UNIVERSIDADE DO MINHO
MIEI

SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO

1º Exercício

Grupo:
João Nunes - a82300
Flávio Martins - a65277
Luís Braga - a82088
Luís Martins - a82298

Braga, Portugal
5 de Abril de 2019

Conteúdo

1 Introdução

A programação em lógica é um paradigma da programação baseado na lógica formal, cujo o conteúdo dos programas se prende em factos e predicados que se associam a esses factos.

Como tal, este tipo de programação tem por base dois conceitos atómicos, a lógica em si utilizada para representar os conhecimentos e informação, e a inferência, que são regras aplicadas à lógica para manipular o conhecimento.

O trabalho prático consiste no desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caraterizar um universo no discurso na área de prestação de cuidados de sáude, sendo utilizada a ferramenta de programação em lógica *Prolog*.

O *Prolog* é uma linguagem de programação que se enquadra no paradigma da programação lógica, sendo uma das primeiras e mais populares linguagens neste tipo específico de programação. É geralmente utilizada em tarefas que beneficiam de consultas baseadas em regras lógicas tais como procurar em base de dados, sistemas de reconhecimento de voz, entre outros.

1.1 Motivação e objectivos

Este primeiro exercício tem como objectivo consolidar a matéria leccionada nas aulas portanto, o objectivo deste primeiro exercício é elaborar um programa capaz de armazenar conhecimento sobre registos de consultas de uma instituição realizados por um utente e solucionar problemas relativos a este tema.

1.2 Estrutura do relatório

Este relatório encontra-se dividido em cinco capítulos, neste capítulo inicial é feita uma breve introdução da linguagem e do tema em que se baseia este exercício.

No segundo capítulo é feita uma descrição da base de conhecimento inicial, sendo esta a base para a implementação das funcionalidades pretendidas.

No terceiro capítulo são descritos os passos necessários para a implementação das funcionalidades.

No quarto capítulo são apresentadas as funcionalidades extras implementadas pelo grupo de trabalho de modo a tornar o conhecimento o mais completo possível.

No final, ou seja, no quinto capítulo, é feita uma conclusão sobre o trabalho, onde estão apresentadas as reflexões do grupo de trabalho sobre o trabalho desenvolvido no decorrer deste exercício.

2 Base de conhecimento

A base de conhecimento define às base de dados ou conhecimento acumulado sobre um determinado assunto, como tal, para este exercício foi necessária a criação de uma base de conhecimento de modo a responder às necessidades do exercício em questão.

2.1 Serviço

De modo a proceder à resolução das funcionalidades propostas, foi criada a seguinte assinatura **servico(IdServ,Descricao,Instituicao,Cidade)**, onde a cada serviço este está associado a um id que o identifica unicamente, uma descrição sobre o serviço prestado e a instituição e cidade onde ele foi processado.

```
servico(1,"Urgencia", "Hospital Sao Joao", "Porto").
servico(2,"Terapia da Fala", "Hospital Sao Joao", "Porto").
servico(3,"Fisioterapia", "Hospital Sao Joao", "Porto").
servico(4, "Pediatria", "Hospital Sao Joao", "Porto").
servico(5,"Ginecologia", "Hospital Sao Joao", "Porto").
servico(6,"Neurologia", "Hospital de Braga", "Braga").
servico(7,"Oncologia", "Hospital de Braga", "Braga").
servico(8,"Urgencia", "Hospital de Braga", "Braga").
servico(9,"Cardiologia", "Hospital de Braga", "Braga").
servico(10,"Neurologia", "Hospital de Braga", "Braga").
```

2.2 Utente

Um requisito necessário a incluir na base de conhecimento são os utentes. Como tal foi criado o predicado **utente** com a seguinte assinatura **utente(IdUt,Nome,Idade,Cidade)**, onde cada utente está identificado com um identificador único, um nome, uma idade e a cidade de onde é natural.

```
utente(1, "Luis Martins", 20, "Braga").
utente(2, "Francisco Lomba", 16, "Braga").
utente(3, "Joao Nunes", 20, "Braga").
utente(4, "Renato Silva", 34, "Porto").
utente(5, "Mateus Oliveira", 22, "Porto").
utente(6, "Joana Santos", 45, "Porto").
utente(7, "Mariana Moreira", 20, "Porto").
utente(8, "Maria Ralha", 36, "Braga").
utente(9, "Patricia Nogueira", 57, "Braga").
utente(10, "Carla Queiros", 23, "Porto").
```

2.3 Consulta

Por fim, foi incluída a base de conhecimento relativa à consulta com a seguinte assinatura **consulta(Data,IdUt,IdServ,IdM,Custo)**. Cada consulta irá ter uma data específica onde a mesma ocorreu, um identificador do utente que usufruiu da consulta, o identificador do serviço prestado e um identificador do médico, cuja base de conhecimento será apresentada posteriormente, tendo esta sido adiconada de modo a tornar o conhecimento o mais completo possível e o custo da consulta.

```
consulta('2019-01-23',1,2,3,25.00).
consulta('2019-01-23',2,10,10,32.50).
consulta('2019-01-23',3,9,9,17.00).
consulta('2019-01-23',4,3,4,50.09).
consulta('2019-01-23',5,1,12,28.00).
consulta('2019-01-23',6,5,1,32.52).
consulta('2019-01-23',7,4,5,29.00).
consulta('2019-01-23',8,7,7,53.59).
consulta('2019-01-23',9,8,8,25.80).
consulta('2019-01-23',10,6,6,38.21).
consulta('2019-01-24',1,8,11,35.00).
consulta('2019-01-24',2,9,9,35.50).
consulta('2019-01-24',3,10,10,13.00).
consulta('2019-01-24',4,3,4,20.09).
consulta('2019-01-24',5,2,3,58.00).
consulta('2019-01-24',6,1,2,42.52).
consulta('2019-01-24',7,5,1,59.00).
consulta('2019-01-24',8,7,7,43.59).
consulta('2019-01-24',9,8,11,26.30).
consulta('2019-01-24',10,6,6,39.11).
```

3 Funcionalidades

3.1 Registar utentes, serviços e consultas

De modo a registar novos utentes, serviços e consultas, foi necessário criar invariantes de modo a garantir que não seria adicionada informação repetida, no caso do utente foi criado o seguinte invariante:

```
+utente(Id,_,_,_) :: (findall(Id,(utente(Id,_,_,_)),L),
 comprimento(L,1)).
```

Onde neste invariante, a sua função é garantir que não é adicionado um novo utente com um Id que já exista na base de conhecimento, sendo utilizado o *findall* para pesquisar na base de conhecimento o utente que se pretende adicionar pelo seu Id, e no final o comprimento da lista terá que ser 1, o que significa que não existem utentes repetidos na base de conhecimento.

Na mesma linha de pensamento, foi criado também o invariante para o serviço, tal como se segue:

```
+servico(Id,_,_,_) :: (findall(Id,(servico(Id,_,_,_)),L),
  comprimento(L,1)).
```

Por outro lado, para adicionar uma consulta é necessário verificar se o utente, o serviço existem e também se o médico que se está a atribuir à consulta existe e pertence ao serviço a que a consulta vai ser dada.

```
+consulta(_,IdUt,IdServ,IdM,_) :: (utente(IdUt,_,_,_),
servico(IdServ,_,_,_),
    medico(IdM,_,_,IdSer)).
```

Na evolução do conhecimento, e juntamente com estes invariantes que garantem a fidelidade do conhecimento, são utilizados os seguintes predicados, sendo o *evolucao* o responsável pela inserção de conhecimento.

```
inserir(E) :- assert(E).
inserir(E) :- retract(E),!,fail.

teste([]).
teste([X|Y]) :- X , teste(Y).

evolucao(E) :- findall(I,+E::I,L),
               inserir(E),
               teste(L).
```

3.2 Remover utentes, serviços e consultas

No que toca à remoção, foram também criados invariantes de modo a garantir a consistência na base de conhecimento, onde foram criados dois invariantes de modo assegurar que não são removidos ou utentes, ou serviços inexistentes na base de conhecimento, com a dificuldade acrescida de que, se tanto o utente ou o serviço existirem, no primeiro caso este não poderá estar associado a consultas, e no segundo não pode estar associado a médicos já existente na base de dados, que por consequência, também não poderá estar associado a consultas.

```
-utente(IdU,_,_,_) :: (findall(IdU,(utente(IdU,_,_,_)),L),
                       comprimento(L,1),
                       findall(IdU, consulta(_,IdU,_,_,_),C),
                       comprimento(C,0)).
-servico(IdS,_,_,_) :: (findall(IdS,(servico(IdS,_,_,_)),L),
                        comprimento(L,1),
                        findall(IdS, medico(_,_,_,IdS),M),
                        comprimento(M,0)).
```

De seguida, foi criado um predicado tendo sempre por base estes invariantes supracitados.

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

involucao(E) :- findall(I,-E::I,L),
 teste(L),
 remove(E).
```

3.3 Identificar as instituições prestadoras de serviços

Para implementar esta funcionalidade, foi necessário encontrar todas as instituições onde podem ser prestados serviços, como tal no serviço apenas iremos pesquisar por instituição.

De seguida, é aplicado o *setof* onde são pesquisadas as instituições que prestam serviços dentro da base de conhecimento do serviço. Não possuíndo instituições repetidas devido ao funcionamento do *setof*.

```
instituicao(I) :- servico(_,_,I,_).
instituicoes(L) :- setof(I,instituicao(I),L).
```

Por exemplo, testando o funcionamento deste predicado, temos:

```
?- instituicoes(L).
L = ["Hospital Sao Joao", "Hospital de Braga"].
```

3.4 Identificar utentes/serviços/consultas por critérios de seleção

De modo a identificar os utentes, serviços e consultas por critério de seleção ou seja, no caso do utente, por ID, nome, idade e cidade é utilizado o *findall*, onde são encontradas todas as soluções com o formato pretendido que satisfazem o critério de seleção.

Como tal, são apresentadas de seguida os predicados necessários para responder à questão apresentada:

```
procuraUID(ID,R) :- findall((ID,NOME,IDADE,CIDADE)
                    ,utente(ID,NOME,IDADE,CIDADE),R).

procuraUName(NOME,R) :- findall((ID,NOME,IDADE,CIDADE)
                        ,utente(ID,NOME,IDADE,CIDADE),R).

procuraUAge(IDADE,R) :- findall((ID,NOME,IDADE,CIDADE)
                        ,utente(ID,NOME,IDADE,CIDADE),R).

procuraUCidade(CIDADE,R) :- findall((ID,NOME,IDADE,CIDADE)
                         ,utente(ID,NOME,IDADE,CIDADE),R).
```

O resultado de utilizar todos estes predicados apresenta-se de seguida:

```
?- procuraUID(1,R).
R = [(1, "Luis Martins", 20, "Braga")].

?- procuraUName("Luis Martins",R).
R = [(1, "Luis Martins", 20, "Braga")].

?- procuraUAge(20,R).
R = [(1, "Luis Martins", 20, "Braga"),
    (3, "Joao Nunes", 20, "Braga"),
    (7, "Mariana Moreira", 20, "Porto")].

?- procuraUCidade("Braga",R).
R = [(1, "Luis Martins", 20, "Braga"),
    (2, "Francisco Lomba", 16, "Braga"),
    (3, "Joao Nunes", 20, "Braga"),
    (8, "Maria Ralha", 36, "Braga"),
    (9, "Patricia Nogueira", 57, "Braga")].
```

O processo de identificar os serviços por critério de seleção é feita de uma forma análoga à anterior, sendo que, neste os critérios de seleção são o ID, a descrição, a instituição e a cidade.

```
procuraSID(ID,R) :- findall((ID,DESCRICAO,INSTITUICAO,CIDADE),
                    servico(ID,DESCRICAO,INSTITUICAO,CIDADE),R).

procuraSDES(DESCRICAO,R) :- findall((ID,DESCRICAO,INSTITUICAO,CIDADE),
                         servico(ID,DESCRICAO,INSTITUICAO,CIDADE),R).

procuraSINS(INSTITUICAO,R) :- findall((ID,DESCRICAO,INSTITUICAO,CIDADE),
                           servico(ID,DESCRICAO,INSTITUICAO,CIDADE),R).

procuraSCID(CIDADE,R) :- findall((ID,DESCRICAO,INSTITUICAO,CIDADE),
                        servico(ID,DESCRICAO,INSTITUICAO,CIDADE),R).
```

Ao executar os predicados anteriores, possuímos o seguintes resultados:

```
?- procuraSID(1,R).
R = [(1, "Urgencia", "Hospital Sao Joao", "Porto")].

?- procuraSDES("Urgencia",R).
R = [(1, "Urgencia", "Hospital Sao Joao", "Porto"),
    (8, "Urgencia", "Hospital de Braga", "Braga")].

?- procuraSINS("Hospital Sao Joao",R).
R = [(1, "Urgencia", "Hospital Sao Joao", "Porto"),
    (2, "Terapia da Fala", "Hospital Sao Joao", "Porto"),
    (3, "Fisioterapia", "Hospital Sao Joao", "Porto"),
    (4, "Pediatria", "Hospital Sao Joao", "Porto"),
    (5, "Ginecologia", "Hospital Sao Joao", "Porto")].

?- procuraSCID("Porto",R).
R = [(1, "Urgencia", "Hospital Sao Joao", "Porto"),
    (2, "Terapia da Fala", "Hospital Sao Joao", "Porto"),
    (3, "Fisioterapia", "Hospital Sao Joao", "Porto"),
    (4, "Pediatria", "Hospital Sao Joao", "Porto"),
    (5, "Ginecologia", "Hospital Sao Joao", "Porto")].
```

Tal como nos anteriores, a mesma lógica se aplica aqui, onde o critério de seleção para as consultas é a data, o id do utente, o id do serviço, o id do médico que mais uma vez irá ser aborado mais a frente nos extras e por fim o custo da consulta.

```
procuraCData(Data,R) :- findall((Data,IdUt,IdServ,IdM,Custo),
                        consulta(Data,IdUt,IdServ,IdM,Custo),R).

procuraCIdUt(IdUt,R) :- findall((Data,IdUt,IdServ,IdM,Custo),
                        consulta(Data,IdUt,IdServ,IdM,Custo),R).

procuraCIdServ(IdServ,R) :- findall((Data,IdUt,IdServ,IdM,Custo),
                            consulta(Data,IdUt,IdServ,IdM,Custo),R).

procuraCIdM(IdM,R) :- findall((Data,IdUt,IdServ,IdM,Custo),
                      consulta(Data,IdUt,IdServ,IdM,Custo),R).

procuraCCusto(Custo,R) :- findall((Data,IdUt,IdServ,IdM,Custo),
                          consulta(Data,IdUt,IdServ,IdM,Custo),R).
```

O resultado de executar estes predicados é como se segue:

```
?- procuraCData('2019-01-23',R).
R = [('2019-01-23', 1, 2, 3, 25.0),
    ('2019-01-23', 2, 10, 10, 32.5),
    ('2019-01-23', 3, 9, 9, 17.0),
    ('2019-01-23', 4, 3, 4, 50.09),
```

```
    ('2019-01-23', 5, 1, 12, 28.0),
    ('2019-01-23', 6, 5, ..., ...),
    ('2019-01-23', 7, ..., ...),
    ('2019-01-23', ..., ...),
    (..., ...)|...].

?- procuraCIdUt(1,R).
R = [('2019-01-23', 1, 2, 3, 25.0),
    ('2019-01-24', 1, 8, 11, 35.0)].

?- procuraCIdServ(1,R).
R = [('2019-01-23', 5, 1, 12, 28.0),
    ('2019-01-24', 6, 1, 2, 42.52)].

?- procuraCCusto(25.00,R).
R = [('2019-01-23', 1, 2, 3, 25.0)].
```

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

De modo a identificar os serviços com esses critérios de seleção, é necessário recorrer ao *setof* cujo resultado é uma lista ordenada sem repetidos, onde apenas se verifica, nos dois primeiros casos, onde são retirados todos as descrições correspondentes ao argumento passado no predicado.

```
servicoInstAux(Descricao,Inst) :- servico(_,Descricao,Inst,_).
servicoInst(Inst, R) :- setof(Descricao, servicoInstAux(Descricao,Inst), R).

servicoCidAux(Descricao,Cidade) :- servico(_,Descricao,_,Cidade).
servicoCid(Cidade, R) :- setof(Descricao, servicoCidAux(Descricao,Cidade), R).
```

Cujos resultados da invocação desses dois predicados são os seguintes:

```
?- servicoInst("Hospital Sao Joao",R).
R = ["Urgencia", "Terapia da Fala",
    "Fisioterapia", "Pediatria", "Ginecologia"].

?- servicoCid("Porto",R).
R = ["Urgencia", "Terapia da Fala", "Fisioterapia",
    "Pediatria", "Ginecologia"].
```

Nos dois últimos casos, é necessário consultar uma base de conhecimento adicional, a consulta uma vez que envolve dados relativos à consulta ou seja, as datas e custos da consulta, servindo como uma *bridge* entre os dois o identificador do serviço ou seja o IdServ de modo a conseguir estabelecer uma conecção entre os argumentos de ambas as funções com os dados guardados na base de conhecimento da consulta, mais uma vez isto é feito com auxílio da função *setof*.

```
servicoDataAux(Descricao,Data) :- (servico(IdServ,Descricao,_,_),
servicoData(Data, R) :- setof(Descricao, servicoDataAux( Descricao, Data), R).

servicoCustoAux(Descricao,Custo) :-  (servico(IdServ,Descricao,_,_),
servicoCusto(Custo, R) :- setof(Descricao, servicoCustoAux(Descricao,Custo) ,R).
```

Cujo resultado da invocação de ambos estes predicados é o seguinte:

```
?- servicoData('2019-01-23',R).
R = ["Cardiologia", "Fisioterapia",
    "Ginecologia", "Neurologia",
    "Oncologia", "Pediatria",
    "Terapia da Fala", "Urgencia"].

?- servicoCusto(25.00,R).
R = ["Terapia da Fala"].
```

3.6 Identificar os utentes de um serviço/instituição

Nesta funcionalidade pretendida para identificar os utentes de um serviço/instituição, foi necessária a criação de dois predicados auxiliares onde em cada um deles é utilizado um invariante para certificar a existência de os utentes, sendo que da base de conhecimento a informação que será retirada será o ID do utente, o nome do utente a a idade deste mesmo.

De seguida é utilizado o *setof*, onde em conjunção com a função auxiliar anteriormente explicada, dá como resultado uma lista ordenada e sem repetições, tal como se pode verificar nos seguinte excerto de código:

```
utenteSerAux(IdSer,IdU,Nome,Idade) :- (utente(IdU,Nome,Idade,_),
                                       consulta(_,IdU,IdSer,_,_)).

utentesSerID(IdSer,R) :- setof((IdU,Nome,Idade),
                         utenteSerAux(IdSer,IdU,Nome,Idade),R).

utenteInstAux(I,IdU,Nome,Idade) :- (utente(IdU,Nome,Idade,_),
                                    consulta(_,IdU,IdSer,_,_),
                                    servico(IdSer,_,I,_)).

utentesInst(I,R) :- setof((IdU,Nome,Idade),
                    utenteInstAux(I,IdU,Nome,Idade),R).
```

Na execução destes predicados possuímos os seguintes resultados:

```
?- utentesSerID(1,R).
R = [(5, "Mateus Oliveira", 22),  (6, "Joana Santos", 45)].

?- utentesInst("Hospital Sao Joao",R).
R = [(1, "Luis Martins", 20),
    (4, "Renato Silva", 34),
```

```
    (5, "Mateus Oliveira", 22),
    (6, "Joana Santos", 45),
    (7, "Mariana Moreira", 20)].
```

3.7 Identificar serviços realizados por utente/instituição/ cidade

Neste caso, o funcionamento é congénere ao anterior, é utilizado novamente o *consulta* de modo a certificar a existência do serviço consoante o argumento utilizado no predicado principal, de seguida também é utilizado o *setof* de modo a retornar uma lista ordenada e sem repetidos.

```
serReaUtAux(IdUt,IdSer,Desc) :- (servico(IdSer,Desc,_,_),
                                 consulta(_,IdUt,IdSer,_,_)).
serReaUt(IdUt,R) :- setof((IdSer, Desc),
                     serReaUtAux(IdUt,IdSer,Desc), R).

serReaInstAux(IdSer,Desc,Inst) :- (servico(IdSer,Desc,Inst,_),
                                   consulta(_,_,IdSer,_,_)).
serReaInst(Inst,R) :- setof((IdSer, Desc),
                       serReaInstAux(IdSer,Desc,Inst),R).

serReaCidAux(Cidade,IdSer,Desc) :- (servico(IdSer,Desc,_,Cidade),
                                    consulta(_,_,IdSer,_,_)).
serReaCid(Cidade,R) :- setof((IdSer, Desc),
                        serReaCidAux(Cidade,IdSer,Desc) ,R).
```

Onde passado com os subsequentes argumentos, o predicado irá demonstar o seguinte comportamento:

```
?- serReaUt(1,R).
R = [(2, "Terapia da Fala"),  (8, "Urgencia")].

?- serReaInst("Hospital Sao Joao",R).
R = [(1, "Urgencia"),
    (2, "Terapia da Fala"),
    (3, "Fisioterapia"),
    (4, "Pediatria"),
    (5, "Ginecologia")].

?- serReaCid("Porto",R).
R = [(1, "Urgencia"),
    (2, "Terapia da Fala"),
    (3, "Fisioterapia"),
    (4, "Pediatria"),
    (5, "Ginecologia")].
```

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

Nesta última funcionalidade proposta, foi também necessário recorrer a algumas funções auxiliares criadas pelo grupo de trabalho, tal como o *somaC* que calcula uma soma acumulada numa lista, função essa que é relevante para calcular os custos totais dos utentes, serviço e por data nas consultas. Sendo portanto, necessário tanto num como no outro, utilizar o *findall* de modo a extrair a informação referente ao custo, recorrendo tal como no anterior ao predicado *consulta* de modo a manter sempre uma congruência ao aceder à base de conhecimento da consulta, e no fim é feita uma soma acumulada da lista resultante usando a função *somaC*.

```
somaC([], 0).
somaC([H|T], R) :- somaC(T, R2), R is H+R2.

custoTUt(IdUt,R) :-
    findall(Custo, consulta(_,IdUt,_,_,Custo), R2), somaC(R2,R).

custoTServ(IdSer,R) :-
    findall(Custo, consulta(_,_,IdSer,_,Custo),R2), somaC(R2,R).

custoTData(Data,R) :-
    findall(Custo, consulta(Data,_,_,_,Custo),R2), somaC(R2,R).
```

O resultado de execução destes três predicados é:

```
?- custoTUt(1,R).
R = 60.0.

?- custoTServ(1,R).
R = 70.52000000000001.

?- custoTData('2019-01-23',R).
R = 331.71000000000004.
```

No caso de calcular o custo total por instituição, foi criada um predicado auxiliar *custoTInstAux* com o intuito de estabelecer um tuplo de comparação entre o serviço e a consulta, onde é passado como argumento a instituição, sendo comparado o id do serviço da instituição com o id de serviço presente na consulta sendo retirado esse custo. De seguida, no predicado principal, é utilizado o *setof*, sendo invocada o predicado auxiliar previamente referido juntamente com o predicado *somaC*, para efetuar a soma dos custos da lista resultante.

```
custoTInstAux(Inst,Custo) :- (servico(IdSer,_,Inst,_),
                              consulta(_,_,IdSer,_,Custo)).
custoTInst(Inst,R) :- setof(Custo, custoTInstAux(Inst,Custo), R2),
                      somaC(R2,R).
```

Com os seguintes resultados:

```
?- custoTInst("Hospital Sao Joao",R).
R = 344.21999999999997.
```

4 Extras

Após a conclusão das funcionalidades exigidas, e de forma a enriquecer o conhecimento, foram implementadas algumas funcionalidades extras.

4.1 Base de conhecimento médico

Para além do conhecimento já dado, foi também implementada mais uma nova base de conhecimento acerca do médico, este é identificado por um ID, possuindo um nome, idade e o serviço sobre o qual poderá atender ou seja, **medico(idM,Nome,Idade,IdServ)**.

```
medico(1,"Flavio Martins",69,5).
medico(2,"Rodrigo Sousa",38,1).
medico(3,"Catia Costa",30,2).
medico(4,"Joaquina Afonsina",45,3).
medico(5,"Maria Joana",42,4).
medico(6,"Marisa Podence",56,6).
medico(7,"Jose Costa",63,7).
medico(8,"Ricardo Teixeira",43,8).
medico(9,"Elisabete Riccardi",49,9).
medico(10,"Isadora Goncalves",62,10).
medico(11,"Luis Braga", 34, 8).
medico(12,"Joao Rodrigues",59,1).
```

Tendo sido também criado, tal como nos outros, um invariante estrutural para garantir que não é inserido conhecimento já existente, ou seja, repetido. Além disso este invariante também só permite a inserção de um médico se o serviço a que está a ser associado existir.

```
+medico(IdM,_,_,IdSer) :: (findall(IdM,(medico(IdM,_,_,_)),L),
                           comprimento(L,1),
                           servico(IdSer, _, _, _)).
```

Para a remoção foi também criado o invariante estrutural que garante que um médico apenas é retirado se não existir consultas associadas ao mesmo.

```
-medico(IdM,_,_,_) :: (findall(IdM,(medico(IdM,_,_,_)),L),
                       comprimento(L,1),
                       findall(IdM, (consulta(_,_,_,IdM,_)),C),
                       comprimento(C,0)).
```

Devido à adição deste novo conhecimento, o outro conhecimento já pré-existente sofreu também alterações na sua estrutura, tal como a consulta que tem também informação sobre o identificador do médico.

4.2 Total ganho de consultas dados por um médico

Nesta funcionalidade, o predicado irá receber como argumento o id do médico cujo utilizador pretende saber o total ganho, sendo utilizado o *findall* com o intuito de formar uma lista com todos os custos das consultas que contêm o id do médico. No final é feita uma soma acumulada, utilizando para tal o *somaC*.

```
ganhoTMed(IdMed, R) :- findall(Custo, consulta(_,_,_,IdMed,Custo), R2),
                       somaC(R2,R).
```

Com o seguinte resultado de execução:

```
?- ganhoTMed(1,R).
R = 91.52000000000001.
```

4.3 Total de consultas dadas por um médico

Para elaborar tal predicado, foi necessário recorrer a um predicado auxiliar *inc*, cujo intuito é contar o número de elementos numa lista, tal como se segue:

```
inc([],0).
inc([_|T], R) :- inc(T,R2), R is R2+1.
```

De seguida, é utilizado novamente o *findall*, onde é pesquisado pelo id do médico passado como argumento nas consultas, de seguida é utilizada o predicado *inc*, previamente abordada, sendo retornado o número de elementos dessa lista.

```
consultasMed(IdMed, R) :- findall(IdMed, consulta(_,_,_,IdMed,_), R2),
                          inc(R2,R).
```

O resultado será o seguinte:

```
?- consultasMed(1,R).
R = 2.
```

4.4 Quantas consultas um utente já usufruiu

Neste predicado também é utilizado o predicado anteriormente abordado, ou seja, o *inc*. Neste predicado, é utilizado o *findall*, onde são percorridas as consultas de modo a encontrar as consultas com cujo id é igual ao id do utente passado como argumento, de seguida, é feita uma contagem do número de entradas da lista resultante usando o *inc*.

```
consultasUt(IdUt,R) :- findall(IdUt, consulta(_,IdUt,_,_,_),R2),
                       inc(R2,R).
```

Ao executar este predicado, possuímos os seguintes resultados:

```
?- consultasUt(1,R).
R = 2.
```

4.5 Datas das consultas de um dado utente

Nesta funcionalidade, é utilizado um id do utente passado como argumento usando o *findall*, de modo a encontrar dentro das consultas as consultas cujo data é igual à data passada como argumento do predicado.

```
datasUt(IdUt,R) :- findall(Data, consulta(Data,IdUt,_,_,_),R).
```

Executando este predicado temos:

```
?- datasUt(1,R).
R = ['2019-01-23', '2019-01-24'].
```

5 Conclusão

Após a conclusão deste trabalho prático, foi possível consolidar os conhecimentos aprendidos nas aulas práticas e teóricas, utilizando para tal o *Prolog* como ferramenta para representar e construir os mecanismos de raciocínio e conhecimento, aplicadas na área de prestação de cuidados de cuidados de saúde.

Antes do grupo se debruçar sobre o exercício em mão, foi necessário estudar o problema apresentado de modo a melhor perceber o que era pretendido. Tendo sido feitas de seguida todas as funcionalidades pretendidas e algumas funcionalidades extra de modo a melhor completar o exercício proposto, aproximando o universo de discurso proposto ainda mais em relação à realidade.