Universidade do Minho
Escola de Engenharia
Departamento de Informática
Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho de Grupo – 2º Exercício

Extensão à Programação em Lógica e Conhecimento Imperfeito

Braga, Maio de 2014


David Manuel de Sá de Angelis
Nº60990


Sérgio Lucas dos Santos Oliveira
Nº61024


Marcos André Oliveira Ramos
Nº61023

Resumo

O presente documento serve de apoio ao segundo trabalho prático de SRCR(Sistemas de representação de conhecimento e raciocínio).

O trabalho tem como objectivo, motivar o interesse pela utilização da extensão à programação em lógica, no âmbito da representação de conhecimento imperfeito.

Para tal foi desenvolvido, utilizando a linguagem de programação PROLOG, um sistema de representação de conhecimento e raciocínio que permite descrever um universo de discurso de âmbito farmacológico.

Este relatório pretende mostrar os métodos adoptados para a resolução do enunciado proposto, pelos professores da unidade curricular, as dificuldades e os objectivos cumpridos com o desenvolvimento e conclusão do projecto.

Índice

Introdução

Com vista a desenvolver os nossos conhecimentos e motivar o nosso interesse pela utilização da extensão à programação em lógica, no âmbito da representação de conhecimento imperfeito, recorrendo à utilização de valores nulos e da criação de mecanismos de raciocínio adequados, foi-nos proposto pelos professores de SRCR(Sistemas de Representação de Conhecimento e Raciocínio) o desenvolvimento, em linguagem PROLOG, de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso de âmbito farmacológico.

Depois de aprendermos os fundamentos da linguagem PROLOG e da programação em lógica, entramos assim na extensão à programação em lógica, utilizada neste trabalho. Pretendemos, sobretudo, compreender e solidificar as matérias leccionadas nas aulas da unidade curricular, nomeadamente na representação de conhecimento imperfeito.

Objectivos

A aplicação em PROLOG deverá implementar as seguintes funcionalidades:

- Representar conhecimento positivo e negativo;
- Representar casos de conhecimento imperfeito, pela utilização de valores nulos de todos os tipos estudados;
- Representar invariantes que designem restrições à inserção e à remoção de conhecimento do sistema;
- Construir os procedimentos adequados a lidar com a problemática da evolução do conhecimento;
- Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas.

Descrição do Trabalho e Análise de Resultados

Interpretação do problema

Segundo o enunciado pretendia-se caracterizar um universo de discurso de âmbito farmacológico. Para tal o grupo de trabalho, depois de discutir a melhor forma de abordar todos os objectivos propostos, decidiu considerar o seguinte conhecimento:

- Tipo de farmácia: local ou de hospital
- Medicamentos: principio activo, nome, formato,apresentação e doença(aplicação clinica)
- Datas: data de colocação no mercado e data de validade
- Preços: normal ou reformado

Base de Conhecimento

Depois de definido o que se pretendia construir foi necessário criar um conjunto de predicados para lidar com a base de conhecimentos.

medicamento: identificação(ID), principio activo, um nome comercial, apresentação farmacêutica e aplicação clinica

data: ID do medicamento, data de introdução no mercado e data de validade

tipoFar: ID do medicamento, tipo de farmácia que pode ser local ou de hospital

preco: ID do medicamento, tipo de preço a aplicar que pode ser normal ou para reformados e o preço respectivo

Medicamentos

Descrito por um código de identificação(ID), um principio activo, um nome comercial, apresentação farmacêutica(pomada, xarope, comprimido, etc.) e uma aplicação clinica. Os medicamentos são o foco principal de todo o projecto, sendo que a eles vão estar associados, por exemplo datas e preços.

Conhecimento perfeito, positivo:

```
medicamento(a001,ibuprofeno,brufen,comprimido,febre).
medicamento(a002,butilescopolamina,buscopan,comprimido,'dor abdominal').
medicamento(a003,iodopovidona,betadine,pomada,feridas).
medicamento(a004,'oxido de zinco',halibut,pomada,queimaduras).
medicamento(a005,bnzidamina,'tantum verde',pastilha,'dores de garganta').
```

Conhecimento perfeito, negativo:

```
-medicamento(a006,valeriana,livetan,comprimido,diarreia).
-medicamento(a007,dietilamina,assagim,creme,gripe).
```

Conhecimento imperfeito, incerto:

```
excepcao( medicamento(a008,sodio,maxfluor,comprimido,algum) ).
excepcao( medicamento(a009,tirotricina,mebocaina,algum,gripe) ).
```

Note-se, por exemplo, que no primeiro exemplo a aplicação clinica é 'algum'. Isto introduz incerteza.

Conhecimento imperfeito, impreciso:

```
excepcao( medicamento(a010,ambroxol,mucosolvan,xarope,expectoracao) ).
excepcao( medicamento(a011,ambroxol,mucodrenol,xarope,expectoracao) ).
```

Aqui os dois medicamentos são exactamente iguais, excepto no seu nome comercial.

Conhecimento imperfeito, interdito:

```
medicamento(a012,interdito,memoria,comprimido,memoria).
```

Note-se que aqui a substância activa é 'interdito'.

Para além disto adicionamos os seguintes **invariantes**, de modo a obter uma correcta remoção e inserção e manter a nossa base de conhecimento consistente.

Garante que não estamos a inserir um medicamento com ID repetido:

```
+medicamento(ID,P,N,F,D) :: (solucoes(ID, medicamento(ID,_,_,_,_), L),
comprimento(L,G), G==1).
```

Garante que não existe conhecimento negativo para o positivo que estamos a inserir:

```
+medicamento(ID,P,N,F,D) :: (solucoes(ID, (-medicamento(ID,P,N,F,D)), L),
comprimento(L,G), G=<1 ).
```

Garante que não existe conhecimento positivo para o negativo que estamos a inserir:

```
+(-medicamento(ID,P,N,F,D)) :: (solucoes(ID, (medicamento(ID,P,N,F,D)),
L),
comprimentos(L,G), G==0 ).
```

Garante que se está remover um medicamento que existe:

```
-medicamento(ID,P,N,F,D) :: (solucoes(ID,(medicamento(ID,P,N,F,D)), L),
comprimento(L,G), G==0 ).
```

Datas

As datas vão estar associadas a um medicamento através do seu ID. Este predicado conterá uma data de introdução do medicamento no mercado e uma data de validade.

Conhecimento perfeito, positivo:

```
data(a001,'01/01/01','01/01/2011').
data(a002,'02/02/02','02/02/2022').
```

Conhecimento perfeito, negativo:

```
-data(a003,'03/03/2003','03/04/2003').
```

Conhecimento imperfeito, incerto:

```
excepcao( data(a004,'04/04/2004',algum) ).
excepcao( data(a006,algum,'06/06/2016') ).
```

Conhecimento imperfeito, impreciso:

```
excepcao( data(a007,'07/07/2007','07/07/2017') ).
excepcao( data(a007,'07/07/2007','08/08/2018') ).
```

Conhecimento imperfeito, interdito:

```
data(a008,'08/08/2008',interdito).
```

Os **invariantes** que se seguem foram usados para a correcta inserção e remoção de datas.

Garante que só existe um registo de data que contém o ID do medicamento na base de conhecimento:

```
+data(IDM,_,_) :: (solucoes( IDM, (data(IDM,_,_)),L ),
comprimento(L,N), N==1 ).
```

Garante que não existe conhecimento negativo para o positivo que estamos a inserir:

```
+data(IDM,DI,DL) :: (solucoes( IDM,(-data(IDM,DI,DL)),L ),
comprimento(L,N), N=<1 ).
```

Garante que não existe conhecimento positivo para o negativo que estamos a inserir:

```
+(-data(IDM,DI,DL)) :: (solucoes( IDM,(data(IDM,DI,DL)),L ),
comprimento(L,N), N==0 ).
```

Garante que existe um registo de data que estamos a remover:

```
-data(IDM,DI,DL) :: (solucoes( (IDM,DI,DL),(data(IDM,DI,DL)),L ),
comprimento(L,N), N==1 ).
```

Tipo de farmácia

O tipo de farmácia associa o ID de um medicamento a um tipo de farmácia que pode ser local ou de hospital. Pretende-se simular a realidade das farmácias portuguesas, onde certos medicamentos não estão disponíveis aos utentes e só podem ser adquiridos por especialistas nas farmácias de hospital.

Conhecimento perfeito, positivo:

```
tipoFar(a001,local).
tipoFar(a002,hospital).
```

Conhecimento perfeito, negativo:

```
-tipoFar(a003,local).
-tipoFar(a004,hospital).
```

Conhecimento imperfeito, incerto:

```
excepcao( tipoFar(a005,algum) ).
```

Conhecimento imperfeito, impreciso:

```
excepcao( tipoFar(a006,local) ).
excepcao( tipoFar(a006,hospital) ).
```

Este **invariante** impede que se insira o mesmo tipo de farmácia para o mesmo medicamento:

```
+tipoFar(IDM,Tipo) :: (solucoes(IDM, tipoFar(IDM,Tipo), L),
comprimento(L,G), G==1).
```

Preço dos medicamentos

A cada medicamento vai estar associado um preço que pode ser de dois tipos, normal ou reformado. Isto pretende caracterizar um universo próximo da realidade em que

determinados grupos populacionais, devido a por exemplo factores financeiros, têm certos direitos e benefícios diferentes da população em geral.

Conhecimento perfeito, positivo:

```
preco(a001,normal,5).
preco(a001,reformado,4).
preco(a002,normal,10).
preco(a002,reformado,8).
```

Conhecimento perfeito, negativo:

```
-preco(a003,reformado,21).
-preco(a003,normal,25).
```

Conhecimento imperfeito, incerto:

```
excepcao( preco(a004,algum,12) ).
excepcao( preco(a005,normal,algum) ).
```

Conhecimento imperfeito, impreciso:

```
excepcao( preco(a006,normal,3) ).
excepcao( preco(a006,normal,4) ).
```

Conhecimento imperfeito, interdito:

```
preco(a009,normal,interdito).
```

Os **invariantes** que se seguem são usados para garantir a correcta remoção e inserção de dados na base de conhecimento.

Garante a inexistência de conhecimento igual ao que queremos inserir:

```
+preco(IDM,T,P)::(solucoes( (IDM,T),(preco(IDM,T,_)),L ),
comprimento(L,N), N==1 ).
```

Garante que para um medicamento para o valor do preço que estamos a inserir:

```
+preco(IDM,T,P) :: (solucoes( IDM,(medicamento(IDM,_,_,_,_)),L ),
comprimento(L,G), G==1 ).
```

Garante que todos os preços são maiores que 0:

```
+preco(IDM,T,P) :: P > 0.
```

Garante que não existe um conhecimento negativo para o que estamos a inserir:

```
+preco(IDM,T,P) :: (solucoes( -preco(IDM,T,_),(-preco(IDM,T,_)),L ),
comprimento(L,N), N=<1 ).
```

Garante que o que estamos a eliminar existe:

```
-(preco(IDM,T,P)) :- (solucoes( (IDM,T,P),(preco(IDM,T,P)),L ),
comprimento(L,N), N==1 ).
```

Predicado Evolução

Extensão do predicado que permite a evolução do conhecimento verificando os diferentes invariantes acima referidos.

```
evolucaonormal( Termo ) :-
solucoes( Invariante,(+Termo::Invariante),Lista ),
insercao( Termo ),
teste( Lista ).
```

Referente ao medicamento:

```
evolucao( medicamento(ID,P,N,F,D) ) :-
solucoes( (medicamento(ID,P,algum,F,D)),(medicamento(ID,P,algum,F,D)),L1
),
solucoes( (medicamento(ID,algum,N,F,D)),(medicamento(ID,algum,N,F,D)),L2
),
solucoes( (medicamento(ID,P,N,algum,D)),(medicamento(ID,P,N,algum,D)),L3
),
solucoes( (medicamento(ID,P,N,F,algum)),(medicamento(ID,P,N,F,algum)),L4
),
solucoes(
(excepcao(medicamento(ID,P,N,F,D))),(excepcao(medicamento(ID,P,N,F,D))),L5
),
junta( L1,L2,R1 ), junta( R1,L3,R2 ), junta( R2,L4,R3 ), junta( R3,L5,RF
),
comprimento( RF,G), G>=0,
evolucaonormal( medicamento(ID,P,N,F,D)).
```

Referente ao data:

```
evolucao( data(IDM,DI,DL) ) :-
solucoes( (data(IDM,algum,DL)),(data(IDM,algum,DL)),L1 ),
solucoes( (data(IDM,DI,algum)),(data(IDF,DI,algum)),L2 ),
solucoes( (excepcao( data(IDM,DI,DL) )),(excepcao( data(IDM,DI,DL) )),L3
),
junta( L1,L2,R1 ),junta( R1,L3,RF ),
comprimento( RF,G ), G>=0,
evolucaonormal( data(IDM,DI,DL)).
```

Referente ao tipoFar:

```
evolucao( tipoFar(IDM,Tipo) ) :-
solucoes((tipoFar(IDM,algum)),(tipoFar(IDM,algum)),L1 ),
solucoes( (excepcao( tipoFar(IDM,Tipo) )),(excepcao(
tipoFar(IDM,Tipo))),L2 ),
junta( L1,L2,RF),
comprimento( RF,G ), G>=0,
evolucaonormal(tipoFar(IDM,Tipo)).
```

Referente ao preco:

```
evolucao( preco(IDM,T,P) ) :-
solucoes( (preco(IDM,T,algum)),(preco(IDM,T,algum)),L1 ),
solucoes( (preco(IDM,algum,P)),(preco(IDM,algum,P)),L2 ),
solucoes( (excepcao( preco(IDM,T,P) )),(excepcao( preco(IDM,T,P) )),L3 ),
junta( L1,L2,R1 ), junta( R1,L3,RF ),
comprimento( RF,G ), G>=0,
evolucaonormal( preco(IDM,T,P) ).
```

Predicado Demo

Para o nosso trabalho foi necessário complementar o predicado demo que aprendemos nas aulas, de modo a lidar com tipos de conhecimento imperfeito diferentes.
Foi desenvolvido o predicado de maneira a permitir a realização de conjunção e disjunção. O demo retorna verdadeiro quando a informação está contida na base de conhecimento, falso quando a informação contida na base de conhecimento é contraditória e desconhecido se não encontrar nenhuma informação contraditória nem válida.

```
demo( (A e B), falso ) :- demo( A, falso ), demo( B, falso ).
demo( (A e B), falso ) :- demo( A, falso ), demo( B, algum ).
demo( (A e B), falso ) :- demo( A, falso ), demo( B, verdadeiro ).
demo( (A e B), falso ) :- demo( A, algum ), demo( B, falso ).
demo( (A e B), desconhecido ) :- demo( A, algum ), demo( B, algum ).
demo( (A e B), desconhecido ) :- demo( A, algum ), demo( B, verdadeiro ).
demo( (A e B), falso ) :- demo( A, verdadeiro ), demo( B, falso ).
demo( (A e B), desconhecido ) :- demo( A, verdadeiro ), demo( B, algum ).
demo( (A e B), verdadeiro ) :- demo( A, verdadeiro ), demo( B, verdadeiro
).

demo( (A ou B), falso ) :- demo( A, falso ), demo( B, falso ).
demo( (A ou B), desconhecido ) :- demo( A, falso ), demo( B, algum ).
demo( (A ou B), verdadeiro ) :- demo( A, falso ), demo( B, verdadeiro ).
demo( (A ou B), desconhecido ) :- demo( A, algum ), demo( B, falso ).
demo( (A ou B), desconhecido ) :- demo( A, algum ), demo( B, algum ).
demo( (A ou B), verdadeiro ) :- demo( A, algum ), demo( B, verdadeiro ).
demo( (A ou B), verdadeiro ) :- demo( A, verdadeiro ), demo( B, falso ).
demo( (A ou B), verdadeiro ) :- demo( A, verdadeiro ), demo( B, algum ).
demo( (A ou B), verdadeiro ) :- demo( A, verdadeiro ), demo( B, verdadeiro
).

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

Predicado Retrocesso

O predicado retrocesso permite a remoção da informação inserida na base conhecimento. Este mesmo predicado terá de respeitar certos invariantes para podermos manter a nossa base de conhecimento.

```
retrocesso( Termo ) :-
solucoes( Invariante,(-Termo::Invariante),Lista ),
(remocao( Termo ); removedesconhecido( Termo)),
teste( Lista ).
```

Predicado removedesconhecido

Este predicado permite lidar com conhecimento imperfeito do tipo incerto.

```
removedesconhecido( medicamento(ID,N,P,F,Do) ) :-
retract( medicamento(ID,A,B,C,D) ),
(A == P ; A == algum),
(B == N ; B == algum),
(C == F ; C == algum),
(D == Do ; D == algum),
(A == algum ; B == algum ;
C == algum ; D == algum).

removedesconhecido( preco(IDM,T,P) ) :-
retract( preco(IDM,A,B) ),
(A == T ; A == algum),
(B == P ; B == algum),
(A == algum ; B == algum).

removedesconhecido( data(IDM,DI,DL) ) :-
retract(data(IDM, A, B)),
(A == DI ; A == algum),
(B == DL ; B == algum),
(A == algum ; B == algum).

removedesconhecido(tipoFar(IDM,T) ) :-
retract(tipoFar(IDM,A)),(A ==T ; A == algum),(A==algum).
```

Conclusão

Este trabalho prático foi realizado com a intenção de solidificar os conhecimentos adquiridos na Unidade Curricular de SRCR, nomeadamente a extensão à programação em lógica, no âmbito da representação de conhecimento imperfeito.

Foram aplicados os conhecimentos de prolog que adquirimos ao longo das aulas, ao mesmo tempo que esses mesmos conhecimentos foram continuamente desenvolvidos à medida que o trabalho ia sendo realizado, uma vez que foi necessário resolver um conjunto de novas situações e problemas.

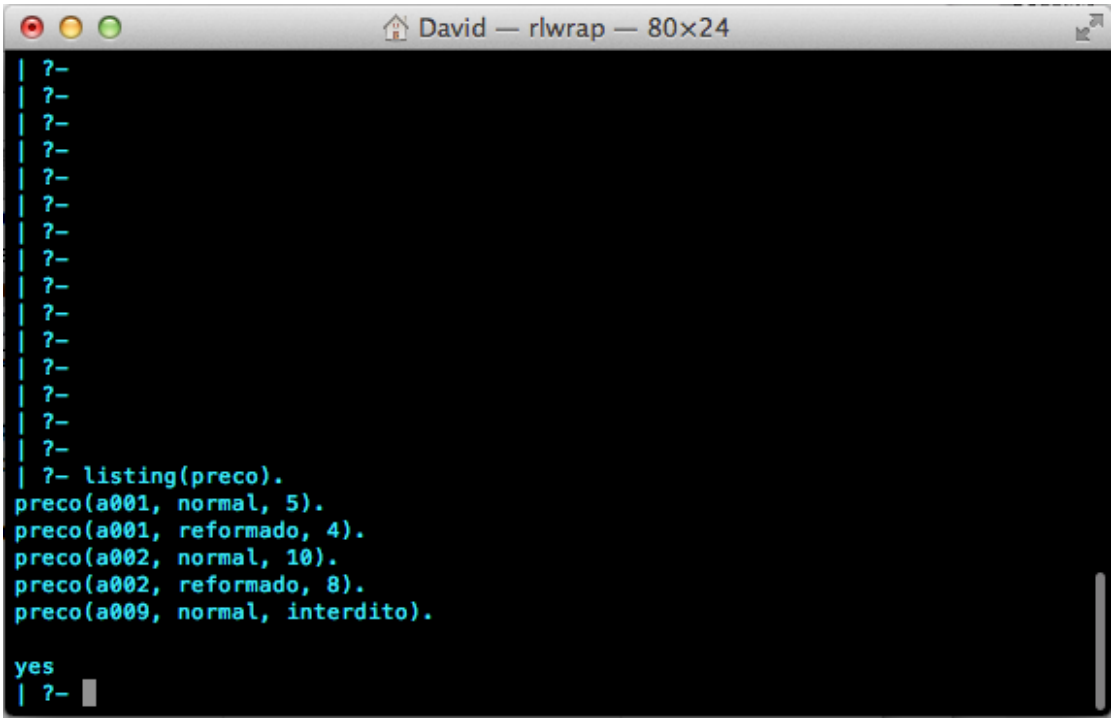
Em geral, pensamos que grupo foi capaz de analisar e resolver prontamente a maioria dos problemas apresentados neste enunciado e consequentemente realizar todos os objectivos a que se propôs aquando do começo deste trabalho.

No futuro, esperamos que os conhecimentos aqui desenvolvidos nos sejam úteis para conseguirmos implementar soluções de representação de conhecimento e raciocínio mais complexas e em contextos mais úteis.

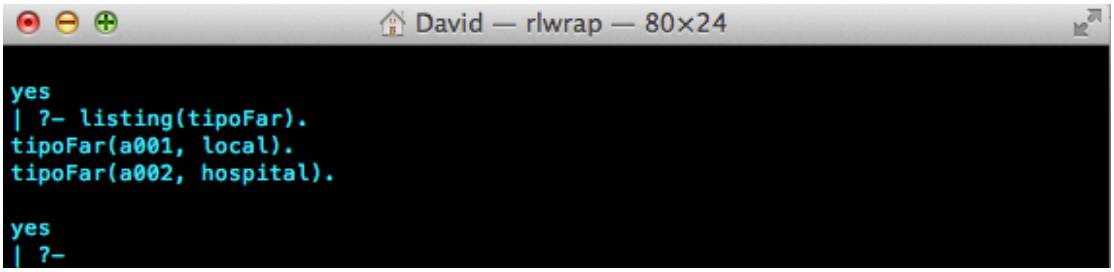
Anexos

Listagem de medicamentos:

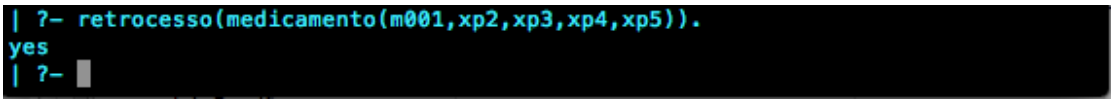
Listagem de preços:



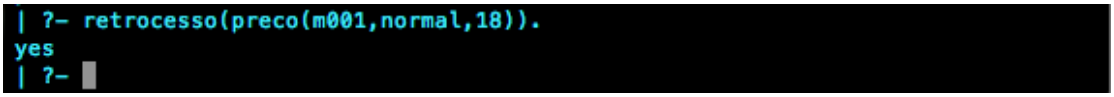
Listagem do tipo de farmácia onde se encontram os medicamentos:



Remoção de um medicamento

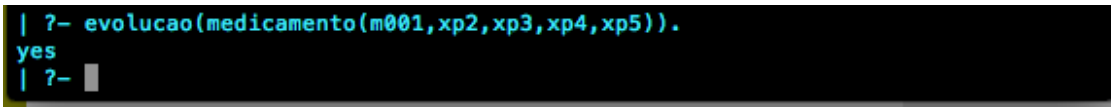


Remoção do preço de um medicamento:



Remoção do tipo de farmácia de um medicamento:

Consulta de um medicamento:



Consulta do preço de um medicamento:

Consulta do tipo de farmácia onde se pode encontrar o medicamento:

Utilização de 'ou'(disjunção):