Universidade do Minho

2º Exercício

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

(2º Semestre / 2016-2017)

A74859 João da Cunha Coelho

A74601 José Miguel Ribeiro da Silva

A73959 Pedro João Novais da Cunha

A74748 Luís Miguel Moreira Fernandes

Braga,

Março de 2017

Resumo

O presente documento tem como objetivo a documentação do segundo exercício prático da unidade curricular de *Sistemas de Representação de Conhecimento e Raciocínio*. Seguindo o modelo do primeiro exercício prático, depois de uma breve introdução, é detalhada a forma como o grupo de trabalho desenvolveu as várias funcionalidades propostas no enunciado do problema. Após a apresentação de alguns resultados, é realizada uma breve crítica ao trabalho realizado.

Índice

Introdução

O segundo exercício prático tem como objetivo a elaboração de um sistema de representação de conhecimento e raciocínio, contextualizado numa situação em que a informação existente sobre o problema não é completa, neste caso na área da prestação de cuidados de saúde.

O exercício prático incide assim sobre a representação de conhecimento imperfeito onde é introduzido um terceiro valor de verdade, o desconhecido. Fazendo uso das características da programação em lógica, em especial, da linguagem PROLOG, fica assim clara a necessidade de desenvolvimento de novos mecanismos capazes de lidar e atuar sobre este novo tipo de conhecimento.

Ao longo do exercício prático, para além do desenvolvimento dos mecanismos referidos acima, são explicitados vários predicados exemplificativos das três variantes do conhecimento imperfeito, bem como invariantes capazes de restringir a inserção e remoção destes.

Preliminares

A realização do primeiro exercício prático da unidade curricular permitiu uma maior facilidade no que diz respeito à manipulação da linguagem utilizada para a realização deste segundo trabalho, na construção de invariantes e predicados exemplificativos dos vários tipos de conhecimento. Por outro lado, a presença por parte de todos os elementos do grupo nas aulas teóricas facilitou, sobretudo, a implementação de mecanismos capazes de lidar com o novo tipo de conhecimento.

Com um papel menos relevante, breves pesquisas na web, a leitura prévia e atenta do enunciado, a revisão da matéria e a distribuição de tarefas pelos diferentes membros do grupo permitiram igualmente que a implementação das funcionalidades propostas se iniciassem de forma tranquila e se prolongasse sem sobressaltos.

Descrição do trabalho e análise dos resultados

- **Inserção de conhecimento**

À semelhança do primeiro exercício prático, o objetivo foi desenvolver um sistema de representação de raciocínio sobre a prestação de cuidados de saúde pela realização de serviços de atos médicos. Para além disso, neste caso em particular, considerámos que a informação existente sobre o problema não é completa. Como consequência, aos valores de verdade já contemplados anteriormente, verdadeiro ou falso, foi introduzido um novo, o desconhecido. Deste modo, as extensões de predicado realizadas no exercício anterior tiveram que ser ligeiramente alteradas. Apesar dos exemplos a seguir apresentados incidirem sobre conhecimento perfeito, mais à frente, no relatório, vai ser exemplificada a inserção de conhecimento imperfeito que justifica a inserção do novo valor de verdade. Assim, no que diz respeito ao conhecimento perfeito, temos:

```
% Extensão do predicado utente: IdUt, Nome, Idade, Sexo, Morada -> {V, F, D}
utente( 1, joao, 20, masculino, 'vila verde' ).
utente( 2, jose, 20, masculino, 'lousada' ).
utente( 3, josefina, 34, feminino, 'aveiro' ).
```

```
utente( 4, luis, 20, masculino, 'vila das aves' ).

utente( 5, pedro, 20, masculino, 'felgueiras' ).
```

```
% Extensão do predicado cuidadoPrestado: IdServ, Descrição, Instituição, Cidade -> {V, F, D}

cuidadoPrestado( 1, 'Medicina Familiar', 'Centro de Saude de Vila Verde', 'Vila Verde').

cuidadoPrestado( 2, 'Radiologia', 'Hospital Sao Joao', 'Porto').

cuidadoPrestado( 3, 'Medicina Familiar', 'Centro de Saude de Lousada', 'Lousada').

cuidadoPrestado( 4, 'Medicina Familiar', 'Centro de Saude de Felgueiras', 'Felgueiras').

cuidadoPrestado( 5, 'Ginecologia', 'Hospital de Braga', 'Braga').

cuidadoPrestado( 6, 'Obstetricia', 'Hospital de Braga', 'Braga').
```

```
% Extensão do predicado atoMedico: Data, IdUt, IdServ, Custo -> {V, F, D}

atoMedico( '14-03-2017', 1, 5, 30 ).

atoMedico( '12-03-2017', 3, 2, 20 ).

atoMedico( '13-03-2017', 4, 4, 5 ).

atoMedico( '14-03-2017', 2, 3, 5 ).

atoMedico( '04-04-2017', 1, 3, 7 ).
```

Os exemplos práticos apresentados em cima referem-se a conhecimento perfeito positivo e apenas diferem no contra-domínio, comparando com o que foi realizado no exercício prático anterior, uma vez que agora existe o terceiro valor de verdade. No entanto, para o presente exercício prático um dos objetivos passou por ser igualmente possível a inserção de conhecimento perfeito negativo. Assim, foi inserido conhecimento perfeito negativo para os três principais predicados, como podemos verificar a seguir:

```
-utente( 17, laura, 24, feminino, 'Faro' ).
-utente( 34, ricardo, 38, masculino, 'Guarda' ).
-Utente( 72, alberto, 83, masculino, 'Setubal' ).
```

Com os predicados explicitados acima, é possível representar que determinado utente não se encontra presente na base de conhecimento. O mesmo pode ser assumido para os seguintes predicados, correspondentes aos cuidados prestados e atos médicos, respetivamente:

```
-cuidadoPrestado( 30, 'Medicina Familiar', 'Hospital de São João', 'Porto').
-cuidadoPrestado( 45, 'Obstetrícia', 'Hospital do Algarve', 'Porto').
-cuidadoPrestado( 98, 'Ginecologia', 'Centro de Saude de Felgueiras', 'Felgueiras').
```

```
-atoMedico( '31-01-2017', 5, 3, 13 ).
-atoMedico( '12-07-2017', 3, 4, 43 ).
-atoMedico( '06-01-2018', 2, 1, 59 ).
-atoMedico( '24-12-2017', 6, 5, 2 ).
```

Apesar de terem sido explicitados alguns exemplos práticos de conhecimento perfeito negativo, para os predicados utente e atoMedico foi utilizado o Pressuposto do Mundo Fechado, que assume a não existência de qualquer utente/atoMedico não definido no universo de discurso, mediante a criação de uma exceção.

```
% Pressuposto do mundo fechado para o predicado utente
-utente(Id,N,I,S,C) :-
    nao(utente(Id,N,I,S,C)),
    nao(excecao(utente(Id,N,I,S,C))).

% Pressuposto Mundo Fechado para o predicado atoMedico
-atoMedico(D,Id,IdS,C) :-
    nao(atoMedico(D,Id,IdS,C)),
    nao(excecao(atoMedico(D,Id,IdS,C))).
```

Já para o predicado cuidadoPrestado, a ausência do PMF leva a que apenas os casos representados como conhecimento negativo na base de conhecimento sejam interpretados como a inexistência do dado cuidado numa certa instituição.

Importa, também, destacar desde já o uso do predicado auxiliar *nao*.

Tal como no primeiro exercício prático, é necessário controlar a inserção e remoção de utentes, cuidados prestados e atos médicos. A inserção de conhecimento perfeito negativo levou inevitavelmente à criação de novos invariantes estruturais e referenciais de inserção e remoção. Mas antes é importante expor os predicados auxiliares utilizados e o que se alterou relativamente à primeira fase:

- o predicado que se segue representa a negação fraca;

```
% Extensao do meta-predicado nao: Questao -> {V,F}
nao( Questao ) :- Questao, !, fail.
nao( Questao ).
```

- sistema de inferência que recebe uma única questão;

```
%Extensao do meta-predicado demo: Questao,Resposta -> {V,F,D}

demo(Questao, verdadeiro) :- Questao.

demo(Questao, falso) :- -Questao.

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

- sistema de inferência capaz de dar resposta a um conjunto de *queries*, utilizando para isso os valores de verdade resultantes da conjunção/disjunção dos resultados das queries. Nota para o facto de, na lista das questões, estarem também incluídos os operadores a usar entre cada par de questões. Temos assim uma lista do género "[utente(…), e, utente(…), ou, …]", como é possível observar na imagem que se segue;

```
% Extensão do meta-predicado myDemo: [Queries],Resposta->{V,F,D}

myDemo( [], verdadeiro ).

myDemo( [Q], Resposta ) :- demo( Q, Resposta ).

myDemo( [Q1, Op | T],Resposta ) :- Op == e,
                                   demo( Q1, V1 ),
                                   myDemo( T, V2 ),
                                   conjuncao(V1,V2,Resposta).

myDemo( [Q1, Op | T],Resposta ) :- Op == ou,
                                   demo( Q1, V1 ),
                                   myDemo( T, V2 ),
                                   disjuncao(V1,V2,Resposta).
```

```
%Extensão do meta-predicado conjuncao: Q1,Q2,Resposta -> {V,F,D}
   conjuncao( verdadeiro, verdadeiro, verdadeiro ).
   conjuncao( verdadeiro, desconhecido, desconhecido ).
   conjuncao( desconhecido, verdadeiro, desconhecido ).
   conjuncao( desconhecido, desconhecido, desconhecido ).
   conjuncao( falso, _, falso ).
   conjuncao( _, falso, falso ).


%Extensão do meta-predicado disjuncao: Q1,Q2,Resposta -> {V,F,D}
   disjuncao( verdadeiro, _, verdadeiro ).
   disjuncao( _, verdadeiro, verdadeiro ).
   disjuncao( falso, falso, falso ).
   disjuncao( falso, desconhecido, desconhecido ).
   disjuncao( desconhecido, falso, desconhecido ).
   disjuncao( desconhecido, desconhecido, desconhecido ).
```

Estas meta-predicados foram concebidos assumindo a seguinte tabela de verdade:

1° argumento	2° argumento	Resultado Conjunção	Resultado Disjunção
Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Desconhecido	Desconhecido	Verdadeiro
Desconhecido	Verdadeiro	Desconhecido	Verdadeiro
Desconhecido	Desconhecido	Desconhecido	Desconhecido
Falso	Falso	Falso	Falso

```
| ?- myDemo([utente(1, joao, 20, masculino, 'vila verde'), e, utente(2, jose, 20, feminino, 'lousada'), e, utente(2, jose, 20, masculino, 'lousada'),e, atoMedico('14-
03-2017',1,5,30)], Resposta).
Resposta = falso ?
yes
| ?- myDemo([utente(1, joao, 20, masculino, 'vila verde'), e, utente(2, jose, 20, feminino, 'lousada'), e, utente(2, jose, 20, masculino, 'lousada'),ou, atoMedico('14
-03-2017',1,5,30)], Resposta).
Resposta = falso ?
yes
| ?- myDemo([utente(1, joao, 20, masculino, 'vila verde'), e, utente(2, jose, 20, feminino, 'lousada'), ou, utente(2, jose, 20, masculino, 'lousada'),ou, atoMedico('1
4-03-2017',1,5,30)], Resposta).
Resposta = verdadeiro ?
yes
| ?-
```

- sistema de inferência que recebe uma lista de questões e é capaz de fornecer a lista de respostas às mesmas;

```
%Extensão do meta-predicado demoLista: [Qs],[Resposta]->{V,F,D}

demoLista( [],[] ).

demoLista( [X|L],[R|S] ) :- demo( X,R ),
                            demoLista( L,S ).
```

- sistemas de inferência que dão resposta à conjunção ou disjunção, respetivamente, de um conjunto de *queries*:

```
% Extensão do meta-predicado demoConj: [Qs], Resposta -> {V,F,D}

demoConj( [],verdadeiro ).
```

```
demoConj( [X|Y], verdadeiro ) :-
    demo( X, verdadeiro ),
    demoConj( Y, verdadeiro ).

demoConj( [X|Y], falso ) :-
    demo( X, falso ),
    demoConj( Y, Z ).

demoConj( [X|Y], falso ) :-
    demo( X, Z ),
    demoConj(Y, falso).

demoConj( [X|Y], desconhecido ) :-
    demo( X, desconhecido ),
    nao(demoConj( Y, falso )).

demoConj( [X|Y], desconhecido ) :-
    nao(demo( X, falso )),
    demoConj( Y, desconhecido ).

% Extensão do meta-predicado demoDisj: [Qs], Resposta -> {V,F,D}
demoDisj( [],falso ).
demoDisj( [X|Y], falso ) :- demo(X, falso ),
```

```
                              demoDisj(Y, falso ).
demoDisj( [X|Y], verdadeiro ) :-
    demo( X, verdadeiro ),
    demoDisj( Y, Z ).

demoDisj( [X|Y], verdadeiro ) :-
    demo( X, Z ),
    demoDisj( Y, verdadeiro ).

demoDisj( [X|Y], desconhecido ) :-
    demo( X, desconhecido ),
    nao(demoDisj( Y, verdadeiro )).

demoDisj( [X|Y], desconhecido ) :-
    nao(demo( X, verdadeiro ),
    demoDisj( Y, desconhecido )).
```

- predicados auxiliares já utilizados no primeiro exercício prático;

```
% Predicado solucoes: F, Q, S -> {V,F}
solucoes(T,Q,S) :- Q, assert(tmp(T)), fail.
solucoes(T,Q,S) :- construir(S, []).

construir(S1, S2) :- retract(tmp(X)),
                     !,
```

```
                    construir(S1, [X|S2]).
construir(S, S).


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

De forma a permitir evolução correspondente a conhecimento negativo, o predicado evolução teve que sofrer alterações. Essas alterações resultaram em:

```
evolucao( F ) :-
    solucoes(I, +F::I, Li),
    testar(Li),
    assert(F).



evolucao( F, Type ) :-
    Type == positivo,
    solucoes(I, +F::I, Li),
    testar(Li),
    assert(F).

evolucao( F, Type ) :-
    Type == negativo,
    solucoes( I, +(-F)::I, Li ),
    testar(Li),
```

```
        assert(-F).
```

Também foi necessário possibilitar a evolução correspondente a conhecimento imperfeito. Relativamente a conhecimento imperfeito incerto, adicionaram-se mais termos ao predicado evolução. Será usado como exemplo o termo que permite adicionar um utente cuja cidade é desconhecida. Todos os restantes são análogos, por isso depois de perceber o exemplo, todos são facilmente percebidos. Exemplo:

```
% permite cidade desconhecida
evolucao( utente( Id,No,Idd,Se,Cid ), Type, Desconhecido ) :-
    Type == incerto,
    Desconhecido == cidade,
    evolucao( utente( Id,No,Idd,Se,Cid ), positivo ),
    assert( (excecao(utente( IdUt,Nome,Idade,Sexo,Cidade )) :-
                utente( IdUt,Nome,Idade,Sexo,Cid ) ) ).
```

Os restantes permitem: utente com idade desconhecida; cuidado prestado com instituição desconhecida; ato médico com custos desconhecidos; ato médico com serviços desconhecidos.

Para o conhecimento imperfeito impreciso, foram adicionados os seguintes termos:

```
   evolucao( [OPT1 | R], Type ) :-
       Type == impreciso,
       solucoes( I, +(excecao(OPT1))::I, Li ),
       testar(Li),
       assert( (excecao( OPT1 )) ),
```

```
        evolucao( R,impreciso ).


    evolucao( [], impreciso ).
```

Representa-se assim o conhecimento relativo a todas as possibilidades na lista.

Ainda para o conhecimento imperfeito impreciso, por exemplo para o utente:

```
    evolucao( utente( Id,No,Idd,Se,Cid ), Type, Impreciso,
MenorValor, MaiorValor ) :-
        Type == impreciso,
        Impreciso == idade,
        solucoes( I, +(excecao(utente( Id,No,Idd,Se,Cid ))::I,
Li ),
        testar(Li),
        assert( (excecao( utente( Id,No,Idd,Se,Cid ) ) :-
                    Idd >= MenorValor,
                    Idd =< MaiorValor ) ).
```

O termo acima permite a evolução relativa a um utente sobre o qual o conhecimento da sua idade é impreciso, estando entre dois valores. Existe também o caso do ato médico, sendo que neste caso é o custo que está entre dois valores.

Por fim, o conhecimento imperfeito interdito. Mais uma vez será utilizado o utente como exemplo, no caso em que não se sabe nem é possível saber a cidade:

```
    % permite cidade interdita
    evolucao( utente( Id,No,Idd,Se,Cid ), Type, Desconhecido ) :-
```

```
        Type == interdito,
        Desconhecido == cidade,
        evolucao( utente( Id,No,Idd,Se,Cid ),positivo ),
        assert(  (excecao(utente(  IdUt,Nome,Idade,Sexo,Cidade  ))
:-
                    utente( IdUt,Nome,Idade,Sexo,Cid ) ) ),
        assert( (nulo(Cid)) ).
```

Os restantes termos, análogos a este: idade do utente, instituição do cuidado prestado; custos dos ato médico; serviços do ato médico.

Havendo agora conhecimento negativo e imperfeito, tal como o predicado evolução, o predicado involução também sofreu alterações.

Para o conhecimento negativo, a alteração é semelhante à do predicado evolução:

```
% involucao: F -> {V,F}
involucao( F ) :- solucoes(I, -F::I, Li),
                  testar(Li),
                  retract(F).

% involucao: F, Type -> {V,F}
involucao( F,Type ) :-
    Type == positivo,
    solucoes(I, -F::I, Li),
        testar(Li),
        retract(F).
```

```
involucao( F,Type ) :-
    Type == negativo,
    solucoes(I, -(-F)::I, Li),
        testar(Li),
        retract(-F).
```

Quanto ao conhecimento imperfeito temos os mesmos casos da evolução:

Incerto: Utente com cidade desconhecida; utente com idade desconhecida; cuidado prestado com instituição desconhecida; ato médico com custos desconhecidos; ato médico com serviços desconhecidos. Exemplo:

```
% permite remover conhecimento incerto sobre idade de utentes
involucao( utente( Id,No,Idd,Se,Cid ),Type,Desconhecido ) :-
    Type == incerto,
    Desconhecido == idade,
    involucao( utente( Id,No,Idd,Se,Cid ), positivo ),
    retract( (excecao(utente( IdUt,Nome,Idade,Sexo,Cidade )) :-
                utente( IdUt,Nome,Idd,Sexo,Cidade )) ).
```

Impreciso: Como se viu até agora, as alterações são muito parecidas com as do predicado evolução. Este caso não é exceção, permite remover várias possibilidades numa lista ou então (no caso da idade do utente ou custo do ato médico) todas entre dois valores. Implementação apenas com o exemplo da idade do utente:

```
involucao( [OPT1 | R], Type ) :-
    Type == impreciso,
    solucoes( I, -OPT1::I, Li ),
    testar(Li),
    retract( (excecao( OPT1 )) ),
    involucao( R,impreciso ).
```

```
involucao( [], impreciso ).


involucao(   utente(   Id,No,Idd,Se,Cid   ),   Type,   Impreciso,
MenorValor, MaiorValor ) :-
    Type == impreciso,
    Impreciso == idade,
    solucoes( I, -(excecao(utente( Id,No,Idd,Se,Cid ))::I, Li
),
    testar(Li),
    retract( (excecao( utente( Id,No,Idd,Se,Cid ) ) :-
                Idd >= MenorValor,
                Idd =< MaiorValor ) ).
```

Interdito: Cidade do utente; idade do utente; instituição do cuidado prestado; custos dos ato médico; serviços do ato médico. Exemplo:

```
% permite remover conhecimento interdito sobre a cidade de
utentes
involucao( utente( Id,No,Idd,Se,Cid ), Type, Desconhecido ) :-
    Type == interdito,
    Desconhecido == cidade,
    retract( (nulo(Cid)) ),
    involucao( utente( Id,No,Idd,Se,Cid ),incerto,cidade ).
```

No que diz respeito aos invariantes, neste segundo exercício recorreu-se a uma abordagem diferente, relativamente ao primeiro exercício prático. Seguindo o conselho do docente, os invariantes são agora responsáveis por permitir ou impedir a adição/remoção de conhecimento, ao invés de verificarem o cumprimento ou incumprimento dos invariantes apenas após a remoção/inserção.

Se no exercício passado foi realizado um invariante que garantia, na inserção de conhecimento perfeito positivo, a unicidade dos Ids dos utentes, o mesmo teve que ser realizado neste exercício. Isto não é apenas verificado para o predicado utente mas também para predicado cuidadoPrestado, não acontecendo para o atoMedico devido à ausência de um identificador único.

```
% Garantia de unicidade nos Ids dos utentes – conhecimento perfeito positivo

    +utente( IdUt,Nome,Idade,Sexo,Morada )::(solucoes( (IdUt), utente(IdUt,N,I,Se,M), S ), comprimento( S,N ), N == 0 ).


% Garantia de unicidade nos Ids dos Serviços – conhecimento perfeito positivo

    +cuidadoPrestado(IdServ,Desc,Inst,Cid)    ::    (solucoes(
    (IdServ),cuidadoPrestado(IdServ,D,I,C),S),
    comprimento(S,N), N == 0).
```

Dado que não foi declarado o PMF para o predicado cuidadoPrestado, é necessário adicionar à base de conhecimento toda a informação não verdadeira conhecida, o que implica a verificação da unicidade, não do Id do serviço, mas do conjunto dos termos do predicado.

```
% Garantia de unicidade na negação do cuidadoPrestado

    +(-cuidadoPrestado(IdServ,Desc,Inst,Cid))  ::  (solucoes(
    (IdServ),-cuidadoPrestado(IdServ,Desc,Inst,Cid),S),
    comprimento(S,N ), N == 0).
```

Para os atos médicos também foi verificada a unicidade, mas de todo o predicado.

```
% Garantir que não existe conhecimento positivo repetido

+atoMedico( Data,IdUt,IdServ,Custo ) :: ( solucoes(
(IdUt,IdServ), atoMedico( Data,IdUt,IdServ,Custo ), S ),
comprimento( S, N ), N == 0 ).
```

Do exercício prático anterior foram também recuperados os seguintes invariantes referenciais de remoção, para além do invariante de inserção que impede a adição de atos médicos para utentes e/ou cuidados não presentes na base de conhecimento.

```
% Não é possível a remoção de utentes se houver algum Ato Médico
para este

-utente( IdUt,Nome,Idade,Sexo,Cidade ) :: ( solucoes( (IdUt),
atoMedico( Data,IdUt,IdServ,Custo ), S ), comprimento( S,N ), N
== 0 ).

% Não é possível a remoção de Serviços se houver algum Ato
Médico marcado que o use

-cuidadoPrestado( IdServ,Desc,Inst,Cid ) :: ( solucoes(
(IdServ), atoMedico( Data,IdUt,IdServ,Custo ), S ), comprimento(
S,N ), N == 0 ).
```

```
% Apenas é possível inserir um atoMedico em que o IdUt esteja registado nos Utentes e o IdServ esteja registado nos Cuidados Prestados

+atoMedico( Data,IdUt,IdServ,Custo ) :: ( solucoes( (IdUt), utente( IdUt,Nome,Idade,Sexo,Morada ), S1 ), comprimento( S1,N1 ), N1 == 1, solucoes( (IdServ), cuidadoPrestado( IdServ,Descricao,Instituicao,Cidade ), S2 ), comprimento( S2,N2 ), N2 == 1 ).
```

A possibilidade de inserção de conhecimento perfeito negativo trouxe igualmente a necessidade de garantir que não existe conhecimento contraditório, ou seja, não é possível a inserção de conhecimento perfeito negativo se na base de conhecimento já existir conhecimento perfeito positivo exatamente com a mesma informação (ou apenas o mesmo Id, nos predicados em que existe). O mesmo se verifica quando existe conhecimento perfeito negativo e queremos inserir conhecimento perfeito positivo com a mesma informação. Para o cuidadoPrestado há a necessidade de um invariante que precave esta situação, porque toda a informação negativa está declarada. Já nos restantes predicados, o PMF faz com que tudo o que não seja conhecimento perfeito positivo declarado seja assumido como falso, pelo que um invariante do género impediria inserções de conhecimento positivo na base de conhecimento. Assim, temos:

```
% Garante que não existe conhecimento positivo contraditótio

+(-utente(IdUt,Nome,Idade,Sexo,Cidade)) :: (solucoes((IdUt), utente(IdUt,Nome,Idade,Sexo,Cidade), S ), comprimento( S,N ), N == 0).

% Garante que não existe conhecimento negativo contraditótio

+cuidadoPrestado(IdServ,Desc,Inst,Cid) :: (solucoes((IdServ), -cuidadoPrestado(IdServ,Desc,Inst,Cid), S ), comprimento( S, N ), N ==0).

% Garante que não existe conhecimento positivo contraditótio
```

```
+(-cuidadoPrestado(IdServ,Desc,Inst,Cid))      ::      (solucoes(
(IdServ),cuidadoPrestado(IdServ,Desc,Inst,Cid), S ),comprimento(
S, N ), N == 0 ).


% Garante que não existe conhecimento positivo contraditótio

+(-atoMedico(Data,IdUt,IdServ,Custo))        ::        (solucoes(
(IdUt,IdServ),atoMedico(Data,IdUt,IdServ,Custo),S), comprimento(
S, N ),N == 0 ).
```

Também para manter a coerência da base de conhecimento, foram adicionados invariantes que controlam a inserção de conhecimento relativo a certos fatores que se encaixam no campo do desconhecido.

```
% Invariante que impede a inserção de conhecimento positivo ou
negativo acerca de conhecimento interdito sobre a cidade de
utentes

+utente( Id,No,I,Se,C ) :: (solucoes( (Id,No,I,Se,C), (utente(
Id,No,I,Se,xpto ), nulo(xpto)), S ), comprimento( S,N ),N == 0).

+(-utente( Id,No,I,Se,C )) :: (solucoes( (Id,No,I,Se,C),(utente(
Id,No,I,Se,xpto ), nulo(xpto)), S ), comprimento( S,N ),N == 0).


% Invariante que permite a inserção de utentes se não houver
exceção relativa à cidade

+utente( IdUt,Nome,Idade,Sexo,Cidade ) :: ( solucoes(
excecao(utente(IdUt,Nome,Idade,Sexo,Cid))),excecao(utente(IdUt,N
ome,Idade,Sexo,Cid)), S ),comprimento( S,N ), N == 0 ).
```

De referir que estes invariantes são replicados para os vários termos suscetíveis de serem desconhecidos. Neste exemplo constam exceções relativas à cidade de um utente, porém poderiam ser relativas à idade ou ao sexo, por exemplo. Por outro lado, também importa referir que para os restantes predicados (cuidadoPrestado e atoMedico) requerem igualmente a definição de invariantes equivalentes a estes, para os seus termos cujo valor pode ser desconhecido. Esta partilha de invariantes equivalentes entre os três predicados que compõem o universo de discurso acontece também com os seguintes, que controlam a inserção direta de exceções. Uma vez mais, o exemplo é aplicado ao predicado utente, nomeadamente ao desconhecimento da cidade.

```
% Garantir que não se adicionam exceções relativas à cidade a
conhecimento perfeito positivo.

+excecao(utente(Id,No,I,Se,C))::(nao(utente(Id,No,I,Se,Cidade)))
.

% Garantia da não inserção de exceções repetidas.

+(excecao(utente(Id,No,I,Se,C)))::(solucoes(
(excecao(utente(Id,No,I,Se,C))),  excecao(utente(Id,No,I,Se,C)),
S), comprimento(S,N), N == 0).
```

- **Inserção de conhecimento imperfeito**

Como já foi dito anteriormente, o presente trabalho prático não assenta apenas sobre conhecimento perfeito. Através do novo valor de verdade – desconhecido - é agora possível a inserção de conhecimento incerto, impreciso e interdito.

No que diz respeito ao **conhecimento imperfeito do tipo interdito,** este ocorre quando determinado facto não se conhece e permanecerá sempre uma incógnita. Seguem-se dois exemplos:

```
% Não se sabe nem é possível vir a saber a morada do utente
% António Costa.

utente(11, 'Antonio Costa', 55, masculino, int0001).

excecao(utente(Id,Nome,Idade,Sexo,Morada))    :-    utente(
Id,Nome,Idade,Sexo,int0001).
```

```
nulo(int0001).
```

```
% Não se sabe nem é possível vir a saber qual o serviço prestado ao utente cujo % IdUt é 10 no dia 20 de Março de 2017 e cujo preço foi 3000€.
```

```
atoMedico( '20-03-2017', 11, int0002, 3000 ).
excecao(atoMedico(Data,IdUt,IdServ,Custo)) :- atoMedico( Data,IdUt,int0002,Custo ).
nulo(int0002).
```

Como podemos verificar por os exemplos acima, os valores desconhecidos são previamente definidos como sendo valores nulos. Tomando como exemplo o primeiro predicado, para o caso do utente 'António Costa' a morada é desconhecida, sendo definida em int0001, como um valor nulo. Para além disto, é necessário garantir que não é possível alterar o valor nulo, referido anteriormente, através de uma inserção. Apesar de já possuirmos um invariante que garante a unicidade dos Ids para os utentes e assim impede a alteração do valor nulo, decidimos, seguindo a metodologia proposta nas aulas práticas, criar invariantes especialmente realizados para cobrir este caso em específico:

```
% Invariante que impede a inserção de conhecimento positivo ou negativo acerca de conhecimento interdito sobre a cidade de utentes
+utente(Id,No,I,Se,C) :: (solucoes((Id,No,I,Se,C),(utente( Id,No,I,Se,xpto ),nulo(xpto)),S),comprimento( S,N ),N==0).
```

```
+(-utente(Id,No,I,Se,C)):: (solucoes((Id,No,I,Se,C), (utente(Id,No,I,Se,xpto),nulo(xpto)),S),comprimento(S,N),
N == 0).
```

No que diz respeito ao **conhecimento imperfeito do tipo impreciso** este ocorre quando determinado facto não se conhece mas a dúvida incide sobre um determinado intervalo de incerteza. Seguem-se vários exemplos:

```
% Não se sabe se o/a utente Dolores é do sexo masculino ou
feminino.
     excecao(utente( 7, dolores, 34, masculino, 'Amadora' )).
     excecao(utente( 7, dolores, 34, feminino, 'Amadora' )).

% Não se sabe se o utente Zeca tem 36 ou 37 anos, nem se é da
Amadora ou de Sintra.
     excecao(utente( 8, zeca, 36, masculino, 'Sintra' )).
     excecao(utente( 8, zeca, 37, masculino, 'Sintra' )).
     excecao(utente( 8, zeca, 36, masculino, 'Amadora' )).
     excecao(utente( 8, zeca, 37, masculino, 'Amadora' )).


% Não se sabe se o utente Alfredo é de Felgueiras ou Lousada.
     excecao(utente( 9, alfredo, 22, masculino, 'felgueiras' )).
     excecao(utente( 9, alfredo, 22, masculino, 'lousada' )).

% Não se sabe se a utente Alzira tem 23 ou 24 anos.
     excecao(utente( 10, alzira, 23, feminino, 'braga' )).
     excecao(utente( 10, alzira, 24, feminino, 'braga' )).
% Não se sabe o preço da consulta que ocorreu no dia 29 de Abril
% de 2017, cujo utente tem o IdUt 2 e o serviço prestado tem o
% IdServ 3, mas sabe-se que o preço foi entre os 3 e os 17
euros.

     excecao(atoMedico('29-04-2017',2,3,C)) :- C>=3, C=<17.
```

Conclusões e Sugestões

A resolução do segundo exercício prático permitiu-nos consolidar os conhecimentos relativos à representação de conhecimento perfeito positivo e negativo e relativos aos diversos tipos de conhecimento imperfeito, utilizando mais uma vez a linguagem de programação PROLOG.

Ao contrário do exercício prático realizado anteriormente, o foco do grupo de trabalho incidiu sobretudo na construção de um sistema de inferência, de mecanismos capazes de lidar com a evolução e invariantes restritivos ajustados ao novo tipo de conhecimento e não tanto na diversidade de predicados capazes de aumentar a representação do universo do problema. Apesar disso, a representação de conhecimento foi realizada de modo completo, sendo capaz de exemplificar de forma elucidativa e completa, para os três predicados base do enunciado do problema, todos os tipos de conhecimento estudados na unidade curricular.

De um modo geral, fazemos uma avaliação positiva do exercício prático realizado, já que todas as funcionalidades propostas no enunciado do problema foram implementadas com sucesso.

Referências

- “PROLOG: Programming for Artificial Intelligence”, Ivan Bratko;

- “Representação de Informação Incompleta”, Cesar Analide, José Neves;

- “Sugestões para a Redacção de Relatórios Técnicos, Cesar Analide, Paulo Novais, José Neves;

Anexos