Sistemas de Representação de Conhecimento e Raciocínio

Cláudio Novais

10 de Junho de 2010

Conteúdo

Capítulo 1

Ficha 1: Introdução

Para a resolução da ficha, irão ser considerados os seguintes predicados:

```
filho:-           Filho, Pai -> {V,F}
pai:-             Pai, Filho -> {V,F}
avo:-             Avo, Neto -> {V,F}
bisavo:-          Bisavo, Bisneto -> {V,F}
trisavo:-         Trisavo, Trisneto -> {V,F}
tetravo:-         Tetravo, Tetraneto -> {V,F}
neto:-            Neto, Avo -> {V,F}
sexo:-            Nome, sexo -> {V,F}
descendente:-     Filho, Pai -> {V,F}
graudesc:-        Filho, Pai, Grau -> {V,F}
avodescendencia:- Avo, Neto -> {V,F}
```

1.1 Resolução da ficha

I. O João é filho do José;

```
filho(joao, jose).
```

II. O José é filho do Manuel;

```
filho(jose, manuel).
```

III. O Carlos é filho do José;

```
filho(jose, manuel).
```

IV. O Paulo é pai do Filipe;

```
pai(paulo,filipe).
```

V. O Paulo é pai da Maria;

```
pai(paulo,maria).
```

VI. O António é avô da Nádia;

```
avo(antonio,nadia).
```

VII. O João é do sexo masculino;

```
sexo(joao, masculino).
```

VIII. O José é do sexo masculino;

```
sexo(jose, masculino).
```

IX. A Maria é do sexo feminino;

```
sexo(maria,feminino).
```

X. A Joana é do sexo feminino;

```
sexo(joana,feminino).
```

XI. Construir a extensão de um predicado capaz de determinar que o indivíduo P é pai do indivíduo F se existir uma prova de que F seja filho de P;

```
pai(P,F):-filho(F,P).
```

XII. Construir a extensão de um predicado capaz de determinar que o indivíduo A é avô de N se existir um indivíduo X de quem N seja filho e de quem A seja pai;

```
avo(A,N):- filho(N,P),
           pai(A,P).
```

XIII. Construir a extensão de um predicado capaz de determinar que o indivíduo N é neto do indivíduo A se existir uma prova de que A seja avô de N;

```
neto(N,A):- avo(A,N).
```

XIV. Construir a extensão de um predicado que permita determinar se uma pessoa X descende de outra pessoa Y;

```
descendente(X,Y):- filho(X,Y).
descendente(X,Y):- pai(Y,X).
descendente(X,Y):- filho(X,P),
                   descendente(P,Y).
descendente(X,Y):- pai(Y,F),
                   descendente(X,F).
```

XV. Construir a extensão de um predicado que permita determinar o grau de descendência entre duas pessoas, X e Y;

```
graudesc(X,Y,1):- filho(X,Y).
graudesc(X,Y,1):- pai(Y,X).
graudesc(X,Y,G):- filho(X,P),
                  graudesc(P,Y, N),
                  G is N+1.
graudesc(X,Y,G):- pai(Y,P),
                  graudesc(X,P, N),
                  G is N+1.
```

XVI. Construir a extensão de um predicado capaz de determinar se o indivíduo A é avô de N pela utilização do predicado que determina o grau de descendência entre dois indivíduos;

```
avodescendencia(A,N):- graudesc(N,A,G),
                       G=2.
```

XVII. Construir a extensão de um predicado capaz de determinar se o indivíduo X é bisavô de Y;

```
bisavo(X,Y):- graudesc(Y,X,G),
                       G=3.
```

XVIII. Construir a extensão de um predicado capaz de determinar se o indivíduo X é trisavô de Y;

```
trisavo(X,Y):- graudesc(Y,X,G),
                        G=4.
```

XIX. Construir a extensão de um predicado capaz de determinar se o indivíduo X é tetraneto de Y.

```
tetravo(X,Y):- graudesc(Y,X,G),
                        G=5.
```

1.2 Testes

De seguida, mostra-se alguns testes. De salientar que ao criar o ficheiro com os predicados anteriores, de forma a utilizar o assert, é necessário pôr no topo do ficheiro o seguinte:

```
:-dynamic filho/2.
```

É utilizado o predicado assert para adicionar mais predicados à Base de Conhecimento de forma a tornar os testes mais legíveis. O predicado listing mostra os predicados que o argumento dado.

```
| ?- assert(filho(filho,pai)).
yes
| ?- assert(filho(pai,avo)).
yes
| ?- assert(filho(avo,bisavo)).
yes
| ?- assert(filho(bisavo,trisavo)).
yes
| ?- assert(filho(trisavo,tetravo)).
yes
| ?- listing(filho).
filho(filho, pai).
filho(pai, avo).
filho(avo, bisavo).
filho(bisavo, trisavo).
filho(trisavo, tetravo).

yes
| ?- tetravo(tetravo,filho).
yes
| ?- tetravo(tetravo,pai).
no
| ?- graudesc(pai,trisavo,Grau).
Grau = 3 ?
yes
| ?- descendente(pai,tetravo).
yes
| ?- etetravo(tetravo,Neto).
Neto = filho ?
yes
| ?- pai(tetravo,Filho).
Filho = trisavo ?
yes
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 1
Fevereiro, 2010

Tema

Programação em Lógica.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Construam procedimentos adequados à resolução de problemas, corporizados na extensão de predicados de uma linguagem de programação em lógica;
- Apliquem o Algoritmo de Resolução à demonstração de teoremas;
- Desenvolvam árvores de prova como forma intuitiva e evidente de que um teorema é derivável de um programa.

Enunciado

Utilizando a linguagem de programação em lógica PROLOG, pretende-se que desenvolva a extensão dos predicados que implementam a resolução dos seguintes enunciados:

i. O João é filho do José;
ii. O José é filho do Manuel;
iii. O Carlos é filho do José;
iv. O Paulo é pai do Filipe;
v. O Paulo é pai da Maria;
vi. O António é avô da Nádia;
vii. O João é do sexo masculino;
viii. O José é do sexo masculino;
ix. A Maria é do sexo feminino;
x. A Joana é do sexo feminino;
xi. Construir a extensão de um predicado capaz de determinar que o indivíduo P é pai do indivíduo F se existir uma prova de que F seja filho de P;
xii. Construir a extensão de um predicado capaz de determinar que o indivíduo A é avô de N se existir um indivíduo X de quem N seja filho e de quem A seja pai;
xiii. Construir a extensão de um predicado capaz de determinar que o indivíduo N é neto do indivíduo A se existir uma prova de que A seja avô de N;
xiv. Construir a extensão de um predicado que permita determinar se uma pessoa X descende de outra pessoa Y;
xv. Construir a extensão de um predicado que permita determinar o grau de descendência entre duas pessoas, X e Y;
xvi. Construir a extensão de um predicado capaz de determinar se o indivíduo A é avô de N pela utilização do predicado que determina o grau de descendência entre dois indivíduos;
xvii. Construir a extensão de um predicado capaz de determinar se o indivíduo X é bisavô de Y;
xviii. Construir a extensão de um predicado capaz de determinar se o indivíduo X é trisavô de Y;
xix. Construir a extensão de um predicado capaz de determinar se o indivíduo X é tetraneto de Y.

Para as questões dadas de seguida, enuncie a fórmula lógica que representa essa questão e desenvolva a árvore de prova que ilustra a prova de cada teorema:

xx. O João é filho do José?

xxi. O José é pai do João?

xxii. O João é do sexo masculino?

xxiii. O José é do sexo feminino?

xxiv. Existe alguém que seja filho do José?

xxv. O José é filho do João?

xxvi. O Manuel é avô do José?

xxvii. O Manuel é avô do João?

xxviii. Existe alguém de quem o Carlos seja neto?

xxix. O João é descendente do Manuel?

xxx. Existe algum filho do José que seja descendente do Manuel?

xxxi. Existe algum descendente do Manuel que seja filho do José?

xxxii. Qual o grau de descendência entre o João e o José?

xxxiii. A descendência entre o João e o José é de 2º grau?

xxxiv. Qual o grau de descendência entre o João e o Manuel?

xxxv. A descendência entre o João e o Manuel é maior do que de 2º grau?

Capítulo 2

Ficha 2: Aritmética

Para a resolução da ficha, irão ser considerados os seguintes predicados:

```
soma:          Real,Real, Resultado -> {V,F}
oparit:        Real,Real,Operacao,Resultado -> {V,F}
somatres:      Real,Real,Real,Resultado -> {V,F}
somalista:     Lista,Resultado -> {V,F}
oparitlista:   Lista,Operacao,Resultado -> {V,F}
maior:         Real,Real, Resultado -> {V,F}
menor:         Real,Real, Resultado -> {V,F}
maiorlista:    Lista, Resultado -> {V,F}
menorlista:    Lista, Resultado -> {V,F}
contavazios:   Lista, Resultado -> {V,F}
nao:           Q -> {V,F}
```

2.1 Resolução da ficha

I. Construir a extensão de um predicado que calcule a soma de dois valores;

```
soma(X,Y,Z):- Z is X + Y.
```

II. Construir a extensão de um predicado que aplique uma operação aritmética (adição, subtracção, multiplicacao, divisão) a dois valores;

```
oparit(X,Y,+,Z):- Z is X + Y.
oparit(X,Y,-,Z):- Z is X - Y.
oparit(X,Y,*,Z):- Z is X * Y.
oparit(X,Y,/,Z):- Z is X / Y.
```

III. Construir a extensão de um predicado que calcule a soma de três valores;

```
somatres(X,Y,Z,R):- R is +(X,+(Y,Z)).

%Ou, alternativamente:
somatresb(X,Y,Z,R):- soma(X,Y,P),
                     soma(P,Z,R).
```

IV. Construir a extensão de um predicado que calcule a soma de um conjunto de valores;

```
somalista([],0).
somalista([H|T],X):- somalista(T,R),
                     X is R+H.
```

V. Construir a extensão de um predicado que aplique uma operação aritmética (adição, subtracção, multiplicacao, divisão) a um conjunto de valores;

```
oparitlista([H|T],+,Z):- oparitlista(T,+,R),
                         Z is R+H.
oparitlista([H|T],-,Z):- oparitlista(T,-,R),
                         Z is R-H.
oparitlista([H|T],*,Z):- oparitlista(T,*,R),
                         Z is R*H.
oparitlista([H|T],/,Z):- oparitlista(T,/,R),
                         Z is R/H.
```

VI. Construir a extensão de um predicado que calcule o maior valor entre dois números;

```
% A solucao seguinte nao podia ser, pois em certos casos de backtracking, ao tentar a segunda↩
      hipotese, falha.
maior2(X,Y,X):- X>Y.
maior2(X,Y,Y).

% Como alternativa, poder-se-ia utilizar o bang (?), no entanto, fazendo uma tabela de ↩
     verdade, das varias hipoteses conclui-se que nao equivalente a solucao optima.
% Solucao optima:
maior2(X,Y,X):- X>Y,!.
maior2(X,Y,Y).

% Solucao optima:
maior(X,Y,X):- X>Y.
maior(X,Y,Y):- X=<Y.
```

VII. Construir a extensão de um predicado que calcule o maior de um conjunto de valores;

```
maiorlista([X],X).
maiorlista([H|T],X):- maiorlista(T,R),
                      maior(H,R,X).
```

VIII. Construir a extensão de um predicado que calcule o menor valor entre dois números;

```
menor(X,Y,Y):- X>Y.
menor(X,Y,X):- X=<Y.
```

IX. Construir a extensão de um predicado que calcule o menor de um conjunto de valores;

```
menorlista([X],X).
menorlista([H|T],X):- menorlista(T,R),
                      menor(H,R,X).
```

X. Construir a extensão de um predicado que contabilize a quantidade de conjuntos vazios que existem num determinado conjunto de ocorrências;

```
contavazios([], 0).
contavazios([[] | T], R) :- contavazios(T, X), R is X + 1.
contavazios([ H | T], R) :- contavazios(T, R).
```

XI. Construir a extensão de um predicado que calcule o valor de verdade contrário à resposta a uma questão.

```
nao(Q):-Q,!,fail.
nao(Q).
```

2.2 Testes

```
| ?- soma(2,3,R).
R = 5 ?
yes
| ?- menor(3,5,R).
R = 3 ?
yes
| ?- maior(3,5,R).
R = 5 ?
yes
| ?- maiorlista([3,5,7,2,1,5,3],R).
R = 7 ?
yes
| ?- menorlista([3,5,7,2,1,5,3],R).
R = 1 ?
yes
| ?- menorlista([3,5,7,2],R).
R = 2 ?
yes
| ?- contavazios([2,3,[],4,[],[],5,[],6],R).
R = 4 ?
yes
| ?- nao(contavazios([2,3,[],4,[],[],5,[],6],R)).
no
| ?- soma(2,3,3).
no
| ?- nao(soma(2,3,3)).
yes
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 2
Fevereiro, 2010

Tema

Programação em Lógica.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Construam procedimentos adequados à resolução de problemas, corporizados na extensão de predicados de uma linguagem de programação em lógica, envolvendo a manipulação de valores aritméticos e de conjuntos de dados;
- Utilizem e definam a extensão de predicados e de meta-predicados.

Enunciado

Utilizando a linguagem de programação em lógica PROLOG, pretende-se que desenvolva a extensão dos predicados que implementam a resolução dos seguintes enunciados:

i. Construir a extensão de um predicado que calcule a soma de dois valores;
ii. Construir a extensão de um predicado que aplique uma operação aritmética (adição, subtracção, multiplicacao, divisão) a dois valores;
iii. Construir a extensão de um predicado que calcule a soma de três valores;
iv. Construir a extensão de um predicado que calcule a soma de um conjunto de valores;
v. Construir a extensão de um predicado que aplique uma operação aritmética (adição, subtracção, multiplicacao, divisão) a um conjunto de valores;
vi. Construir a extensão de um predicado que calcule o maior valor entre dois números;
vii. Construir a extensão de um predicado que calcule o maior de um conjunto de valores;
viii. Construir a extensão de um predicado que calcule o menor valor entre dois números;
ix. Construir a extensão de um predicado que calcule o menor de um conjunto de valores;
x. Construir a extensão de um predicado que contabilize a quantidade de conjuntos vazios que existem num determinado conjunto de ocorrências;
xi. Construir a extensão de um predicado que calcule o valor de verdade contrário à resposta a uma questão.

Para as questões dadas de seguida, enuncie a fórmula lógica que representa essa questão e desenvolva a árvore de prova que ilustra a prova de cada teorema:

xii. Qual é a soma entre 1 e 3?

xiii. Qual é a soma entre 1, 3 e 5?

xiv. Qual é o resultado de se aplicar a multiplicação aos valores 2 e 4?

xv. Qual é o resultado de se aplicar a adição ao conjunto de valores 5, 3 e 1?

xvi. Qual é o maior valor entre 1 e 3?

xvii. Qual é o maior valor entre 3 e 1?

xviii. O maior valor entre 3 e 1 é menor do que 2?

xix. Qual é o maior valor do conjunto formado pelos números 5, 3 e 7?

xx. Qual é o menor valor do conjunto formado pelos números 2,4 e 6?

Capítulo 3

Ficha 3: Listas

Para a resolução da ficha, irão ser considerados os seguintes predicados:

```
soma:          Real,Real, Resultado -> {V,F}
nao:           Q -> {V,F}
pertence:      Elemento, Lista, Resultado -> {V,F}
comprimento:   Lista, Resultado -> {V,F}
quantos:       Elemento, Lista, Resultado -> {V,F}
apagar:        Elem, Lista, Resultado -> {V,F}
apagartudo:    Elem, Lista, Resultado -> {V,F}
adicionar:     Elem, Lista, Resultado -> {V,F}
concatenar:    Lista1, Lista2, Resultado -> {V,F}
inverter:      Lista, Resultado -> {V,F}
sublista:      Sublista, Lista -> {V,F}
Solucoes:      Termo, Questao, Conjunto de solucoes -> {V,F}
Construir:     Conjunto, Lista -> {V,F}
```

3.1 Resolução da ficha

I. Construir a extensão de um predicado que calcule o maior valor entre dois números;

```
soma(X,Y,Z):- Z is X + Y.
```

II. Construir a extensão do predicado que implementa a negação por falha na prova;

```
nao(Q):- Q,
         !,
         fail.
nao(Q).
```

III. Construir a extensão do predicado «pertence» que verifica se um elemento existe dentro de uma lista de elementos;

```
pertence(E, [E|_]).
pertence(E, [_|T]) :- pertence(E, T).
```

IV. Construir a extensão do predicado «comprimento», capaz de calcular o número de elementos existentes numa lista;

```
comprimento([], 0).
comprimento([_|T], R) :- comprimento(T, X),
                         R is X + 1.
```

V. Construir a extensão do predicado «quantos», capaz de calcular a quantidade de elementos diferentes que existem numa lista;

```
quantos(E,[], 0).
quantos(E,[E|T], R) :- quantos(E,T, X),
                       R is X + 1.
quantos(E,[_|T], R) :- quantos(E,T, R).
```

VI. Construir a extensão do predicado «apagar», que apaga a primeira ocorrência de um elemento de uma lista;

```
apagar(_, [], []).
apagar(E, [E|T], T).
apagar(E, [H|T], [H|L]) :- apagar(E, T, L).
```

VII. Construir a extensão do predicado «apagartudo», de modo a apagar todas as ocorrências de um elemento numa lista;

```
apagartudo(_, [], []).
apagartudo(E, [E|T], L):- apagartudo(E, T, L).
apagartudo(E, [H|T], [H|L]) :- apagartudo(E, T, L).
```

VIII. Construir a extensão do predicado «adicionar», capaz de inserir um elemento numa lista, sem repetidos;

```
adicionar(E, [], [E]).
adicionar(E, [E|T], [E|T]).
adicionar(E, [H|T], [H|L]) :- adicionar(E, T, L).
```

IX. Construir a extensão do predicado «concatenar», que resulta na concatenação dos elementos da lista L1 com os elementos da lista L2;

```
concatenar([],L2,L2).
concatenar([H|L1],L2,[H|T]):- concatenar(L1,L2,T).
```

X. Construir a extensão do predicado «inverter», que inverte a ordem dos elementos de uma lista;

```
inverter([],[]).
inverter([H|T],R):- inverter(T,X),
                    concatenar(X,[H],R).
```

XI. Construir a extensão do predicado «sublista», que deve determinar se uma lista S é considerada uma sublista de uma outra lista L;

```
sublista([],_).
sublista([H|S],[H|L]):- sublista(S,L).
sublista(S,[X|L]):- sublista(S,L).

% Alternativa:
sublista2(S,L):- concatenar(L1,L2,L),
                 concatenar(S,L3,L2).
```

XII. Construir a extensão de um predicado capaz de encontrar todas as possibilidades de prova de um teorema.

```
solucoes( T, Q, S):- Q,
                     assert(temp(T)),
                     fail.
solucoes( T, Q, S):- construir(S,[]).

construir( S, Li ):- retract(temp(X)),
                     construir(S,[X|Li]).
construir(S,S).
```

3.2 Testes

```
| ?- comprimento([3,4,5,6,2,4,5],R).
R = 7 ?
yes
| ?- quantos(3,[4,5,3,1,3,5,3,2,3],R).
R = 4 ?
yes
| ?- apagar(3,[2,4,3,5,3],R).
R = [2,4,5,3] ?
yes
| ?- apagartudo(3,[2,4,3,5,3],R).
R = [2,4,5] ?
yes
| ?- adicionar(3,[2,3,4],R).
R = [2,3,4] ?
yes
| ?- adicionar(3,[4,5,6],R).
R = [4,5,6,3] ?
yes
| ?- concatenar([1,2,3],[4,5,6],R).
R = [1,2,3,4,5,6] ?
yes
| ?- inverter([2,3,4,5,6,7,8,9],R).
R = [9,8,7,6,5,4,3,2] ?
yes
| ?- sublista([2,3],[1,2,2,3,4]).
yes
| ?- assert(filho(joao,jose)).
yes
| ?- assert(filho(jose,manuel)).
yes
| ?- assert(filho(carlos,jose)).
yes
| ?- listing(filho).
filho(joao, jose).
filho(jose, manuel).
filho(carlos, jose).
| ?- solucoes( X, filho(X,jose), S).
S = [carlos,joao] ?
yes
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 3
Fevereiro, 2010

Tema

Programação em Lógica.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Construam procedimentos adequados à resolução de problemas que envolvem a manipulação de termos complexos como, por exemplo, listas ou estruturas;
- Utilizem e definam a extensão de predicados e de meta-predicados;
- Desenvolvam procedimentos de procura de todas as possibilidades de prova de um teorema.

Enunciado

Utilizando a linguagem de programação em lógica PROLOG, pretende-se que desenvolva a extensão dos predicados que implementam a resolução dos seguintes enunciados:

i. Construir a extensão de um predicado que calcule o maior valor entre dois números;
ii. Construir a extensão do predicado que implementa a negação por falha na prova;
iii. Construir a extensão do predicado «pertence» que verifica se um elemento existe dentro de uma lista de elementos;
iv. Construir a extensão do predicado «comprimento», capaz de calcular o número de elementos existentes numa lista;
v. Construir a extensão do predicado «quantos», capaz de calcular a quantidade de elementos diferentes que existem numa lista;
vi. Construir a extensão do predicado «apagar», que apaga a primeira ocorrência de um elemento de uma lista;
vii. Construir a extensão do predicado «apagartudo», de modo a apagar todas as ocorrências de um elemento numa lista;
viii. Construir a extensão do predicado «adicionar», capaz de inserir um elemento numa lista, sem repetidos;
ix. Construir a extensão do predicado «concatenar», que resulta na concatenação dos elementos da lista L1 com os elementos da lista L2;
x. Construir a extensão do predicado «inverter», que inverte a ordem dos elementos de uma lista;
xi. Construir a extensão do predicado «sublista», que deve determinar se uma lista S é considerada uma sublista de uma outra lista L;
xii. Construir a extensão de um predicado capaz de encontrar todas as possibilidades de prova de um teorema.

Para as questões dadas de seguida, enuncie a fórmula lógica que representa essa questão e desenvolva a árvore de prova que ilustra a prova de cada teorema:

xiii. Qual é o maior valor entre 1 e 3?
xiv. Qual é o maior valor entre 3 e 1?
xv. O maior valor entre 3 e 1 é menor do que 2?
xvi. É verdade que o João não é filho do Manuel?
xvii. É verdade que o João não é filho do José?
xviii. É verdade que o João não é descendente do José?
xix. É verdade que o João não é descendente de 1º grau do José?
xx. O elemento 'b' pertence à lista de elementos [a,b,c]?
xxi. O elemento '1' pertence à lista de elementos [a,b,c]?
xxii. Algum dos elementos pertencentes à lista [a,b,c] é o elemento 'b'?
xxiii. Qual é o comprimento de uma lista vazia de elementos?
xxiv. Qual é o resultado de apagar o elemento '2' da lista [a,b,c]?
xxv. Qual é o resultado de apagar todos os elementos '2' da lista [a,b,c]?
xxvi. A concatenação da lista [1,2] com a lista [a,b,c] é [1,2,a,b,c]?
xxvii. A lista [2,3] é uma sublista de [1,2,3,4,5]?
xxviii. A lista [3,2] é uma sublista de [1,2,3,4,5]?
xxix. A lista [2,4] é uma sublista de [1,2,3,4,5]?
xxx. Existe alguém que seja filho do José?
xxxi. Quem são os filhos do José?
xxxii. Quantos são os filhos do José?
xxxiii. Quem são os filhos do Manuel?
xxxiv. O Manuel tem mais de 2 filhos?
xxxv. Quem são os filhos do João?
xxxvi. O João tem filhos?

Capítulo 4

Ficha 4: Invariantes

Para a resolução da ficha, irão ser considerados os seguintes predicados:

```
filho:          Filho, Pai -> {V,F}
pai:            Pai, Filho -> {V,F}
neto:           Neto, Avo -> {V,F}
avo:            Avo, Neto -> {V,F}
descendente:    Descendente, Ascendente, Grau -> {V,F}
```

Relativamente às flags, serão utilizadas as seguintes:

```
:-set_prolog_flag(discontiguous_warnings,off).
:-set_prolog_flag(single_var_warnings,off).
:-set_prolog_flag(unknown,fail).
```

De forma a ser possível mudar a base de conhecimento, utilizando os assert's e retract's, é necessário adicionar as seguintes definições iniciais ao prolog:

```
:-op(900,xfy,'::').
:-dynamic filho/2.
:-dynamic pai/2.
:-dynamic neto/2.
:-dynamic avo/2.
:-dynamic descendente/3.
:-dynamic temp/1.
```

Esta ficha tem como objectivo criar invariantes que depois sejam utilizados no predicado evolução, descrito a seguir:

```
evolucao(Termo) :- solucoes(Invariante,+Termo::Invariante,Lista),
                   inserir(Termo),
                   teste(Lista).

inserir(Termo):- assert(Termo).
inserir(Termo):- retract(Termo), !, fail.

teste([]).
teste( [R|LR] ):- R, teste( LR ).

comprimento([],0).
comprimento([X|L],R):- comprimento(L,N), R is N+1.

solucoes(X,Y,Z):- findall(X,Y,Z).

%Infelizmente, o solucoes2, explicado pelo professor, e que tem exactamente o mesmo output que o ↩
     solucoes de cima, nao funciona direito no evolucao.
solucoes2( T, Q, S):- Q,
                      assert(temp(T)),
                      fail.
solucoes2( T, Q, S):- construir(S,[]).

construir( S, Li ):- retract(temp(X)),
                     construir(S,[X|Li]).
construir(S,S).
```

4.1 Resolução da ficha

I. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação filho/2;

```
+filho(F,P)::(solucoes( (F,P), ( filho(F,P) ), S),
              comprimento(S,N),
              N==1).
```

II. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação pai/2;

```
+pai(P,F)::(solucoes( (P,F), ( pai(P,F) ), S),
            comprimento(S,N),
            N==1).
```

III. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação neto/2;

```
+neto(N,A)::(solucoes( (N,A), ( neto(N,A) ), S),
             comprimento(S,C),
             C==1).
```

IV. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação avô/2;

```
+avo(A,N)::(solucoes( (A,N), ( neto(A,N) ), S),
            comprimento(S,C),
            C==1).
```

V. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação descendente/3;

```
+descendente(X,Y,G)::(solucoes( (X,Y,G), ( descendente(X,Y,G) ), S),
                      comprimento(S,C),
                      C==1).
```

VI. Não é admissível existirem mais do que 2 progenitores para um determinado indivíduo, na relação filho/2;

```
+filho(F,P)::(solucoes( Ps, ( filho(F,Ps) ), S),
              comprimento(S,N),
              N=<2).
```

VII. Não é admissível existirem mais do que 2 progenitores para um determinado indivíduo, na relação pai/2;

```
+pai(P,F)::(solucoes( Ps, ( pai(Ps,F) ), S),
            comprimento(S,N),
            N=<2).
```

VIII. Não poderá acontecer existirem mais do que 4 indivíduos identificados como «avô» na relação neto/2;

```
+neto(N,A)::(solucoes( As, ( neto(N,As) ), S),
             comprimento(S,C),
             C=<4).
```

IX. Não poderá acontecer existirem mais do que 4 indivíduos identificados como «avô» na relação avô/2;

```
+avo(A,N)::(solucoes( As, ( avo(As,N) ), S),
            comprimento(S,C),
            C=<4).
```

X. A identificação do «grau» de descendência na relação descendente/3 deverá pertencer ao conjunto dos números naturais N.

```
+descendente(X,Y,G)::(number(G)).
```

4.2 Testes

```
| ?- evolucao(pai(joao,carlos)).
yes
| ?- evolucao(pai(joao,carlos)).
no
| ?- evolucao(pai(joao,carlos)).
no
| ?- listing(pai).
pai(joao, carlos).

yes
| ?- evolucao(descendente(afonso,claudio,40)).
yes
| ?- evolucao(descendente(afonso,claudio,40)).
no
| ?- evolucao(descendente(afonsinho,ze,g)).
no
| ?- evolucao(filho(carlos,joao)).
yes
| ?- evolucao(filho(carlos,joao)).
no
| ?- evolucao(filho(carlos,maria)).
yes
| ?- evolucao(filho(carlos,eduardo)).
no
| ?- evolucao(neto(andre,antonio)).
yes
| ?- evolucao(neto(andre,maria)).
yes
| ?- evolucao(neto(andre,manuel)).
yes
| ?- evolucao(neto(andre,josefina)).
yes
| ?- evolucao(neto(andre,edmundo)).
no
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 4
Março, 2010

Tema

Invariantes Estruturais e Referenciais.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Desenvolvam procedimentos adequados à resolução de problemas, corporizados na extensão de predicados de uma linguagem de programação em lógica;
- Utilizem e apliquem invariantes estruturais e referenciais para a manutenção da verdade;
- Construam invariantes.

Enunciado

Utilizando a linguagem de programação em lógica PROLOG, e para um sistema de representação de conhecimento e de raciocínio onde estão presentes os predicados:

filho: Filho, Pai $\rightarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

pai: Pai, Filho $\rightarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

neto: Neto, Avô $\rightarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

avô: Avô, Neto $\rightarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

descendente: Descendente, Ascendente, Grau $\rightarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

entre outros, pretende-se que desenvolva os invariantes que descrevam os significados que se enunciam de seguida:

i. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação filho/2;
ii. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação pai/2;
iii. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação neto/2;
iv. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação avô/2;
v. Não se admitirá a existência de mais do que uma ocorrência da mesma evidência na relação descendente/3;
vi. Não é admissível existirem mais do que 2 progenitores para um determinado indivíduo, na relação filho/2;
vii. Não é admissível existirem mais do que 2 progenitores para um determinado indivíduo, na relação pai/2;
viii. Não poderá acontecer existirem mais do que 4 indivíduos identificados como «avô» na relação neto/2;
ix. Não poderá acontecer existirem mais do que 4 indivíduos identificados como «avô» na relação avô/2;
x. A identificação do «grau» de descendência na relação descendente/3 deverá pertencer ao conjunto dos números naturais $\mathbb{N}$.

Capítulo 5

Ficha 5: Negação forte

Neste capítulo é introduzido o conceito de negação forte que altera o domínio dos predicados até agora utilizados. Neste capítulo a novidade está na adição do "desconhecido"que outrora não existia. Antes, toda a informação que não fosse mensionada era falsa. Agora, para determinada informação ser falsa, é necessário que haja uma prova que demonstre a sua falsidade. Tudo o resto que não pode ser provado como verdadeiro ou falso, é desconhecido.

De um modo conceptual, a interpretação de questões (prova de teoremas), será da forma:

```
Tsim: solucoes(X, q(X), S), comp(S,N), N >= 1
Tnao: solucoes(X, -q(X), S), comp(S,N), N =< 1
Ttalvez: solucoes(X, q(X) ou -q(X), S), comp(S,N), N=0
```

Em PROLOG, a interpretação das questões será feita através do predicado "demo"que tem a forma seguinte:

```
demo(Questao, verdadeiro):-
     Questao.
demo(Questao, falso):-
     -Questao.
demo(Questao, desconhecido):-
     nao(Questao).
     nao(-Questao).
```

Relativamente à negação por falha na prova, não será considerada na resolução da ficha pois basta pôr a prova verdadeira dentro do predicado "nao".

Relativamente ao novo método de definição do que é falso, usando o hífen como símbolo da negação, é necessário adicionar o seguinte ao cabeçalho do ficheiro PROLOG para ser possível adicionar conhecimento:

```
:-dynamic '-'/1.
```

Para a resolução da ficha, irão ser considerados os seguintes predicados:

```
par:          X       -> {V,F,D}
impar:        X       -> {V,F,D}
natural:      X       -> {V,F,D}
inteiro:      X       -> {V,F,D}
arcoiris:     Cor     -> {V,F,D}
corvitoria:   Cor     -> {V,F,D}
atravessar:   Caminho -> {V,F,D}
nodoterminal: Nodo    -> {V,F,D}
```

5.1 Resolução da ficha

I. Construa a extensão de um predicado capaz de caracterizar os números pares.

```
par(0).
par(X):- Y is X-2,
         Y>=0,
         par(Y).

-par(X):- nao( par(X) ).
```

II. Construa a extensão de um predicado capaz de caracterizar os números ímpares.

```
impar(1).
impar(X):- Y is X-2,
           Y>=0,
           impar(Y).

-impar(X):- nao( impar(X) ).

%alternativamente, como temos o predicado "par" definido, podia-se fazer o seguinte:
impar2(X):- -par(X).
-impar2(X):- par(X).
```

III. Construa a extensão de um predicado que caracterize o conjunto dos números naturais.

```
natural(1).
natural(X):-  Y is X-1,
              Y>=1,
              natural(Y).

-natural(X):- nao(natural(X)).
```

IV. Construa a extensão de um predicado que caracterize o conjunto dos números inteiros.

```
inteiro(0).
inteiro(X):-  Y is X-1,
              Y >= 0,
              inteiro(Y).
inteiro(X):-  Y is X+1,
              Y =< 0,
              inteiro(Y).

-inteiro(X):- nao(inteiro(X)).
```

V. Construa a extensão de um predicado capaz de caracterizar as cores do arco-iris.

```
% Uma das formas de caracterizar as cores do arco-iris passa por definir cada cor para um ↩
    determinado predicado e definir tambem as cores que nao sao para as provas falsas. Esta ↩
    resposta foi baseada no artigo da wikipedia: http://pt.wikipedia.org/wiki/Arco-%C3%ADris

arcoiris(vermelho).
arcoiris(laranja).
arcoiris(amarelo).
arcoiris(verde).
arcoiris(azul).
arcoiris(anil).
arcoiris(violeta).
-arcoiris(cinzento).
-arcoiris(preto).
-arcoiris(branco).
```

VI. Construa a extensão de um predicado com capacidade para identificar as cores dos equipamentos oficial e alternativos do Vitória.

```
corvitoria(branco).
corvitoria(preto).
-corvitoria(X).
```

VII. Construa um programa capaz de representar a autorização de atravessar a estrada baseado na não existência de automóveis em aproximação, e, ainda, a autorização de atravessar o caminho-de-ferro pela confirmação da inexistência de um comboio em aproximação.

```
atravessar(estrada):- nao(vem_automovel).
atravessar(caminho_de_ferro):- -vem_comboio.
```

VIII. Tendo em consideração o grafo ilustrado na Figura 1, desenvolva um programa com a capacidade para definir o significado de «nodo terminal», baseado na descrição de grafos pela definição de nodos e de arcos.

```
% Este predicado, inicialmente procura todas as ligacoes para fora que o nodo em questao tem.↩
      No caso de nao ter nenhuma ligacao, considera-se <nodo terminal>.
% Nao sei se sera este o conceito de nodo terminal.
nodoterminal(X):- solucoes(Y, ligacao(X, Y), Ln),
                  comprimento(Ln,N),
                  N==0.

ligacao(b,a).
ligacao(b,c).
ligacao(c,a).
ligacao(c,d).
ligacao(f,g).
```

5.2 Testes

```
| ?- demo(par(6),X).
X = verdade ?
yes
| ?- demo(par(5),X).
X = falso ?
yes
| ?- demo(par(-3),X).
X = desconhecido ?
yes
| ?- demo(inteiro(6),R).
R = verdade ?
yes
| ?- demo(inteiro(-4),R).
R = verdade ?
yes
| ?- demo(arcoiris(vermelho),R).
R = verdade ?
yes
| ?- demo(arcoiris(laranja_claro),R).
R = desconhecido ?
yes
| ?- demo(arcoiris(preto),R).
R = falso ?
yes
| ?- demo(corvitoria(branco),R).
R = verdade ?
yes
| ?- demo(corvitoria(vermelho),R).
R = falso ?
yes
| ?- atravessar(estrada).
yes
| ?- atravessar(caminho_de_ferro).
no
| ?- assert(-vem_comboio).
yes
| ?- atravessar(caminho_de_ferro).
yes
| ?- nodoterminal(b).
no
| ?- nodoterminal(a).
yes
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 5
Março, 2010

Tema

Extensão à Programação em Lógica.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Distingam e utilizem as duas formas de negação presentes numa extensão à programação em lógica;
- Apliquem o pressuposto do mundo fechado à extensão de predicados;
- Construam a extensão de predicados no contexto desta extensão à programação em lógica;
- Utilizem meta-predicados para a construção de mecanismos de raciocínio adequados às problemáticas específicas de um sistema ou ambiente de programação.

Enunciado

Utilizando a linguagem de programação em lógica PROLOG, pretende-se que desenvolva um sistema com capacidade para utilizar as duas formas de negação – negação forte e negação por falha na prova – no sentido de desenvolver a extensão de predicados capazes de implementar a resolução dos seguintes enunciados:

i. Construa a extensão de um predicado capaz de caracterizar os números pares.
ii. Construa a extensão de um predicado capaz de caracterizar os números ímpares.
iii. Construa a extensão de um predicado que caracterize o conjunto dos números naturais.
iv. Construa a extensão de um predicado que caracterize o conjunto dos números inteiros.
v. Construa a extensão de um predicado capaz de caracterizar as cores do arco-iris.
vi. Construa a extensão de um predicado com capacidade para identificar as cores dos equipamentos oficial e alternativos do Vitória.
vii. Construa um programa capaz de representar a autorização de atravessar a estrada baseado na não existência de automóveis em aproximação, e, ainda, a autorização de atravessar o caminho-de-ferro pela confirmação da inexistência de um comboio em aproximação.
viii. Tendo em consideração o grafo ilustrado na Figura 1, desenvolva um programa com a capacidade para definir o significado de «nodo terminal», baseado na descrição de grafos pela definição de nodos e de arcos.

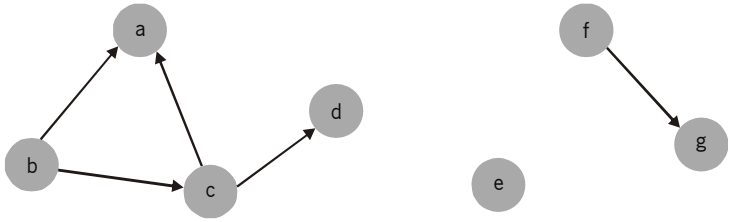

Figura 1
Grafo dirigido.

Deverá ser desenvolvido, ainda, o sistema de inferência, na forma de um meta-predicado, capaz de dar corpo ao mecanismo de raciocínio subjacente ao contexto desta extensão à programação em lógica.

Capítulo 6

Ficha 6: Excepções e NF

A ficha 6 tem uma particularidade extra em relação à ficha 5: adição de excepções às questões. Como exemplo das excepções tem-se as avestruzes que apesar de serem aves não voam e voar é uma característica normal das aves. O mesmo acontece com as galinhas e pinguins que também são aves e não voam ou certos mamíferos que, por o serem não são considerados aves, no entanto, têm uma característica das aves: voar (por exemplo, os morcegos são mamíferos e voam).

Assim, para resolver estas excepções, deverão sempre ser consideradas nos predicados caso seja necessário (ver resolução da primeira questão). A definição das excepções deverão ser feitas da seguinte forma:

```
excepcao: termo/predicado -> {V,F,D}
```

Para a resolução da ficha, irão ser considerados os seguintes predicados:

```
voa:      X -> {V,F,D}
ave:      X -> {V,F,D}
mamifero: X -> {V,F,D}
```

6.1 Resolução da ficha

I. Se a entidade X é uma ave então a característica do voo está presente nessa entidade;

```
%  A resposta correcta a esta questao em particular seria atraves do predicado "voa2", no ↩
    entanto,o predicado desta questao deve ter em conta excepcoes que existem na Classe Ave,↩
    nomeadamente aves que nao voam. Por isso, o predicado correcto tem de ser o "voa"
voa(X):- ave(X),
         nao( excepcao( voa(X) ) ).

voa2(X):- ave(X).
```

II. Se X é um mamífero então a característica do voo não está presente em X;

```
-voa(X):- mamifero(X).
```

III. O Tweety não voa;

```
-voa(tweety).
```

IV. O Pitiguí é uma ave;

```
ave(pitigui).
```

V. Os canários são aves;

```
ave(X):- canario(X).
```

VI. Os periquitos são aves;

```
ave(X):- piriquito(X).
```

VII. O Piupiu é um canário;

```
canario(piupiu).
```

VIII. O Silvestre é um mamífero;

```
mamifero(silvestre).
```

IX. Os cães são mamíferos;

```
mamifero(X):- cao(X).
```

X. Os gatos são mamíferos;

```
mamifero(X):- gato(X).
```

XI. O Boby é um cão;

```
cao(boby).
```

XII. As avestruzes são aves;

```
% nesta questao foi necessario evoluir o predicado "-voa" de forma a poder ser possivel ↩
    provar que a avestruz nao voa
ave(X):- avestruz(X).
excepcao( voa(X) ):- avestruz(X).
-voa(X):- excepcao( voa(X) ).
```

XIII. Os pinguins são aves;

```
ave(X):- pinguim(X).
```

XIV. A Trux é uma avestruz;

```
avestruz(trux).
```

XV. O Pingú é um pinguim.

```
pinguim(pingu).
```

XVI. Os morcegos são mamíferos.

```
% neste caso foi necessario evoluir o predicado "voa" de forma a atender as excecpoes dos ↩
    animais que voam e nao sao aves
mamifero(X):- morcego(X).
excepcao( -voa(X) ):- morcego(X).
voa(X):- excepcao( -voa(X) ).
```

XVII. O Bateméne é um morcego.

```
morcego(batemene).
```

6.2 Testes

```
| ?- demo(ave(piupiu),R).
R = verdade ?
yes
| ?- demo(voa(tweety),R).
R = falso ?
yes
| ?- demo(voa(trux),R).
R = falso ?
yes
| ?- demo(voa(piupiu),R).
R = verdade ?
yes
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 6
Março, 2010

Tema

Extensão à Programação em Lógica.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Distingam e utilizem as duas formas de negação presentes numa extensão à programação em lógica;
- Apliquem o pressuposto do mundo fechado à extensão de predicados;
- Construam a extensão de predicados no contexto desta extensão à programação em lógica.

Enunciado

Utilizando a linguagem de programação em lógica PROLOG e tendo desenvolvido o sistema de inferência capaz de dar corpo ao mecanismo de raciocínio subjacente à extensão à programação em lógica, pretende-se que desenvolva a resolução dos seguintes problemas:

i. Se a entidade X é uma ave então a característica do voo está presente nessa entidade;
ii. Se X é um mamífero então a característica do voo não está presente em X;
iii. O Tweety não voa;
iv. O Pitiguí é uma ave;
v. Os canários são aves;
vi. Os periquitos são aves;
vii. O Piupiu é um canário;
viii. O Silvestre é um mamífero;
ix. Os cães são mamíferos;
x. Os gatos são mamíferos;
xi. O Boby é um cão;
xii. As avestruzes são aves;
xiii. Os pinguins são aves;
xiv. A Trux é uma avestruz;
xv. O Pingú é um pinguim.
xvi. Os morcegos são mamíferos.
xvii. O Bateméne é um morcego.

Considere a possibilidade de descrever excepções às conclusões de um predicado, como, por exemplo, as avestruzes são excepções às conclusões do predicado que determina o voo de uma qualquer entidade.

Capítulo 7

Ficha 7: R. Conhecimento Imperfeito

7.1 Extensão à programação em lógica

- **Adoptar a negação forte:** ¬
- **Abandono do PMF** (Pressuposto do Mundo Fechado)
- **Conhecimento imperfeito**:
  - **Conhecimento incerto** (Não se conhece o contradomínio);

```
excepcao(filho(F,P):- filho(F,alguem).
```

  - **Conhecimento impreciso** (Conhece-se o contradomínio, mas não o valor exacto, por exemplo, o nome do pai de alguém ou é X ou Y);

```
excepcao( filho(maria, faria) ).
excepcao( filho(maria, garcia) ).
```

  - **Conhecimento interdito** (Nunca é conhecido o valor real);

```
excepcao(filho(F,P):- filho(bebe,P).
```

- **Valor nulo**: constante que não representa um valor do contradomínio. É utilizado para indicar algo diferente, Por exemplo "alguém".

Tendo em conta o descrito anteriormente, a prova para a negação do predicado filho será:

```
-filho(F,P):-  nao( filho(F,P) ),
               nao( excepcao(filho(F,P)) ).
```

Como interpretador das questões, utiliza-se o mesmo "demo"das fichas anteriores:

```
demo( filho( belem, xico ), R ).
```

7.2 Pré-Requisitos da ficha

Relativamente à evolução do conhecimento, com base no descrito na segunda página da ficha prática, é necessário criar invariantes que controlem o sistema de gestão e nomeação de árbitros:

```
% jogo: ID, Arbitro, Custo -> {V,F,D}
-jogo(ID,Nome,Valor):- nao(jogo(ID,Nome,Valor)),
                       nao(excepcao(jogo(ID,Nome,Valor))).

% excepcao: P -> {V,F,D}
excepcao( jogo(ID,Nome,Valor) ):- jogo( ID, Nome, desconhecido ).

nulo(interdito).
```

Para além disso, será necessário um interpretador de questões:

```
demo(Q, verdade):- Q.
demo(Q, falso):- -Q.
demo(Q, desconhecido):- nao(Q), nao(-Q).

nao(Q):- Q, !, fail.
nao(Q).
```

7.3 Resolução da ficha

I. O árbitro Almeida Antunes apitou o primeiro jogo do campeonato, no qual recebeu 500€ como ajudas de custo;

```
jogo(1,almeidaantunes,500).
```

II. O árbitro Baltazar Borges apitou o segundo jogo, tendo recebido a título de ajudas de custo um valor que ainda ninguém conhece;

```
jogo(2,baltazarborges,desconhecido).
```

III. Consta na ficha de jogo da terceira partida, que o árbitro Costa Carvalho recebeu 500€, mas a comunicação social alega ter-lhe sido pago mais 2.000€ (como compensação por danos no seu veículo); instado a pronunciar-se sobre o assunto, o árbitro não confirma nem desmente nenhum dos valores noticiados;

```
excepcao(jogo(3,costacarvalho,500)).
excepcao(jogo(3,costacarvalho,2500)).
```

IV. O árbitro Duarte Durão apitou o quarto jogo, tendo recebido como ajudas de custo um valor que ronda os 250€ a 750€, desconhecendo-se qual a quantia exacta;

```
excepcao(jogo(4,duartedurao,X)):- X=<750, X>=250.
```

V. No quinto jogo apitado pelo árbitro Edgar Esteves, ocorreram tumultos no final do encontro tendo desaparecido as ajudas de custo da carteira do árbitro, pelo que se torna impossível vir-se a conhecer esse valor;

```
% visto que vai ser impossivel determinar o valor, entao o conhecimento eh interdito e por ↩
    isso mesmo eh necessario adicionar um invariante.
excepcao(jogo(5,edgaresteves,interdito)).
+jogo(N,Nome,V)::(solucoes(V, (jogo(5,edgaresteves,V), nao(nulo(V)) ),[])).
```

VI. O árbitro do sexto jogo, Francisco França recebeu, como ajudas de custo, o valor de 250€; no entanto, entre amigos, refere ter "encaixado" nesse dia para cima de 5.000€;

```
jogo(6,franciscofranca,250).
excepcao(jogo(6,franciscofranca,X)):- X>5000.
```

VII. O árbitro Guerra Godinho que apitou o sétimo jogo, declara ser falso que alguma vez tenha recebido os 2.500€ que a comunicação social refere como tendo entrado na sua conta bancária; contudo, este árbitro nunca confirmou o valor exacto das ajudas de custo que recebeu;

```
jogo(7,guerragodinho,desconhecido).
-jogo(7,guerragodinho,2500).
```

VIII. Não se conhecendo, exactamente, o valor das ajudas de custo entregues ao árbitro Hélder Heitor no oitavo jogo, aceita-se ter sido um valor cerca dos 1.000€;

```
% Nesta questao eh necessario definir quanto eh o "acerca". Deve ser definido atraves de uma ↩
    percentagem, por exemplo 10%, e, por isso, deve ser criado um predicado capaz de ↩
    devolver o valor do acerca.
% acercaUP: X,Y -> {V,F}
acerca(X,Y):- Z is X*0.95,
              W is X*1.05,
              Y=<W,
              Y>=Z.

excepcao(jogo(8,helderheitor,X)):- acercaUP(1000,X).
```

IX. Apesar de não se conhecer o valor exacto das ajudas de custo pagas ao árbitro do nono jogo, Ivo Inocêncio, este terá recebido uma quantia muito próxima dos 3.000€;

```
% Para responder a esta questao, podia-se usar perfeitamente o acerca anterior, no entanto, ↩
    para diferenciar o acerca do "muito proximo", considerou-se outro predicado, muito ↩
    semelhante mas mais restritivo na exactidao:
% muitoProximo: X,Y -> {V,F}
muitoProximo(X,Y):- Z is X*0.98,
                    W is X*1.02,
                    Y=<W,
                    Y>=Z.

excepcao(jogo(8,ivoinocencio,X)):- muitoProximo(3000,X).
```

7.4 Testes

```
| ?- demo(jogo(3,costacarvalho,500),R).
R = desconhecido ?
yes
| ?- demo(jogo(3,costacarvalho,501),R).
R = falso ?
yes
| ?- demo(jogo(2,baltazarborges,230),R).
R = desconhecido ?
yes
| ?- demo(jogo(4,duartedurao,500),R).
R = desconhecido ?
yes
| ?- demo(jogo(4,duartedurao,5000),R).
R = falso ?
yes
| ?- evolucao(jogo(26,zacarias,2400)).
yes
| ?- listing(jogo).
jogo(1, almeidaantunes, 500).
jogo(2, baltazarborges, desconhecido).
jogo(5, edgaresteves, interdito).
jogo(6, ff, 250).
jogo(7, gg, desconhecido).
jogo(26, zacarias, 2400).

yes
| ?- evolucao(jogo(5, edgaresteves,400)).
no
demo(jogo(5,edgaresteves,300),R).
R = desconhecido ?
yes
| ?- demo(jogo(7,guerragodinho,250),R).
R = desconhecido ?
yes
| ?- demo(jogo(7,guerragodinho,2500),R).
R = falso ?
yes
| ?- demo(jogo(8,hh,950),R).
R = desconhecido ?
yes
| ?- demo(jogo(8,hh,940),R).
R = falso ?
yes
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 7
Abril, 2010

Tema

Conhecimento Imperfeito.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Distingam diversos géneros de conhecimento imperfeito;
- Utilizem valores nulos para caracterizar situações relacionadas com a representação de conhecimento imperfeito;
- Utilizem invariantes estruturais e referenciais em problemas cujo âmbito dá corpo à problemática da evolução do conhecimento;
- Utilizem meta-interpretadores para a construção de mecanismos de raciocínio adequados às problemáticas específicas de um sistema ou ambiente de programação.

Enunciado

Recorrendo à problemática da representação de conhecimento imperfeito, pela caracterização de situações inconclusivas através de valores nulos e à utilização de invariantes para definição das condições de consistência do conhecimento, resolva o seguinte exercício:

i. O árbitro Almeida Antunes apitou o primeiro jogo do campeonato, no qual recebeu 500€ como ajudas de custo;
ii. O árbitro Baltazar Borges apitou o segundo jogo, tendo recebido a título de ajudas de custo um valor que ainda ninguém conhece;
iii. Consta na ficha de jogo da terceira partida, que o árbitro Costa Carvalho recebeu 500€, mas a comunicação social alega ter-lhe sido pago mais 2.000€ (como compensação por danos no seu veículo); instado a pronunciar-se sobre o assunto, o árbitro não confirma nem desmente nenhum dos valores noticiados;
iv. O árbitro Duarte Durão apitou o quarto jogo, tendo recebido como ajudas de custo um valor que ronda os 250€ a 750€, desconhecendo-se qual a quantia exacta;
v. No quinto jogo apitado pelo árbitro Edgar Esteves, ocorreram tumultos no final do encontro tendo desaparecido as ajudas de custo da carteira do árbitro, pelo que se torna impossível vir-se a conhecer esse valor;
vi. O árbitro do sexto jogo, Francisco França recebeu, como ajudas de custo, o valor de 250€; no entanto, entre amigos, refere ter "encaixado" nesse dia para cima de 5.000€;
vii. O árbitro Guerra Godinho que apitou o sétimo jogo, declara ser falso que alguma vez tenha recebido os 2.500€ que a comunicação social refere como tendo entrado na sua conta bancária; contudo, este árbitro nunca confirmou o valor exacto das ajudas de custo que recebeu;
viii. Não se conhecendo, exactamente, o valor das ajudas de custo entregues ao árbitro Hélder Heitor no oitavo jogo, aceita-se ter sido um valor cerca dos 1.000€;
ix. Apesar de não se conhecer o valor exacto das ajudas de custo pagas ao árbitro do nono jogo, Ivo Inocêncio, este terá recebido uma quantia muito próxima dos 3.000€;

Para além das situações descritas, aplicam-se as seguintes regras de funcionamento ao sistema de gestão e nomeação dos árbitros:

x. Num mesmo jogo não pode existir mais do que um árbitro nomeado;
xi. Um árbitro não pode apitar mais do que 3 partidas do campeonato;
xii. O mesmo árbitro não pode apitar duas partidas consecutivas.

O sistema deverá ser capaz de implementar o mecanismo de raciocínio adequado ao enquadramento do problema e, ainda, mostrar capacidade para lidar com situações de aquisição de novo conhecimento.

Capítulo 8

Ficha 8: R. Conhecimento Imperfeito

Esta ficha é continuação da mesma teoria da ficha anterior: Representação de Conhecimento Imperfeito. Antes de resolver a ficha, é necessário manter a consistência da base de conhecimento:

```
% Nao se admitira a existencia de mais do que uma ocorrencia da mesma evidencia na relacao filho/3:
+filho(F,P,M)::(solucoes( (F,P,M), ( filho(F,P,M) ), S),
                comprimento(S,N),
                N==1).

% Nao eh admissivel existirem mais do que 2 progenitores para um determinado individuo, na relacao ↩
    filho/3;
+filho(F,P,M)::(solucoes( (Ps,Ms), ( filho(F,Ps,Ms) ), S),
                comprimento(S,N),
                N=<2).

% Predicado para provar factos falsos:
-filho(F,P,M):- nao( filho(F,P,M) ),
                nao(excepcao( filho(F,P,M) )).

% Predicado para provar factos falsos:
-nasceu(N,D):- nao( nasceu(N,D) ),
               nao(excepcao( nasceu(N,D) )).

% Nao se admitira a existencia de mais do que uma data de nascimento:
+nasceu(N,D)::(solucoes( Ds, ( nasceu(N,Ds) ), S),
               comprimento(S,C),
               C==1).

% Necessario para os pais desconhecidos
excepcao(filho(F,P,M)):- filho(F,desconhecido,M).
excepcao(filho(F,P,M)):- filho(F,P,desconhecido).
excepcao(nasceu(F,P)):- nasceu(F,interdito).

% Para o conhecimento interdito:
nulo(interdito).
```

De salientar que para a resolução desta ficha, foi necessário considerar algumas modificações sobre o predicado conhecido nas aulas referente à relação Filho/Pai. Outrora, o predicado "filho" tinha apenas dois termos. No entanto, para considerar o pai ou a mãe desconhecidos (ver exercício 8) é necessário modificar a base de conhecimento de forma a conseguir, de uma forma simples, considerar essa informação correctamente.

O interpretador de questões será o seguinte (igual ao da ficha anterior):

```
demo(Q, verdade):- Q.
demo(Q, falso):- -Q.
demo(Q, desconhecido):- nao(Q), nao(-Q).

nao(Q):- Q, !, fail.
nao(Q).
```

Predicados utilizados na resolução da ficha:

```
filho:  Filho, Pai, Mae    -> {V,F,D}
nasceu: Nome, date     -> {V,F,D}
date:   dia, mes, ano -> {V,F}
```

8.1 Resolução da ficha

I. O Abel e a Alice são os pais da Ana; a Ana nasceu no dia 01/01/2010;

```
filho(ana,abel,alice).
nasceu(ana,date(01,01,2010)).
```

II. O António e a Alberta são os pais do Aníbal; o Aníbal nasceu no dia 02/01/2010;

```
filho(anibal,antonio,alberta).
nasceu(anibal,date(02,01,2010)).
```

III. O Brás e a Belém são os pais da Berta e do Berto; a Berta e o Berto nasceram no dia 02/02/2010;

```
filho(berta,bras,belem).
filho(berto,bras,belem).
nasceu(berta,date(02,02,2010)).
nasceu(berto,date(02,02,2010)).
```

IV. A Cátia nasceu no dia 03/03/2010;

```
nasceu(catia,date(03,03,2010)).
```

V. A Cátia é a mãe do Crispim, mas desconhece-se se o pai é o Celso ou o Caio;

```
excepcao( filho(crispim,celso,catia)).
excepcao( filho(crispim,caio,catia)).
```

VI. O Daniel é o pai do Danilo, mas desconhece-se o nome da mãe; o Danilo nasceu no dia 04/04/2010;

```
filho(danilo,daniel,desconhecido).
nasceu(danilo,date(04,04,2010)).
```

VII. O Elias e a Elsa são os pais do Eurico; O Eurico nasceu no mês de Maio de 2010, embora se desconheça se foi no dia 5 ou no dia 6;

```
filho(eurico,elias,elsa).
excepcao( nasceu(eurico,date(X,05,2010)) ):- X>=1, X=<31.
```

VIII. O Fausto não sabe se a filha se chama Fábia ou Octávia e desconhece-se o nome da mãe;

```
excepcao( filho(fabia,fausto,desconhecido)).
excepcao( filho(octavia,fausto,desconhecido)).
```

IX. O Guido e a Guida são os pais do Golias, mas como faleceram antes de registarem o filho, nunca será possível saber a data de nascimento do Golias;

```
filho(golias,guido,guida).
nasceu(golias,interdito).
+nasceu(N,V)::(solucoes(V, (nasceu(golias,V), nao(nulo(V)) ),[])).
```

X. Embora não se conheça a data de nascimento do Hélder, sabe-se que não nasceu no dia 8 de Agosto de 2010 (imagine que, nesse dia, a sua mãe foi vista ainda grávida).

```
%Devia ser mais robusto, mas nesse caso teria-se de fazer um controlo total de datas.
excepcao( nasceu(helder,date(X,8,2010)) ):- X>=9,
                                            X=<31.
excepcao( nasceu(helder,date(X,Y,2010)) ):- X>=1,
                                            X=<31,
                                            Y>=9,
                                            Y=<12.
excepcao( nasceu(helder,date(X,Y,Z)) ):-    X>=1,
                                            X=<31,
                                            Y>=1,
                                            Y=<12,
                                            Z>=2011.
```

8.2 Testes

```
| ?- demo(filho(berto,bras,belem),R).
R = verdade ?
yes
| ?- demo(nasceu(berta,D),R).
D = date(2,2,2010),
R = verdade ?
yes
| ?- demo(filho(crispim,celso,catia),R).
R = desconhecido ?
yes
| ?- demo(filho(crispim,manuel,catia),R).
R = falso ?
yes
| ?- demo(filho(danilo,daniel,diana),R).
R = desconhecido ?
yes
| ?- demo(nasceu(eurico,(date(23,05,2010))),R).
R = desconhecido ?
yes
| ?- demo(nasceu(eurico,(date(23,05,2009))),R).
R = falso ?
yes
| ?- demo(filho(fabia,fausto,flabia),R).
R = falso ?
yes
| ?- demo(filho(fabia,fausto,desconhecido),R).
R = desconhecido ?
yes
| ?- demo(filho(joana,fausto,desconhecido),R).
R = falso ?
yes
| ?- demo(nasceu(helder,date(9,9,2010)),R).
R = desconhecido ?
yes
| ?- demo(nasceu(helder,date(9,9,2012)),R).
R = desconhecido ?
yes
| ?- demo(nasceu(helder,date(9,9,2002)),R).
R = falso ?
yes
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 8
Abril, 2010

Tema

Conhecimento Imperfeito.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Distingam diversos géneros de conhecimento imperfeito;
- Utilizem valores nulos para caracterizar situações relacionadas com a representação de conhecimento imperfeito;
- Utilizem invariantes estruturais e referenciais em problemas cujo âmbito dá corpo à problemática da evolução do conhecimento;
- Utilizem meta-interpretadores para a construção de mecanismos de raciocínio adequados às problemáticas específicas de um sistema ou ambiente de programação.

Enunciado

Recorrendo à problemática da representação de conhecimento imperfeito, pela caracterização de situações inconclusivas através de valores nulos e à utilização de invariantes para definição das condições de consistência do conhecimento, resolva o seguinte exercício:

i. O Abel e a Alice são os pais da Ana; a Ana nasceu no dia 01/01/2010;
ii. O António e a Alberta são os pais do Aníbal; o Aníbal nasceu no dia 02/01/2010;
iii. O Brás e a Belém são os pais da Berta e do Berto; a Berta e o Berto nasceram no dia 02/02/2010;
iv. A Cátia nasceu no dia 03/03/2010;
v. A Cátia é a mãe do Crispim, mas desconhece-se se o pai é o Celso ou o Caio;
vi. O Daniel é o pai do Danilo, mas desconhece-se o nome da mãe; o Danilo nasceu no dia 04/04/2010;
vii. O Elias e a Elsa são os pais do Eurico; O Eurico nasceu no mês de Maio de 2010, embora se desconheça se foi no dia 5 ou no dia 6;
viii. O Fausto não sabe se a filha se chama Fábia ou Octávia e desconhece-se o nome da mãe;
ix. O Guido e a Guida são os pais do Golias, mas como faleceram antes de registarem o filho, nunca será possível saber a data de nascimento do Golias;
x. Embora não se conheça a data de nascimento do Hélder, sabe-se que não nasceu no dia 8 de Agosto de 2010 (imagine que, nesse dia, a sua mãe foi vista ainda grávida).

O sistema deverá ser capaz de implementar o mecanismo de raciocínio adequado ao enquadramento do problema.

Capítulo 9

Ficha 9 - Herança

Nesta ficha, tal como na maior partes das fichas para resolução de problemas em PROLOG, é necessário tomar decisões conforme as nossas necessidades para responder a determinados tipos de questões. Nesta ficha em concreto, é necessário decidir que tipo de herança devemos escolher: ou herança que engloba todas as propriedades, mesmo aquelas do mesmo tipo; ou herança que apenas tem em conta as propriedades novas do mesmo tipo, esquecendo as propriedades que deveria herdar de entidades hierárquicas superiores.

Assim, foram feitos dois predicados diferentes para interpretação de questões. O primeiro, tem em conta todas as propriedades definidas na base de conhecimento, ou seja, existe uma herança total:

```
demo(A,Q):-
  agente(A,T),
  processar(Q,T).
demo(A,Q):-
  e_um(A,D),
  demo(D,Q).
```

Relativamente ao predicado para interpretação de questões que tem em conta apenas as propriedades mais novas, ou seja, existe herança, mas é substituível por propriedades novas, definiu-se da seguinte maneira:

```
demo2(A,Q):-
  agente(A,T),
  processar(Q,T).
demo2(A,Q):-
  agente(A,T),
  nao(pertence(Q,T)),
  e_um(A,D),
  demo2(D,Q).
```

Aparte dessa diferença, em que num caso considera todas as propriedades e o outro, se houver propriedades novas do mesmo tipo, apenas considera as mais novas, os dois predicados são iguais. Ou seja, inicialmente verifica se existe algum termo que corresponda ao agente. Caso não encontre nenhum termo, verifica se o agente hierarquicamente acima tem alguma teoria que resposda.
Os termos anteriores fazem uso também dos seguintes predicados para procura de teorias:

```
processar(X,X).
processar(X,Lista) :- pertence(Lista,X).

pertence([H|_],H).
pertence([_|T],H) :- pertence(T,H).
```

Como existem casos em que será melhor uma herança que substitua as propriedades do mesmo tipo, então, para esta ficha, será considerado o predicado "**demo2**" como interpretador de questões.
Como se pode ver nos dois predicados de interpretação de questões, existirá um predicado que relacionará as hierarquias: o predicado **e_um**. Este predicado irá relacionar a primeira entidade como uma sub-entidade da segunda. Quanto ao predicado "**agente**", este irá ser o predicado que definirá cada entidade e as suas características.

Para a resolução da ficha, irão ser considerados os seguintes predicados:

```
demo2:  Agente, Questao -> {V,F}

agente: Agente, Teoria  -> {V,F}
e_um:   Origem, Destino -> {V,F}
```

9.1 Resolução da ficha

Tendo em conta todos os pressupostos anteriores, a resolução da ficha passa a ser bastante simples. Bastando, para isso, preencher o predicado "**agente**" por cada entidade da hierarquia, com todas as suas propriedades; e para cada agente, deverá ser considerado um relacionamento binário do predicado "**e_um**" para determinar a hierarquia das entidades. Assim, relativamente às propriedades das entidades tem-se o seguinte:

```
agente(   ave,    [ coberto(penas),      movimento(voo) ]).
agente( canario,  [ cor(amarelo),        som(chilro)    ]).
agente( papagaio, [ comida(pao),         som(fala),     cor(verde),  cor(vermelho) ]).
agente(  boby,    [ cor(preto),          cor(branco)    ]).
agente(  lulu,    [ alimento(tremocos),  coberto(pelos) ]).
```

Relativamente à hierarquia das entidades, basta ter o seguinte:

```
e_um(boby, canario).
e_um(canario,ave).
e_um(papagaio,ave).
e_um(lulu,papagaio).
```

9.2 Testes

```
| ?- demo2(canario,som(X)).
X = chilro ?
yes
| ?- demo(boby,cor(X)).
X = preto ? n
X = branco ? n
X = amarelo ? n
no
| ?- demo2(boby,cor(X)).
X = preto ? n
X = branco ? n
no
| ?- demo2(lulu,cor(preto)).
no
| ?- demo2(lulu,cor(verde)).
yes
| ?- demo2(lulu,cobertura(X)).
X = pelos ? n
no
| ?- demo(lulu,cobertura(X)).
X = pelos ? n
X = penas ? n
no
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 9
Abril, 2010

Tema

Raciocínio por Defeito em Sistemas Hierárquicos, com Herança.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Entendam a utilização de pressupostos de raciocínio por defeito, como a herança, no desenvolvimento de sistemas hierárquicos de representação de conhecimento;
- Reconheçam as capacidades subjacentes à representação de conhecimento em termos hierárquicos;
- Abordem a resolução de problemas descritos através de sistemas hierárquicos de representação de conhecimento, utilizando pressupostos baseados na herança para a construção de mecanismos de raciocínio adequados.

Enunciado

Fazendo uso da linguagem de programação em lógica PROLOG, pretende-se que aborde o problema apresentado na Figura 1, no sentido de explorar as suas capacidades em termos da implementação de mecanismos de raciocínio por defeito, característicos dos sistemas hierárquicos de representação de conhecimento.

Deve ter-se em consideração que a estratégia para a implementação do mecanismo de raciocínio por defeito, neste tipo de sistemas, recorrerá à implementação de procedimentos que levam em linha de conta a capacidade de as diferentes entidades presentes no universo de discurso poderem herdar conhecimento de outras, a um nível hierarquicamente superior.

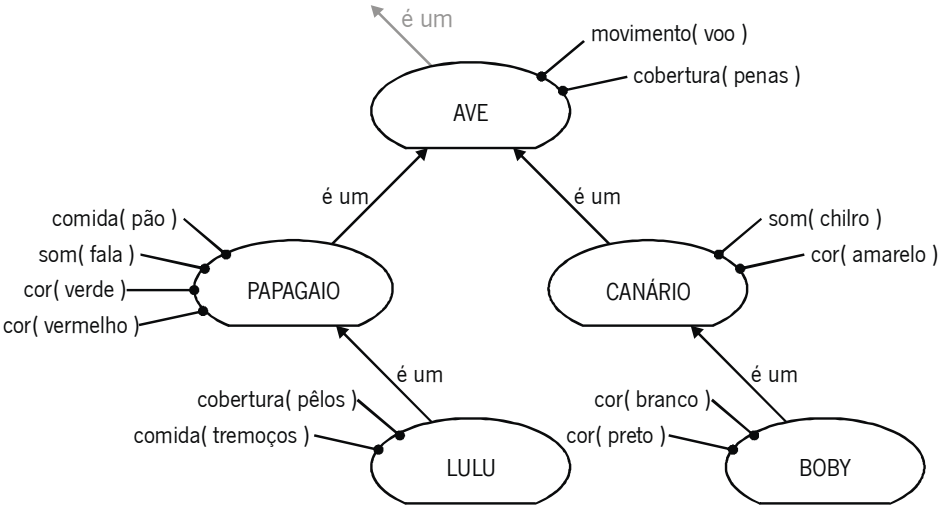

Figura 1

Ilustração de um sistema de estrutura hierárquica, com herança.

Desta forma, deverá ser possível questionar o sistema no sentido de saber qual o som emitido pelo canário, qual a cobertura dos papagaios, se o Boby tem cor amarela ou se o Lulu tem cor preta.

Capítulo 10

Ficha 10 - Herança múltipla

10.1 Resolução da ficha

Nesta ficha, será feito um sistema hierárquico de representação de conhecimento. Haverá uma hierarquia semelhante à feita na ficha anterior, no entanto, existe uma evolução na hierarquia: a multiplicidade.

Também diferente da ficha anterior, é o interpretador de questões. Ao contrário da ficha anterior que bastava apresentar os termos de cada agente, nesta ficha é necessário provar condições. Sendo assim, para além do interpretador (**demo/2**), haverá um predicado "**prova/2**" para testar as várias condições:

```
%demo: Agente, Questao -> {V,F}
demo(A,Q):-
  agente(A,T),
  prova(Q,T).
demo(A,Q):-
  agente(A,T),
  nao(pertence(Q,T)),
  e_um(A,D),
  demo(D,Q).

%Prova: Questao, Teoria -> {V,F}
prova(Q,[Q|T]).
prova(Q,[(Q:-Condicoes)|T]) :- Condicoes.
prova(Q,[X|T]):-Q \==X, prova(Q,T).

%pertence: Lista, valor -> {V,F}
pertence([H|_],H).
pertence([_|T],H) :- pertence(T,H).
```

Tal como a ficha anterior, serão considerados os seguintes predicados:

```
demo:  Agente, Questao -> {V,F}
agente: Agente, Teoria  -> {V,F}
e_um:   Origem, Destino -> {V,F}
```

Para além disso, serão utilizados predicados auxiliares para a prova das condições que alguns dos agentes têm, nomeadamente um predicado que cria listas, outro que foi feito no exercídio IV. da ficha 2 para fazer o somatório de uma lista e outro predicado feito no exercício IV. da ficha 3:

```
%somar: Lista, Resultado -> {V.F}
somar([],0).
somar([H|T],X):- somar(T,R),
                 X is H+R.

%crialista
crialista(Comprimento,1,[Comprimento]).
crialista(Comprimento,Lados,[Comprimento|Lista]) :- LadosTemp is Lados - 1, crialista(Comprimento,↩
    LadosTemp,Lista).

%comprimento
compriment([],0).
compriment([H|T],R):- R is X+1,
                      compriment(T,X).
```

Relativamente às entidades ou agentes, estes têm a particularidade de ter predicados que serão condições para se obter resultados. Nesta ficha, para além do cálculo de propriedades geométricas, ainda pede que seja apresentada uma descrição de cada figura geométrica. Essa descrição será feita através do predicado **write/1**:

```
agente(poligono(Lista),    [(descricao:-    comprimento(Lista,X),
                                            write('Poligono com '),
                                            write(X),
                                            write('lados.')),
                            (perimetrop(P):- somar(Lista,P))]).
agente(rectangulo(L1,L2), [(descricao:-     A is L1*L2,
                                            write('Rectangulo de area '),
                                            write(A)),
                            (area(A):-      A is L1 * L2)]).
agente(polregular(L,N),    [(descricao:-    P is N*L,
                                            write('Poligono Regular com '),
                                            write(N),
                                            write(' lados de tamanho '),
                                            write(L),
                                            write(' e perimetro igual a '),
                                            write(P)),
                            (perimetro(P):- P is N*L)]).
agente(quadrado(L),        [(descricao:-    write('Quadrado com lado '),
                                            write(L))]).
agente(pentagono(L),       []).
```

Considerações: o pentagono, na figura, não tem qualquer condição, por isso não lhe foi dada nenhuma teoria; o predicado **write/1**, só recebe um argumento de cada vez, por isso, para criar as várias frases, foi preciso utilizá-lo várias vezes.

Quanto à hierarquização dos agentes que tem a particularidade de ser múltipla, é a seguinte:

```
e_um(polregular(L,N),poligono(Lista)):- crialista(L,N,Lista).
e_um(rectangulo(L1,L2),poligono(Lista)):- Lista = [L1,L1,L2,L2].
e_um(pentagono(L),polregular(L,5)).
e_um(quadrado(L),rectangulo(L,L)).
e_um(quadrado(L),polregular(L,4)).
```

10.2 Testes

Os testes demonstrados a seguir, respondem às questões feitas no penúltimo parágrafo da ficha:

```
| ?- demo(quadrado(4),area(A)).
A = 16 ?
yes
| ?- demo(pentagono(5),perimetro(P)).
P = 25 ?
yes
| ?- demo(polregular(12,3),descricao).
Poligono Regular com 12 lados de tamanho 3 e perimetro igual a 36
yes
```

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 10
Abril, 2010

Tema

Raciocínio por Defeito em Sistemas Hierárquicos, com Herança.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Abordem a resolução de problemas descritos através de sistemas hierárquicos de representação de conhecimento, utilizando pressupostos baseados na herança para a construção de mecanismos de raciocínio adequados;
- Explorem as capacidades, limitações e aplicações da utilização de pressupostos de raciocínio baseados na herança.

Enunciado

Pretende-se que aborde o problema apresentado na Figura 1, explorando as suas capacidades em termos da implementação de mecanismos de raciocínio por defeito, característicos dos sistemas hierárquicos de representação de conhecimento.

Deve ter-se em consideração que a estratégia para a implementação do mecanismo de raciocínio por defeito recorrerá à implementação de procedimentos que levam em linha de conta pressupostos baseados na herança, bem como a possibilidade de o corpo de conhecimento das diversas entidades presente na estrutura poder ser descrito em termos de um qualquer paradigma de representação de conhecimento.

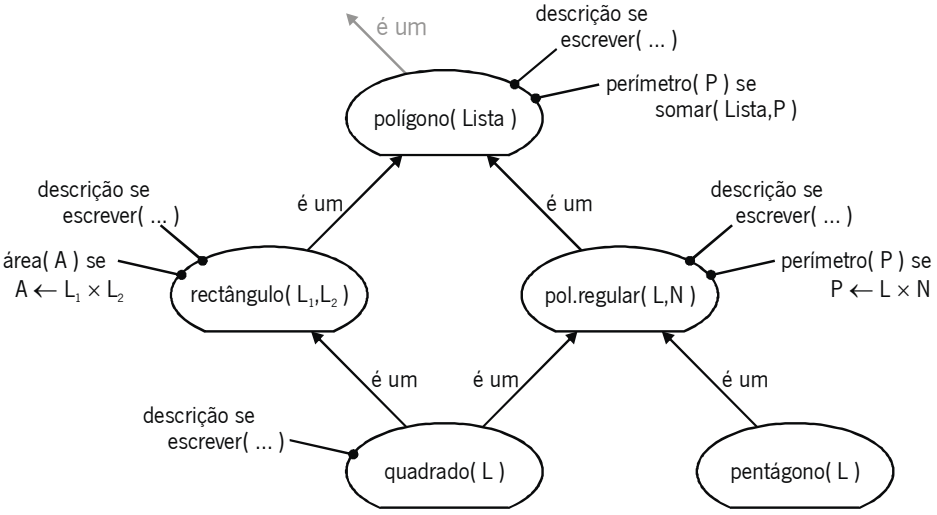

Figura 1

Ilustração de um sistema de estrutura hierárquica, com herança múltipla.

Desta forma, será possível questionar o sistema no sentido de saber qual a área de um quadrado cujos lados medem 4 unidades de comprimento (UC), qual o perímetro de um pentágono com lados de comprimento igual a 5 UC ou qual a descrição que o sistema atribui a um polígono regular de 12 lados, com comprimento de 3 UC.

Deverá ser estudado, também, um método de controlo sobre o mecanismo de herança múltipla como o que se encontra presente a partir da entidade quadrado( ).

Capítulo 11

Ficha 12

11.1 Quadros Negros (LINDA)

Quadros negros são espaços de partilha de memória com um determinado objectivo. Cada quadro negro tem o seu socket.

11.1.1 Protocolo de comunicação (linguagem)

Não é TCP/IP, é uma linguagem, um conjunto de termos que serão conhecidos pelos vários agentes inteligentes. Essa organização de termos conhecidos para cada agente forma a linguagem de comunicação. Por exemplo a linguagem desta ficha é descrita pelo seguinte:

```
        n: X
   pedido: Op
resultado: Op, Res
```

11.1.2 Linda/Server

Um quadro negro, no Sicstus Prolog, é um servidor linda. Assim, para criar um servidor, basta abrir uma sessão do sicstus e executar o predicado **linda/0** do módulo **linda/server**.

Tendo em conta o descrito no parágrafo anterior, para a resolução desta ficha, como é necessário um Quadro Negro para partilha dos dados entre os vários agentes, então é necessário criar um ficheiro que será o servidor (**servidor.pl**) e que será uma das várias sessões abertas. Esse ficheiro terá apenas o conteúdo seguinte:

```
:-use_module(library('linda/server')),linda.
```

11.1.3 Linda/Client

Cada agente que se vai ligar ao servidor, irá utilizar o módulo chamado **linda/client**. Este módulo contém vários predicados que serão utilizados para ligar, enviar, ler e retirar dados dos quadro negros. Os vários predicados existentes no módulo estão apresentados na Tabela 11.1.

11.2 Resolução da ficha

Para resolução da ficha, será abordado cada agente em separado, tal como eles funcionarão depois. O funcionamento protocolar entre os agentes será baseado na linguagem especificada anteriomente, ou seja, serão enviados valores para o servidor, que serão posteriormente calculados. Depois dos valores serem inseridos pela interface, então, através da mesma, serão pedidos os cálculos aos agentes que retornarão

Ligação:	**linda_client(Maq:Porta).**	Permite ligar a uma determinada máquina com uma determinada porta
	close_client.	Permite fechar a ligação que tem para com uma determinada máquina
Leitura:	**in(X).**	Este predicado consome o que houver para ler conforme o argumento
	rd(X).	Lê conforme o predicado, mas não consome a informação do quadro negro, não a apaga
	in_noblock(X).	Ao contrário dos anteriores, não bloqueia se não satisfizer a unificação. Aparte disso, é semelhante ao predicado in(X).
	rd_noblock(X).	Tal como anterior, não bloqueia se não satisfizer a unificação e funciona da mesma maneira que o rd(X).
Escrita:	**out(X).**	Escreve no Quadro Negro

Tabela 11.1: Tabela do módulo Linda/Client

os resultados para o Quadro negro, onde posteriormente deverão ser consumidos para visualização dos resultados.

Assim, para testar todo este sitema inteligente multi-agente, serão utilizados os seguintes predicados:

```
% Para ligar ao servidor:
ligar: maquina, porta

% para enviar numeros (X) a serem calculados:
in( n(X) ).

% para fazer um pedido de accao/processamento, ser feito da seguinte maneira:
in ( pedido( ACCAO) ).

% para visualizar o resultado de um determinado calculo:
in ( resultado(ACCAO,S) ).

% para por um agente em espera, usa-se o predicado demo:
demo.
```

Os agentes autónomos estarão sempre à espera que lhe mandem uma determinada ordem, que foi defenida desta forma (para o caso do somatorio):

```
pedido(somatorio)
```

Sempre que receber este pedido, o agente vai consumir esse pedido e vai ler (não vai consumir) todos os valores inseridos no quadro negro com o formato:

```
n(X)
```

Essa leitura será feita através do predicado **bagof_rd_noblock/3**, que funciona da mesma forma que o predicado **soluções/3** feito nas aulas, com a única diferença que este vai ao Quadro Negro. Depois do referido predicado encontrar os vários números (pode ser nenhum, pois é *noblock*) é calculado o valor através do predicado **somatorio/2** ou **produtorio/2**, conforme o agente. Por fim, o resultado **R** será inserido no Quadro Negro, sob a forma seguinte (no caso do somatorio):

```
resultado(somatorio,R)
```

11.2.1 Agente Somatorio

Depois de todas as considerações referidas, o agente (**somatorio.pl**) deverá ter o seguinte aspecto:

```
:-use_module(library('system')).
:-use_module(library('linda/client')).

qn(A):-linda_client: bagof_rd_noblock(B,B,A).

%Predicado especialista no calculo do SOMATORIO

demo:-
   write('Esperando um pedido de SOMATORIO.'),nl,
   in(pedido(somatorio)),
   write('SOMATORIO em actividade...'),nl,
   bagof_rd_noblock(X,n(X),L), somatorio(L,S),
   sleep(1),
   out(resultado(somatorio,S)),
   write('...Actividade concluida!'),nl,
   demo.

somatorio([],0).
somatorio([A|B],PART) :- somatorio(B,PART2), PART is PART2+A.
```

Nota: Depois do calculo ser feito, o predicado volta a chamar-se a si mesmo para voltar a ficar à espera de um novo pedido. Relativamente ao **sleep(1)**, apenas está presente para fazer de conta que o cálculo em questão é demorado. Este predicado está incluído no módulo **system**.

11.2.2 Agente Produtório

Para o agente **produtório.pl**, o funcionamento é exactamente igual. A diferença está apenas no cálculo que neste caso em vez de se utilizar o predicado **somatorio/2**, utiliza-se o predicado **produtorio/2**:

```
:-use_module(library('system')).
:-use_module(library('linda/client')).

qn(A):-linda_client: bagof_rd_noblock(B,B,A).

%Predicado especialista no calculo do PRODUTORIO

demo:-
   write('Esperando um pedido de PRODUTORIO.'),nl,
   in(pedido(produtorio)),
   write('PRODUTORIO em actividade...'),nl,
   bagof_rd_noblock(X,n(X),L), produtorio(L,S),
   sleep(1),
   out(resultado(produtorio,S)),
   write('...Actividade concluida!'),nl,
   demo.

produtorio([],1).
produtorio([A|B],PART) :- produtorio(B,PART2), PART is PART2*A.
```

11.2.3 Agente Interface

Neste momento temos um agente **servidor.pl**, que é Quadro Negro e está, por isso, bloqueado e temos também bloqueados os dois agentes autónomos que fazem os cálculos: **produtorio.pl** e **somatorio.pl**.

Ora como estes agentes estão bloqueados, não é possível fazer nada. É, portanto, necessário um outro agente que possa fazer os pedidos aos agentes inteligentes através do quadro negro. O nome desse agente é **interface.pl** e tem como objectivo, interagir e ser simples para o utilizador. Sendo assim, foram criados alguns predicados para tornar mais simples o processo. Há predicados para fazer pedidos ao QN e para inserir dados no QN (conforme a linguagem anteriormente definida):

```
:-use_module(library('linda/client')).

qn(A):-bagof_rd_noblock(B,B,A).

% INTERFACE

insert(X):-          out(n(X)).
inserlista([]).
inserlista([H|T]):-  out(n(H)),
                     inserlista(T).

ligar(X):-   linda_client(X).

pedir(X):-     out(pedido(X)),
               in(resultado(X,S)),
               write(S).
```

11.3 Testes

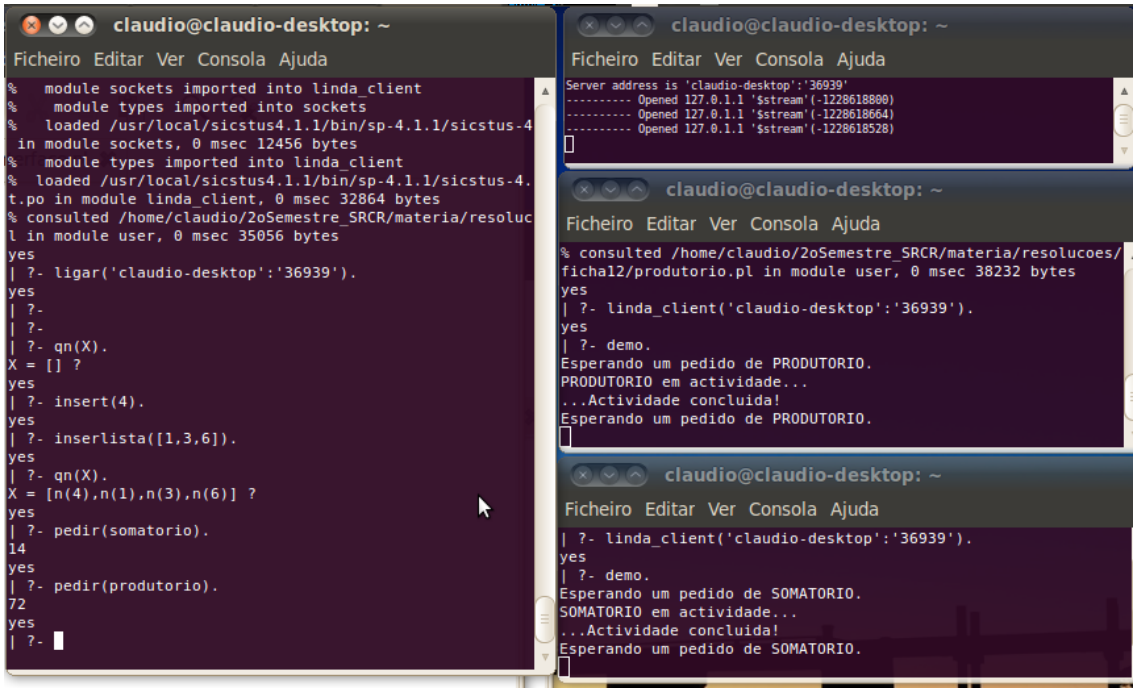

Figura 11.1: Testes entre os agentes da ficha 12

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 12
Maio, 2010

Tema

Sistemas Baseados em Quadros Negros.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Adoptem um modelo de distribuição da computação baseado em Quadros Negros para a implementação de sistemas inteligentes;
- Utilizem e desenvolvam sistemas inteligentes multi-agente.

Enunciado

Recorrendo à linguagem de programação em lógica PROLOG, e às bibliotecas LINDA do SICStus PROLOG, pretende-se que seja construído um ambiente distribuído de computação para implementar uma máquina calculadora sobre séries de números.

Atenda à Figura 1 onde se esboça uma possível arquitectura para o sistema multi-agente a desenvolver.

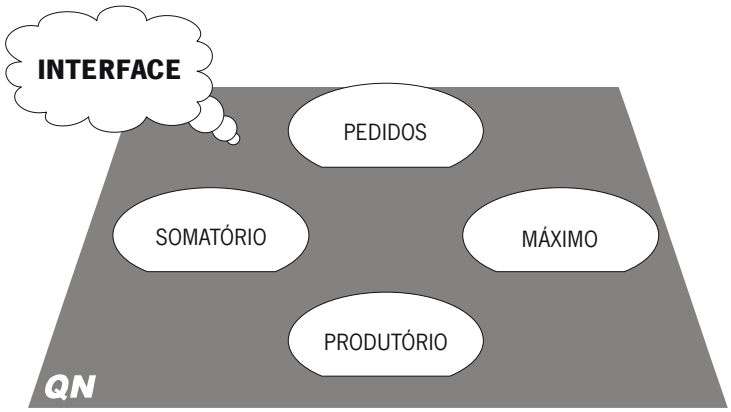

Figura 1

Esboço da arquitectura do SMA.

Para o cenário descrito, desenvolva:

i. A linguagem de comunicação a ser observada por todos os elementos constituintes do sistema;
ii. O corpo de conhecimento de cada um dos agentes.

Capítulo 12

Ficha 13

Esta ficha é bastante semelhante à ficha anterior. A grande diferença prende-se no facto da anterior haver um agente que está à espera de uma comunição para exercer o seu processo e neste caso os agentes estão à espera também de pedidos, conforme a linguagem definida, e irão responder se forem capazes. No caso dos agentes não serem capazes de responder, conforme os seus predicados de definição de hierarquias, deverão perguntar aos agentes hierarquicamente acima.

12.1 Modelos adoptados

12.1.1 Linguagem de comunicação

Tal como na ficha anterior, foi necessário definir um protocolo de comunicação de forma a que os vários agentes "presentes" no Quadro Negro possam interagir e responder a perguntas que sejam feitas através de uma *interface*. Sendo assim, como modelo de comunicação tem-se o seguinte:

```
% As perguntas sao feitas atraves da insercao de um meta-predicado "demo" que contem dois ↩
     argumentos: o nome do agente e a questao
demo(agente,questao)

% As respostas dos agentes, quando forem encontradas, deverao ser da seguinte forma:
prova(agente,resposta)
```

12.1.2 Modelo de distribuição

Nesta ficha, o objectivo é criar modelos de distribuição da computação baseados em Quadrdos Negros. Conforme a Figura 1 da ficha, conclui-se que uma das formas possíveis de resolução deste sistema hierárquico é criando um agente autónomo por cada elemento da figura. Desta forma, cada agente deverá saber responder às questões que ele lhe dizem respeito, conforme a figura referida.

Caso um determinado agente não saiba responder a uma determinada questão colada a ele, então ele deverá fazer a mesma perguntar ao agente hierarquicamente a seguir conforme o protocolo definido na secção anterior.

Todos os agentes presentes no quadro negro que representam o sistema hierárquico da figura, depois de questionados, deverão ou perguntar ao agente seguinte e voltar a ficar à espera de nova questão, ou deverão dar uma resposta para o QN, conforme o protocolo de comunicação, e voltar ao modo de espera.

12.2 Implementação do sistema

12.2.1 Servidor (server.pl)

Sendo este sistema inteligente constituído por vários agentes autónomos em comunicação através de um Quadro Negro, é necessário criar um servidor, ou seja, um Quadro Negro que possibilite a comunicação entre agentes. Tal como a ficha anterior, o Quadro Negro resume-se ao seguinte:

```
:-use_module(library('linda/server')),linda.
```

12.2.2 Interface (interface.pl)

A interface que possibilitará o utilizador comunicar com o sistema inteligente multi-agente, é constituída por alguns predicados que simplificam o processo de perguntas. Sendo assim, o utilizador não precisa de saber como funciona o protocolo de comunicação. Apenas precisa fazer o seguinte para perguntar algo a um agente:

```
p( AGENTE, PERGUNTA ).
```

O ficheiro **interface.pl** ficou com o seguinte aspecto:

```
:-use_module(library('linda/client')).

qn(A):-bagof_rd_noblock(B,B,A).

% INTERFACE

ligar(X):-              linda_client(X).

p(AGENTE,PERGUNTA):- out(demo(AGENTE,PERGUNTA)),
                     in(prova(X,S)),
                     write(S),nl.
```

12.2.3 Agentes autónomos

Relativamente aos agentes autónomos, todos eles têm exactamente a mesma estrura: carregamento dos módulos, predicado que os inicializa, predicado que faz o processamento (verifica se existe resposta na teoria do agente) e no caso de não conseguir responder pergunta ao agente a seguir (conforme o predicado que o define chamado **e_um**), a teoria do agente.

Relativamente ao carregamento dos módulos, utiliza-se apenas o módulo **Linda** para a comunicação com o Quadro Negro. Podia-se utilizar também o módulo **System** para fazer os *sleeps* referidos na ficha anterior.

```
:-use_module(library('linda/client')).
```

Quanto ao predicado que inicaliza o agente, pondo-o à espera de perguntas dirigidas a si, tem o seguinte aspecto:

```
demo:-  write('Sou uma AGENTE'),nl,
    in(demo(AGENTE,Questao)),
    write('demo(AGENTE,Questao)'),nl,
    demo(AGENTE,Questao),
    demo.
```

Em que "AGENTE" deve ser o termo com o nome do agente.

Relativamente ao predicado que tenta buscar a resposta na teoria do agente e no caso de não encontrar pergunta ao agente hierarquicamente a seguir, ele tem o aspecto seguinte:

```
demo(Agente,Questao):-
   Agente::Questao,
   write((1,Agente::Questao)),nl,
   out(prova(Agente,Questao)).
demo(Agente,Questao):-
   e_um(Agente,Classe),
   write((2,e_um(Agente,Classe))),nl,
   out(demo(Classe,Questao)).
```

Lulu

Depois do definido acima, cada agente apenas terá a teoria que os distinguirá uns dos outros. A teoria é composta pelas propriedas definidas na imagem. Assim, o ficheiro **lulu.pl** ficou com o aspecto seguinte:

```
:-op(900,xfy,'::').
:-use_module(library('linda/client')).
qn(A):-linda_client: bagof_rd_noblock(B,B,A).

%-----------------------------------------
%teoria
lulu::cobertura(pelos).
lulu::comida(tremocos).

e_um(lulu,papagaio).

%-----------------------------------------
%inicializa   o da vida do agente

demo:-
   write('Sou o lulu'),nl,
   in(demo(lulu,Questao)),
   write('demo(lulu,Questao)'),nl,
   demo(lulu,Questao),
   demo.
%-----------------------------------------
%Extensao do meta predicado demo

demo(Agente,Questao):-
   Agente::Questao,
   write((1,Agente::Questao)),nl,
   out(prova(Agente,Questao)).
demo(Agente,Questao):-
   e_um(Agente,Classe),
   write((2,e_um(Agente,Classe))),nl,
   out(demo(Classe,Questao)).
```

Boby

Relativamente ao ficheiro **boby.pl**, este ficou com o seguinte aspecto:

```
:-op(900,xfy,'::').
:-use_module(library('linda/client')).
qn(A):-linda_client: bagof_rd_noblock(B,B,A).

%-----------------------------------------
%teoria
boby::cor(branco).
boby::cor(preto).

e_um(boby,canario).

%-----------------------------------------
%inicializa   o da vida do agente

demo:-   write('Sou o boby'),nl,
   in(demo(boby,Questao)),
   write('demo(boby,Questao)'),nl,
   demo(boby,Questao),
   demo.
%-----------------------------------------
%Extensao do meta predicado demo
```

```
demo(Agente,Questao):-
   Agente::Questao,
   write((1,Agente::Questao)),nl,
   out(prova(Agente,Questao)).
demo(Agente,Questao):-
   e_um(Agente,Classe),
   write((2,e_um(Agente,Classe))),nl,
   out(demo(Classe,Questao)).
```

Papagaio

Quanto ao aspecto do ficheiro **papagaio.pl**, ficou assim:

```
:-op(900,xfy,'::').
:-use_module(library('linda/client')).
qn(A):-linda_client: bagof_rd_noblock(B,B,A).

%-----------------------------------------
%teoria
papagaio::comida(pao).
papagaio::som(fala).
papagaio::cor(verde).
papagaio::cor(vermelho).

e_um(papagaio,ave).


%-----------------------------------------
%inicializa   o da vida do agente

demo:-   write('Sou o papagaio'),nl,
   in(demo(papagaio,Questao)),
   write('demo(papagaio,Questao)'),nl,
   demo(papagaio,Questao),
   demo.
%-----------------------------------------
%Extensao do meta predicado demo

demo(Agente,Questao):-
   Agente::Questao,
   write((1,Agente::Questao)),nl,
   out(prova(Agente,Questao)).
demo(Agente,Questao):-
   e_um(Agente,Classe),
   write((2,e_um(Agente,Classe))),nl,
   out(demo(Classe,Questao)).
```

Canário

O **canario.pl** ficou com este aspecto:

```
:-op(900,xfy,'::').
:-use_module(library('linda/client')).
qn(A):-linda_client: bagof_rd_noblock(B,B,A).

%-----------------------------------------
%teoria
canario::som(chilro).
canario::cor(amarelo).

e_um(canario,ave).

%-----------------------------------------
%inicializa   o da vida do agente

demo:-   write('Sou o canario'),nl,
   in(demo(canario,Questao)),
   write('demo(canario,Questao)'),nl,
   demo(canario,Questao),
   demo.
%-----------------------------------------
%Extensao do meta predicado demo

demo(Agente,Questao):-
```

```
    Agente::Questao,
    write((1,Agente::Questao)),nl,
    out(prova(Agente,Questao)).
demo(Agente,Questao):-
    e_um(Agente,Classe),
    write((2,e_um(Agente,Classe))),nl,
    out(demo(Classe,Questao)).
```

Ave

Por fim, relativamente ao agente **Ave**, este agente tem uma pequena diferença em relação aos agentes autónomos anteriores. Enquanto que os anteiores têm sempre de perguntar ao agente a hierarquicamente a seguir caso não saibam a resposta, no caso deste, como não há mais nenhum acima dele, precisa de considerar que a resposta é "inconclusiva". Assim, no ficheiro **ave.pl**, para além da secção "teoria", também tem o predicado **demo/2** diferente:

```
:-op(900,xfy,'::').
:-use_module(library('linda/client')).
qn(A):-linda_client: bagof_rd_noblock(B,B,A).

%-----------------------------------------------
%teoria
ave::movimento(voo).
ave::cobertura(penas).

%-----------------------------------------------
%inicializa   o da vida do agente

demo:-    write('Sou uma ave'),nl,
    in(demo(ave,Questao)),
    write('demo(ave,Questao)'),nl,
    demo(ave,Questao),
    demo.
%-----------------------------------------------
%Extensao do meta predicado demo

demo(Agente,Questao):-
    Agentee::Questao,
    write((1,Agente::Questao)),nl,
    out(prova(Agente,Questao)).
demo(Agente,Questao):-
    write('Nao sei responder'),nl,
    out(prova(Agente,inconclusivo)).
```

12.3 Testes

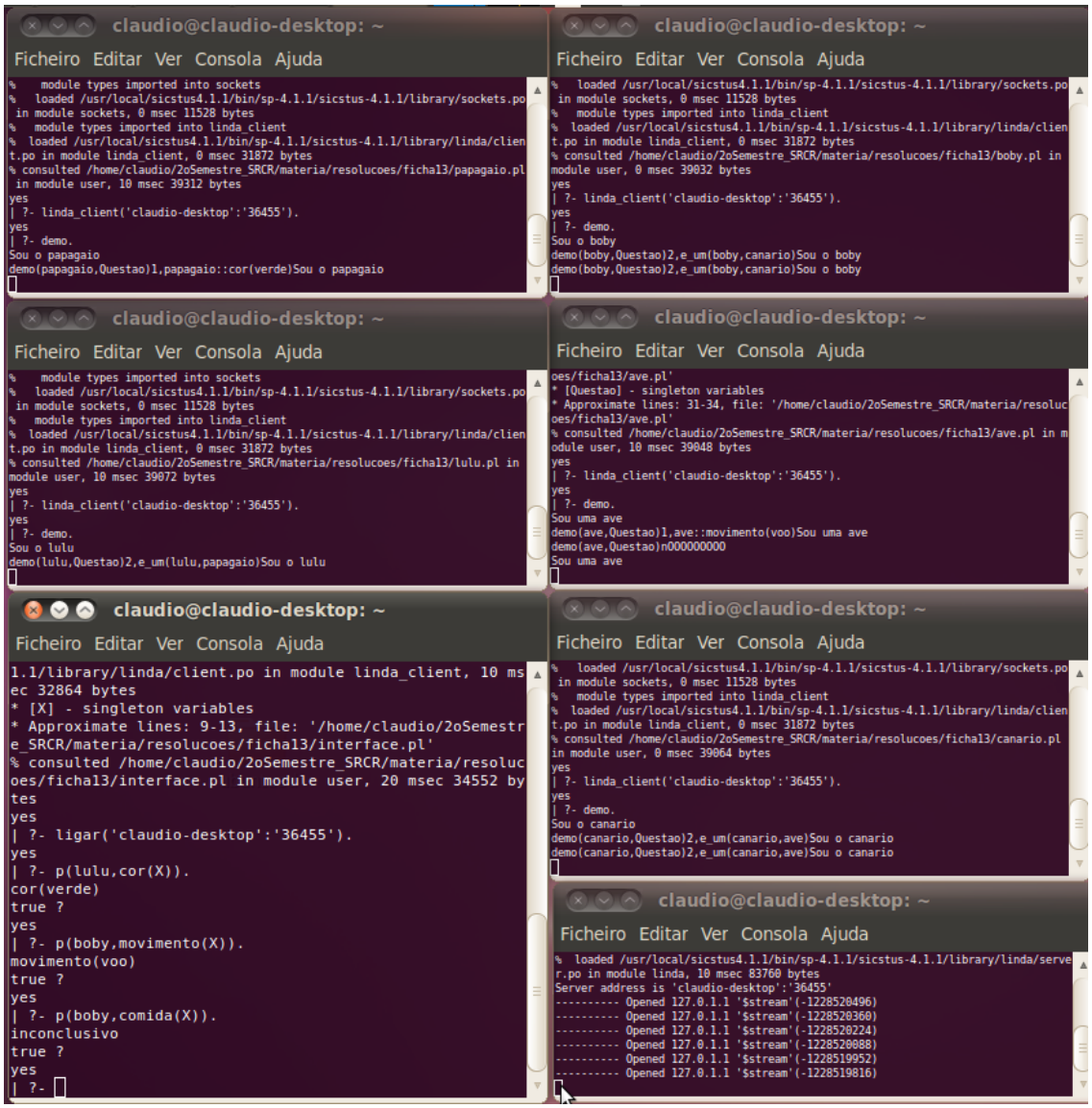

Figura 12.1: Testes entre os agentes da ficha 13

Universidade do Minho
Conselho dos Cursos de Engenharia
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
3º Ano, 2º Semestre
Ano lectivo 2009/2010

Ficha prática nº 13
Maio, 2010

Tema

Computação Distribuída – Sistemas Baseados em Quadros Negros (LINDA)
Sistemas Hierárquicos, com Herança.

Objectivos de aprendizagem

Com a realização desta ficha prática pretende-se que os alunos:

- Adoptem um modelo de distribuição da computação baseado em Quadros Negros para a implementação de sistemas inteligentes;
- Utilizem e desenvolvam sistemas inteligentes multi-agente.

Enunciado

Recorrendo às bibliotecas LINDA do SICStus PROLOG que implementam um modelo de computação distribuída baseado em Quadros Negros (QN), pretende-se que aborde o problema apresentado na Figura 1, que descreve um sistema hierárquico de raciocínio por defeito com base em mecanismos de herança.

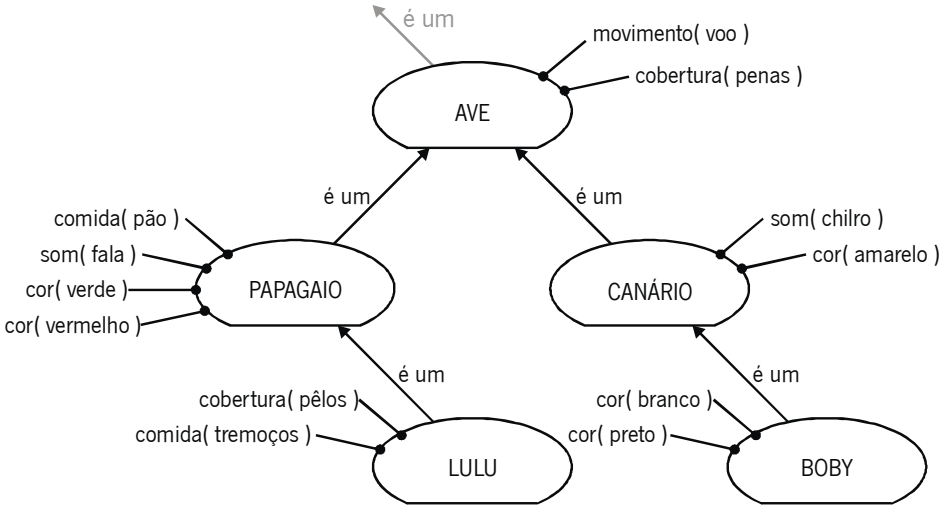

Figura 1

Sistema hierárquico a implementar segundo um modelo de computação distribuída.

Em particular, devem ser discutidos os seguintes temas:

- Qual o modelo de distribuição da computação a adoptar para a resolução deste problema;
- Qual a linguagem a adoptar pelos elementos constituintes do sistemas;
- De acordo com a linguagem definida e o modelo de distribuição da computação adoptado, qual o funcionamento implementado pelo sistema.