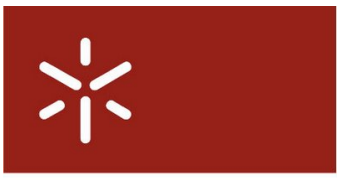


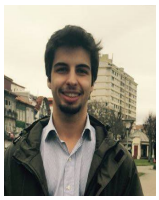
Universidade do Minho

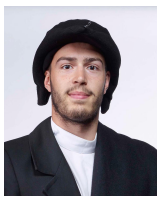
Sistemas de Representação de Conhecimento e Raciocínio

MIEI - 3º ano - 2º semestre
Universidade do Minho

Trabalho I
Programação em lógica e Invariantes

Grupo 10


Dinis Peixoto
A75353


José Bastos
A74696


Ricardo Pereira
A74185


Marcelo Lima
A75210

16 de Novembro de 2017

1. *Resumo*

O presente documento corresponde ao relatório elaborado em conjunto com a resolução do primeiro exercício da componente prática exigida na unidade curricular *Sistemas de Representação de Conhecimento e Raciocínio*. Com o tema *Programação em Lógica e Invariantes*, a resolução do exercício pretende motivar os alunos para a linguagem de programação em lógica *PROLOG*, exigindo destes o desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde pela realização de serviços e atos médicos a utentes de diferentes estabelecimentos.

Conteúdo

2. *Introdução*

Este relatório é o resultado da elaboração do primeiro exercício prático proposto na unidade curricular de *Sistemas de Representação de Conhecimento e Raciocínio*, com o tema *Programação em lógica e Invariantes*.
Tal como o próprio nome indica, este trabalho tem por base a utilização da programação em lógica, mais concretamente da linguagem de programação *PROLOG* e os seus invariantes.
No enunciado é-nos pedido que seja desenvolvido um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar todo um vasto cenário centralizado na área da prestação de cuidados de saúde, mais propriamente na realização de serviços de atos médicos. Assim, existem três principais fontes de conhecimento: os utentes, os cuidados prestados e os atos médicos. Desta forma, o objetivo é criar um sistema capaz de receber e guardar todas as informações associadas a cada um destes, e processar toda esta informação, de modo a responder às diversas questões presentes no enunciado do exercício.

3. *Preliminares*

Tal como dito anteriormente, neste sistema, que pretende representar um universo de realização de serviços de atos médicos, existem três principais fontes de conhecimento, nas quais se vai basear todo o seu cenário. Primeiramente temos o utente, a única personagem interveniente neste sistema, a qual tem associado a si um ID, Nome, Idade e Morada. De seguida temos o cuidado prestado (serviço), que tem associado a si um ID do serviço, uma Descrição desse serviço, a Instituição onde o serviço foi efectuado e a Cidade onde este se realizou. Por último, temos o ato médico, que basicamente faz a junção dos dois conhecimentos anteriores, ou seja, diz-nos quando e com que custo um determinado serviço foi efectuado a um determinado utente. Assim, tem associado a si uma Data, o ID do utente em questão, o ID do serviço efectuado, e por último, o Custo desse ato médico. Deste modo, temos as seguintes definições iniciais:

- **utente**: (#ID_Utente, Nome, Idade, Morada) → {V,F}
- **serviço**: (#ID_Serviço, Descrição, Instituição, Cidade) → {V,F}
- **ato**: (Data, #ID_Utente, #ID_Serviço, Custo) → {V,F}

É fácil entender que tanto o ID do utente como do serviço têm de ser únicos e identificadores deles mesmos, isto é, não pode haver nem utentes nem serviços diferentes com o mesmo ID. Por outro lado, em relação aos atos médicos, visto que estes são a associação entre utentes e serviços prestados, é possível haver atos médicos com um conjunto de ID's semelhantes, uma vez que um determinado utente pode ser submetido ao mesmo cuidado médico várias vezes. Agora sim, depois de todas estas considerações iniciais, encontramo-nos aptos a prosseguir com a elaboração do trabalho propriamente dito.

4. *Descrição do Trabalho e Análise de Resultados*

4.1 Predicados de evolução/retrocesso de conhecimento

Para o correto funcionamento da nossa base de conhecimento foram inseridos uma série de invariantes para a resolução de alguns problemas relativos à informação que a base de conhecimento iria conter. Desta forma, no momento da inserção e remoção de informação é necessário utilizar os invariantes para garantir o controlo dessa mesma informação. Para tal, foram criados os predicados *evolucao* e *retrocesso* para controlar a inserção e remoção, respetivamente.

```
remove(T) :- retract(T).

insercao(T) :- assert(T).
insercao(T) :- retract(T), !,fail.

testar([]).
testar([I|L]) :- I, testar(L).

evolucao(F) :-
    solucoes(I,+F::I,L),
    insercao(F),
    testar(L).

retrocesso(F) :-
    solucoes(I,-F::I,L),
    testar(L),
    remove(F).

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

Como é visível, estes predicados, no momento de inserção ou remoção de informação, testam os invariantes, e impedem qualquer ação, caso um invariante não se verifique, garantindo assim a consistência da nossa base de conhecimento.

4.2 Invariantes

Como foi referido anteriormente, para o correto funcionamento da nossa base de conhecimento é necessária a implementação de invariantes para o controlo da informação presente na mesma. Para tal, implementamos dois conjuntos de invariantes: invariantes associados à inserção e invariantes associados à remoção.
No conjunto de invariantes associados à inserção, encontramos um conjunto que impede a repetição de conhecimento (neste caso, um utente, servico ou médico) repetido, isto é, após a inserção, verifica se o ID do predicado dado é único, caso contrário, remove o mesmo da base de conhecimento. Optou-se por não verificar repetição de atos uma vez que o grupo considerou a possibilidade de existirem atos repetidos.

```
+utente(IDU,N,I,M) :: (solucoes(IDU,utente(IDU,_,_,_),S),
                       comprimento(S,X),
                       X == 1).

+servico(IDS,D,I,C) :: (solucoes(IDS,servico(IDS,_,_,_),S),
                        comprimento(S,X),
                        X == 1).

+medico(ID,N,I,M,E) :: (solucoes(ID,medico(ID,_,_,_,_),S),
                        comprimento(S,X),
                        X == 1).
```

Para além disso, existem mais dois invariantes importantes, para o controlo da inserção de informação. Quando inserimos um servico à base de conhecimento, é preciso ter em atenção que não exista já um serviço, com a mesma descrição, instituição e cidade, e para tal, é necessário o seguinte invariante:

```
+servico(IDS,D,I,C) :: (solucoes((D,I),servico(_,D,I,C),S),
                        comprimento(S,X),
                        X == 1).
```

Outro invariante é associado ao ato, que não pode possuir IDs, de utente, servico e médico, inexistentes, além disto, o médico deve possuir especialização para o servico prestado:

```
+ato(D,IDU,IDS,C,IDMED) :: (utente(IDU,_,_,_),
                            servico(IDS,D,_,_),
                            medico(IDMED,_,_,_,D)).
```

O outro conjunto de invariantes é referente à remoção. No momento da remoção é necessário controlar a existência do conhecimento a ser removido (utente, servico, medico e ato):

```
-utente(IDU,N,I,M) :: (solucoes(IDU,utente(IDU,_,_,_),S),
                       comprimento(S,X),
                       X == 1).

-servico(IDS,D,I,C) :: (solucoes(IDS,servico(IDS,_,_,_),S),
                        comprimento(S,X),
                        X == 1).

-medico(ID,N,I,M,E) :: (solucoes(ID,medico(ID,_,_,_,_),S),
                        comprimento(S,X),
                        X == 1).

-ato(D,IDU,IDS,C,IDMED) :: (solucoes((D,IDU,IDS),ato(D,IDU,IDS,_,IDMED),S),
                            comprimento(S,X),
                            X==1).
```

Para além disso é necessário ter em atenção a consistência da base de conhecimento, isto é, não permitir a remoção de conhecimento que possui referência noutro lado. Estes casos acontecem com o ato, que possui os ID's do utente, servico e do médico. Para tal, é necessário verificar antes da remoção, se existe algum ato com o ID de um predicado, caso exista, não deve ser removido.

```
-utente(IDU,N,I,M) :: (solucoes(IDU,ato(_,IDU,_,_,_),S),
                       comprimento(S,X),
                       X==0).

-servico(IDS,D,I,C) :: (solucoes(IDS,ato(_,_,IDS,_,_),S),
                        comprimento(S,X),
                        X==0).

-medico(ID,N,I,M,E) :: (solucoes(ID,ato(_,_,_,_,ID),S),
                        comprimento(S,X),
                        X==0).
```

4.3 Predicados gerais/auxiliares

Nesta seção faremos referência aos predicados gerais de todo este exercício. Estes, serão auxiliares e indispensáveis para a futura realização de outros predicados mais adiante.

- **sum**

  O predicado *sum* terá como função fazer o somatório de todos os elementos presentes numa lista.

```
% Extensão do predicado sum: Lista, Resultado -> {V,F}
sum([],0).
sum([X|Y],G) :- sum(Y,R), G is R+X.
```

- **concat**

  O predicado *concat* resultará numa lista obtida através da concatenação de duas listas fornecidas.

```
% Extensão do predicado concat: Lista_1, Lista_2, Resultado -> {V,F}
concat([],R,R).
concat([X|L],R,[X|S]) :- concat(L,R,S).
```

- **comprimento**

  O predicado *comprimento* terá como função retribuir o comprimento de uma determinada lista.

```
% Extensão do predicado comprimento: Lista, Resultado -> {V,F}
comprimento([],0).
comprimento([H|T],N) :-
    comprimento(T,N1),
    N is N1+1.
```

- **eliminarRepetidos**

  O predicado *eliminarRepetidos* servirá para efetuar a remoção de todos os elementos repetidos numa determinada lista.

```
% Extensão do predicado eliminarRepetidos: Lista, Resultado -> {V,F}
eliminarRepetidos([], []) .
eliminarRepetidos([H|T], Res) :- eliminarElemento(T, H, E),
                                 eliminarRepetidos(E, R),
                                 Res = [H|R].
```

- **eliminarElemento**

O predicado *eliminarElemento* será utilizado maioritariamente como predicado auxiliar ao anterior, e a sua função é eliminar um determinado elemento de uma dada lista.

```
% Extensão do predicado eliminarElemento: Lista, Elemento, Resultado -> {V,F}
eliminarElemento([], _, []).
eliminarElemento([H|T], H, R) :- eliminarElemento(T, H, R).
eliminarElemento([H|T], E, Res) :- H\== E,
                                   eliminarElemento(T, E, R),
                                   Res = [H|R].
```

4.4 Conhecimento extra - Médico

De modo a acrescentar conhecimento ao mínimo requisitado pelo enunciado do exercício em questão, o grupo optou por criar uma nova fonte de conhecimento capaz de se enquadrar no caso em questão estudado: prestação de cuidados de saúde pela realização de serviços e atos médicos, sendo esta o **Médico** de cada ato realizado. Portanto, foi necessário efetuar algumas alterações quanto à estrutura inicialmente fornecida pelo enunciado, sendo o resultado o seguinte:

- **utente**: (#ID_Utente, Nome, Idade, Morada) → {V,F}
- **serviço**: (#ID_Serviço, Descrição, Instituição, Cidade) → {V,F}
- **ato**: (Data, #ID_Utente, #ID_Serviço, Custo, #ID_Médico) → {V,F}
- **médico**: (#ID_Médico, Nome, Idade, Morada, Especialização) → {V,F}

Seguem as definições iniciais correspondentes às fontes de conhecimento anteriores.

```
:- op( 900,xfy,'::' ).
:- dynamic utente/4.        % (ID_Utente, Nome, Idade, Morada)
:- dynamic servico/4.       % (ID_Serviço, Descrição, Instituição, Cidade).
:- dynamic ato/5.           % (Data, ID_Utente, ID_Serviço, Custo, ID_Médico).
:- dynamic medico/5.        % (ID_Médico, Nome, Idade, Morada, Especialização).
```

4.5 Base de conhecimento inicial

É necessário começar com uma base de conhecimentos inicial, de forma a que possámos logo testar a nossa informação sem ter que inserir conhecimento. Ao preenchê-la, tivemos cuidado com as definições das fontes de conhecimento. Também quisemos abranger alguns casos mais específicos, como por exemplo os dois primeiros utentes representados, em que ambos têm o mesmo nome, mesma idade e mesma morada. Como se pode constatar também implementámos utentes envolvidos em mais do que um ato e médicos em mais do que um serviço.

```
utente(1,'Renato Portoes',32,'Rua Nova Santa Cruz').
utente(2,'Renato Portoes',32,'Rua Nova Santa Cruz').
utente(3,'Ricardo Azevedo',20,'Rua Velha Santa Cruz').
utente(4,'Ana Martins',22,'Rua do Outeiro').
utente(5,'Jose Vilaca',12,'Rua das Cegonhas').
utente(6,'Sara Alberto',45,'Rua dos Marcianos').
utente(7,'Joao Guilherme',89,'Avenida Central').
utente(8,'Afonso Aragao',26,'Rua dos Palacetes').
utente(9,'Jose Silva',73,'Estreito Largo').
utente(10,'Frederico Pereira',4,'Rua dos Carretos').
utente(11,'Rita Gameiro',52,'Rua Engenheiro Antonio Filipe').
utente(12,'Luis Marcio',15,'República das Bananas').
utente(13,'Pedro Luis',53,'Largo do Bigode').
utente(14,'Marco Cardoso',33,'Avenida Principal').


servico(1,'Ortopedia','Centro de Saude Santa Tecla','Braga').
servico(2,'Cardiologia','Centro de Saude Santa Tecla','Braga').
servico(3,'Neurocirurgia','Hospital Privado de Braga','Braga').
servico(4,'Pediatria','Posto medico de Fao','Esposende').
servico(5,'Otorrinolaringologia','Hospital Santa Maria','Porto').
servico(6,'Oftalmologia','Clinica de Santa Madalena','Felgueiras').
servico(7,'Urologia','Casa de Saude de Caldelas','Amares').
servico(8,'Ginecologia','Clinica do Tubarao','Viana do Castelo').
servico(9,'Ortopedia','Espaco Saude Beirao Rendeiro','Moledo').


medico(1,'Dr. Eugenio Andrade',54,'Viana do Castelo','Ortopedia').
medico(2,'Dr. Firmino Cunha',63,'Guimaraes','Cardiologia').
medico(3,'Dr. Jorge Costa',38,'Santo Tirso','Neurocirurgia').
medico(4,'Dr. Bruno Hermenegildo',48,'Vila das Aves','Pediatria').
medico(5,'Dra. Alberta Mendes',57,'Matosinhos','Otorrinolaringologia').
medico(6,'Dr. Cristiano Soares',34,'Terras de Bouro','Oftalmologia').
medico(7,'Dra. Rosalina Oliveira',49,'Vieira do Minho','Urologia').
medico(8,'Dr. Carlos Mota',44,'Maia','Ginecologia').


ato('12-01-2017',1,1,19,1).
ato('13-01-2017',14,2,10,2).
ato('14-01-2017',2,3,15,3).
ato('15-01-2017',13,4,5,4).
ato('16-01-2017',3,6,12,6).
ato('17-01-2017',12,6,14,6).
ato('15-01-2017',4,7,5,7).
ato('16-01-2017',11,5,12,5).
ato('17-01-2017',5,8,100,8).
ato('24-01-2017',6,2,35,2).
ato('15-01-2017',7,7,80,7).
ato('19-01-2017',8,2,200,2).
ato('11-01-2017',9,6,10,6).
ato('25-01-2017',10,2,55,2).
ato('26-01-2017',11,3,40,3).
ato('29-01-2017',12,4,90,4).
ato('31-01-2017',13,9,50,1).
```

4.6 Predicados de registo/remoção de utentes, cuidados prestados, atos médicos e médicos

Os predicados que permitem inserir um utente, cuidado prestado (Serviço), ato médico ou médico são os seguintes:

```
% Extensão do predicado registar: Termo -> {V,F}
registar(T) :- evolucao(T).

% Extensão do predicado registarUtente: Id_Utente, Nome, Idade, Morada -> {V,F}
registarUtente(IDU,N,I,M) :- evolucao(utente(IDU,N,I,M)).

% Extensão do predicado registarServico: Id_Serviço, Descrição, Instituição, Cidade -> {V,F}
registarServico(IDS,D,I,C) :- evolucao(servico(IDS,D,I,C)).

% Extensão do predicado registarAto: Data, Id_Utente, Id_Serviço, Custo, Id_Médico -> {V,F}
registarAto(D,IDUT,IDSE,C,IDMED) :- evolucao(ato(D,IDUT,IDSE,C,IDMED)).

% Extensão do predicado registarMedico: Id_Médico, Nome, Idade, Morada, Especialização -> {V,F}
registarMedico(ID,N,I,M,E) :- evolucao(medico(ID,N,I,M,E)).
```

Estes predicados, basicamente, recebem os parâmetros de um predicado qualquer (utente,

servico, ato ou medico), e passam para o predicado *evolucao(T)*, que insere na base de conhecimento de forma controlada.
Por outro lado, existem também os predicados que permitem remover os referidos anteriormente:

```
% Extensão do predicado remover: Termo -> {V,F}
remover(T) :- retrocesso(T).

% Extensão do predicado removerUtente: Id_Utente, Nome, Idade, Morada -> {V,F}
removerUtente(ID) :- retrocesso(utente(ID,N,I,M)).

% Extensão do predicado removerServico: Id_Serviço, Descrição, Instituição, Cidade -> {V,F}
removerServico(ID) :- retrocesso(servico(ID,D,I,C)).

% Extensão do predicado removerAto: Data, Id_Utente, Id_Serviço, Id_Médico -> {V,F}
removerAto(D,IDUT,IDSE,IDMED) :- retrocesso(ato(D,IDUT,IDSE,C,IDMED)).

% Extensão do predicado removerMedico: Id_Médico, Nome, Idade, Morada, Especialização -> {V,F}
removerMedico(ID) :- retrocesso(medico(ID,N,I,M,E)).
```

Do mesmo modo que a inserção, utilizamos o predicado *retrocesso(T)*, que remove da base de conhecimento, qualquer conhecimento de forma controlada.

4.7 Predicados de listagem de conhecimento

4.7.1 Identificar os utentes por critérios de seleção

Para identificar os utentes, recorremos a 4 critérios de seleção: ID, Nome, Idade e Morada. Desta forma, usamos os seguintes predicados, para representar cada caso de seleção.

```
% Extensão do predicado utenteID: ID, Resultado -> {V,F}
utenteID(ID,R) :- solucoes((ID,N,I,M),utente(ID,N,I,M),R).

% Extensão do predicado utentesNome: Nome, Resultado -> {V,F}
utentesNome(N,R) :- solucoes((ID,N,I,M),utente(ID,N,I,M),R).

% Extensão do predicado utentesIdade: Idade, Resultado -> {V,F}
utentesIdade(I,R) :- solucoes((ID,N,I,M),utente(ID,N,I,M),R).

% Extensão do predicado utentesMorada: Morada, Resultado -> {V,F}
utentesMorada(M,R) :- solucoes((ID,N,I,M),utente(ID,N,I,M),R).
```

Para representar o conjunto de Utentes, recorremos ao predicado auxiliar *solucoes*, que nos agrupa o conjunto de parâmetros (ID, Nome, Idade e Morada do Utente), que satisfazem o predicado *utente(ID, N, I, M)* que possuem o parâmetro de seleção (ID, Nome, Idade ou Morada) dado inicialmente.

Exemplo de output:

```
| ?- utenteID(1,R).
R = [(1,'Renato Portoes',32,'Rua Nova Santa Cruz')] ?
yes
```

4.7.2 Identificar as instituições prestadoras de cuidados de saúde

Do mesmo modo que na secção anterior, para obter as instituições prestadoras de cuidados de saúde, usamos o predicado *solucoes*, que nos determina, o conjunto de todas as instituições existentes na base de conhecimento, isto é, todos os parâmetros (Instituição) que satisfazem o predicado *utente(_,_,I,_)*, onde I é a instituição prestadora de

cuidados de saúde. Como é de esperar, o conjunto, irá conter elementos repetidos. Por isso, implementamos o predicado *eliminarRepetidos*, que recebe a lista das instituições, e elimina todos os elementos repetidos.

```
% Extensão do predicado instituicoes: Resultado -> {V,F}
instituicoes(R) :- solucoes(I,servico(_,_,I,_),S),
                   eliminarRepetidos(S,R).
```

Exemplo de output:

```
| ?- instituicoes(R).
R = ['Centro de Saude Santa Tecla','Hospital Privado de Braga','Posto medico de Fao','Hospital Santa Maria',
'Clinica de Santa Madalena','Casa de Saude de Caldelas','Clinica do Tubarao','Espaco Saude Beirao Rendeiro']
 ?
yes
```

4.7.3 Identificar os cuidados prestados por instituição ou cidade

Para este problema, recorremos ao mesmo raciocínio das secções anteriores, isto é, utilizamos o predicado *solucoes*, para obter o conjunto pretendido. Neste caso, cada predicado recebe um parâmetro de seleção dentro da base de conhecimento. Os parâmetros de seleção são a Instituição e a Cidade. Sendo assim, recorrendo ao predicado *solucoes*, obtemos a lista com o conjunto de parâmetros que satisfazem o predicado *servico(ID,D,I,C)*, para uma dada Instituição (I) ou Cidade (C). Os parâmetros para uma dada **Instituição** são o ID, Descrição e Cidade, enquanto que para a **Cidade**, os parâmetros são o ID, Descrição e Instituição.

```
% Extensão do predicado servicoInstituicao: Instituição, Resultado -> {V,F}
servicoInstituicao(I,R) :- solucoes((ID,D,C),servico(ID,D,I,C),R).

% Extensão do predicado servicoCidade: Cidade, Resultado -> {V,F}
servicoCidade(C,R) :- solucoes((ID,D,I),servico(ID,D,I,C),R).
```

Exemplo de output:

```
| ?- servicoCidade('Braga',R).
R = [(1,'Ortopedia','Centro de Saude Santa Tecla'),(2,'Cardiologia','Centro de Saude Santa Tecla'),(3,'Neuro
cirurgia','Hospital Privado de Braga')] ?
yes
```

4.7.4 Identificar os utentes de uma instituição ou serviço

Na listagem dos utentes de uma instituição, foi necessário implementar um novo predicado, *institUtente(I,IDU)*, para podermos obter a lista dos Utentes pretendida. Este novo predicado, verifica se uma dada instituição e ID de um utente, possuem alguma relação.
Desta forma, e através do predicado *solucoes*, obtemos a lista de todos os ID's, que satisfazem o predicado *institUtente*, com o parâmetro instituição (I) fornecido. Tendo a lista com todos os ID's dos Utentes, passamos essa lista pelo predicado auxiliar *eliminarRepetidos*, para remover possíveis repetições de ID's dos Utentes.
Por fim, para obter a lista de todos os utentes, com os parâmetros que indentificam cada um, utilizamos o predicado *findUtentesServico*, para criar a lista de Utentes, onde

cada utente é identificado com o seguinte formato : (ID,Nome,Idade,Morada). O predicado *findUtentesServico(L,R)*, percorre a lista L (lista de ID's de Utentes), e faz a concatenação do conjunto de tuplos de Utentes encontrados.

```
% Extensão do predicado instituicaoUtentes: Instituição, Resultado -> {V,F}
instituicaoUtentes(I,R) :- solucoes(IDU,institUtente(I,IDU),S),
                           eliminarRepetidos(S,W),
                           findUtentesServico(W,R).

findUtentesServico([],[]).
findUtentesServico([X],R) :- solucoes((X,N,I,M),utente(X,N,I,M),R).
findUtentesServico([X|T],R) :- solucoes((X,N,I,M),utente(X,N,I,M),S),
                               findUtentesServico(T,W),
                               concat(S,W,R).

institUtente(I,IDU) :- servico(IDS,D,I,C),
                       ato(Data,IDU,IDS,Custo,IDMED).
```

Por outro lado, para a listagem dos utentes de um serviço, utilizamos o mesmo racíocinio aplicado ao caso anterior, no entanto não recorremos a nenhum novo predicado. Neste caso, inicialmente obtemos todos os ID's dos Utentes, através dos predicados *solucoes* e *ato(_,IDU,IDS,_,_)*, e do parâmetro fornecido inicialmente (servico(IDS)). Desta forma, apenas pegamos na lista dos ID's, e aplicamos o mesmo método que no caso anterior, isto é, removemos possíveis repetições de ID's dos Utentes com o predicado *eliminarRepetidos*, e posteriormente, com o predicado *findUtentesServico*, obtemos a lista com os tuplos dos Utentes, como explicado anteriormente.

```
% Extensão do predicado servicoUtentes: Servico, Resultado -> {V,F}
servicoUtentes(IDS,R) :- solucoes(IDU,ato(_,IDU,IDS,_,_),S),
                         eliminarRepetidos(S,W),
                         findUtentesServico(W,R).

findUtentesServico([],[]).
findUtentesServico([X],R) :- solucoes((X,N,I,M),utente(X,N,I,M),R).
findUtentesServico([X|T],R) :- solucoes((X,N,I,M),utente(X,N,I,M),S),
                               findUtentesServico(T,W),
                               concat(S,W,R).
```

Exemplo de output:

```
| ?- servicoInstituicao('Centro de Saude Santa Tecla',R).
R = [(1,'Ortopedia','Braga'),(2,'Cardiologia','Braga')] ?
yes
```

4.7.5 Identificar os atos médicos realizados, por utente, instituição ou serviço

Para a listagem dos atos de um dado Utente ou Servico, recorremos ao nosso método aplicado em todas as alíneas anteriores, isto é, usar o predicado *solucoes*. No entanto, criamos um novo predicado, *atoServicoInfo(D,IDU,IDS,C,IMED,Des)*, que basicamente é uma expansão do predicado *ato*, onde inclui a descrição do cuidado prestado (serviço). Desta forma, com o uso do predicado *solucoes* e *atoServicoInfo*, obtemos uma lista de tuplos com o seguinte formato: (em(Data), de(ID Utente),id(ID Serviço), servico(Descrição), custo(Custo)). A adição dos predicados *em(), de(), id(), servico(), custo()* é para uma melhor leitura do resultado obtido.
No entanto, para identificar os atos associados a uma Instituição dada, é preciso combinar alguma informação na nossa base de conhecimento. Para tal, recorremos ao predicado *solucoes* para obter a lista de ID's dos servicos efectuados na Instituição dada.

Posteriormente, implementamos o predicado *findAto(L,R)*, que recebe a lista dos ID's dos serviços, e cria a lista com todos os atos associados aqueles serviços, neste formato: (em(Data),de(ID Utente),idServico(ID Servico),custo(Custo)).

```
% Extensão do predicado atoUtente: Id_Utente, Resultado -> {V,F}
atoUtente(IDU,R) :- solucoes((em(D),de(IDU),servico(Des),custa(C)),atoServicoInfo(D,IDU,IDS,C,IDMED,Des),R).

% Extensão do predicado atoServico: Id_Serviço, Resultado -> {V,F}
atoServico(IDS,R):- solucoes((em(D),id(IDS),servico(Des),custo(C)),atoServicoInfo(D,IDU,IDS,C,IDMED,Des),R).

% Extensão do predicado atoInstituicao: Instituição, Resultado -> {V,F}
atoInstituicao(I,R) :- solucoes(IDS,servico(IDS,_,I,_),L),
                       findAto(L,R).

findAto([],[]).
findAto([X],R) :- solucoes((em(D),de(IDU),idServico(X),custo(C)),ato(D,IDU,X,C,IDMED),R).
findAto([H|T],R) :- solucoes((em(D),de(IDU),idServico(H),custo(C)),ato(D,IDU,H,C,IDMED),S),
                    findAto(T,W),
                    concat(S,W,R).

atoServicoInfo(D,IDU,IDS,C,IDMED,Des) :- ato(D,IDU,IDS,C,IDMED), servico(IDS,Des,Inst,Cidade).
```

Exemplo de output:

```
| ?- atoInstituicao('Centro de Saude Santa Tecla',R).
R = [(em('12-01-2017'),de(1),idServico(1),custo(19)),(em('13-01-2017'),de(14),idServico(2),custo(10)),(em('2
4-01-2017'),de(6),idServico(2),custo(35)),(em('19-01-2017'),de(8),idServico(2),custo(200)),(em('25-01-2017')
,de(10),idServico(2),custo(55))] ?
yes
```

4.7.6 Determinar todas as instituições ou serviços a que um utente já recorreu

Nesta alínea, existem dois casos, em que ambos recebem o mesmo parâmetro (ID do Utente), no entanto, devolvem listas diferentes. Um caso, o predicado *utenteInstituicoes(U,R)*, inicialmente lista todos os ID's dos Serviços, através do predicado *solucoes* com auxilio do predicado *ato*, isto é, verifica que ID's dos Serviços estão associados ao ID do Utente dado. De seguida, eliminamos possíveis ID's repetidos com o predicado *eliminarRepetidos*, e por fim construimos a lista com todas as Instituições, que satisfazem o predicado servico, com o ID Serviço dado. A lista é formada por tuplos com o seguinte formato: (Instituição,Cidade).

```
% Extensão do predicado utenteInstituicoes: Id_Utente, Resultado -> {V,F}
utenteInstituicoes(U,R) :- solucoes(IDS,ato(_,U,IDS,_,_),S),
                           eliminarRepetidos(S,W),
                           findInst(W,R).

findInst([],[]).
findInst([X],R) :- solucoes((I,C),servico(X,D,I,C),R).
findInst([H|T],R) :- solucoes((I,C),servico(H,D,I,C),S),
                     findInst(T,W),
                     concat(S,W,R).
```

O outro caso, é idêntico ao anterior, isto é, recebe o mesmo parâmetro (ID do Utente), forma inicialmente uma lista com os ID's dos Serviços que satisfazem o predicado *ato(_,IDU,IDS,_,_)*, e elimina dessa lista os possíveis ID's repetidos, com o predicado auxiliar *eliminarRepetidos*. Neste passo, ocorre algo diferente, comparado ao caso anterior. Neste caso, em vez de listar Instiuições, listamos os serviços, com o predicado *findServico*. O formato dos tuplos da lista é o seguinte: (ID Serviço, Descrição, Instituição, Cidade).

```
% Extensão do predicado utenteServico: Id_Serviço, Resultado -> {V,F}
utenteServico(U,R) :- solucoes(IDS,ato(_,U,IDS,_,_),S),
                      eliminarRepetidos(S,W),
                      findServico(W,R).

findServico([],[]).
findServico([X],R) :- solucoes((X,D,I,C),servico(X,D,I,C),R).
findServico([H|T],R) :- solucoes((H,D,I,C),servico(H,D,I,C),S),
                        findServico(T,W),
                        concat(S,W,R).
```

Exemplo de output:

```
| ?- utenteServico(4,R).
R = [(7,'Urologia','Casa de Saude de Caldelas','Amares')] ?
yes
```

4.7.7 Calcular o custo total dos atos médicos por utente, serviço, instituição ou data

Esta listagem de conhecimento foi abordada de maneira relativamente diferente das anteriores, uma vez que, neste caso, a informação não precisava apenas de ser listada conforme o conhecimento base, esta teria primeiro de ser interpretada, que no caso, seria efetuar o somatório de todos os custos conseguidos em cada uma destas diferentes situações.
Assim sendo, para cada uma destas situações foi construída uma lista com os custos relacionados com o parâmetro dado, através do uso do predicado *solucoes* em conjunto com o predicado *ato(D,IDU,IDS,Custo,IDMED)*, percorrendo assim todos os **atos** presentes na base de conhecimento e selecionando apenas o custo associado aos atos que faziam corresponder com o respectivo parâmetro fornecido. Houve, no entanto, uma exceção: o caso da **Instituição**, uma vez que esta não consegue ser obtida diretamente através do ato, foi necessário utilizar o predicado adicional *findCusto* para, através do ID de um serviço, obter o seu custo associado conforme o respectivo ato.
Por fim, restava apenas efetuar o somatório dos custos incluídos nesta lista, conseguindo assim o custo total pretendido com a utilização do predicado *sum*.

```
% Extensão do predicado custoPorUtente: Id_Utente, Resultado -> {V,F}
custoPorUtente(IDU,R) :- solucoes(Custo,ato(D,IDU,IDS,Custo,IDMED),S),
                         sum(S,R).

% Extensão do predicado custoPorServico: Id_Serviço, Resultado -> {V,F}
custoPorServico(IDS,R) :- solucoes(Custo,ato(D,IDU,IDS,Custo,IDMED),S),
                          sum(S,R).

% Extensão do predicado custoPorData: Data, Resultado -> {V,F}
custoPorData(Data,R) :- solucoes(Custo,ato(Data,IDU,IDS,Custo,IDMED),S),
                        sum(S,R).

% Extensão do predicado custoPorInstituicao: Instituição, Resultado -> {V,F}
custoPorInstituicao(I,R) :- solucoes(IDS,servico(IDS,D,I,C),S),
                            findCusto(S,W),
                            sum(W,R).

findCusto([],[]).
findCusto([X],R) :- solucoes(C,ato(_,_,X,C,_),R).
findCusto([X|T],R) :- solucoes(C,ato(_,_,X,C,_),S),
                      findCusto(T,W),
                      concat(S,W,R).
```

Exemplo de output:

```
| ?- custoPorUtente(11,R).
R = 52 ?
yes
```

4.7.8 Listagem de informação sobre Médicos

Como adição aos requisitos mínimos do exercício, o grupo resolveu realizar uma série de predicados extra relacionados com a fonte de conhecimento adicionada: o **Médico**.
Deste modo, inicialmente foram criados os predicados de identificação de médicos por critérios de seleção, tal como anteriormente o foram para os utentes.

```
% Extensão do predicado medicoID: ID, Resultado -> {V,F}
medicoID(ID,R) :- solucoes((ID,N,I,M,E),medico(ID,N,I,M,E),R).

% Extensão do predicado medicosNome: Nome, Resultado -> {V,F}
medicosNome(N,R) :- solucoes((ID,N,I,M,E),medico(ID,N,I,M,E),R).

% Extensão do predicado medicosIdade: Idade, Resultado -> {V,F}
medicosIdade(I,R) :- solucoes((ID,N,I,M,E),medico(ID,N,I,M,E),R).

% Extensão do predicado medicosMorada: Morada, Resultado -> {V,F}
medicosMorada(M,R) :- solucoes((ID,N,I,M,E),medico(ID,N,I,M,E),R).

% Extensão do predicado medicosEspecialização: Especialização, Resultado -> {V,F}
medicosEspecializacao(E,R) :- solucoes((ID,N,I,M,E),medico(ID,N,I,M,E),R).
```

Tal como podemos verificar pela imagem anterior, os predicados anteriores seguem a mesma estrutura que os apresentados anteriormente para os critérios de seleção dos utentes, na secção **4.7.1**.
Posteriormente foi construído o predicado *atosMedico*, com o simples objetivo de apresentar estruturadamente todos os atos nos quais um determinado médico participou. Para conseguir isto, foi utilizado o predicado *solucoes* em conjunto com o predicado auxiliar, também já utilizado anteriormente, *atoServicoInfo(D,IDU,IDS,C,IDMED,Des)*, só assim conseguindo apresentar toda a informação necessária sobre o respectivo ato e ainda a descrição associada ao correspondente serviço.

```
% Extensão do predicado atosMedico: Id_Médico, Resultado -> {V,F}
atosMedico(IDMED,R) :- solucoes((em(D),id_utente(IDU),servico(Des),custa(C)),
                                atoServicoInfo(D,IDU,IDS,C,IDMED,Des),R).
```

De seguida, o grupo elaborou o predicado *custoMedioPorMedico* que, através de um dado ID de um médico, apresentaria o custo médio por ato realizado por este. Assim, para calcular corretamente este valor, seria necessário acumular numa lista o custo associado a cada ato onde o médico ,dado como input, fizesse correspondência, através do predicado *solucoes* em conjunto com o predicado *ato(D,IDU,IDS,Custo,IDMED)*. Consequentemente, seria apenas necessário fazer o somatório e o tamanho da lista resultante, efetuando, por fim, a divisão dos respectivos valores, conseguindo assim o valor pretendido.

```
% Extensão do predicado custoMedioPorAto: Id_Médico, Resultado -> {V,F}
custoMedioPorMedico(IDMED,R) :- solucoes(Custo,ato(D,IDU,IDS,Custo,IDMED),S),
                                sum(S,Sum),
                                comprimento(S,L),
                                R is Sum/L.
```

Após este, foi elaborado o predicado *instituicoesMedico* que, através do ID de um dado médico era responsável por apresentar todas as instituições onde este já teria prestado qualquer tipo de serviço. Para conseguir isto seria necessário conjugar o predicado *solucoes* com o predicado *ato* de modo a percorrer todos os atos da base de conhecimento,

filtrando estes pelos atos em que o respectivo médico era interveniente. Foi ainda utilizado, como auxiliar, o predicado *findInst* para reconher a instituição do serviço a que o ato correspondia, fazendo assim uma lista de instituições queria seria, por fim, submetida ao predicado *eliminarRepetidos* para eliminiar, em caso de existência, instituições repetidas na lista.

```
% Extensão do predicado instituicoesMedico: Id_Médico, Resultado -> {V,F}
instituicoesMedico(M,R) :- solucoes(IDS,ato(_,_,IDS,_,M),S),
                           findInst(S,W),
                           eliminarRepetidos(W,R).
```

Por fim, foi construído o predicado *medicosInstituicao* com o objetivo de, dado uma determinada Instituição da base de conhecimento, fornecer todos os médicos cujos atos/serviços tivessem ocorrido nessa. Para conseguir alcançar isto, foi necessária, uma vez mais, a utilização do predicado *solucoes* que em conjunto com o predicado *institMed* permitiu filtrar os médicos pelos atos/serviços na respectiva Instituição. Restava apenas, utilizar o predicado *eliminarRepetidos* para eliminar eventuais repetições e o predicado *findMedicosServico* de modo a conseguir apresentar a informação resultante o mais completa possível.

```
% Extensão do predicado medicosInstituicao: Instituição, Resultado -> {V,F}
medicosInstituicao(I,R) :- solucoes(IDMED,institMed(I,IDMED),S),
                           eliminarRepetidos(S,W),
                           findMedicosServico(W,R).

institMed(I,IDMED) :- servico(IDS,D,I,C),
                      ato(Data,IDU,IDS,Custo,IDMED).

findMedicosServico([],[]).
findMedicosServico([X],R) :- solucoes((X,N,I,M,E),medico(X,N,I,M,E),R).
findMedicosServico([X|T],R) :- solucoes((X,N,I,M,E),medico(X,N,I,M,E),S),
                               findMedicosServico(T,W),
                               concat(S,W,R).
```

Exemplo de output:

```
| ?- instituicoesMedico(1,R).
R = [('Centro de Saude Santa Tecla','Braga'),('Espaco Saude Beirao Rendeiro','Moledo')] ?
yes
```

5. *Conclusões*

Sendo este o primeiro exercício do trabalho prático da Unidade Curricular, foi, de certa forma, uma novidade para nós, uma vez que nunca havíamos realizado nenhum projeto na linguagem de programação lógica *PROLOG*, para além dos exercícios elaborados nas aulas práticas.
Deste modo, inicialmente deparamo-nos com várias dificuldades no que toca à descoberta de estratégias que melhor se ajustassem à implementação de alguns requisitos. A nossa maior dificuldade foi conseguir relacionar as diferentes fontes de informação, ou seja, a partir de uma fonte conseguir chegar a outras que estivessem relacionadas com esta, para deste modo, filtrar a informação necessária. No entanto, depois de alguma dedicação e pesquisa, conseguimos ultrapassar esta dificuldade e várias outras com que mais à frente nos deparamos.
Assim, a realização deste exercício foi bastante importante para começarmos a entender como é que esta linguagem de programação lógica funciona e que problemas podem ser resolvidos e analisados com ela. Conseguimos, desta forma, consolidar não só todos os conhecimentos adquiridos nas aulas práticas, como também vários outros que foram sendo necessários durante a realização deste exercício. Concluindo, no geral estamos satisfeitos e orgulhosos com o trabalho aqui desenvolvido e esperemos que assim continue nos próximos que se avizinham.