-ésimo argumento do termo Term.

Solução:

```
arg_(N,Term,Arg) :- Term=..[F|Args], position(N,Args,Arg).

position(1,[X|_],X).
position(N,[_|Xs],Y) :- N>1, N1 is N-1, position(N1,Xs,Y).
```




Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2003/2004
LEIC
(3º Ano)
1º Sem

Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira

Exercícios – Prolog Avançado

Exercício PAV 1. Predicados de Segunda Ordem

Escreva um programa em PROLOG, e coloque-lhe as questões apropriadas a resolver o seguinte problema:

Os Srs. Azul, Verde e Castanho estão sentados num parque. Um deles a dado momento exclama:

- Os nossos nomes correspondem às cores dos nossos olhos. Nós temos olhos azuis, verdes e castanhos.

- Sim - replica o Sr. com olhos castanhos - mas só um tem o nome igual à cor dos olhos.

- Estás certo - diz o Sr. Azul.

O Sr. Verde tem olhos verdes ?

Solução:

/ Encontrar as possíveis correspondências entre nomes e cor dos olhos respeitando as seguintes restrições:*

1. O Sr. Azul não pode ter olhos castanhos.

2. Exactamente um tem o nome correspondente à cor dos olhos.

A correspondência vai ser representada no formato: [nome1/cor1, nome2/cor2, ...] */

```
nomes([verde, azul, castanho]).
```

```
possiveis_cores(Lista):-
```

```
    nomes(Nomes), nomes(Cores),
```

```
    findall(
```

```
        Nomes_Cores,
```

```
        (permuta(Cores,CoresP),
```

```
        corresp(Nomes,CoresP,Nomes_Cores),
```

```
        not( member(azul/castanho,Nomes_Cores)),
```

```
        member(X/X,Nomes_Cores),
```

```
        not( member(Y/Y,Nomes_Cores), Y\==X)) )
```

```
    Lista).
```

/ findall/3 produz uma lista arg3 de todos os objectos arg1 que satisfazem a propriedade arg2 */*

```
findall(X, P, _):-
```

```

        asserta(encontrou(base)),      % marca de base da pilha P,
        asserta(encontrou(X)),        % X é instancia do obj. que satisfaz G
        fail.
findall(_ , _ , L) :-
    recolhe_encontrados([], R), !, L = R.
recolhe_encontrados(Aux,L) :-
    recolhe_seguinte(X), !,
    recolhe_encontrados([X| Aux], L).
recolhe_encontrados(L,L).
recolhe_seguinte(X):-
    retract(encontrou(X)), !, X =\= base.
corresp([],[],[]).
corresp([Nome|Nomes],[Cor|Cores],[Nome/Cor|Ns-Cs]):-
    corresp(Nomes,Cores,Ns-Cs).
?- possiveis_cores(Lista), member(Ns-Cs,Listas), member(verde/verde,Ns-Cs).

```

Exercício PAV 2. Derivada de uma Expressão Matemática

Escreva um programa em Prolog, que calcula a derivada de uma expressão matemática. Esta expressão pode incluir as operações aritméticas de negação, adição, subtração, multiplicação e potência.

Exemplo:

```

expressão: -x^2+3*x+1
derivada: -2*x+3

```

Exercício PAV 3. Chave do Totobola

Uma chave do totobola pode ser representada em uma lista de 13 elementos. As múltiplas são representadas por listas interiores.

```
Ex: [[1,x,2],x,1,1,2,1,[1,2],2,[x,2],1,2,[1,x,2],1]
```

Implemente um predicado Prolog, que determina todas as chaves simples que correspondem a uma determinada chave múltipla.

Solução:

```

chave(C,L):- chave(C,[],L).
chave([],L,L).
chave([C1|O],Lant,L):-
    decompoe(C1,Lant,Laux),
    chave(O,Laux,L).
% elemento e chave multipla
decompoe([],_,[]).
decompoe([X1|O],Lant,L):-
    junta(X1,Lant,Lx),
    decompoe(O,Lant,R),

```

```

        append(Lx,R,L).
%elemento e chave simples
decompoe(X,Lant,L):- atomic(X), junta(X,Lant,L).
junta(_,[],[]).
junta(X,[L1|OL],[L11|OLr]):-
    append(L1,[X],L11),
    junta(X,OL,OLr).

```

Exercício PAV 4. Simplificação de Expressões Matemáticas

Escreva um predicado Prolog para simplificação de expressões matemáticas simples, contendo apenas o operador soma. Os termos não variáveis são somados, e os termos variáveis são agrupados por variável distinta.

Nota: A associatividade do operador + é yfx.

Sugestão: Guarde os termos variáveis da expressão numa lista, a ser tratada posteriormente.

Exemplos:

```

se Expressão = 2+A+1+4+B+3      então Expressão Simplificada = A+B+10
se Expressão = A+3+B+A+1         então Expressão Simplificada = 2*A+B+4

```

Solução:

```

simplifica(L,Ls):-
    simplifica(L,Lv,Valor),
    trata(Lv,_,Lv1),Ls=Lv1+Valor.

simplifica(X+Y,[Y|R],V):- var(Y),simplifica(X,R,V).
simplifica(X+Y,X1,V):- simplifica(X,X1,V1), V is V1+Y.
simplifica(X,[X],0):- var(X).
simplifica(X,[],X).

trata([],Expr,Expr).
trata([X|T],Expr1,ExprF):-
    nv_iguais(T,X,N,R),
    ( ( N<2,Expr=X ) ; Expr=N*X ),
    ( ( var(Expr1),Expr2=Expr ) ; Expr2=Expr1+Expr ),
    trata(R,Expr2,ExprF).

nv_iguais([],_,1,[]).
nv_iguais([Xa|T],X,N,Rt):- Xa==X, nv_iguais(T,X,N1,Rt), N is N1+1.
nv_iguais([Y|T],X,N,[Y|Rt]):- nv_iguais(T,X,N,Rt).

```

Exercício PAV 5. Potências Nucleares

Dois países p_1 e p_2 , possuidores de mísseis de diferentes potências (identificadas por um valor inteiro), estão a negociar a destruição dessas armas.

Cada passo da negociação implica a destruição, por parte de um país, de um míssil; e do outro país de um ou dois mísseis cuja soma de potências seja igual à do míssil que o primeiro país destruiu. Em qualquer instante da negociação a diferença de quantidade de mísseis entre os dois países não pode ser superior a uma unidade.

O objectivo é eliminar todos os mísseis de um ou ambos os países.

Inicialmente o país p_1 possui N mísseis e o país p_2 $N-1$ mísseis.

- a) Descreva, em Prolog, as estruturas de dados que utilizaria na identificação de um estado.
- b) Escreva um programa simples em Prolog, comentado, para resolução deste problema.

Exercício PAV 6. Navegação Robótica

Considere a planta de um espaço navegável por um robot representada por uma quadrícula (linhas×colunas), onde as linhas verticais e horizontais de qualquer quadrícula são locais de passagem. O comprimento do lado de qualquer quadrícula é de 1 unidade. A partir de um ponto (cruzamento de linhas) só é possível atingir pontos adjacentes a este na vertical ou horizontal. Os obstáculos impedem a passagem em determinados pontos (vértices da quadrícula).

- a) Escreva os factos Prolog necessários para representar a planta de um espaço navegável.
- b) Implemente uma função heurística para o valor a atribuir à posição (x_a, y_a) da quadrícula, quando o objectivo é atingir um ponto fixo (x, y) . (Sugestão: use a distância euclidiana entre os dois pontos.)
- c) Implemente um programa Prolog comentado, que calcula um caminho a percorrer pelo robot, quando este parte de um ponto inicial (x_i, y_i) , com o objectivo de atingir um ponto final (x_f, y_f) . Utilize o algoritmo A^* , e a função heurística da alínea anterior.

Exercício PAV 7. Transmissão de um Carro

A transmissão de um carro é composta por rodas dentadas que transmitem a potência do motor às rodas do veículo. Podemos inferir a velocidade e direcção das rodas, partindo de factos sobre que rodas dentadas estão em contacto com outras rodas dentadas. As rodas dentadas estão referenciadas por $rd1$, $rd2$, ..., rdn .

O número de dentes de cada roda dentada é especificado por factos com o formato:

`<nome_da_roda_dentada> tem_num_dentes <número_de_dentes>.`

As rodas que estão montadas no mesmo eixo são representadas por factos com o formato:

`<nome_da_roda_dentada2> no_mesmo_eixo_que <nome_da_roda_dentada1>.`

As rodas que estão em contacto são especificados por factos com o formato:

`<nome_da_roda_dentada2> em_contacto_com <nome_da_roda_dentada1>.`

- a) Pretende-se abordar a questão da velocidade angular e sentido das rodas dentadas. Defina um formato para a representação destes parâmetros relativamente a cada roda.
- b) Sempre que duas rodas estão montadas no mesmo eixo a sua velocidade angular é idêntica. Escreva uma regra em PROLOG que permita inferir a velocidade angular de uma roda sobre

um eixo a partir da velocidade angular de outra roda sobre esse mesmo eixo. Use o formato definido em a).

- c) Sempre que duas rodas estão em contacto o produto da velocidade angular pelo número de dentes é igual para as duas rodas, com a ressalva de rodarem em direcções opostas. Escreva uma regra em PROLOG que permita inferir a velocidade angular de uma roda a partir da velocidade angular de outra em contacto com a primeira, assumindo que o número de dentes das duas rodas é conhecido. Use o formato definido em a).
- d) Suponha o seguinte jogo de rodas dentadas. A roda rd1 tem uma velocidade de 5000 rpm na direcção dos ponteiros do relógio. Suponha que sobre o mesmo eixo de rd1 existe rd2, rd2 está em contacto com rd3 e com rd4. Suponha que rd1 tem 100 dentes, rd2 tem 30, rd3 tem 60 e rd4 tem 90.

Questione o interpretador sobre a velocidade angular e direcção da roda rd4.

Mostre os passos dados pelo interpretador de PROLOG para responder à questão posta.

- e) Explique em que situação, caso não tenha os necessários cuidados na especificação dos factos, podem surgir ciclos infinitos no raciocínio sobre rodas dentadas.
- f) Suponha que para uma dada combinação de rodas dentadas e seguindo uma determinada cadeia de raciocínio a roda rd8 tem velocidade angular de 1200 rpm. Seguindo outra cadeia de raciocínio rd8 tem velocidade angular de 800 rpm. Que há a dizer sobre a ocorrência desta situação?

Solução:

a)

<roda> tem_velocidade <velocidade_angular>

c/ veloc. positiva se roda na direcção dos ponteiros do relógio.

b)

```
:-op(300,xfx,[tem_num_dentes,no_mesmo_eixo_que,em_contacto_com,tem_velocidade]).
```

```
R tem_velocidade V :-
```

```
R no_mesmo_eixo_que R1,
```

```
R1 tem_velocidade V.
```

c)

```
R tem_velocidade V:-
```

```
R em_contacto_com R1,
```

```
R tem_num_dentes ND,
```

```
R1 tem_num_dentes ND1,
```

```
R1 tem_velocidade V1,
```

```
V is 0-(V1*ND1/ND).
```

d)

```
rd1 tem_velocidade 5000.
```

```
rd2 no_mesmo_eixo_que rd1.
```

```
rd3 em_contacto_com rd2.
```

```
rd4 em_contacto_com rd2.
```

```
rd1 tem_num_dentes 100.
```

```
rd2 tem_num_dentes 30.  
rd3 tem_num_dentes 60.  
rd4 tem_num_dentes 90.
```

e) Podem ocorrer ciclos se incluir factos como por exemplo

```
rd2 em_contacto_com rd1.  
rd1 em_contacto_com rd2.
```

Nota: nos factos que consideramos está presente uma ideia de direcção de transmissão de potência.

f) As rodas não rodavam ou autodestruíam-se. Esta situação não ocorre no sistema que descrevemos em d).

Exercício PAV 8. Unidades Arquitectónicas

Escreva um programa que determina a configuração de unidades arquitectónicas. Essas unidades obedecem às seguintes especificações:

- i) cada unidade tem dois quartos quadrados;
- ii) cada quarto tem uma janela e uma porta interior;
- iii) os quartos estão ligados pelas portas interiores;
- iv) um dos quartos tem também uma porta para o exterior;
- v) cada parede só pode ter uma janela ou uma porta;
- vi) nenhuma janela pode estar voltada a Norte;
- vii) As janelas não podem estar em lados opostos da unidade arquitectónica.

Exemplo:

```
?- plano(oeste, P1, J1, P2, J2). /* em que direcção ficam as portas interiores e  
janelas dos quartos, estando a porta exterior voltada a Oeste ? */  
P1 = este  
J1 = sul  
P2 = oeste  
J2 = sul
```

Solução:

```
plano_arq(PortaExt, P1, J1, P2, J2) :-  
    quarto_frente(PortaExt, P1, J1),  
    oposto(P1, P2) % portas interiores viradas uma para a outra  
    quarto(P2, J2),  
    nao_oposto(J1, J2).  
quarto_frente(PE, P, J) :-  
    quarto(P, J),  
    direccao(PE),  
    PE =\= P,  
    PE =\= J.  
quarto(P, J) :-
```

```

    direcacao(P),
    direcacao(J),
    P =\= J,
    J =\= norte.
direcacao(norte).
direcacao(sul).
direcacao(oeste).
direcacao(este).
oposto(norte, sul).
oposto(sul, norte).
oposto(este, oeste).
oposto(oeste, este).
nao_oposto(P1, P2) :-
    oposto(P1, P3), P2 =\= P3.

```

Exercício PAV 9. Corrida de Robôs

Um percurso está marcado com uma série de marcos numerados (inteiros crescentes). Junto a cada marco, há um cronometrista.

Decidiu-se fazer uma corrida de robôs nesse percurso. Tal como nos contra-relógios do ciclismo e nas provas dos rallys, sai um robô de cada vez, e ganha o que conseguir fazer o percurso em menos tempo, na condição de que tenha sido cronometrado em cada um dos marcos do percurso (se "falhar" um marco, é desclassificado).

As cronometragens realizadas são armazenadas em computador no formato:

```
pos(Robot, Marco, Tempo)
```

Exemplo:

```

pos(ra, 1, 0).
pos(ra, 2, 10).
pos(ra, 3, 15).
pos(rb, 1, 5).
pos(rb, 2, 14).
pos(rb, 3, 21).
pos(rc, 1, 10).
pos(rc, 3, 23).

```

Neste exemplo, ra fez o percurso em 15 unidades de tempo, rb em 16 e rc foi desclassificado.

a) Construa um programa `prova(Robot, L)`, onde `L` é uma lista contendo estruturas com a forma

```
Marco:Tempo
```

A lista deverá ficar ordenada por valores crescentes dos marcos.

Exemplos:

```

?- prova(ra, L).
L = [1:0, 2:10, 3:15]

```

```
?- prova(rc, L).
```

```
L = [1:10, 3:23]
```

b) Construa um programa `totais(Robot, NMarcos, Total)` que, dado um Robô e o número de marcos do percurso (NMarcos), calcule o tempo total do percurso do robô. O programa deverá devolver o valor zero caso o concorrente seja desclassificado.

Exemplos:

```
?- totais(ra, 3, T).
```

```
T = 15
```

```
?- totais(rc, 3, T).
```

```
T = 0
```




Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2003/2004
LEIC
(3º Ano)
1º Sem

Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira

Exercícios – Pesquisa de Soluções

Exercício PESQ1. Pesquisa de Ligação num Grafo

Escreva um programa em Prolog que determine um caminho entre dois nós de um Grafo.

- a) Utilizando pesquisa em profundidade (evitando ciclos);
- b) Utilizando pesquisa em largura.

De forma a experimentar o programa, considere que o grafo é definido pelos seguintes factos:

```
ligado(a,b).          ligado(f,i).
ligado(a,c).          ligado(f,j).
ligado(b,d).          ligado(f,k).
ligado(b,e).          ligado(g,l).
ligado(b,f).          ligado(g,m).
ligado(c,g).          ligado(k,n).
ligado(d,h).          ligado(l,o).
ligado(d,i).          ligado(i,f).
```

Solução:

```
/* Para testar, utilize:
   ?- resolve_larg(No_inicial, No_meta, Solucao).
   ?- resolve_prof(No_inicial, No_meta, Solucao).
   onde No_inicial é o inicio do grafo e No_meta é a meta que se deseja atingir.
   Solucao retorna o caminho entre No_inicial e No_meta, na forma de uma lista.*/
```

// Utilitários de manipulação de Listas

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

concatena([], L, L).
concatena([X|Y], L, [X|Lista]) :- concatena(Y, L, Lista).

inverte([X], [X]).
inverte([X|Y], Lista) :- inverte(Y, Lista1), concatena(Lista1, [X], Lista).
```

// a) Pesquisa em Profundidade

```
// Encontra o caminho Solucao entre No_inicial e No_meta
resolve_prof(No_inicial, No_meta, Solucao) :-
    profundidade([], No_inicial, No_meta, Sol_inv),
    inverte(Sol_inv, Solucao).
```

```
// Realiza a pesquisa em profundidade
profundidade(Caminho, No_meta, No_meta, [No_meta|Caminho]).
profundidade(Caminho, No, No_meta, Sol):-
    ligado(No, No1),
    not membro(No1, Caminho),          % previne ciclos
    profundidade([No|Caminho], No1, No_meta, Sol).

// b) Pesquisa em Largura

// Acha todos os X onde Y esta satisfeito e retorna numa lista Y
ache_todos(X, Y, Z):- bagof(X, Y, Z), !.
ache_todos(_, _, []).

// Estende a fila ate um filho N1 de N, verificando se N1
// não pertence à fila, prevenindo, assim, ciclos
estende_ate_filho([N|Trajectory], [N1,N|Trajectory]):-
    ligado(N, N1),
    not membro(N1, Trajectory).

// Encontra o caminho Solucao entre No_inicial e No_Meta
resolva_larg(No_inicial, No_meta, Solucao):-
    largura([No_inicial], No_meta, Sol1),
    inverte(Sol1, Solucao).

// Realiza a pesquisa em largura
largura([No_meta|T]|_, No_meta, [No_meta|T]).
largura([T|Fila], No_meta, Solucao):-
    ache_todos(ExtensaoAteFilho, estende_ate_filho(T, ExtensaoAteFilho), Extensoes),
    concatena(Fila, Extensoes, FilaExtendida),
    largura(FilaExtendida, No_meta, Solucao).
```

Exercício PESQ2. Pesquisa de Ligação mais Rápida num Grafo

Altera o Programa PESQ1 de forma a que cada ligação tenha um custo. Escreva um programa que lhe permita encontrar o caminho mais rápido (de menor custo) entre dois nós do grafo.

Exercício PESQ3. Pesquisa de Ligação com Visita a uma Lista de Nós

Altera o Programa PESQ1 ou PESQ2 de forma a encontrar um caminho entre dois nós do grafo que visite todos os nós contidos numa lista de nós fornecida como parâmetro de entrada ao algoritmo.

Exercício PESQ4. Todos os Trajectos

Altera o Programa anterior de forma a que retorne todos os trajectos possíveis entre dois nós do grafo

Exercício PESQ5. Disponibilidades para Reuniões

Uma empresa pretende realizar um encontro num determinado mês que reúna alguns responsáveis por várias unidades dessa empresa. As disponibilidades em termos de dias por parte de cada membro a estar presente na reunião são dadas por factos do tipo:

```
disponibilidade(nome, lista_dias_disponíveis).
```

onde cada membro da lista de dias disponíveis é do tipo:

```
disp(primeiro_dia, último_dia).
```

Por exemplo:

```
disponibilidade(pedro, [disp(2,4), disp(12,20), disp(25,28)])
```

indica que Pedro está disponível dos dias 2 a 4, dos dias 12 a 20 e dos dias 25 a 28.

a) Escreva um programa Prolog que receba como argumento um dia e retorne todos os nomes de pessoas disponíveis nesse dia para uma reunião.

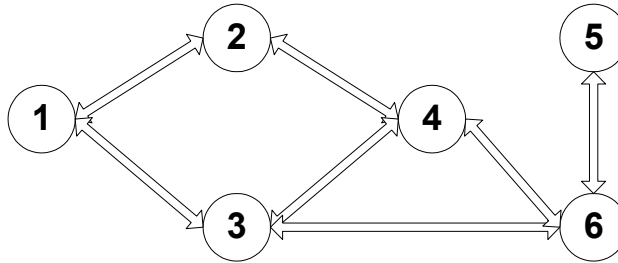
b) Escreva um programa Prolog que receba uma lista de nomes e o número mínimo de dias consecutivos para realizar uma reunião e crie uma lista com elementos do tipo `poss(primeiro_dia, último_dia)` representando as possibilidades de realização da reunião envolvendo todas as pessoas da lista de nomes.

Exercício PESQ6. Pesquisa de Ligação com Visita a uma Lista de Nós

Considere que um grafo não dirigido é representado por um conjunto de cláusulas unitárias da forma **ligacao(No1, No2)**.

Exemplo:

```
ligacao(1, 2).
ligacao(1, 3).
ligacao(2, 4).
ligacao(3, 4).
ligacao(3,6).
ligacao(4, 6).
ligacao(5,6).
```



a) Escreva um predicado **caminho(+NoInicio, +NoFim, -Lista)**, que dados dois nós do grafo, calcule um possível caminho (não necessariamente o mais curto) entre esses nós. *Nota: Suponha que a dimensão máxima do caminho é 5.*

Exemplos:

```
?- caminho(2, 3, Lista).
```

```
Lista = [2,4,3] ; Lista = [2,4,6,3] ; Lista = [2,1,3] ; no
```

```
?- caminho(1, 5, Lista) %Uma possível solução é: Vai do nó 1 para o 2, depois para o 4, para o 6 e finalmente para o 5.
```

```
Lista = [1,2,4,6,5] ; Lista = [1,2,4,3,6,5] ; Lista = [1,3,4,6,5] ; Lista = [1,3,6,5] ; no
```

```
?- caminho(2, 2, Lista).
```

```
Lista = [2,4,2] ; Lista = [2,4,6,3,1,2] ; Lista = [2,4,3,1,2] ; Lista = [2,1,2] ; Lista = [2,1,3,4,2] ; Lista = [2,1,3,6,4,2] ; no
```

b) Escreva um predicado **ciclos(+No, +Comp, -Lista)**, que dado um nó, calcule todos os ciclos possíveis, com comprimento inferior a **Comp**, desse nó. *Sugestão: Utilize o predicado caminho (alínea anterior) como base para a resolução.*

Exemplo:

```
?- ciclos(4, 3, Lista).
```

```
Lista = [[4,3,6], [4,6,3]]
```

```
?- ciclos(4, 5, Lista).
```

```
Lista = [[4,3,6], [4,6,3], [4,2,1,3], [4,3,1,2]]
```

Solução:

```
% a) caminho(+NoInicio, +NoFim, -Lista),
ligacao2(X,Y) :- ligacao(X,Y).
ligacao2(X,Y) :- ligacao(Y,X).
```

```

caminho(NoInicio, NoFim, Lista):-
    caminho(NoInicio, NoFim, [NoInicio], Lista, 5).

caminho(NoInicio, NoFim, Lista, ListaFim, _):-
    ligacao2(NoInicio, NoFim),
    append(Lista, [NoFim], ListaFim).
caminho(NoInicio, NoFim, Lista, ListaFim, N):-
    N>0,
    ligacao2(NoInicio, NoInterm),
    NoInterm \= NoFim,
    \+(member(NoInterm, Lista)),
    append(Lista, [NoInterm], Lista2),
    N2 is N-1,
    caminho(NoInterm, NoFim, Lista2, ListaFim, N2).

% Outra versão. Sera' que funciona? Porque?

caminho2(NoIni, NoFim, Lista):- caminho2(NoIni, NoFim, Lista, 5).

caminho2(_,_,_,0):- !,fail.
caminho2(NoIni, NoFim, [NoIni,NoFim], _):- ligacao2(NoIni,NoFim).
caminho2(NoIni, NoFim, [NoIni|Rest], N):-
    N2 is N-1,
    ligacao2(NoIni, NoInt),
    caminho2(NoInt, NoFim, Rest, N2).

% Ainda outra Versão!

caminho3(NoIni, NoFim, Lista):- caminho3(NoIni, NoFim, Lista, 0).

caminho3(NoIni, NoFim, [H|Rest], N):-
    N < 4,
    ligacao2(NoIni, NoInt),
    (NoInt = NoFim, [H|Rest] = [NoIni, NoFim] ;
    H = NoIni, N2 is N+1, caminho3(NoInt, NoFim, Rest, N2)).

b) ciclos(+No, +Comp, -Lista),

ciclos(No, Comp, Lista):-
    findall(Ciclo, caminho(No, No, [], Ciclo, Comp), Lista).

```



Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação
Programação em Lógica

2003/2004
LEIC
(3º Ano)
1º Sem

Docentes: Luís Paulo Reis e Eugénio da Costa Oliveira

Exercícios – JOGOS E PUZZLES

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.

- Especifique a estrutura de dados que utilizaria em Prolog para a representação de um estado de um tabuleiro de Xadrez.
- 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 numa 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 numa 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" se 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|T1]):- explicacao(E1,E2), nl, escrever([E2|T1]).

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 $N \times N$ 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,[O,C,R,O,S,S],[O,O,R,O,A,D],[D,A,N,G,E,R]).
puzzle(2,[O,S,E,N,D],[O,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, Digits, Digits).
soma([D1|R1], [D2|R2], Cant, Cap, Digits, Ddisp):-
    soma(R1, R2, R, Cant, Cap1, Digits, 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 única 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.