UNIVERSIDADE DO MINHO
ESCOLA DE ENGENHARIA

Sistemas de Representação de Conhecimento e Raciocínio

1º EXERCÍCIO - GRUPO 17

Adriana Henriques Esteves Teixeira Meireles A82582
Ana Marta Santos Ribeiro A82474
Carla Isabel Novais da Cruz A80564
Jéssica Andreia Fernandes Lemos A82061

MIEI - 3º Ano - 2º Semestre

Braga, 31 de março de 2019

1 Resumo

Este relatório tem como objetivo documentar o primeiro exercício proposto na Unidade Curricular de *Sistemas Representação Raciocínio e Conhecimento* usando como auxílio a linguagem PROLOG.

Primeiramente é apresentada uma introdução do projeto. Posteriormente, é explicado o raciocínio do grupo para cada uma das alíneas do 1º exercício bem como para o conhecimento extra adicionado. Por fim, é feita uma apreciação crítica do trabalho efetuado.

Índice

2 Introdução

Este relatório surge na elaboração do primeiro exercicío prático proposto na unidade curricular de *Sistemas de Representação de Conhecimento e Raciocínio*. O objetivo pretendido é desenvolver um sistema de representação de conhecimento e raciocínio capaz de caracterizar um universo na área de prestação de cuidados de saúde. Este deverá ser realizado recorrendo à linguagem de progamação PROLOG.

O sistema desenvolvido deverá ser capaz de gerir a base de conhecimento existente, isto é, permitir adicionar e remover conhecimento, bem como processar toda a sua informação de forma a responder a todas as questões do enunciado. O conhecimento base é composto por utente, serviço e consulta, no entanto optamos por adicionar médicos, enfermeiros, turno e escalonamento uma vez que consideramos ser fundamental na área de cuidados de saúde. Para além do conhecimento, acrescentamos também alguns predicados que verificamos serem importantes nesta área.

3 Preliminares

O sistema que foi desenvolvido ao longo deste projeto pretende caracterizar o universo de discurso na área de prestação de cuidados médicos. Para que este projeto pudesse ser iniciado, foi necessária a obtenção de conhecimento base na área. Para tal, este foi adquirido através das aulas frequentadas, permitindo um maior contacto com a linguagem utilizada. Foram realizadas pesquisas neste âmbito e após a leitura detalhada do enunciado foi possível a distribuição do trabalho a realizar. Isto possibilitou uma maior organização do grupo e que o trabalho fosse concluído na data limite proposta.

4 Descrição do trabalho e análise de resultados

4.1 Base de conhecimento

Neste projeto caracterizamos o conhecimento da seguinte forma:

- **Utente:** IdUt, Nome, Idade, Cidade –> {V,F}

O IdUt será um inteiro que identificará este utilizador. A Idade será também um número. O Nome e a Cidade serão Strings.

```
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
```

- **Servico:** IdServ, Descrição, Instituição, Cidade –> {V,F}

Um serviço será identificado pelo inteiro IdServ. Terá uma descrição, a instituição onde decorre o serviço bem como a cidade, que serão Strings.

```
servico(1, 'Pediatria', 'Hospital de Braga', 'Braga').
servico(2, 'Cirurgia', 'Hospital de Braga', 'Braga').
servico(3, 'Reabilitacao', 'Hospital da Luz', 'Guimaraes').
```

Tendo em conta que os cuidados de saúde serão prestados por médicos e enfermeiros, decidimos acrescentar este conhecimento.

- **Medico**: IdMed, Nome, Idade -> {V, F}

O IdMed será o identificador do médico que será um inteiro. O Nome e a Idade serão Strings.

```
medico(1,'Maria Lemos',35).
medico(2,'Carla Sousa',40).
medico(3,'Catarina Machado',35).
```

Estes profissionais irão realizar consultas de determinados serviços.

- **Consulta:** Data, IdUt, IdServ, Custo, IdMed –> {V,F}

Uma consulta terá uma data cujo o formato é Ano-Mês-Dia. A Consulta conterá o identificador do utilizador bem como o do serviço e o custo da consulta. Estes serão todos um número. Além disso, terá uma inteiro que corresponde ao identificador do médico.

```
consulta(2019-01-02,1,1,40,1).
consulta(2019-01-02,2,1,50,1).
consulta(2019-01-03,1,2,90,2).
```

- **Enfermeiro**: IdEnf, Nome, Idade -> {V, F}

O enfermeiro será identificado pelo inteiro IdEnf. O seu Nome será uma String e a Idade será um inteiro.

```
enfermeiro(1,'Joao Antunes',32).
enfermeiro(2,'Catia Aveiro',29).
enfermeiro(3,'Alexandre Torres',47).
enfermeiro(4,'Silvia Sousa',50).
```

Uma vez que os enfermeiros prestam cuidados de saúde por turnos, decidimos inserir este conhecimento. Assim, primeiro serão elaborados os turnos de trabalho possíveis e de seguida a forma como são distribuídos.

- **Turno**: IdTurno, Horas -> {V, F}

Um turno será caracterizado pelo IdTurno um inteiro e terá uma String com as horas.

```
turno(1,'01h-07h').
turno(2,'07h-13h').
turno(3,'13h-19h').
turno(4,'19h-01h').
```

Cada enfermeiro terá um turno num determinado dia numa determinada instituição, pelo que se tornou imperativo acrescentar este conhecimento que se caracteriza da seguinte forma:

- **Escalonamento**: Data, IdEnf, IdTurno, Instituicao -> {V, F}

A data terá o formato Ano-Mês-Dia. A instituição será uma String enquanto que o identificador do enfermeiro e do turno serão inteiros.

```
escalonamento(2019-01-02,1,1,'Hospital de Braga').
escalonamento(2019-01-03,2,2,'Hospital de Braga').
escalonamento(2019-01-03,3,2,'CUF Coimbra').
escalonamento(2019-01-04,4,3,'Santa Maria').
```

4.2 Invariantes

Os invariantes são responsáveis por garantir que o conhecimento é consistente, pelo que são indispensáveis. Assim, desenvolvemos dois tipos de invariantes os estruturais e os referenciais.

Para se definir estes invariantes recorreu-se ao predicado listaSolucoes que devolve todas as soluções encontradas que satisfazem uma determinada condição. Este predicado é essencial para o desenvolvimento de todos os que irão ser criados ao longo deste exercício.

```
% Predicado listaSolucoes:
% Termo, Predicado, Lista -> {V,F}
listaSolucoes(T,G,B) :-
    findall(T,G,B).
```

De seguida seguem-se os inúmeros invariantes estruturais que impedem a repetição de conhecimento (utente, serviço, médico, enfermeiro ou turno), ou seja, após a inserção verifica se o identificador do predicado dado é único e, caso contrário, remove o mesmo da base de conhecimento, como veremos mais à frente.

```
+utente(IdUt,_,_,_)::(listaSolucoes(IdUt,utente(IdUt,_,_,_),L),
                                    comprimento(L,R),
                                    R==1).

+servico(Id,_,_,_):: (listaSolucoes(Id,servico(Id,_,_,_),L),
                                    comprimento(L,R),
                                    R == 1).

+medico(IdMed,_,_):: (listaSolucoes(IdMed,medico(IdMed,_,_),L),
                                    comprimento(L,R),
                                    R == 1).
```

```
12
13 +enfermeiro(IdEnf,_,_)::(listaSolucoes(IdEnf,enfermeiro(IdEnf,_,_),L),
14                                        comprimento(L,R),
15                                        R == 1).
16
17 %invariantes para n o ter id igual
18 +turno(IdT,_)::(listaSolucoes(IdT,turno(IdT,_),L),
19                                        comprimento(L,R),
20                                        R==1).
21
22 %invariante para n o ter data igual
23  +turno(_,H)::(listaSolucoes(H,turno(_,H),L),
24                                        comprimento(L,R),
25                                        R==1).
26
27 +escalonamento(D,IdEnf,IdT,I)::
28 (listaSolucoes(IdT,escalonamento(D,IdEnf,IdT,I),L),
                                       comprimento(L,R),
29                                        R == 1).
30
31 +servico(_,D,I,C):: (listaSolucoes((D,I),servico(_,D,I,C),L),
32                                        comprimento(L,R),
33                                        R==1).
```

Para além dos 8 invariantes referidos em cima, foi necessário adicionar mais 2 para garantir consistência lógica e deste modo permitir uma correta inserção de conhecimento. Por exemplo, não deve ser possível inserir consultas com identificadores de serviço e utente inexistentes. Além disso, deve também impedir que sejam adicionados escalonamentos com enfermeiros, turnos e instituições inexistentes.

```
1 +consulta(_,IdU,IdS,_,IdMed)::  (utente(IdU,_,_,_),
2                            servico(IdS,_,_,_),
3                            medico(IdMed,_,_)).
4
5
6 +escalonamento(_,IdEnf,IdT,I):: (enfermeiro(IdEnf,_,_),turno(IdT,_),
    servico(_,_,I,_)).
```

Os invariantes estruturais definidos para a remoção, têm como objetivo não permitir a eliminação de conhecimento que não exista. Desta forma, tornou-se imperativo

definir um para cada, como pode ser observado de seguida:

```
-utente(Id,N,I,C):: (listaSolucoes(Id,utente(Id,N,I,C),L),
                     comprimento(L,R),
                     R == 1).

-servico(Id,S,I,C):: (listaSolucoes(Id,servico(Id,S,I,C),L),
                     comprimento(L,R),
                     R == 1).

-consulta(D,IdU,IdS,C,IdMed):: (listaSolucoes((D,IdU,IdS),
                     consulta(D,IdU,IdS,C,IdMed),L),
                     comprimento(L,R),
                     R == 1).

-medico(IdMed,N,I):: (listaSolucoes(IdMed,
                     medico(IdMed,N,I),L),
                     comprimento(L,R),
                     R == 1).

-enfermeiro(IdEnf,N,I):: (listaSolucoes(IdEnf,
                     enfermeiro(IdEnf,N,I),L),
                     comprimento(L,R),
                     R == 1).

-turno(IdTurno,H):: (listaSolucoes(IdTurno,turno(IdTurno,H),L),
                     comprimento(L,R),
                     R == 1).

-escalonamento(D,IdEnf,IdT,I):: (listaSolucoes(IdT,
                     escalonamento(D,IdEnf,IdT,I),L),
                     comprimento(L,R),
                     R == 1).
```

Os invariantes referenciais devem manter a consistência lógica. Por exemplo, ao nível da remoção devem assegurar que o conhecimento que será eliminado não está a ser utilizado. Assim, antes de eliminar um utente temos de verificar se este não tem consultas marcadas e para remover um serviço também é necessário que este não contenha consultas. Além disso, não se pode eliminar um enfermeiro com um escalonamento associado, um turno utilizado num escalonamento ou um médico com consultas.

Deste modo, surgem os seguintes invariantes:

```
-utente(Id,_,_,_):: (listaSolucoes(Id,consulta(_,Id,_,_),L),
                     comprimento(L,R),
                     R == 0).

-servico(Id,_,_,_):: (listaSolucoes(Id,consulta(_,Id,_,_),L),
                      comprimento(L,R),
                      R == 0).

-medico(IdMed,_,_):: (listaSolucoes(IdMed,consulta(_,_,_,_,IdMed),L),
                      comprimento(L,R),
                      R == 0).

-enfermeiro(IdEnf,_,_) :: ( listaSolucoes(IdEnf,
                     escalonamento(_,IdEnf,_,_), L ),
                     comprimento(L,R),
                     R == 0 ).

-turno(IdTurno,_):: (listaSolucoes(IdTurno,
                     escalonamento(_,_,IdTurno,_),L),
                     comprimento(L,R),
                     R == 0).
```

4.3 Predicados

4.3.1 Registar utentes/serviços/consultas/médicos/enfermeiros/turnos/escalonamentos

De modo a adicionar conhecimento é necessário averiguar se a inserção é válida. Para isso, aplicou-se o predicado listaSolucoes, que é responsável por listar todos os invariantes associados. Depois, é adicionado o termo pretendido e são testados os invariantes através do testaInv, que se encontra representado em baixo.

```
testaInv([]).
testaInv([H|T]) :-
    H, testaInv(T).
```

```
insere(T) :- assert(T).
insere(T) :- retract(T), !,fail.
```

```
3
4 adiciona(Termo) :-
5         listaSolucoes(Inv,+Termo::Inv,S),
6         insere(Termo),
7         testaInv(S).
```

Como foi apresentado na secção anterior, foi criado um invariante que impede que utentes com identificadores iguais sejam inseridos. Assim, segue um exemplo em que se vai tentar inserir um novo utente com um id já existente (1) e não será permitido.

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo', 9, 'Porto').
utente(8, 'Joel Fernandes', 34, 'Porto').
utente(9, 'Jessica Pereira', 41, 'Guimaraes').
utente(10, 'Adriana Teixeira', 85, 'Braga').
utente(11, 'Roberto Leal', 7, 'Guimaraes').
utente(12, 'Anibal Esteves', 24, 'Coimbra').
utente(13, 'Joana Peixoto', 17, 'Porto').
utente(14, 'Raquel Gomes', 50, 'Braga').
utente(15, 'Hugo Silva', 71, 'Coimbra').

yes
| ?-
```

```
| ?- adiciona(utente(1,'Paulo Silva',18,'Braga')).
no
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
```

```
utente(7, 'Rodrigo', 9, 'Porto').
utente(8, 'Joel Fernandes', 34, 'Porto').
utente(9, 'Jessica Pereira', 41, 'Guimaraes').
utente(10, 'Adriana Teixeira', 85, 'Braga').
utente(11, 'Roberto Leal', 7, 'Guimaraes').
utente(12, 'Anibal Esteves', 24, 'Coimbra').
utente(13, 'Joana Peixoto', 17, 'Porto').
utente(14, 'Raquel Gomes', 50, 'Braga').
utente(15, 'Hugo Silva', 71, 'Coimbra').

yes
| ?-
```

Posteriormente, é mostrado um exemplo que insere um utente com um id que ainda não existe.

```
| ?- adiciona(utente(16,'Carla Isabel',20,'Algarve')).
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo', 9, 'Porto').
utente(8, 'Joel Fernandes', 34, 'Porto').
utente(9, 'Jessica Pereira', 41, 'Guimaraes').
utente(10, 'Adriana Teixeira', 85, 'Braga').
utente(11, 'Roberto Leal', 7, 'Guimaraes').
utente(12, 'Anibal Esteves', 24, 'Coimbra').
utente(13, 'Joana Peixoto', 17, 'Porto').
utente(14, 'Raquel Gomes', 50, 'Braga').
utente(15, 'Hugo Silva', 71, 'Coimbra').
utente(16, 'Carla Isabel', 20, 'Algarve').

yes
| ?-
```

O invariante dá uso ao predicado listaSolucoes e testa se após a inserção do conhecimento pretendido, existe apenas um utente na base de conhecimento com o identifi-

cador que foi inserido. Caso não seja satisfeito, este conhecimento é removido.

4.3.2 Remoção utentes/serviços/consultas/médicos/enfermeiros/turnos/escalonamentos

De forma a eliminar conhecimento é necessário verificar se a remoção é válida. Para tal, recorre-se ao predicado listaSolucoes, que lista todos os invariantes associados. De seguida, através do testaInv averiguamos se estes são respeitados. Na eventualidade de não serem a remoção falha.

```
remove(T) :- retract(T).
remove(T) :- assert(T),!,fail.

elimina(Termo) :- listaSolucoes(Inv, -Termo :: Inv, S),
                  testaInv(S),
                  remove(Termo).
```

De seguida será ilustrado o exemplo em que o utente não pode ser removido, dado que possui consultas marcadas como nos indica o predicado consultas_utente.

```
| ?- consultas_utente(1,R).
S =
    [(2019-01-02,1,1,40,1),(2019-01-03,1,2,90,2),(2019-01-06,1,4,40,5)]
    ?
yes
| ?- elimina(utente(1, 'Joana Antunes', 18, 'Braga')).
no
| ?
```

4.3.3 Identificar as instituições prestadoras de serviços

De modo a facilitar a elaboração deste predicado e de outros que possam surgir, foi necessário implementar um predicado auxiliar que permitisse retirar os elementos repetidos de uma lista. Assim, surge o semRepetidos:

```
% Predicado semRepetidos:
% ListaA, ListaB -> {V,F}
semRepetidos([],[]).
semRepetidos([H|T], R) :-
    pertence(H,T),
```

```
    semRepetidos(T,R).
semRepetidos([H|T], [H|R]) :-
    nao(pertence(H,T)),
    semRepetidos(T,R).
```

Para desenvolver o prestadoras_servicos, recorre-se à listaSolucoes que permitirá listar as instituições e as respetivas cidades onde são prestados serviços ao longo do país.

```
% Predicado prestadoras_servicos:
% Resultado -> {V,F}
prestadoras_servicos(R1) :-
    listaSolucoes((Inst,Cidade), servico(_, _, Inst, Cidade), R),
    semRepetidos(R, R1).
```

Assim, de seguida, são apresentadas todas as instituções que a base de conhecimento contém até ao momento.

```
| ?- prestadoras_servicos(R).
R = [('Hospital de Braga','Braga'),('Hospital da
   Luz','Guimaraes'),('Santa Maria','Porto'),('CUF
   Coimbra','Coimbra'),('Sao Joao','Porto')] ?
yes
| ?-
```

4.3.4 Identificar utentes/serviços/consultas/médicos/enfermeiros/turnos/escalonamentos por critérios de seleção

Para a realização desta funcionalidade, tornou-se imperativo a construção de regras que manipulam dois conhecimentos, o critério de seleção bem como a lista de utentes, serviços, consultas, médicos, enfermeiros, turnos e escalonamento dependendo do pretendido.

- **Utentes**

No caso do utente, foram escolhidos 4 critérios: identificador, nome, idade e cidade. Dado que os predicados elaborados para cada um dos critérios são bastante semelhantes, é apresentado de seguida um exemplo no qual identificamos um utente pelo nome. Assim, os conhecimentos são o nome do utilizador e uma lista de utentes.

```
% Predicado utente_nome:
% Nome, Lista de Utentes -> {V,F}
utente_nome(Nome,U) :-
    listaSolucoes(utente(IdUt, Nome, Idade, Cidade),utente(IdUt,
        Nome, Idade, Cidade),U).
```

Como podemos observar o sistema identifica todos os utilizadores com o nome Joana Antunes.

```
| ?- utente_nome('Joana Antunes',U).
U = [utente(1,'Joana Antunes',18,'Braga')] ?
yes
| ?
```

- **Serviços**

Para o serviço os critérios também são 4: identificador, descrição, instituição e cidade. No exemplo que se segue temos como conhecimento uma descrição do serviço e uma lista de serviços.

```
% Predicado servico_instituicao:
% Instituicao, Lista de Servico -> {V,F}
servico_instituicao(Instituicao,S) :-
    listaSolucoes(servico(IdServ, Descricao, Instituicao,
        Cidade),servico(IdServ, Descricao, Instituicao, Cidade),S).
```

O sistema consegue identificar todos os serviço existentes numa determinada instituição, neste caso na CUF Coimbra.

```
| ?- servico_instituicao('CUF Coimbra',S).
S = [servico(6,'Cirurgia','CUF Coimbra','Coimbra')] ?
yes
| ?-
```

- **Consultas**

No caso das consultas, os critérios são a data, o identificador do utente, do médico e do serviço bem como o custo. Para obter, por exemplo, as consultas realizadas numa determinada data, temos como conhecimento uma data e uma lista de consultas.

```
% Predicado consulta_data:
% Data, Lista de Consultas -> {V,F}
consulta_data(Data,C) :-
    listaSolucoes(
    consulta(Data, IdUt, IdServ, Custo,IdMed),
    consulta(Data, IdUt, IdServ, Custo,IdMed),
    C).
```

Assim, podemos verificar que o sistema devolve a lista de todas as consultas cuja data é 2019-01-02.

```
| ?- consulta_data(2019-01-02,C).
C = [consulta(2019-01-02,1,1,40,1),consulta(2019-01-02,2,1,50,1)]?
yes
| ?-
```

- **Médicos**

No caso do médico os critérios escolhidos foram o identificador, nome bem como a idade. No exemplo apresentado de seguida temos como conhecimento o nome do médico e uma lista de médicos.

```
% Predicado medico_nome:
% Nome Medico, Lista de Medicos -> {V,F}
medico_nome(Nome,M) :-
    listaSolucoes(medico(IdMed, Nome, Idade),
    medico(IdMed, Nome, Idade),M).
```

Como esperado, o sistema indica a lista de médicos com o nome Maria Lemos.

```
| ?- medico_nome('Maria Lemos',M).
M = [medico(1,'Maria Lemos',35)] ?
yes
| ?-
```

- **Enfermeiros**

À semelhança dos médicos, escolhemos como critérios o identificador, o nome e a idade. Tal como apresentado anteriormente elaboramos regras que manipulam o critério selecionado bem como uma lista de enfermeiros. Assim, estes predicados irão

diferir dos anteriores na medida em que os critérios não são os mesmo e o predicado a aceder será o enfermeiro. Assim, de seguida poderemos observar um exemplo em que o sistema irá listar os enfermeiros com uma determinada idade.

```
% Predicado enfermeiro_idade:
% Idade, Lista de Enfermeiro -> {V,F}
enfermeiro_idade(Idade,M) :-
    listaSolucoes(enfermeiro(IdEnf, Nome, Idade),
    enfermeiro(IdEnf, Nome, Idade),M).
```

Desta forma, conseguimos obter os enfermeiros com 50 anos.

```
| ?- enfermeiro_idade(50,E).
E = [enfermeiro(4,'Silvia Sousa',50)] ?
yes
| ?-
```

- **Turnos**

Neste caso elaboramos predicados tendo como critérios o identificador e a hora do turno. Desta forma, apresentamos um desses como forma de exemplo.

```
% Predicado turno_idTurno:
% Id turno, Lista de Turnos -> {V,F}
turno_idTurno(IdTurno,T) :-
    listaSolucoes(turno(IdTurno, Horas),turno(IdTurno, Horas),T).
```

Fornecendo o identificador de um turno o sistema irá indicar todas as informações deste, como podemos observar.

```
| ?- turno_idTurno(1,T).
T = [turno(1,'01h-07h')] ?
yes
```

- **Escalonamentos**

Os critérios escolhidos para o escalonamento são: a data, o identificador do enfermeiro, o identificador do turno e a instituição. Para estes predicados teremos como conhecimento um critério e uma lista de escalonamentos. Como estes são bastante semelhantes apresentamos um exemplo.

```
% Predicado escalonamento_instituicao:
% Instituicao, Lista de Escalonamentos -> {V,F}
escalonamento_instituicao(Instituicao,T) :-
    listaSolucoes(escalonamento(Data, IdEnf, IdTurno,
       Instituicao),escalonamento(Data, IdEnf, IdTurno,
       Instituicao),T).
```

Neste caso, o sistema consegue indicar os escalonamentos realizados no Hospital de Braga.

```
| ?- escalonamento_instituicao('Hospital de Braga',E).
E = [escalonamento(2019-1-2,1,1,'Hospital de
   Braga'),escalonamento(2019-1-3,2,2,'Hospital de Braga')] ?
yes
| ?-
```

4.3.5 Identificar serviços prestados por instituição/cidade/datas/custo/utente

Foi utilizado o predicado listaSolucoes, para adquirir a lista pretendida com o conjunto de parâmetros que satisfazem o predicado servico (IdServico, Descricao, Instituicao, Cidade) para uma dada instituicao, ou uma dada cidade, ou numa dada data ou com um determinado custo.

Para este ponto foi utilizado o predicado auxiliar semRepetidos, apresentado anteriormente.

- **Instituição**

```
% Extensao do predicado servico_Instituicao: Instituicao, Resultado
   -> {V,F}
servico_Instituicao(Instituicao,S) :-
    listaSolucoes(D,servico(IdServ,D,Instituicao,C),S).
```

No exemplo que se segue são devolvidos todos os serviços efetuados no Hospital de Braga.

```
| ?- servico_Instituicao('Hospital de Braga',R).
R = ['Pediatria','Cirurgia'] ?
yes
| ?-
```

- **Cidade**

```
% Extensao do predicado servico_Cidade: Cidade, Resultado -> {V,F}
servico_Cidade(Cidade,S) :-
    listaSolucoes(D,servico(IdServ,D,I,Cidade),S).
```

No exemplo que se segue são devolvidos todos os serviços efetuados na cidade do Porto.

```
| ?- servico_Cidade('Porto',R).
R = ['Oftalmologia','Ortopedia',Neurologia'] ?
yes
| ?-
```

- **Datas**

```
% Extensao do predicado servico_Data: Data, Resultado -> {V,F}
servico_Data(Data,S) :-
        listaSolucoes(D,(consulta(Data,_,IdServ,_,_),
        servico(IdServ,D,I,C)),L),
         semRepetidos(L,S).
```

Neste caso, são retornados todos os serviços que foram realizados na data 2019-01-02.

```
| ?- servico_Data(2019-01-02,R).
R = ['Pediatria'] ?
yes
| ?-
```

- **Custo**

```
% Extensao do predicado servico_Custo: Custo, Resultado -> {V,F}
servico_Custo(Custo,S) :-
        listaSolucoes(D,(consulta(_,_,IdServ,Custo,_),
        servico(IdServ,D,I,C)),L),
        semRepetidos(L,S).
```

Por último, podemos observar os todos os serviços que tiveram um custo de 40 euros.

```
| ?- servico_Custo(40,R).
R = ['Pediatria','Oftalmologia'] ?
yes
| ?-
```

- **Medico**

```
%Predicado servico_medico:
% Id medico, Lista de Servicos -> {V,F}
servico_Medico(IdM,S) :-
        listaSolucoes(D,(medico(IdM,_,_),
        consulta(_,_,IdServ,_,IdM),
        servico(IdServ,D,_,_)),L),
        semRepetidos(L,S).
```

Este exemplo, permite listar os serviços efetuados pelo médico cujo Id é 1.

```
| ?- servico_Medico(1,R).
R = ['Pediatria'] ?
yes
| ?-
```

- **Utente**

```
%Predicado servico_utente:
%IdUtente, Lista de Servicos ->{V,F}
servico_Utente(IdU,S) :-
        listaSolucoes(D,(utente(IdU,_,_,_),
        consulta(_,IdU,IdServ,_,_),
        servico(IdServ,D,_,_)),L),
        semRepetidos(L,S).
```

O exemplo devolve os serviços realizados pelo utente cujo identificador é 1.

```
| ?- servico_Utente(1,R).
R = ['Pediatria','Cirurgia','Oftalmologia'] ?
yes
| ?-
```

4.3.6 Identificar os utentes de um serviço/instituição/médico

De modo a facilitar a elaboração destes predicados, foi necessário utilizar o predicado semRepetidos como apresentado previamente.

- **Instituição**

Este predicado devolve todos os utentes que frequentaram uma dada instituição.

```
utente_instituicao(Instituicao,S):-
        listaSolucoes((IdU,N,I,C),
        (consulta(_,IdU,IdS,_,_),
        servico(IdS,_,Instituicao,_),
        utente(IdU,N,I,C)),L),
        semRepetidos(L,S).
```

O sistema irá indicar-nos todos os utentes com consultas no Hospital de Braga, tal como esperado.

```
| ?- utente_instituicao('Hospital de Braga', R).
R = [(2, 'Manuel Sousa', 37, 'Guimaraes'),(1, 'Joana Antunes', 18,
    'Braga')] ?
yes
| ?-
```

- **Serviço**

Este predicado devolve todos os utentes com consultas de um determinado serviço.

```
utente_servico(IdS,S) :-
        listaSolucoes((IdU,N,I,C),
        (consulta(_,IdU,IdS,_,_),utente(IdU,N,I,C)),L),
        semRepetidos(L,S).
```

No exemplo seguinte, são os pacientes que recorrem ao serviço com identificador 1.

```
| ?- utente_servico(1, R).
R = [(1, 'Joana Antunes', 18, 'Braga'),(2, 'Manuel Sousa', 37,
    'Guimaraes')] ?
yes
| ?-
```

- **Médico**

Este predicado, é bastante útil, uma vez que permite obter todos os utentes de um determinado médico.

```
utente_medico(Id,S) :-
        listaSolucoes((IdU,N,I,C),
        (consulta(_,IdU,_,_,Id),utente(IdU,N,I,C)),L),
        semRepetidos(L,S)..
```

O exemplo que se segue devolve os utentes do médico cujo identificador é 1.

```
| ?- utente_medico(3,R).
R = [(3,'Ricardo Pimenta',25,'Braga'),(4,'Carla
    Ribeiro',67,'Coimbra'),(5,'Paula Costa',21,'Porto')] ?
yes
| ?-
```

4.3.7 Identificar consultas realizadas por utente/instituição/cidade/médico/data/custo

- **Utente**

De modo a identificar todas as consultas realizadas por um utente com identificador IdUt, foi utilizado o predicado listaSolucoes que irá devolver uma lista com todas as consultas relativas ao utente apresentado.

```
% Predicado consultas_utente:
% Id Utente, Resultado -> {V,F}
consultas_utente(IdUt, R) :-
    listaSolucoes((Data, IdUt, IdServ, Custo, IdMed),
    (utente(IdUt, _, _, _), consulta(Data, IdUt, IdServ, Custo,
       IdMed)), R).
```

Assim, no exemplo que se segue, o sistema apresenta todos as consultas realizadas pelo utente com identificador 1.

```
| ?- consultas_utente(1, R).
R = [(2019-1-2,1,1,40,1),(2019-1-3,1,2,90,2),(2019-1-6,1,4,40,5)] ?
yes
| ?-
```

- **Instituição**

```
% Predicado consultas_instituicao:
% Instituicao, Resultado -> {V,F}
consultas_instituicao(Instituicao, R) :-
    listaSolucoes((Data, IdUt, IdServ, Custo, IdMed), (consulta(Data,
        IdUt, IdServ, Custo, IdMed), servico(IdServ, _, Instituicao,
        _)), R).
```

À semelhança do anterior, este predicado devolve todas as consultas realizadas por dada instituição. No exemplo apresentado, são descritas as consultas realizadas no Hospital de Braga.

```
| ?- consultas_instituicao('Hospital de Braga', R).
R = [(2019-1-2,1,1,40,1),(2019-1-2,2,1,50,1),(2019-1-3,1,2,90,2)] ?
yes
| ?-
```

- **Cidade**

```
% Predicado consultas_cidade:
% Cidade, Resultado -> {V,F}
consultas_cidade(Cidade, R) :-
    listaSolucoes((Data, IdUt, IdServ, Custo, IdMed), (consulta(Data,
        IdUt, IdServ, Custo, IdMed), servico(IdServ, _, _, Cidade)),
        R).
```

Neste caso, são apresentadas as consultas realizadas na cidade de Braga, com as informações relativas às mesmas.

```
| ?- consultas_cidade('Guimaraes', R).
R = [(2019-1-3,3,3,70,3),(2019-1-3,4,3,70,3),
(2019-1-4,5,3,80,3),(2019-1-4,6,3,50,4)] ?
yes
| ?-
```

- **Médico**

```
% Predicado consultas_medico:
% Id Medico, Resultado -> {V,F}
consultas_medico(IdMed, R) :-
    listaSolucoes((Data, IdUt, IdServ, Custo, IdMed), (medico(IdMed,
       _, _), consulta(Data, IdUt, IdServ, Custo, IdMed)), R).
```

Neste predicado são apresentadas as consultas realizadas por um médico com certo identificador. Podemos verificar no seguinte exemplo as consultas prestadas pelo médico com identificador 3.

```
| ?- consultas_medico(3, R).
R = [(2019-1-3,3,3,70,3),(2019-1-3,4,3,70,3),(2019-1-4,5,3,80,3)] ?
yes
| ?-
```

- **Data**

```
% Predicado consulta_data:
% Data, Resultado -> {V,F}
consultas_data(Data,C) :-
        listaSolucoes((Data, IdUt, IdServ, Custo,
           IdMed),consulta(Data,IdUt,IdServ,Custo, IdMed),C).
```

São apresentadas as consultas realizadas numa certa data com o formato Ano-Mes-Dia. Como podemos observar são devolvidas as informações das consultas relativas ao dia 2019-01-04.

```
| ?- consultas_data(2019-1-4,U).
U = [(2019-1-4,5,3,80,3),(2019-1-4,6,3,50,4)] ?
yes
| ?-
```

- **Custo**

```
% Predicado consulta_custo:
% Custo, Resultado -> {V,F}
consultas_custo(Custo,C) :-
        listaSolucoes((Data, IdUt, IdServ, Custo,
           IdMed),consulta(Data,IdUt,IdServ,Custo, IdMed),C).
```

Com base no custo introduzido como conhecimento é possível obter todas as consultas relativas a esse preço. No exemplo apresentado, o valor utilizado é de 50.

```
| ?- consultas_custo(50,U).
U = [(2019-1-2,2,1,50,1),(2019-1-4,6,3,50,4)] ?
yes
| ?-
```

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

Para auxiliar na implementação da última funcionalidade requerida, tornou-se necessário a criação do predicado auxiliar soma.

```
% Predicado soma:
% Lista, valor -> {V,F}
soma([],0).
soma([H|L],R) :-
    soma(L,X), R is X+H.
```

Este será utilizado em todos os predicados a construir nesta funcionalidade. Verificámos que os predicados para o cálculo do custo total dos cuidados de saúde por utente, serviço e data serão semelhantes uma vez que o predicado consulta contém todas as informações necessárias, ou seja, o identificador do utente e do serviço bem como a data. Sendo assim, apresentamos de seguida um exemplo para cada.

- **Utente**

```
% Predicado custo_utente:
% Id utente, Custo total -> {V,F}
custo_utente(IdUt, CustoTotal) :-
    listaSolucoes(Custo, consulta(_, IdUt, _, Custo,_), C),
    soma(C, CustoTotal).
```

O sistema indica que o custo total dos cuidados de saúde do utente cujo identificador é 1, como era expectável.

```
| ?- custo_utente(1,C).
C = 170 ?
```

```
yes
| ?-
```

- **Serviço**

```
% Predicado custo_servico:
% Id servico, Custo total -> {V,F}
custo_servico(IdServ, CustoTotal) :-
    listaSolucoes(Custo, consulta(_, _, IdServ, Custo,_), C),
    soma(C, CustoTotal).
```

Assim, o sistema informa qual o custo total em cuidados de saúde num determinado serviço.

```
| ?- custo_servico(2,C).
C = 90 ?
yes
| ?-
```

- **Instituição**

Contudo, para o cálculo dos custos por instituição já será necessário aceder ao predicado serviço.

```
custo_instituicao(Instituicao, CustoTotal) :-
    listaSolucoes(
    Custo,
    (consulta(_, _, IdServ, Custo,_),
       servico(IdServ,_,Instituicao,_)),
    C
    ),
    soma(C, CustoTotal).
```

Como podemos verificar, o sistema indica o custo total dos cuidados de saúde numa dada instituição.

```
| ?- custo_instituicao('CUF Coimbra',C).
C = 190 ?
yes
| ?-
```

- **Data**

```
% Predicado custo_data:
% Data, Custo total -> {V,F}
custo_data(Data, CustoTotal) :-
    listaSolucoes(Custo, consulta(Data, _, _, Custo,_), C),
    soma(C, CustoTotal).
```

Como podemos verificar o sistema indica o custo total em cuidados numa determinada data.

```
| ?- custo_data(2019-1-4,C).
C = 130 ?
yes
| ?-
```

5 Funcionalidades Extra

5.1 Predicados

Para além dos predicados e do conhecimento extra acrescentados na secção anterior, decidimos adicionar novos predicados que complementam a nossa base de conhecimento com novas funcionalidades.

5.1.1 Identificar médicos por instituição

Este predicado permite obter todos os médicos de uma determinada instituição.

```
% Predicado Medico_ instituicao:
% Instituicao, Resultado -> {V,F}
medico_instituicao(Instituicao,S) :-
        listaSolucoes((IdM,N,I),(servico(IdServ,_,Instituicao,_),
        consulta(_,_,IdServ,_,IdM),medico(IdM,N,I)),L),
        semRepetidos(L,S).
```

Assim, no exemplo apresentado de seguida é possível observar todos os médicos que dão consultas no Hospital Santa Maria.

```
| ?- medico_instituicao('Santa Maria',R).
```

```
R = [(5,'Ana Sousa',56),(6,'Josefina Costa',34),(8,'Andre
    Barbosa',32)] ?
yes
| ?-
```

5.1.2 Identificar serviços entre datas

O predicado que se pode observar em baixo, devolve todos os serviços que ocorreram entre duas datas. Para tal, foi necessário criar dois predicados auxiliares, a data_Maior e a testa_Data_Consulta. O primeiro, dadas duas datas devolve a maior delas. O segundo, verifica se uma determinada consulta está entre duas datas.

```
%Determinar a maior entre duas datas
data_Maior(Y1-M1-D1,Y2-M2-D2,Y2-M2-D2) :-
        Y2 > Y1;
        (Y2 == Y1, M2 > M1);
        (Y2 == Y1, M2 == M1, D2 >= D1).

data_Maior(Y1-M1-D1,Y2-M2-D2,Y1-M1-D1) :-
        Y2 < Y1;
        (Y2 == Y1, M2 < M1);
        (Y2 == Y1, M2 == M1, D2 < D1).

%Verifica se a data de uma determinada consulta esta entre duas datas
testa_Data_Consulta(D1,D2,C) :-
    data_Maior(D1,C,C),data_Maior(D2,C,D2).
```

```
%Extensao do predicado servico_entre_Datas:
%Data1, Data2, Resultado ->{V,F}
servico_entre_Datas(Data1,Data2,S) :-
        listaSolucoes((IdServ,D,I,C),(servico(IdServ,D,I,C),
        consulta(DC,_,IdServ,_,_),
        testa_Data_Consulta(Data1,Data2,DC)),L),
        semRepetidos(L,S)
```

Com o exemplo que se segue, são obtidos os serviços que se realizaram entre 1 e 2 de Janeiro de 2019.

```
| ?- servico_entre_Datas(2019-01-02,2019-01-03,R).
R = [(1,'Pediatria','Hospital de
```

```
    Braga','Braga'),(2,'Cirurgia','Hospital de
    Braga','Braga'),(3,'Reabilitacao','Hospital da Luz','Guimaraes')] ?
yes
| ?-
```

5.1.3 Idade média dos utentes por instituição/serviço

Para implementar estes predicados, tornou-se fundamental definir um predicado auxiliar, capaz de calcular a média dos elementos de uma lista. Deste modo, elaboramos a media:

```
media([],0).
media(L,R) :-
    soma(L,X),
    comprimento(L,S),
    R is (div(X,S)).
```

- **Instituição**

```
% Predicado instituicao_idade_media:
% Instituicao, Idade -> {V,F}
instituicao_idade_media(Instituicao,M) :-
    listaSolucoes(Idade,(consulta(_,IdUt,IdServ,_,_),
    servico(IdServ,_,Instituicao,_), utente(IdUt,_,Idade,_)),S),
    semRepetidos(S,R),
    media(R,M).
```

Neste exemplo, o sistema irá devolver a idade média dos utentes que têm consultas no Hospital de Braga.

```
| ?- instituicao_idade_media('Hospital de Braga',R).
R = 27 ?
yes
| ?-
```

- **Serviço**

```
servico_idade_media(IdServ,M) :-
    listaSolucoes(Idade,(consulta(_,IdUt,IdServ,_,_),
    utente(IdUt,_,Idade,_)),S),
    semRepetidos(S,R),
    media(R,M).
```

Permite calcular a idade média de todos os utentes que possuem uma consulta de um determinado serviço. No caso ilustrado abaixo é para o serviço cujo identificador é 1.

```
| ?- servico_idade_media(1,R).
R = 27 ?
yes
| ?-
```

5.1.4 Consultas de um determinado médico numa determinada data

Com a definição deste predicado é possível obter as consultas que um determinado médico possui no dia fornecido. Assim, conseguimos obter a descrição do serviço que este médico irá prestar no dia correspondente.

```
consultas_data_nome(Data, Nome, R1) :-
        listaSolucoes((Data, IdUt, IdServ, Custo, IdMed),
        (medico(_, Nome, _),
        consulta(Data, IdUt, IdServ, Custo, IdMed)), R),
        semRepetidos(R, R1).
```

Como é verificado no exemplo em baixo, a médica Josefina Costa, no dia 3 de janeiro de 2019 possui uma consulta de cirurgia.

```
| ?- consultas_data_nome(2019-01-03, 'Josefina Costa', R).
R = [(2019-1-3,1,2,90,2),(2019-1-3,3,3,70,3),(2019-1-3,4,3,70,3)] ?
yes
| ?-
```

5.1.5 Enfermeiros por data/instituição

- **Data**

Este predicado permite obter os enfermeiros que trabalham numa dada data. Caso os enfermeiros possuam escalonamento no dia indicado, serão apresentados os seus dados.

```
% Predicado enfermeiros_data:
% Data, Resultado -> {V,F}
enfermeiros_data(Data, R) :-
        listaSolucoes((IdEnf, Nome, Idade),
        (enfermeiro(IdEnf, Nome, Idade),
        escalonamento(Data, IdEnf, _, _)), R).
```

Neste exemplo, podemos observar que no dia 4 de janeiro de 2019, a enfermeira Silvia Sousa encontra-se de serviço.

```
| ?- enfermeiros_data(2019-01-04, R).
R = [(4,'Silvia Sousa',50)] ?
yes
| ?-
```

- **Instituição**

Este predicado permite-nos conhecer quais os enfermeiros que exercem as suas funções numa dada instituição.

```
% Predicado enfermeiros_instituicao:
% Instituicao, Resultado -> {V,F}
enfermeiros_instituicao(Instituicao, R) :-
        listaSolucoes((IdEnf, Nome, Idade),
        (enfermeiro(IdEnf, Nome, Idade),
        escalonamento(_, IdEnf, _, Instituicao)), R).
```

Assim, devido à existência deste predicado, sabemos que os enfermeiros Joao Antunes e Catia Aveiro se encontram em serviço no Hospital de Braga.

```
| ?- enfermeiros_instituicao('Hospital de Braga', R).
R = [(1,'Joao Antunes',32),(2,'Catia Aveiro',29)] ?
yes
| ?-
```

5.1.6 Custo médio por serviço/instituição

- **Serviço**

De modo a ser possível ter noção do custo médio de um determinado serviço, decidimos que seria relevante a construção do seguinte predicado.

```
% Predicado servico_custo_medio:
% Id servico, Custo medio -> {V,F}
servico_custo_medio(IdServ,M) :-
    listaSolucoes(Custo,consulta(_,_,IdServ,Custo,_),S),
    media(S,M).
```

Assim, o sistema será capaz de indicar o custo médio por serviço tendo como conhecimento o identificador do serviço e uma lista de serviços.

```
| ?- servico_custo_medio(1,U).
U = 45 ?
yes
| ?-
```

- **Instituição**

Tal como pensado para a funcionalidade anterior, construímos o próximo predicado que devolverá o custo médio por instituição. Este difere do referido previamente na medida em que teremos de aceder ao predicado servico.

```
% Predicado instituicao_custo_medio:
% Instituicao, Custo medio -> {V,F}
instituicao_custo_medio(Instituicao,M) :-
    listaSolucoes(Custo,
    (consulta(_,_,IdServ,Custo,_),
    servico(IdServ,_,Instituicao,_)),S),
    media(S,M).
```

Desta forma, conseguimos verificar o custo médio em cuidados de saúde no Hospital de Braga.

```
| ?- instituicao_custo_medio('Hospital de Braga',I).
I = 60 ?
yes
| ?-
```

5.1.7 Obter as instituições existentes numa cidade

Devolve uma lista com as instituições existentes numa determinada cidade, pelo que se revela bastante útil.

```
1 instituicao_cidade(Cidade,L) :-
2     listaSolucoes(Instituicao,
3     servico(_,_,Instituicao,Cidade),S),
4     semRepetidos(S,L).
```

No exemplo que se pode observar de seguida, constatamos que as instituições existentes na cidade do Porto são o Santa Maria e o São João.

```
| ?- instituicao_cidade('Porto',S).
S = ['Santa Maria','Sao Joao'] ?
yes
| ?-
```

6 Conclusão

A resolução deste primeiro exercício prático permitiu a consolidação dos conhecimentos adquiridos nas aulas relativamente à linguagem de programação em lógica, PROLOG. Como podemos verificar o nosso projeto é capaz de representar o conhecimento pedido. Para além disto, optamos por adicionar novas funcionalidades e conhecimento extra, tais como os médicos, enfermeiros, turnos e escalonamento, de modo a ser possível uma melhor aproximação à realidade. Contudo, consideramos que seria possível expandir ainda mais a base de conhecimento elaborada.

Em suma, os objetivos estabelecidos foram cumpridos. Tendo em conta o trabalho realizado bem como os resultados obtidos consideramos que este primeiro exercício foi resolvido com sucesso.