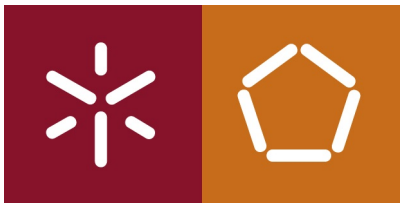
Universidade do Minho



Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

3º Ano, 2º Semestre, 2018/2019

Programação em Lógica e Invariantes

A81712 Ana Filipa Pereira
A81368 Inês Alves
A81580 Francisco Freitas
A81611 Maria Dias
A80975 Pedro Freitas

April 9, 2019

1 Resumo

O seguinte relatório, proposto pelos docentes da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, tem como objetivo dar resposta às diversas questões abordadas no enunciado deste mesmo exercício. Como alunos do 3.º ano do Mestrado Integrado em Engenharia Informática, procuramos, com este exercício, aprofundar os nossos conhecimentos no que toca à linguagem de programação lógica *PROLOG*. Com o intuito de atingir este objetivo, o grupo procura, ao longo deste documento, explicar e exemplificar a estruturação, de raíz, 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.

Índice

Lista de figuras

2 Introdução

No âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, foi-nos proposto que explorássemos as nossas competências ao nível da linguagem de programação lógica *PROLOG*, começando, para isso, por resolver um primeiro exercício especificado neste relatório, cujo tema é *Programação em Lógica e Invariantes*.

Com este exercício, pretende-se que seja desenvolvido um Sistema de Representação de Conhecimento e Raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde.

Esclarecido o cenário, facilmente identificamos as três principais fontes de conhecimento deste exercício: o Utente, o Serviço e a Consulta. Assim, para além de possibilitar dar resposta às questões apresentadas no enunciado, pretende-se também criar um sistema que esteja apto a guardar e processar informações relativamente a estas três fontes.

3 Preliminares

Como era de esperar, todo o sistema a desenvolver vai ser focado nas três fontes de conhecimento identificadas, pelo que é essencial que as aprofundemos um pouco, visto que cada uma destas fontes possui um conjunto de atributos que as caracterizam.

Começando pela única fonte de conhecimento que interage diretamente com o sistema, o Utente, possui, associado a si próprio, um ID, um Nome, uma Idade e uma Cidade.

De seguida, o Serviço, neste caso um cuidado de saúde, é caracterizado pela associação de um ID, uma Descrição, uma Instituição e uma Cidade.

Por fim, a Consulta tem associada a si uma Data, o ID de um Utente, o ID de um Serviço e ainda um Custo. Prestando um pouco de atenção nesta última fonte de conhecimento, podemos concluir que, uma vez que tem associado a si IDs de outras fontes, a Consulta é a ponte entre as outras duas fontes. Ou seja, é com esta que conseguimos determinar quando é que um utente necessitou de um certo serviço e ainda qual o custo dessa prestação de cuidados.

Posto isto, temos o seguinte panorama:

- **Utente** : $\#IdUtente, Nome, Idade, Cidade \rightsquigarrow \{V, F\}$
- **Servico** : $\#IdServico, Descricao, Instituicao, Cidade \rightsquigarrow \{V, F\}$
- **Consulta** : $Data, \#IdUtente, \#IdServico, Custo \rightsquigarrow \{V, F\}$

De notar que, de modo a que todo este sistema faça sentido, tanto o ID do Utente, como o ID do Serviço são identificadores únicos das suas fontes de conhecimentos, sendo que é a partir destes que é possível identificarmos um determinado Utente ou Serviço, inequivocamente. Deste modo, é impossível que haja mais do que um Utente ou um Serviço com o mesmo ID.

Já no que toca à Consulta efetuada, é razoável a existência de vários serviços com o mesmo *#IDUtente* e/ou mesmo *#IDServiço*.

Apesar de existirem diversos casos, a título de exemplo, imaginemos que um determinado utente X necessita constantemente do serviço Y. Neste caso, é natural que existam vários registos deste utente com aquele serviço, sendo a data e o custo da consulta fatores variantes nesta equação.

Concluída a fase de desmistificação das diversas fontes de conhecimento utilizadas para este sistema, consideramos que podemos, agora, avançar para a fase seguinte do exercício: a execução, propriamente dita, do mesmo.

4 Descrição do trabalho e Análise de resultados

4.1 Evolução e involução de conhecimento

Em prolog, as regras especificam algo que pode ser verdade, se uma dada condição for satisfeita[1]. Para introduzir e remover informação da base de conhecimento, é necessário recorrer a algumas regras, de forma a controlar as entradas e saídas de informação.

A regra para introdução de conhecimento na base de conhecimento denomina-se **evolucao**, que assegura que não existe já o elemento que estamos a tentar introduzir. A regra **involucao**, por outro lado, assegura que não é removido um elemento que não existe. Estes factos são garantidos pelas regras através do teste que se faz aos invariantes - caso algum dos invariantes falhe, a inserção/remoção falha também.

A implementação de invariantes no nosso programa é fulcral para o controlo da informação presente na base de conhecimento. É necessário garantir que não existe informação repetida ou inconsistente nesta.

```
evolucao( T ) :- solucoes( I,+T::I,L ),
                    insercao( T ), teste( L ).

involucao( T ) :- solucoes( I,-T::I,L ),
                teste( L ), remocao( T ).

remocao( T ) :- retract( T ).

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

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

4.2 Invariantes

Os invariantes que se seguem garantem que não existem dois utentes, dois serviços ou dois médicos com o mesmo ID.

```
+utente( ID,N,I,C ) :: ( solucoes( ID,(utente(ID,_,_,_)),S ),
                  comprimento( S,N ), N == 1).

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

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

Ao remover ocorrências da base de conhecimento, é necessário garantir que os elementos que queremos eliminar existem, de facto. Os próximos invariantes denotam essa característica.

```
-utente( ID,NO,I,C ) :: ( solucoes( ID,utente(ID,_,_,_),S ),
                    comprimento( S,N ), N == 1).

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

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

A consulta não possui invariante que controla inserções, pois não existe nenhum conjunto de atributos que a possam identificar univocamente, uma vez que podem ocorrer várias consultas do mesmo cliente a receber o mesmo serviço, até no mesmo dia e pelo mesmo preço. Por isso, achamos que não fazia sentido construir esse invariante.

No entanto, no que toca às consultas, não podem haver ocorrências com identificadores de utilizadores e de serviços inexistentes. Para além disso, um médico não pode dar uma consulta que não seja da sua especialidade.

```
+consulta( D,IDU,IDS,C,IDM ) ::
                        utente( IDU,_,_,_ ),
                        servico( IDS,_,_,_ ),
                        medico( IDM,_,_,_ ).

+consulta( D,IDU,IDS,C,IDM ) ::
                        ( solucoes( (IDM,E),(medico(IDM,_,E,_)),S ),
                        comprimento( S,N ), N >= 1).

-consulta( D,IDU,IDS,C,IDM ) ::
           ( solucoes( (D,IDU,IDS),(consulta(D,IDU,IDS,_,_)),S ),
           comprimento( S,N ), N == 1).
```

No que toca a remover utentes, serviços ou médicos da base de conhecimento, é preciso ter em atenção de que poderão existir consultas em nome desses mesmos utentes, serviços e médicos. De forma a manter o programa consistente, aquando destas remoções, deve ser feita uma verificação de que não existem consultas em conflito com essa ação. Ou seja, não podemos apagar um utente da nossa base de conhecimento se esse utente tiver alguma consulta registada. O mesmo raciocínio serve para os serviços e médicos.

```
-utente( ID,NO,I,C ) :: (solucoes( ID,consulta(_,ID,_,_,_),S ),
                         comprimento( S,N ), N == 0).

-servico( ID,D,I,C ) :: (solucoes( ID,consulta(_,_,ID,_,_),S ),
                         comprimento( S,N ), N == 0).

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

4.3 Fonte de Conhecimento extra: Médico

Após uma leitura atenta pelo enunciado do projeto, a fim de acrescentar conhecimento ao sistema, os elementos do grupo decidiram criar uma nova fonte de conhecimento, estando esta, obviamente, enquadrada no cenário a explorar. A nova fonte de conhecimento idealizada, assim como o utente, interage diretamente com o sistema, sendo esta o **Médico** de cada cuidado de saúde prestado.

Com esta nova aquisição, o panorama estrutural inicialmente definido pelo enunciado do projeto, sofreu alterações:

- **Utente** : $\#IdUtente, Nome, Idade, Cidade \rightsquigarrow \{V, F\}$
- **Servico** : $\#IdServico, Descricao, Instituicao, Cidade \rightsquigarrow \{V, F\}$
- **Consulta** : $Data, \#IdUtente, \#IdServico, Custo, \#IdMedico \rightsquigarrow \{V, F\}$
- **Médico** : $\#IdMedico, Nome, Especialidades, Instituicoes \rightsquigarrow \{V, F\}$

4.4 Base de Conhecimento

Uma vez que este sistema foi inteiramente criado de raíz pelo grupo, é necessário garantirmos que conseguimos testar todas as funcionalidades e respostas que dermos. Para esta finalidade, torna-se importante termos uma base de conhecimento com que possamos trabalhar. De notar que esta base procura abranger diversos casos omissos a olho nu. Em anexo encontra-se a nossa base de conhecimento inicial.

4.5 Apresentação e Análise de Resultados

4.5.1 Registar utentes, serviços, consultas e médicos

Seguem-se os predicados que permitem registar um utente, um serviço, uma consulta ou um médico:

```
% Extensao do predicao registar_utente: IdUtente, Nome, Idade, Morada -> {V, F}

registar_utente(X,Y,W,Z) :- evolucao(utente(X,Y,W,Z)).

% Extensao do predicao registar_servico: IdServico, Descricao, Instituicao, Cidade -> {V,F}

registar_servico(X,Y,W,Z) :- evolucao(servico(X,Y,W,Z)).

% Extensao do predicao registar_consulta: Data, IdUtente, IdServico, Custo, IdMedico -> {V,F}

registar_consulta(X,Y,W,Z,M) :- evolucao(consulta(X,Y,W,Z,M)).

% Extensao do predicao registar_medico: IdMedico, Nome, Especialidades, Instituicoes -> {V,F}

registar_medico(X,Y,W,Z) :- evolucao(medico(X,Y,W,Z)).
```

Figura 1: Predicados de registo das diversas fontes de conhecimento

Estes predicados recebem os atributos de uma qualquer fonte de conhecimento das quatro referidas e passa-o ao predicado *evolucao*, que, por sua vez, trata de inserir os novos dados na base de conhecimento.

De seguida, apresentamos o output do registo de um utente. Para os serviços, consultas e médicos, o processo é claramente análogo.

```
2 ?- listing(utente).
:- dynamic utente/4.

utente(1, 'Ana Santos', 34, 'Braga').
utente(2, 'Bruno Mendonça', 23, 'Porto').
utente(3, 'Carla Martins', 29, 'Guimarães').
utente(4, 'Beatriz Jesus', 12, 'Lisboa').
utente(5, 'Júlio Carvalho', 36, 'Braga').
utente(6, 'Carlos Silva', 19, 'Porto').
utente(7, 'Xavier Teixeira', 51, 'Braga').
utente(8, 'Luis Almeida', 32, 'Braga').
utente(9, 'Pedro Lima', 20, 'Braga').
utente(10, 'André Campos', 42, 'Lisboa').
utente(11, 'Teresa Cerqueira', 54, 'Guimarães').
utente(12, 'Sérgio Gonçalves', 16, 'Coimbra').
utente(13, 'João Pereira', 24, 'Braga').
utente(14, 'Henrique Castro', 83, 'Porto').
utente(15, 'Sara Fernandes', 27, 'Lisboa').

true.

3 ?- registar_utente(15,'Diana Fontes',20,'Braga').
false.
```

Figura 2: Exemplo de output para o predicado de registar utentes (1)

```
4 ?- registar_utente(16,'Diana Fontes',20,'Braga').
true .

5 ?- listing(utente).
:- dynamic utente/4.

utente(1, 'Ana Santos', 34, 'Braga').
utente(2, 'Bruno Mendonça', 23, 'Porto').
utente(3, 'Carla Martins', 29, 'Guimarães').
utente(4, 'Beatriz Jesus', 12, 'Lisboa').
utente(5, 'Júlio Carvalho', 36, 'Braga').
utente(6, 'Carlos Silva', 19, 'Porto').
utente(7, 'Xavier Teixeira', 51, 'Braga').
utente(8, 'Luis Almeida', 32, 'Braga').
utente(9, 'Pedro Lima', 20, 'Braga').
utente(10, 'André Campos', 42, 'Lisboa').
utente(11, 'Teresa Cerqueira', 54, 'Guimarães').
utente(12, 'Sérgio Gonçalves', 16, 'Coimbra').
utente(13, 'João Pereira', 24, 'Braga').
utente(14, 'Henrique Castro', 83, 'Porto').
utente(15, 'Sara Fernandes', 27, 'Lisboa').
utente(16, 'Diana Fontes', 20, 'Braga').

true.
```

Figura 3: Exemplo de output para o predicado de registar utentes (2)

Como podemos ver na Figura 2, ao tentar fazer o registo de um utente com um ID já ocupado, a operação não é permitida. Inserindo um ID válido, o utente é então inserido normalmente na base de conhecimento.

4.5.2 Remover utentes, serviços, consultas e médicos

De forma semelhante ao registo das diferentes fontes de conhecimento, temos a remoção das mesmas:

```
% Extensao do predicao remover_utente: IdUtente, Nome, Idade, Morada -> {V, F}

remover_utente(X,Y,W,Z) :- involucao(utente(X,Y,W,Z)).

% Extensao do predicao remover_servico: IdServico, Descricao, Instituicao, Cidade -> {V,F}

remover_servico(X,Y,W,Z) :- involucao(servico(X,Y,W,Z)).

% Extensao do predicao remover_consulta: Data, IdUtente, IdServico, Custo, IdMedico -> {V,F}

remover_consulta(X,Y,W,Z,M) :- involucao(consulta(X,Y,W,Z,M)).

% Extensao do predicao remover_medico: IdMedico, Nome, Especialidades, Instituicoes -> {V,F}

remover_medico(X,Y,W,Z) :- involucao(medico(X,Y,W,Z)).
```

Figura 4: Predicados de remoção das diversas fontes de conhecimento

No exemplo que se segue podemos ver que, ao tentar remover um serviço que tem a si associado pelo menos uma consulta, a operação falha. Após remover a consulta relativa àquele serviço podemos, então, removê-lo com sucesso.

```
38 ?- listing(servico).
:- dynamic servico/4.

servico(1, 'Oftalmologia', 'Hospital de Braga', 'Braga').
servico(2, 'Ginecologia', 'Centro de Saúde de Maximinos', 'Braga').
servico(3, 'Cardiologia', 'Hospital da Senhora da Oliveira Guimarães', 'Guimarães').
servico(4, 'Ortopedia', 'Hospital S.João', 'Porto').
servico(5, 'Oftalmologia', 'Centro de Saúde de Maximinos', 'Braga').
servico(6, 'Cardiologia', 'Hospital de Braga', 'Braga').
servico(7, 'Otorrinolaringologia', 'Hospital de Braga', 'Braga').
servico(8, 'Oftalmologia', 'Centro Hospitalar e Universitário de Coimbra', 'Coimbra').
servico(9, 'Cardiologia', 'Hospital S.João', 'Porto').
servico(10, 'Oftalmologia', 'Hospital de Santa Maria', 'Porto').
servico(11, 'Dermatologia', 'Hospital S.José', 'Lisboa').
servico(12, 'Otorrinolaringologia', 'Hospital S.José', 'Lisboa').
servico(13, 'Ginecologia', 'Hospital da Senhora da Oliveira Guimarães', 'Guimarães').
servico(14, 'Psiquiatria', 'Hospital da Luz', 'Lisboa').
servico(15, 'Dermatologia', 'Hospital de Santa Maria', 'Porto').

true.

39 ?- remover_servico(14,_,_,_).
false.

40 ?- remover_consulta(_,4,14,_,_).
true.

41 ?- remover_servico(14,_,_,_).
true.
```

Figura 5: Exemplos de output para a remoção de serviços

4.5.3 Identificar as instituições prestadoras de serviços

De modo a identificar todas as instituições prestadoras de serviços contidos na nossa base de conhecimento foi necessário usar o predicado *solucoes*. Assim, obtemos o conjunto de todos os valores do parâmetro Instituição presentes nos serviços existentes. Por fim, aplicamos o predicado auxiliar *removeRepetidos* que remove os repetidos da lista de instituições.

```
% Extensao do predicado identificaInstituicoes: ListaInstituicoes -> {V,F}

identificaInstituicoes(Lista) :- solucoes(I,servico(_,_,I,_),L),removeRepetidos(L,Lista).
```

Figura 6: Predicado de identificação de instituições prestadoras de serviços

Como podemos ver na figura 7, esta regra retorna *false* se for dada como input uma lista com instituições que não prestam serviços. Se todos os elementos da lista forem instituições prestadoras, então o resultado é *true*.

```
2 ?- identificaInstituicoes(['Hospital de Braga','Hospital Privado de Braga','Hospital da Boa Ajuda']).
false.

3 ?- identificaInstituicoes(['Hospital de Braga']).
true .

4 ?- identificaInstituicoes(R).
R = ['Hospital de Braga', 'Centro de Saúde de Maximinos', 'Hospital da Senhora da Oliveira Guimarães', 'Hospital S.João',
'Centro Hospitalar e Universitário de Coimbra', 'Hospital de Santa Maria', 'Hospital S.José', 'Hospital da Luz'] .
```

Figura 7: Output para a identificação de instituições prestadoras de serviços

4.5.4 Identificar utentes/serviços/consultas por critérios de seleção

Estes predicados têm o intuito de identificar utentes, serviços e consultas pelos seus atributos, ou seja, dado um ID, Nome, Cidade, etc, é apresentado o conjunto de utentes/serviços/consultas que correspondem a esses parâmetros.

- **Identificar utentes por critérios de seleção**

```
% Extensao do predicado identificar_utenteID: IDUt,Utente -> {V,F}

identificar_utenteID(ID, R) :- solucoes((ID, N, I, C), utente(ID, N, I, C), R).

% Extensao do predicado identificar_utenteNome: Nome,ListaUtentes -> {V,F}

identificar_utenteNome(NOME, R) :- solucoes((ID, NOME, I, C), utente(ID, NOME, I, C), R).

% Extensao do predicado identificar_utenteIdade: Idade,ListaUtentes -> {V,F}

identificar_utenteIdade(IDADE, R) :- solucoes((ID, N, IDADE, C), utente(ID, N, IDADE, C), R).

% Extensao do predicado identificar_utenteCidade: Cidade,ListaUtentes -> {V,F}

identificar_utenteCidade(CIDADE, R) :- solucoes((ID, N, I, CIDADE), utente(ID, N, I, CIDADE), R).
```

Figura 8: Predicado de identificação de utentes por critérios de seleção

```
5 ?- identificar_utenteID(1,U).
U = [(1, 'Ana Santos', 34, 'Braga')].

6 ?- identificar_utenteNome('Pedro Lima',U).
U = [(9, 'Pedro Lima', 20, 'Braga')].

7 ?- identificar_utenteIdade(24,U).
U = [(13, 'João Pereira', 24, 'Braga')].

8 ?- identifcar_utenteCidade('Guimarães',U).
Correct to: "identificar_utenteCidade('Guimarães',U)"?
Please answer 'y' or 'n'? yes
U = [(3, 'Carla Martins', 29, 'Guimarães'),  (11, 'Teresa Cerqueira', 54, 'Guimarães')].
```

Figura 9: Output para a identificação de utentes por critérios de seleção

- **Identificar serviços por critérios de seleção**

```
% Extensao do predicado identificar_servicoID: IDServ,Servico -> {V,F}

identificar_servicoID(ID, R) :- solucoes((ID, D, I, C), servico(ID, D, I, C), R).

% Extensao do predicado identificar_servicoDescricao: Descricao,Servico -> {V,F}

identificar_servicoDescricao(DESC, R) :- solucoes((ID, DESC, I, C), servico(ID, DESC, I, C), R).

% Extensao do predicado identificar_servicoInstituicao: Instituicao,ListaServicos -> {V,F}

identificar_servicoInstituicao(INST, R) :- solucoes((ID, D, INST, C), servico(ID, D, INST, C), R).

% Extensao do predicado identificar_servicoCidade: Cidade,ListaServicos -> {V,F}

identificar_servicoCidade(CITY, R) :- solucoes((ID, D, I, CITY), servico(ID, D, I, CITY), R).
```

Figura 10: Predicado de identificação de serviços por critérios de seleção

```
9 ?- identificar_servicoID(4,S).
S = [(4, 'Ortopedia', 'Hospital S.João', 'Porto')].

10 ?- identificar_servicoDescricao('Psiquiatra',S).
S = [].

11 ?- identificar_servicoInstituicao('Hospital S.João',S).
S = [(4, 'Ortopedia', 'Hospital S.João', 'Porto'),  (9, 'Cardiologia', 'Hospital S.João', 'Porto')].

12 ?- identificar_servicoCidade('Guimarães',S).
S = [(3, 'Cardiologia', 'Hospital da Senhora da Oliveira Guimarães', 'Guimarães'),  (13, 'Ginecologia', 'Hospital da Senhora
 da Oliveira Guimarães', 'Guimarães')].
```

Figura 11: Output para o predicado de identificação de serviços por critérios de seleção

- **Identificar consultas por critérios de seleção**

```
% Extensao do predicado identificar_consultaData: Data,ListaConsultas -> {V,F}

identificar_consultaData(DATE, R) :- solucoes((DATE, Idutente, Idservico, Custo,Idmedico),
                                     consulta(DATE, Idutente, Idservico, Custo,Idmedico), R).

% Extensao do predicado identificar_consultaIDUtente: IdUt,ListaConsultas -> {V,F}

identificar_consultaIDUtente(IDU, R) :- solucoes((D, IDU, Idservico, Custo,Idmedico),
                                        consulta(D, IDU, Idservico, Custo,Idmedico), R).

% Extensao do predicado identificar_consultaIDServico: IdServ,ListaConsultas -> {V,F}

identificar_consultaIDServico(IDS, R) :- solucoes((D, Idutente, IDS, Custo,Idmedico),
                                         consulta(D, Idutente, IDS, Custo,Idmedico), R).

% Extensao do predicado identificar_consultaCusto: Custo,ListaConsultas -> {V,F}

identificar_consultaCusto(CUSTO, R) :- solucoes((D, Idutente, Idservico, CUSTO,Idmedico),
                                       consulta(D, Idutente, Idservico, CUSTO,Idmedico), R).
```

Figura 12: Predicado de identificação de consulta por critérios de seleção

```
13 ?- identificar_consultaData('12-10-2018',C).
C = [('12-10-2018', 15, 11, 47, 1)].

14 ?- identificar_consultaIDUtente(11,C).
C = [('10-12-2018', 11, 3, 30, 4),  ('02-03-2019', 11, 3, 30, 4)].

15 ?- identificar_consultaIDServico(5,C).
C = [('23-05-2018', 7, 5, 25, 6)].

16 ?- identificar_consultaCusto(35,C).
C = [('16-09-2018', 5, 6, 35, 7),  ('29-01-2019', 8, 6, 35, 7),  ('25-09-2018', 14, 9, 35, 11)].
```

Figura 13: Output do predicado de identificação de consultas por critérios de seleção

4.5.5 Identificar serviços prestados por instituição/cidade/datas/custo

Da mesma forma que na secção anterior, recorremos ao predicado *solucoes* para identificar os serviços efetuados por uma dada instituição. Então, o predicado *servicosInstituicao* recebe uma instituição I e adquire o conjunto de serviços(S) que satisfazem o predicado *servico(ID,S,I,C)*. Analogamente se obtém os serviços prestados em determinada cidade.

Similarmente, para identificar os serviços ocorridos em determinada data D ou com um dado custo C, são adquiridos os conjuntos de IDs dos serviços que satisfazem o

predicado *consulta(D,IDUt,IDServ,C,IDMed)*. Sendo aplicado de seguida o predicado auxiliar *descServicos* com o intuito de destacar as descrições dos serviços com os IDs do conjunto obtido.

```
% Extensao do predicado servicosInstituicao: Instituicao, ListaServicos -> {V,F}

servicosInstituicao(I,Lista) :- solucoes(S,servico(_,S,I,_),R),removeRepetidos(R,Lista).


% Extensao do predicado servicosCidade: Cidade, ListaServicos -> {V,F}

servicosCidade(C,Lista) :- solucoes(S,servico(_,S,_,C),R), removeRepetidos(R,Lista).


% Extensao do predicado servicosData: Data, ListaServicos -> {V,F}

servicosData(D,Lista) :- solucoes(IDServ,consulta(D,_,IDServ,_,_),R), descServicos(R,T),removeRepetidos(T,Lista).


% Extensao do predicado servicosCusto: Custo, ListaServicos -> {V,F}

servicosCusto(C,Lista) :- solucoes(IDServ,consulta(_,_,IDServ,C,_),L), descServicos(L,T), removeRepetidos(T,Lista).
```

Figura 14: Predicados de identificação dos serviços prestados por instituição, cidade, data e custo

```
17 ?- servicosInstituicao('Centro de Saúde de Maximinos',S).
S = ['Ginecologia', 'Oftalmologia'].

18 ?- servicosCidade('Braga',S).
S = ['Oftalmologia', 'Ginecologia', 'Cardiologia', 'Otorrinolaringologia'] .

19 ?- servicosData('29-01-2019',S).
S = ['Cardiologia'] .

20 ?- servicosCusto(47,S).
S = ['Ortopedia', 'Dermatologia'] .
```

Figura 15: Exemplo de output do predicado de identificação de serviços prestados por instituição, data, cidade e custo

4.5.6 Identificar os utentes de um serviço/instituição

Para obter a listagem dos utentes de um dado serviço criamos um predicado *utentesServico(I,S)*. Com este predicado, obtemos todos os IDs dos Utentes através dos predicados *solucoes* e *consulta*, e do parâmetro fornecido inicialmente, ID de um serviço *I*.

Depois de já termos a lista dos IDs dos utentes, removemos possíveis repetições de IDs com o predicado *removeRepetidos* e, posteriormente, para obter a lista de todos os utentes, com a informação que os identificam, (ID, Nome, Idade e Cidade), utilizamos o predicado *encontrarUtentes*. Este predicado percorre a lista de IDs de Utentes e, através do predicado *identificar_utenteID*, obtém o tuplo que contém a informação relativa ao mesmo. Por fim, faz-se a concatenação do conjunto de tuplos de utentes encontrados.

Por outro lado, para obter a listagem dos utentes de uma instituição, criamos o predicado *utentesInstituicao(I,R)*. Neste predicado, analogamente ao predicado *utentesServico*, obtemos todos os IDs de serviços prestados na instituição *I* através dos predicados

solucoes, *servico* e do parâmetro fornecido inicialmente, instituição *I*, resultando na lista de IDs de serviços prestados nessa instituição.

Tendo a lista de IDs de serviços, para que fosse possível obter a lista de todos os utentes, criamos uma predicado *servicosUtentes* que recebe uma lista de IDs de serviços e devolve uma lista de tuplos de utentes. Este predicado aplica a cada elemento da lista de IDs o predicado *utentesServico*, já explicado anteriormente e, de seguida, faz a concatenação de tuplos de utentes.

Por fim, utilizamos o predicado *removeRepetidos*, para remover tuplos repetidos.

```
% Extensao do predicado utentesServico: Servico, ListaUtentes -> {V,F}

utentesServico(I,S):- solucoes(IDU,consulta(_,IDU,I,_,_),W),removeRepetidos(W,U),encontrarUtentes(U,S).


% Extensao do predicado utentesInstituicao: Instituicao,ListaUtentes -> {V,F}

utentesInstituicao(I,R) :- solucoes(IDS,servico(IDS,_,I,_),S),servicosUtentes(S,R).


%--- Auxiliar da 6

% Extensao do predicado encontrarUtentes: ListaIDUt,ListaUtentes -> {V,F}

encontrarUtentes([],[]).
encontrarUtentes([H|T],R) :- identificar_utenteID(H,W),encontrarUtentes(T,S),concatenar(S,W,R).


% Extensao do predicado servicosUtentes: ListaServicos,ListaUtentes -> {V,F}

servicosUtentes([],[]).
servicosUtentes([H|T],R) :- utentesServico(H,W),servicosUtentes(T,S),concatenar(W,S,Z),removeRepetidos(Z,R).
```

Figura 16: Predicados de identificação dos utentes de um serviço e de uma instituição

```
21 ?- utentesServico(9,U).
U = [(14, 'Henrique Castro', 83, 'Porto')] .

22 ?- utentesInstituicao('Hospital de Braga',U).
U = [(13, 'João Pereira', 24, 'Braga'),  (8, 'Luis Almeida', 32, 'Braga'),  (5, 'Júlio Carvalho', 36, 'Braga'),  (9, 'Pedro
Lima', 20, 'Braga')] .
```

Figura 17: Exemplo de output do predicado de identificação de utentes de um serviço e de uma instituição

4.5.7 Identificar consultas realizados por utente/instituição/cidade

De forma a obtermos a lista das consultas consoante o desejado temos dois tipos de processos diferentes.

Se quiséssemos todas as consultas de um determinado cliente, bastaria fornecer o ID do mesmo e procurar todas as consultas, selecionando apenas aquelas cujo *IDUtente* correspondia ao *IDU* passado, por outras palavras, aquelas consultas que satisfaziam a condição - *consulta(D,IDU,IDServ,Custo,IDMed)*.

Se a pesquisa fosse por instituição ou cidade o processo é semelhante. Primeiramente filtramos todas os serviços que ocorreram nessa determinada instituição/cidade guardando os seus IDs numa lista. Depois para cada elemento da lista vamos procurar a consulta cujo *IDServico* corresponde a cada ID da lista.

```
% Extensao do predicado consultaUtente: IdUt,ListaServicos ->{V,F}

consultaUtente([],[]).
consultaUtente(IDU,LR) :- solucoes((X,IDU,Y,Z,P),consulta(X,IDU,Y,Z,P),LR).


% Extensao do predicado consultaInstituicao: Instituicao,ListaServicos ->{V,F}

consultaInstituicao([],[]).
consultaInstituicao(I,LR) :- solucoes(ID,servico(ID,_,I,_),R),removeRepetidos(R,L),encontrarConsultaS(L,LR).


% Extensao do predicado consultaCidade: Instituicao,ListaServicos ->{V,F}

consultaCidade([],[]).
consultaCidade(C,LR) :- solucoes(ID,servico(ID,_,_,C),R),removeRepetidos(R,L),encontrarConsultaS(L,LR).


%--- Auxiliar da 7
encontrarConsultaS([],[]).
encontrarConsultaS([H|T],R) :- identificar_consultaIDServico(H,W),encontrarConsultaS(T,S),concatenar(S,W,R).
```

Figura 18: Predicado de identificação das consultas realizados por um utente/ em uma instituição/ em uma cidade

```
25 ?- consultaInstituicao('Hospital da Senhora da Oliveira Guimarães',C).
C = [('09-02-2019', 3, 13, 30, 5),  ('10-12-2018', 11, 3, 30, 4),  ('02-03-2019', 11, 3, 30, 4),  ('28-09-2018', 3, 3, 25, 4
)] .

26 ?- consultaUtente(10,C).
C = [('16-09-2018', 10, 11, 47, 1)].

27 ?- consultaCidade('Porto',C).
C = [('18-07-2018', 2, 15, 50, 9),  ('11-04-2018', 6, 10, 22, 9),  ('25-09-2018', 14, 9, 35, 11),  ('18-03-2018', 2, 4, 47,
10)] .
```

Figura 19: Exemplo de output do predicado de identificação de consultas realizados numa cidade, numa instituição ou por um utente

4.5.8 Calcular o custo total dos cuidados de saúde por utente/serviço/ instituição/data.

Para a resolução deste problema, foi novamente aplicado o predicado *solucoes* de modo a obter a lista com os custos de todas as consultas efetuadas pelo utente. Portanto, conseguimos o conjunto de todos os custos de um utente com dado *ID* que satisfazem o predicado *consulta(D,ID,IDServ,C,IDMed)*, somando por fim todos os custos da lista através do predicado auxiliar *somaLista*.

De modo a calcular os custos incorridos por um certo serviço ou em uma qualquer data, o processo é idêntico ao anteriormente explicado. Já para calcular os custos de uma dada instituição I é necessário obter o conjunto de IDs dos serviços prestados pela mesma, isto é, o conjunto que satisfaz o predicado *servico(IDServ,D,I,C)*, sendo posteriormente utilizado o predicado auxiliar *custoListaServ* para calcular os custos dos serviços com os IDs contidos na lista obtida.

```
% Extensao do predicado custoUtente: IDUt,Custo -> {V,F}

custoUtente(ID,R) :- solucoes(C,consulta(_,ID,_,C,_),L), somaLista(L,R).


% Extensao do predicado custoServico: IDServ,Custo -> {V,F}

custoServico(ID,R) :- solucoes(C,consulta(_,_,ID,C,_),L), somaLista(L,R).


% Extensao do predicado custoData: Data,Custo -> {V,F}

custoData(Data,R) :- solucoes(C,consulta(Data,_,_,C,_),L), somaLista(L,R).


% Extensao do predicado custoInstituicao: Instituicao,Custo -> {V,F}

custoInstituicao(I,R) :- solucoes(IDServ,servico(IDServ,_,I,_),Lista), custoListaServ(Lista,R).


%--- Auxiliar da 8
% Extensao do predicado custoListaServ: ListaServicos,Custo -> {V,F}

custoListaServ([],0).
custoListaServ([H | T],Res) :- custoServico(H,S), custoListaServ(T,B), Res is S+B.
```

Figura 20: Predicado para o cálculo do custo total dos cuidados de saúde por utente, serviço, instituição e data

```
28 ?- custoUtente(15,C).
C = 72.

29 ?- custoServico(7,C).
C = 15.

30 ?- custoInstituicao('Hospital de Santa Maria',C).
C = 72.
```

Figura 21: Exemplo de output do predicado que calcula o custo total dos cuidados de saúde por utente, serviço, instituição ou data

4.5.9 Predicados Extra

⇒ Identificar as consultas efetuadas por médico

Este predicado pretende obter toda a informação sobre as consultas efetuadas por um dado médico com determinado *IDMed*. Para tal, é novamente aplicado o predicado *solucoes* que percorre todas as consultas na nossa base de conhecimento e coloca na lista resultado as que foram prestadas pelo médico em questão.

```
% Extensao do predicado consultasMedico: IDMedico,ListaConsultas -> {V,F}

consultasMedico(IDMed,R) :- solucoes((D,IDUt,IDServ,C),consulta(D,IDUt,IDServ,C,IDMed),R).
```

Figura 22: Predicado de identificação de consultas realizadas por médico

```
31 ?- consultasMedico(1,R).
R = [('16-09-2018', 10, 11, 47),  ('12-10-2018', 15, 11, 47)].
```

Figura 23: Exemplo de output para o predicado de identificação de consultas por médico

⇒ Identificar médicos por especialidade

Cada médico é caracterizado pelas suas especialidades, então dada uma especialidade E é possível encontrar todos os médicos que praticam essa mesma especialidade. Para isso recorremos ao predicado *solucoes* e através deste identificamos o conjunto dos nomes dos médicos que satisfazem os predicados *medico(ID,Nome, LE,LI)* e *pertence(E,LE)*, onde LE representa a lista de especialidades de um determinado médico.

```
% Extensao do predicado medicosEspecialidade: Especialidade,ListaMedicos -> {V,F}

medicosEspecialidade(E,L) :- solucoes(Nome,(medico(_,Nome,LE,_),pertence(E,LE)),L).
```

Figura 24: Predicado de identificação de médicos por especialidade

```
32 ?- medicosEspecialidade('Ortopedia',R).
R = ['José Moreira', 'Mariana Sousa', 'Manuel Marques'].
```

Figura 25: Exemplo de output para o predicado de identificação de médicos por especialidade

⇒ Identificar médicos por instituição

A resolução deste predicado é análoga à do predicado anterior. Dado uma instituição I, aplicamos o *solucoes* e retiramos o conjunto de médicos que satisfazem os predicados *medico(ID,Nome, LE,LI)* e *pertence(I,LI)*, onde LI representa a lista de intituições que empregam um determinado médico.

```
% Extensao do predicado medicosInstituicao: Instituicao,ListaMedicos -> {V,F}

medicosInstituicao(I,L) :- solucoes(Nome,(medico(_,Nome,_,LI),pertence(I,LI)),L).
```

Figura 26: Predicado de identificação de médicos por instituição

```
33 ?- medicosInstituicao('Hospital de Braga',R).
R = ['Mariana Sousa', 'Susana Costa', 'Guilherme Cruz'].
```

Figura 27: Exemplo de output para o predicado de identificação de médicos por instituição

⇒ Calcular o custo total dos cuidados de saúde por médico

Este predicado tem como objetivo calcular o custo total dos cuidados de saúde por algum médico com um qualquer *IDMed*. Assim, conseguimos a lista dos custos

das consultas efetuados pelo médico percorrendo-as e guardando numa lista o custo destas se o predicado *consultas(D,IDUt,IDServ,Custo,IDMed)* for satisfeito. Por fim, aplicamos o predicado auxiliar *somaLista*, responsável por somar todos os custos da lista.

```
% Extensao do predicado medicosCusto: IdMedico,Custo -> {V,F}

medicosCusto(IDMed,C) :- solucoes(Custo,consulta(_,_,_,Custo,IDMed),R), somaLista(R,C).
```

Figura 28: Predicado de cálculo do custo total dos cuidados de saúde por médico

```
34 ?- medicosCusto(4,R).
R = 85.
```

Figura 29: Exemplo de output para o predicado de identificação de médicos por especialidade

⇒ Identificar utentes por especialidade

Este predicado pretende identificar os utentes aos quais foram prestados cuidados de saúde de uma qualquer especialidade *E*. Então, conseguimos inicialmente uma lista com os *IDServ* que correspondem à especialidade em questão e depois aplicamos um dos predicados auxiliares efetuado anteriormente, *servicosUtentes*, que nos dá a informação dos utentes que dispuseram desses serviços.

```
% Extensao do predicado utentesEspecialidade: Especialidade,ListaUtentes -> {V,F}

utentesEspecialidade(E,L) :- solucoes(IdServ,servico(IdServ,E,_,_),S), servicosUtentes(S,L).
```

Figura 30: Predicado de identificação de utentes por especialidade

```
35 ?- utentesEspecialidade('Ginecologia',R).
R = [(1, 'Ana Santos', 34, 'Braga'),  (3, 'Carla Martins', 29, 'Guimarães')] .
```

Figura 31: Exemplo de output para o predicado de identificação de utentes por especialidade

5 Conclusões e Sugestões

Sendo este o projeto pioneiro, para o grupo, que requeria conhecimentos acerca da linguagem de programação lógica *PROLOG*, todo o seu desenvolvimento se tornou, de certa forma, um desafio aliciante. Partimos apenas do conhecimento adquirido com base em exercícios mais simples realizados nas aulas práticas da Unidade Curricular e, a partir desse, exploramos, cada vez mais, esta linguagem e as suas funcionalidades.

No entanto, como seria de esperar, enfrentámos algumas dificuldades no arranque deste primeiro exercício. No sentido de garantir a coerência das respostas às questões propostas, a implementação de invariantes foi um ponto fulcral para atingir esse objetivo. Foi aqui, então, que necessitamos de assegurar que, com esta nova implementação, as nossas respostas não ficariam comprometidas. Não obstante, após uma troca de ideias e de reflexão sobre a matéria lecionada, o grupo conseguiu chegar a um consenso e a uma melhor compreensão acerca da temática a abordar, ultrapassando este obstáculo.

Posto isto, conseguimos agora, uniformemente, reconhecer a importância deste exercício para um melhor entendimento sobre a nova linguagem de programação lógica introduzida. Para além da utilidade da mesma, conseguimos ainda perceber algumas das suas formas de aplicação e que problemas podem ser resolvidos com ela.

Assim, consolidamos os conhecimentos previamente adquiridos nas já referidas aulas práticas e exploramos a nossa autonomia aquando da necessidade de resolver novos problemas. Em suma, estamos satisfeitos com o resultado final deste exercício, contando que este seja um primeiro passo promissor para a realização dos exercícios seguintes.

Referências

[1] Bratko, Ivan
Prolog Programming for Artificial Intelligence,
Addison-Wesley Publishing Company, Great Britain, 1986.

Anexos

A Predicados Auxiliares

Com o objetivo de simplificar a abordagem a tomar para responder às questões propostas, foram criados alguns predicados auxiliares.

- **comprimento**: O predicado *comprimento* tem como resultado o valor do comprimento de uma determinada lista.

```
% Extensao do predicado comprimento: Lista, Resultado -> {V,F}

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

Figura 32: Predicado auxiliar comprimento

- **pertence**: O predicado *pertence* informa se um determinado elemento está, ou não, contido numa lista.

```
% Extensao do predicado pertence: Elemento,Lista -> {V,F}

pertence(X,[H | T]) :- X = H.
pertence(X,[H | T]) :- pertence(X,T),X \= H.
```

Figura 33: Predicado auxiliar pertence

- **apagaT**: O predicado *apagaT* elimina, de uma lista inicial, todas as repetições de um determinado elemento, resultando uma lista sem esse elemento.

```
% Extensao do predicado apagaT: Elemento,ListaInicial,ListaFinal -> {V,F}

apagaT(_,[],[]).
apagaT(X,[H | T], [H | L]) :- X \= H, apagaT(X,T,L).
apagaT(X,[H | T], L) :- apagaT(X,T,L).
```

Figura 34: Predicado auxiliar apagaT

- **removeRepetidos**: O predicado *removeRepetidos* remove de uma lista todos os elementos repetidos, ou seja, deste predicado resulta uma lista sem elementos repetidos.

```
% Extensao do predicado removeRepetidos: ListaInicial,ListaFinal -> {V,F}

removeRepetidos([],[]).
removeRepetidos([H | T],R) :- pertence(H,T), apagaT(H,T,S), removeRepetidos(S,B), R = [H|B].
removeRepetidos([H | T],R) :- not(pertence(H,T)),removeRepetidos(T,S), R = [H|S].
```

Figura 35: Predicado auxiliar removeRepetidos

- **somaLista**: O predicado *somaLista* soma todos os elementos de uma dada lista.

```
% Extensao do predicado somaLista: ListaNum,Resultado -> {V,F}

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

Figura 36: Predicado auxiliar somaLista

- **concatenar**: O predicado *concatenar* concatena duas listas, ou seja, deste predicado resulta uma lista que resulta da junção de duas listas fornecidas.

```
% Extensao do predicado concatenar: ListaX,ListaY,ListaFinal -> {V,F}

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

Figura 37: Predicado auxiliar concatenar

B Base de Conhecimento Inicial

```
utente(1,'Ana Santos',34,'Braga').
utente(2,'Bruno Mendonça',23,'Porto').
utente(3,'Carla Martins',29,'Guimarães').
utente(4,'Beatriz Jesus',12,'Lisboa').
utente(5,'Júlio Carvalho',36,'Braga').
utente(6,'Carlos Silva',19,'Porto').
utente(7,'Xavier Teixeira',51,'Braga').
utente(8,'Luis Almeida',32,'Braga').
utente(9,'Pedro Lima',20,'Braga').
utente(10,'André Campos',42,'Lisboa').
utente(11,'Teresa Cerqueira',54,'Guimarães').
utente(12,'Sérgio Gonçalves',16,'Coimbra').
utente(13,'João Pereira',24,'Braga').
utente(14,'Henrique Castro',83,'Porto').
utente(15,'Sara Fernandes',27,'Lisboa').


servico(1,'Oftalmologia','Hospital de Braga','Braga').
servico(2,'Ginecologia','Centro de Saúde de Maximinos','Braga').
servico(3,'Cardiologia','Hospital da Senhora da Oliveira Guimarães','Guimarães').
servico(4,'Ortopedia','Hospital S.João','Porto').
servico(5,'Oftalmologia','Centro de Saúde de Maximinos','Braga').
servico(6,'Cardiologia','Hospital de Braga','Braga').
servico(7,'Otorrinolaringologia','Hospital de Braga','Braga').
servico(8,'Oftalmologia','Centro Hospitalar e Universitário de Coimbra','Coimbra').
servico(9,'Cardiologia','Hospital S.João','Porto').
servico(10,'Oftalmologia','Hospital de Santa Maria','Porto').
servico(11,'Dermatologia','Hospital S.José' ,'Lisboa').
servico(12,'Otorrinolaringologia','Hospital S.José' ,'Lisboa').
servico(13,'Ginecologia','Hospital da Senhora da Oliveira Guimarães','Guimarães').
servico(14,'Psiquiatria','Hospital da Luz','Lisboa').
servico(15,'Dermatologia','Hospital de Santa Maria','Porto').
```

```
medico(1,'José Moreira',['Dermatologia','Ortopedia'],['Hospital S.José']).
medico(2,'Cristina Félix',['Psiquiatria','Otorrinolaringologia'],['Hospital S.José','Hospital da Luz']).
medico(3,'Helena Pereira',['Oftalmologia','Dermatologia'],['Centro Hospitalar e Universitário de Coimbra']).
medico(4,'Rodrigo Vieira',['Cardiologia'],['Hospital da Senhora da Oliveira Guimarães']).
medico(5,'Vitória Pinto',['Ginecologia'],['Hospital da Senhora da Oliveira Guimarães','Centro de Saúde de Maximinos']).
medico(6,'Mariana Sousa',['Oftalmologia','Ortopedia'],['Centro de Saúde de Maximinos','Hospital de Braga']).
medico(7,'Susana Costa',['Cardiologia'],['Hospital de Braga']).
medico(8,'Guilherme Cruz',['Otorrinolaringologia'],['Hospital de Braga']).
medico(9,'Sofia Lopes',['Dermatologia','Oftalmologia'],['Hospital de Santa Maria']).
medico(10,'Manuel Marques',['Ortopedia'],['Hospital S.João']).
medico(11,'Adriana Oliveira',['Cardiologia'],['Hospital S.João']).


consulta('10-11-2018',1,2,30,5).
consulta('18-03-2018',2,4,47,10).
consulta('09-02-2019',3,13,30,5).
consulta('28-01-2019',4,14,19,2).
consulta('16-09-2018',5,6,35,7).
consulta('11-04-2018',6,10,22,9).
consulta('23-05-2018',7,5,25,6).
consulta('29-01-2019',8,6,35,7).
consulta('11-04-2018',9,7,15,8).
consulta('16-09-2018',10,11,47,1).
consulta('10-12-2018',11,3,30,4).
consulta('03-11-2018',12,8,25,3).
consulta('15-03-2018',13,1,20,6).
consulta('25-09-2018',14,9,35,11).
consulta('12-10-2018',15,11,47,1).
consulta('18-07-2018',2,15,50,9).
consulta('02-03-2019',11,3,30,4).
consulta('28-09-2018',3,3,25,4).
consulta('12-01-2019',15,12,25,2).
```