Universidade do Minho

1º 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 primeiro exercício prático da unidade curricular de *Sistemas de Representação de Conhecimento e Raciocínio*. Após uma breve introdução, é detalhada a forma como o grupo de trabalho solucionou as várias alíneas do problema proposto. Após a apresentação de alguns resultados, é realizada uma breve crítica do trabalho realizado.

Índice

Introdução

A programação em lógica assume-se como um tipo específico de programação, bastante distinta das linguagens mais convencionais como C++ e Java. Para além de ser considerada bastante mais simples de utilizar é, geralmente, descrita como uma linguagem declarativa, apesar de ter inevitavelmente algumas características procedimentais (série de instruções executadas umas após as outras). Assim, ao contrário de especificarmos a maneira de como atingir um certo objetivo, especificamos a situação (existem dois componentes em qualquer programa: factos e regras) e o objetivo esperado é que o interpretador encontre a solução.

O primeiro exercício prático tem como objetivo a elaboração de um sistema de representação de conhecimento e raciocínio capaz de caracterizar o discurso na área da prestação de cuidados de saúde. Fazendo uso das características da programação em lógica, em especial, da linguagem PROLOG, e manipulando invariantes que designem restrições à inserção e remoção de conhecimento, são solucionadas as alíneas propostas pelo problema e adicionadas funcionalidades adicionais.

Preliminares

A presença de todos os elementos do grupo de trabalho nas aulas teóricas e práticas da unidade curricular permitiu uma maior facilidade no que diz respeito à manipulação da linguagem utilizada para a realização do exercício prático. Para além disto, 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, permitiu que a resolução do problema proposto se iniciasse de forma tranquila e se prolongasse sem sobressaltos.

Descrição do trabalho e análise dos resultados

- **Inserção de conhecimento**

Ao longo da realização do exercício prático, uma vez que o objetivo foi desenvolver um sistema de representação de raciocínio sobre a prestação de cuidados de saúde pela realização de serviços de atos médicos, foram desenvolvidas as seguintes extensões de predicados:

```
% Extensão do predicado utente:
%            IdUt, Nome, Idade, Sexo, Morada -> {V, F}

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}
```

```
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}


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 ).
```

Estes foram os três factos introduzidos que serviram como base para as regras que respondem à maioria das funcionalidades do enunciado. Nota para a subtil alteração do predicado “utente”, comparativamente ao do enunciado, com a introdução do parâmetro “Sexo” que identifica o género do utente.

Para controlar a inserção e remoção de utentes, cuidados prestados e atos médicos, foram criados invariantes estruturais e referenciais de inserção e remoção, os quais apresentaremos de seguida, mas antes é importante expor os predicados auxiliares utilizados. Estes são predicados criados nas aulas pelo docente e que ofereceram suporte à representação de conhecimento ao longo do exercício prático:

```
% Predicado solucoes:
%                T, Q, S -> {V,F}

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



% Predicado construir:
%                S1, S1 -> {V,F}

construir(S1, S2) :- retract(tmp(X)),
                     !,
                     construir(S1, [X|S2]).
construir(S, S).

% Predicado comprimento:
%                L, Comp -> {V, F}

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

% Predicado evolucao:
%                F -> {V,F}

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

```
                    fail.

% Predicado testar:
%                L -> {V,F}
testar([]).
testar([I|Li]) :- I,
                  testar(Li).

% Predicado involucao:
%                F -> {V,F}

involucao( F ) :- solucoes(I, -F::I, Li),
                  retract(F),
                  testar(Li).
involucao( F ) :- assert( F ),
                  !,
                  fail.
```

Agora sim, apresentamos os invariantes:

Tal como foi falado nas aulas práticas, é necessário criar invariantes que definam regras de evolução da base de conhecimento (de salientar que evolução aqui é referente a inserção ou remoção de conhecimento).

A forma de funcionamento do predicado evolucao (inserção de conhecimento) passa por inserir o conhecimento pretendido, testar o invariante e, se este não se verificar, remover o conhecimento anteriormente adicionado. De forma análoga, o predicado involucao (remoção de conhecimento) remove o conhecimento pretendido, testa o invariante e, caso não se verifique, volta a inserir o conhecimento anteriormente removido.

Quanto à evolução da base de conhecimento que satisfaz o predicado **utente,** foram criados dois invariantes, um para inserção de conhecimento e outro para a remoção.

```
% Invariante Estrutural (Alínea 1) e 9))
```

```
% Garantia de unicidade nos Ids dos utentes

+utente( IdUt,Nome,Idade,Sexo,Morada )::(solucoes( (IdUt),
utente(IdUt,N,I,Se,M), S ), comprimento( S,N ), N == 1 ).

% Invariante Referencial

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

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

O invariante para a inserção impede que sejam inseridos utentes com Ids iguais, garantindo a unicidade dos utentes e formando assim um invariante estrutural.

```
| ?-
| ?- listing(utente).
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).

yes
% source_info
| ?- evolucao(utente(1, joao, 20, masculino, 'vila verde')).
no
% source_info
| ?- listing(utente).
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).
utente(1, joao, 20, masculino, 'vila verde').

yes
% source_info
| ?-
```

O invariante dá uso ao predicado solucoes e verifica se, após a inserção do conhecimento pretendido, existe apenas um utente na base de conhecimento com o Id que foi inserido. Caso não seja satisfeito, este conhecimento é removido.

Quanto ao invariante para a remoção, este impede que um utente com um ato médico seja removido da base de conhecimento, sendo este um invariante referencial.

```
% source_info
| ?- listing(atoMedico).
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).
atoMedico('14-03-2017', 1, 5, 30).

yes
% source_info
| ?- involucao(utente(1,joao, 20, masculino, 'vila verde')).
no
% source_info
| ?- listing(utente).
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).
utente(1, joao, 20, masculino, 'vila verde').

yes
% source_info
| ?-
```

Passando à satisfação do predicado cuidadoPrestado, foram criados três invariantes, sendo, destes, dois para inserção e um para remoção de conhecimento.

```
% Invariante Estrutural (Alínea 1) e 9))

% Garantia de unicidade nos Ids dos Serviços

+cuidadoPrestado( IdServ,Desc,Inst,Cid )::( solucoes( (IdServ), cuidadoPrestado(IdServ,D,I,C), S ), comprimento( S,N ), N == 1).


% Apenas é possível a inserção de um Serviço numa certa instituição uma vez

+cuidadoPrestado( IdServ,Desc,Inst,Cid )::( solucoes( (Desc,Inst), cuidadoPrestado(V,Desc,Inst,C), S ), comprimento( S, N), N == 1 ).


% Invariante Referencial

% 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 ).
```

Tal como para o predicado utente, foi criado um invariante estrutural que garante a unicidade dos Ids dos cuidados médicos.

Para além deste, foi também criado um invariante estrutural de inserção que garante que não é repetido conhecimento na base, não permitindo que seja inserida uma certa especialidade de uma certa instituição mais que uma vez, mesmo tendo Ids diferentes.

E, mais uma vez como no predicado utente, foi criado um invariante referencial que garante que não é removido da base de conhecimento um cuidado que seja utilizado num ato médico.

Quanto ao predicado atoMedico, foi criado um único invariante estrutural de inserção responsável por garantir que não é inserido um atoMedico em que o IdUt (Id do utente) e o IdServ (Id do Serviço) sejam Ids não atribuídos a utentes e serviços, respetivamente, na base de conhecimento.

```
% Invariante Estrutural (Alínea 1) e 9))

% 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 ).
```

```
% source_info
| ?- listing(atoMedico).
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).
atoMedico('14-03-2017', 1, 5, 30).

yes
% source_info
| ?- evolucao('15-03-2017', 7, 5, 10).
no
% source_info
| ?- listing(atoMedico).
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).
atoMedico('14-03-2017', 1, 5, 30).

yes
% source_info
| ?-
```

- **Representação de conhecimento**

Começando agora a abordar as alíneas às quais o enunciado pede resposta, importa referir que a primeira ("Registar utentes, cuidados prestados e atos médicos") e a última ("Remover utentes, cuidados e atos médicos") são alíneas cuja resposta é dada por um predicado auxiliar: "**evolucao**" e "**involucao**", respetivamente. Abordando, novamente e de forma breve cada um destes predicados, o primeiro, responsável pela inserção de conhecimento, insere o facto que lhe é passado como argumento e testa se os invariantes referentes a esse mesmo facto continuam a ser cumpridos, removendo o conhecimento inserido em caso contrário. Por sua vez, o segundo, que acarreta a função de retirar conhecimento do nosso universo de discurso, faz algo semelhante: retira o facto passado como argumento, testa os invariantes e, caso haja violação de algum deles, repõe o estado anterior.

A interpretação da segunda funcionalidade requerida ("Identificar os utentes por critérios de seleção") levou à construção de regras que manipulam apenas dois conhecimentos: o critério de seleção e uma lista de utentes. Foram 4 os critérios escolhidos – nome, idade, sexo e morada – porém, dada a semelhança entre os predicados, vamos expor apenas dois exemplos:

```
% Identificação dos utentes por Nome
% Extensão do predicado utentesPorNome:
%                                Nome, [Utente] -> {V, F}

     utentesPorNome( Nome, S ) :-
                solucoes(  (IdUt,  Nome,  Idade,  Sexo,  Morada),
                utente(IdUt, Nome, Idade, Sexo, Morada), S ).

% Identificação dos utentes por Idade
% Extensão do predicado utentesPorIdade:
%                                Idade, [Utente] -> {V, F}

utentesPorIdade( Idade, S ) :-
```

```
        solucoes( (IdUt, Nome, Idade, Sexo, Morada),
        utente(IdUt, Nome, Idade, Sexo, Morada), S ).
```

Para identificar as instituições prestadoras de cuidados de saúde, construiu-se o seguinte predicado:

```
% Extensão do predicado listarInstComCuidadosDeSaude:
%                    [Instituição, DescCuidado] -> {V, F}

listarInstComCuidadosDeSaude( S ) :-
    solucoes( (Instituicao, Descricao), cuidadoPrestado(
    IdServ, Descricao, Instituicao, Cidade), S ).
```

Apenas manipula o conhecimento referente à lista das instituições e da descrição do tipo de cuidado médico prestado. Optou-se por esta estrutura para os elementos da lista, porque este predicado utiliza os factos **cuidadoPrestado**, logo a mesma instituição poderá aparecer mais do que uma vez na lista, no entanto associada a um tipo de cuidado de saúde diferente.

Na alínea 4 (identificar os cuidados prestados por instituição/cidade), o processo foi semelhante, não havendo novamente necessidade de criar regras adicionais que permitissem manipular o conhecimento pretendido, ou seja, os cuidados prestados por instituição/cidade. Aqui mostra-se apenas o predicado para identificação dos cuidados prestados por instituição, dado que a mudança para a identificação por cidade passa apenas por alterar o primeiro argumento do predicado para a designação da cidade.

```
% Identificação dos cuidados prestados por instituição
% Extensão do predicado cuidadosPorInstituicao:
                % Instituicao, [DescCuidado] -> {V, F}

cuidadosPorInstituicao( Instituicao, S ) :-
        solucoes( (DescCuidado), cuidadoPrestado(
        IdServ, DescCuidado, Instituicao, Cidade ), S ).
```

```
| ?- cuidadosPorInstituicao('Hospital de Braga', S).
S = ['Obstetricia','Ginecologia'] ?
yes
| ?-
```

Já na alínea 5, identificar os utentes de uma instituição/serviço acarreta maior dificuldade, porque implica cruzar a informação dos três predicados base deste trabalho – utente, atoMedico e cuidadoPrestado – para se saber quais os utentes que possuem atos médicos para uma dada instituição ou serviço. Os **atosMedicos** introduzem o conhecimento relativo ao utente e ao serviço (através dos seus Id's) e os **cuidadoPrestados** a informação da Instituição onde esse serviço opera.

```
%  Identificação dos utentes com atos médicos no cuidado com a Descrição apontada

%  Extensão do predicado utentesPorCuidado:

%                          DescCuidado, Utente -> {V, F}


utentesPorCuidado( DescCuidado, (IdUt, Nome, Idade, Sexo, Morada) ) :- cuidadoPrestado( IdServ, DescCuidado, Inst, Cidade ), atoMedico( Data, IdUt, IdServ, Custo ), utente( IdUt, Nome, Idade, Sexo, Morada ).


% Identificação dos utentes com atos médicos no cuidado com a Descrição apontada

% Extensão do predicado listarUtentesPorCuidado:

%                          DescCuidado, [Utente] -> {V, F}



listarUtentesPorCuidado( DescCuidado, S ) :-

    solucoes( (IdUt, Nome, Idade, Sexo, Morada), utentesPorCuidado( DescCuidado, (IdUt, Nome, Idade, Sexo, Morada) ), S ).
```

Como vemos, foram necessários dois predicados para dar resposta a esta funcionalidade: um que indica apenas um utente com ato médico no serviço cuja descrição é fornecida e outro que, recorrendo ao predicado auxiliar **solucoes**, associa uma lista de utentes à descrição do cuidado passado como primeiro argumento.

Agora, eis o predicado que identifica os utentes por instituição:

```
% Identificação dos utentes com atos médicos na instituição
apontada

% Extensão do predicado utentesPorInstituicao:

%                          Instituicao, Utente -> {V, F}


utentesPorInstituicao( Instituicao, (IdUt, Nome, Idade,
Sexo, Morada) ) :-

          cuidadoPrestado( IdServ, Desc, Instituicao,
          Cidade ), atoMedico( Data, IdUt, IdServ, Custo
          ), utente( IdUt, Nome, Idade, Sexo, Morada ).


% Identificação dos utentes com atos médicos na instituição
apontada

% Extensão do predicado listarUtentesPorInstituicao:

%                          Instituicao, [Utente] -> {V, F}


listarUtentesPorInstituicao( Instituicao, S ) :-

     solucoes( (IdUt, Nome, Idade, Sexo, Morada),
     utentesPorInstituicao( Instituicao, (IdUt, Nome,
     Idade, Sexo, Morada) ), S ).
```

```
| ?- listarUtentesPorInstituicao('Hospital de Braga', S).
S = [(1,joao,20,masculino,'vila verde')] ?
yes
| ?-
```

O processo de identificação dos utentes por instituição é muito semelhante ao de identificação por serviço médico prestado.

Apesar de a estrutura deste documento apresentar uma secção destinada às funcionalidades adicionais que o grupo criou, apresentamos agora dois tipos adicionais de identificação de utentes, dada a semelhança com o objetivo desta alínea.

Primeiro, os dois predicados que identificam os utentes com atos médicos numa determinada data; em seguida, apresentamos uma identificação dos utentes com idade de criança (=< 12 anos).

```
% Identificação dos utentes com atos médicos na data
apontada

% Extensão do predicado utentes PorData:
```

```
%                                   Data, Utente -> {V, F}


utentesPorData( Data, (IdUt, Nome, Idade, Sexo, Morada)) :-
      atoMedico( Data, IdUt, IdServ, Custo ),
      utente( IdUt, Nome, Idade, Sexo, Morada ).


% Identificação dos utentes com atos médicos na data
apontada
% Extensão do predicado listarUtentesPorData:
%                              Data, [Utente] -> {V, F}


listarUtentesPorData( Data, S ) :-
            solucoes( (IdUt, Nome, Idade, Sexo, Morada),
            utentesPorData( Data, (IdUt, Nome, Idade, Sexo,
            Morada) ), S ).
```

```
| ?- listarUtentesPorData('14-03-2017', S).
S = [(2,jose,20,masculino,lousada),(1,joao,20,masculino,'vila verde')] ?
yes
| ?-
```

```
%-------------------------------------------------------------
% Identificação dos utentes com idade de criança (<= 12)
com atos médicos
% Extensão do predicado criancaComAtoMedico:
%                                   Utente -> {V, F}


criancaComAtoMedico( IdUt, Nome, Idade, Sexo, Morada ) :-
            atoMedico( Data, IdUt, IdServ, Custo ),
            utente( IdUt, Nome, Idade, Sexo, Morada ),
            Idade =< 12.
```

```
%  Identificação das crianças com atos médicos
%  Extensão do predicado listarCriancasComAtoMedico:
%                                        [Utente] -> {V, F}

listarCriancasComAtoMedico( S ) :-
            solucoes( (IdUt, Nome, Idade, Sexo, Morada),
            criancaComAtoMedico( IdUt, Nome, Idade, Sexo,
            Morada ), S ).
```

Na sexta alínea, com o objetivo de identificar os atos médicos realizados por utente/instituição/serviço, decidiu-se manipular um maior número de conhecimentos de modo a incluir uma informação mais detalhada, relativamente aos atos médicos, nos elementos das listas. Para a identificação por utente, a estrutura escolhida para a lista foi (Descrição, Data, Instituição, Custo), o que implica a reunião de informação relativamente aos factos associados a atoMedico e cuidadoPrestado.

```
%Extensão do Predicado atoMedicoPorUtente:
%DescCuidado, Id Utente, Data, Instituição, Custo -> {V, F}

atoMedicoPorUtente( Desc,IdUt,Data,Instituicao,Custo ):-
      atoMedico( Data,IdUt,IdServ,Custo ),
      cuidadoPrestado( IdServ,Desc,Instituicao,Cidade ).

% Extensão do Predicado listarAtoMedicoPorUtente:
%          IdUt, [(Desc, Data, Inst, Custo)] -> {V, F}
listarAtoMedicoPorUtente( IdUt, S ) :-
      solucoes( (DescCuidado, Data, Instituicao, Custo),
      atoMedicoPorUtente(    DescCuidado,    IdUt,    Data,
      Instituicao, Custo), S ).
```

Já para identificar os atos médicos por instituição, pretendeu-se expressar o nome do utente, para além da descrição do cuidado prestado uma vez mais, portanto houve necessidade de cruzar a informação dos três predicados base.

```
% Extensão do Predicado atoMedicoPorInstituicao:
%          Desc, Inst, IdUt, Nome, Data, Custo -> {V, F}

atoMedicoPorInstituicao(Desc,Inst,IdUt,Nome,Data,Custo) :-
      utente( IdUt, Nome, Idade, Sexo, Morada ),
      cuidadoPrestado( IdServ, Desc, Inst, Cidade ),
      atoMedico( Data, IdUt, IdServ, Custo).

% Extensão do Predicado listarAtoMedicoPorInst:
%                          Inst, [Ato Médico] -> {V, F}
listarAtoMedicoPorInst( Instituicao, S ) :-
      solucoes( (Desc, IdUt, Nome, Data, Custo),
      atoMedicoPorInstituicao(  Desc,  Instituicao,  IdUt,
      Nome, Data, Custo), S ).
```

```
| ?- listarAtoMedicoPorInst('Centro de Saude de Lousada', S).
S = [('Medicina Familiar',2,jose,'14-03-2017',5),('Medicina Familiar',1,joao,'04-04-2017',7)] ?
yes
| ?-
```

Para os atos médicos por serviço, o processo anterior repete-se, com a alteração resultante da inclusão do nome da instituição por troca com a descrição do serviço prestado.

```
% Extensão do Predicado atoMedicoPorServico:
% Desc, Instituição, Id Utente, Nome, Data, Custo -> {V, F}
atoMedicoPorServico( Desc,Inst,IdUt,Nome,Data,Custo ) :-
      utente( IdUt, Nome, Idade, Sexo, Morada ),
      cuidadoPrestado( IdServ, Desc, Inst, Cidade ),
      atoMedico( Data, IdUt, IdServ, Custo).
```

```
% Extensão do Predicado listarAtoMedicoPorServ:
%                          Serviço, [Ato Médico] -> {V, F}
listarAtoMedicoPorServ( AtoMedico, S ) :-
    solucoes( (Instituicao, IdUt, Nome, Data, Custo),
    atoMedicoPorInstituicao(AtoMedico, Instituicao, IdUt,
    Nome, Data, Custo), S ).
```

A sétima funcionalidade requerida consiste em determinar todas as instituições/serviços a que um utente já recorreu. Em ambos os casos, é necessário saber quais os atos médicos do utente cujo Id foi fornecido, uma vez que incluem o Id do tipo de cuidado prestado nesse ato médico. Depois, associando a este Id a informação do **cuidadoPrestado** correspondente, consegue-se alcançar o conhecimento pretendido (`cuidadoPrestado`: `IdServ, Descrição, Instituição, Cidade -> {V, F}`).

```
% Determinação de todas as instituições a que um utente já
recorreu
% Extensão do Predicado instituicaoPorUtente:
%                          Utente, Instituicao -> {V, F}

instituicaoPorUtente( IdUt, Instituicao ) :-
    utente( IdUt, Nome, Idade, Sexo, Morada ),
    atoMedico( Data, IdUt, IdServ, Custo),
    cuidadoPrestado( IdServ, Desc, Instituicao, Cidade ).

% Extensão do Predicado listarInstituicoesPorUtente:
%                          Utente, [Instituicao] -> {V, F}

listarInstituicoesPorUtente( IdUt, S ) :-
    solucoes( (Instituicao), instituicaoPorUtente( IdUt,
    Instituicao ), S ).

% ------------------------------------------------------------

% Determinação de todas os serviços a que um utente já
recorreu
```

```
% Extensão do Predicado serviçoPorUtente:
%                                   Utente, Servico -> {V, F}

servicoPorUtente( IdUt, (IdServ, Desc, Instituicao) ) :-
      utente( IdUt, Nome, Idade, Sexo, Morada ),
      atoMedico( Data, IdUt, IdServ, Custo),
      cuidadoPrestado( IdServ, Desc, Instituicao, Cidade ).
% Extensão do Predicado listarServicosPorUtente:
% Utente, [(IdServ, DescCuidado, Instituicao)] -> {V, F}

listarServicosPorUtente( IdUt, S ) :-
      solucoes(    (IdServ,    DescCuidado,    Instituicao),
      servicoPorUtente(    IdUt,    (IdServ,    DescCuidado,
      Instituicao) ), S ).
```

Tendo em conta que a última alínea do enunciado já foi abordada anteriormente, termina-se a análise das funcionalidades requeridas com o cálculo do custo total dos atos médicos por utente/serviço/instituição/data. Para auxiliar nestes cálculos construiu-se o predicado seguinte:

```
% Extensão do predicado auxiliar somatorio:
%                                       L, Soma -> {V, F}
somatorio([], 0).
somatorio([X | Y], R) :- somatorio(Y,Z),
                              R is X+Z.
```

Este predicado será usado em todas as versões desta funcionalidade. Em todas as versões serão também necessárias mais duas regras, sendo que para os casos do cálculo do custo total por utente, data e serviço estas são muito semelhantes, pelo que mostrar-se-á apenas para o caso do utente. Essa semelhança deve-se ao facto de o predicado atoMedico conter informação relativa à data, ao Id de utente e ao Id de serviço, para além do necessário custo, logo não é necessário aceder à informação de nenhum dos outros factos base.

```
% Cálculo do custo total dos atos médicos por utente
% Extensão do Predicado custosPorUtente:
%                                   IdUt, Custo -> {V, F}
```

```
custosPorUtente( IdUt, Custo ) :-
      atoMedico( Data, IdUt, IdServ, Custo ).



% Extensão do Predicado listarCustosPorUtente:
%                           IdUt, L, Total -> {V, F}


listarCustosPorUtente( IdUt, [X|T], Total ) :-
      solucoes( (Custo), custosPorUtente( IdUt, Custo ),
      [X|T] ), somatorio( T, R ), Total is X + R.
```

```
| ?- listarCustosPorUtente(1, S, X).
S = [7,30],
X = 37 ?
yes
| ?-
```

Alterar o primeiro parâmetro de ambos os predicados anteriores para data ou serviço seria suficiente para podermos calcular o custo total dos atos médicos de acordo com estes fatores.

Por outro lado, para o custo total por instituição já se torna necessário aceder à informação dos predicados **cuidadoPrestado**:

```
% Cálculo do custo total dos atos médicos por instituição
% Extensão do Predicado custosPorInstituicao:
%                           Instituicao, Custo -> {V, F}


custosPorInstituicao( Instituicao, Custo ) :-
      atoMedico( Data, IdUt, IdServ, Custo ),
      cuidadoPrestado( IdServ, DescCuidado, Instituicao,
      Cidade ).


% Extensão do Predicado listarCustosPorInstituicao:
%                           Instituicao, L, Total -> {V, F}
```

```
listarCustosPorInstituicao( Instituicao, [X|T], Total ) :-
    solucoes( (Custo), custosPorInstituicao( Instituicao, Custo ), [X|T] ),
    somatorio( T, R ),
    Total is X + R.
```

- **Funcionalidades adicionais**

Como foi encorajado, incluiu-se algumas novas funcionalidades no nosso universo de discurso, de modo a tornar o problema mais abrangente na área de prestação de cuidados de saúde.

Pela introdução dos factos **medico**, foi possível listar os médicos consoante o cuidado prestado e também pela instituição em que operam.

```
% Extensão do predicado medico:
%                IdMed, Nome, Idade, Sexo, IdServ -> {V, F}

medico( 1, 'Antonio Lemos', 52, masculino, 3  ).
medico( 2, 'Anibal Mota', 59, masculino, 1 ).
medico( 3, 'Catarina Paiva', 35, feminino, 5 ).
medico( 4, 'Jose Garcia', 39, masculino, 2 ).
medico( 5, 'Carla Perez', 41, feminino, 6 ).

% Extensão do predicado medicoPorCuidado:
%                                Descricao, IdMed, Nome -> {V, F}

medicoPorCuidado( Descricao, IdMed, Nome ) :-
    medico( IdMed, Nome, Idade, Sexo, IdServ ),
    cuidadoPrestado( IdServ, Descricao, Inst, Cidade ).

% Extensão do predicado listarMedicosPorCuidado:
%                            Descricao, [(IdMed, Nome)] -> {V, F}
```

```
listarMedicosPorCuidado( Descricao, S ) :-
      solucoes( (IdMed, Nome), medicoPorCuidado( Descricao,
      IdMed, Nome ), S ).


% Extensão do predicado medicoPorInstituicao:
%                          Instituicao, IdMed, Nome -> {V, F}

medicoPorInstituicao( Instituicao, IdMed, Nome ) :-
      medico( IdMed, Nome, Idade, Sexo, IdServ ),
      cuidadoPrestado( IdServ, Descricao, Instituicao, Cidade ).


% Extensão do predicado listarMedicosPorInstituicao:
%                     Instituicao, [(IdMed, Nome)] -> {V, F}

listarMedicosPorInstituicao( Instituicao, S ) :-
      solucoes( ( IdMed, Nome ), medicoPorInstituicao(
      Instituicao, IdMed, Nome ), S ).
```

Como não poderia deixar de ser, construíram-se também alguns invariantes de inserção e remoção para o predicado **medico**. Os dois primeiros, de inserção, impossibilitam que sejam inseridos médicos com Id's iguais e médicos cujo Id do serviço prestado não existe, respetivamente. Já o de remoção impede que a remoção de um médico torne nulo o número de médicos que efetuam o mesmo serviço.

```
% Invariante Estrutural

+medico( IdMed,Nome,Idade,Sexo,IdServ )::( solucoes( (IdMed),
medico(IdMed,N,I,Se,Servico), S ), comprimento( S,N ),
N == 1 ).

+medico( IdMed,Nome,Idade,Sexo,IdServ ) :: ( solucoes( (IdServ),
cuidadoPrestado( IdServ,Desc,Inst,Cid ) , S ), comprimento( S,N
), N == 1 ).

-medico( IdMed,Nome,Idade,Sexo,IdServ ) :: ( solucoes( (IdServ),
medico( Id,N,I,S,IdServ ) , S ), comprimento( S,N ), N >= 1 ).
```

Criaram-se também os predicados **enfermeiro**, **turno** e **destacamento**, que acrescentaram algumas funcionalidades à nossa base de conhecimento, como são a listagem de enfermeiros por instituição e dos enfermeiros destacados para um determinado turno numa determinada data.

```
% Extensão do predicado enfermeiro:
%          IdEnf, Nome, Idade, Sexo, Instituicao -> {V, F}

enfermeiro(1, 'Catia Vanessa', 32, feminino, 'Centro de
Saude de Lousada').
enfermeiro(2, 'Gabriela Soares', 26, feminino, 'Centro de
Saude de Vila Verde').
enfermeiro(3, 'Renato Ribeiro' , 37, masculino, 'Hospital
de Braga').
enfermeiro(4, 'Alexandra Cunha', 29, feminino, 'Hospital de
Braga').
enfermeiro(5, 'Jorge Ferreira', 44, masculino, 'Hospital
Sao Joao').

% Listar enfermeiros de uma dada Instituicao
% Extensão do predicado listarEnfermeirosPorInstituicao:
%                    Instituicao, [enfermeiro] -> {V, F}

listarEnfermeirosPorInstituicao( Instituicao, S ) :-
solucoes( (IdEnf, Nome, Idade, Sexo), enfermeiro( IdEnf,
Nome, Idade, Sexo, Instituicao ), S ).

% Extensão do predicado turno:
%                    IdTurno, Horas, Instituicao -> {V, F}

turno( 1, '19h-01h', 'Hospital de Braga' ).
turno( 2, '07h-13h', 'Hospital de Braga' ).
turno( 3, '13h-19h', 'Hospital de Braga' ).
turno( 4, '14h-19h', 'Hospital de Sao Joao' ).
turno( 5, '19h-01h', 'Centro de Saude de Vila Verde' ).
```

```
% Enfermeiro(a) destacado(a) para o dado turno em
determinada data

% Extensão do predicado destacamento:

%                          Data, IdTurno, IdEnf -> {V, F}


destacamento( '13-03-2017', 2, 4 ).

destacamento( '19-03-2017', 1, 3 ).

destacamento( '5-04-2017', 3, 4 ).

destacamento( '1-04-2017', 5, 2 ).

destacamento( '19-02-2017', 4, 5 ).



% Enfermeiros destacados num dado dia

% Extensão do predicado enfermeiroPorData:

%          Data, (IdEnf, Nome, IdTurno, Horas) -> {V, F}


enfermeiroPorData( Data, IdEnf, Nome, IdTurno, Horas ) :-

      destacamento( Data, IdTurno, IdEnf ),

      turno( IdTurno, Horas, Instituicao ),

      enfermeiro( IdEnf, Nome, Idade, Sexo, Instituicao ).


% Extensão do predicado listarEnfermeirosPorData:

%          Data, [(IdEnf, Nome, IdTurno, Horas)] -> {V, F}


listarEnfermeirosPorData( Data, S ) :-

      solucoes( (IdEnf, Nome, IdTurno, Horas),
      enfermeiroPorData( Data, IdEnf, Nome, IdTurno,
      Horas), S ).


      | ?- listarEnfermeirosPorData('5-04-2017', S).
      S = [(4,'Alexandra Cunha',3,'13h-19h')] ?
      yes
      | ?-
```

Não esquecendo os invariantes que mantêm a coerência na nossa base de conhecimento, para o predicado **enfermeiro** criou-se um invariante de inserção que impede Id's repetidos e um de remoção que impede a remoção de enfermeiros destacados para determinado turno.

```
% Invariante Estrutural

+enfermeiro( IdEnf,Nome,Idade,Sexo,Instituicao ) :: (
solucoes( (IdEnf), enfermeiro(IdEnf,N,I,Se,Inst), S ),
comprimento( S,N ), N == 1 ).

-enfermeiro( IdEnf,Nome,Idade,Sexo,Instituicao ) :: (
solucoes( (IdEnf), destacamento(Data,IdT,IdEnf), S ),
comprimento( S,N ), N == 0 ).
```

Para o **turno**, foi apenas necessário impedir a inserção de turnos com Id's repetidos.

```
% Invariante Estrutural

+turno( IdTurno,Horas,Instituicao ) :: ( solucoes(
(IdTurno), turno(IdTurno,H,Inst), S ), comprimento( S,N ),
N == 1 ).
```

Por último, pensou-se no predicado transplante, com o qual respondemos à identificação dos utentes em lista de espera para transplante de um dado órgão. O invariante de inserção associado ao transplante impede que o mesmo utente esteja representado mais do que uma vez na lista de espera de um dado órgão.

```
% Invariante Estrutural

+transplante( IdUt, Orgao )::( solucoes( (IdUt, Orgao),
transplante(IdUt, Orgao), S ), comprimento( S,N ), N == 1).
```

```
% Extensão do predicado transplante:

%                                IdUt, Orgao -> {V, F}

transplante( 3, 'Rim' ).

transplante( 1, 'Figado' ).

transplante( 2, 'Rim' ).
```

```
% Extensão do predicado utenteEmEspera:
%                                   Orgao, Utente -> {V, F}

utenteEmEspera( Orgao, IdUt, Nome, Idade, Sexo, Morada ) :-
      transplante( IdUt, Orgao ),
      utente( IdUt, Nome, Idade, Sexo, Morada ).

% Extensão do predicado listaDeEspera:
%                                   Orgao, [Utente] -> {V, F}

listaDeEspera( Orgao, S ) :-
      solucoes( (IdUt, Nome, Idade, Sexo, Morada),
      utenteEmEspera( Orgao, IdUt, Nome, Idade, Sexo,
      Morada ), S ).
```

```
| ?- listaDeEspera('Rim', S).
S = [(2,jose,20,masculino,lousada),(3,josefina,34,feminino,aveiro)] ?
yes
| ?-
```

Conclusões e Sugestões

A resolução do primeiro exercício prático permitiu-nos consolidar os conhecimentos adquiridos nas aulas relativamente à programação em lógica, utilizando PROLOG, e à implementação de invariantes. Apesar de termos aproveitado algum código realizado pelo docente nas aulas práticas, o restante trabalho foi feito de forma individual, tendo por base as fichas realizadas nas aulas práticas e pequenas pesquisas na web.

Fazemos uma avaliação positiva do trabalho, onde os requisitos mínimos foram cumpridos sem grande dificuldade (o que se deve em grande parte a uma preparação prévia) e onde foram adicionadas funcionalidades extra.

Referências

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

- “A Inteligência Artificial em 25 Lições”, Hélder Coelho;

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

Anexos