Universidade do Minho
Escola de Engenharia

Sistemas de Representação de Conhecimento e Raciocínio

TRABALHO PRÁTICO Nº 1

Mestrado Integrado em Engenharia Informática

Grupo 30

78416 Francisco José Moreira Oliveira

79617 Raul Vilas Boas

79175 Vitor Emanuel Carvalho Peixoto

Ano letivo 2017/2018

Braga, Março de 2018

RESUMO

Este trabalho prático foi desenvolvido com o intuito de desenvolver e evoluir as competências adquiridas, na unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, na programação em lógica com o PROLOG.

Este primeiro trabalho reflete sobre uma área de prestação de cuidados de saúde e o objetivo deste é desenvolver um sistema para este caso de estudo.

Este relatório foi desenvolvido com o intuito de explicar o processo de desenvolvimento, bem como as escolhas tomadas no decorrer deste.

ÍNDICE

1. INTRODUÇÃO

Este primeiro trabalho, consiste na realização de um conjunto de exercícios envolvendo um sistema desenvolvido à volta da área de prestação de cuidados de saúde e o objetivo deste é desenvolver um sistema de representação de conhecimento e raciocínio para este caso de estudo. Logo, para isso, vai ser necessário caracterizar o conhecimento como apresentado no enunciado.

Vai ser necessário representar utentes, prestadores e cuidados de saúde, onde cada um possui a sua própria informação. Para representar esse conhecimento foram definidos os predicadas para cada uma dessas entidades.

2. PRELIMINARES

Neste caso em específico, consideramos que não há a necessidade de explicar qualquer tipo de preliminares, uma vez que todo o conhecimento aplicado no desenvolvimento deste trabalho prático foi obtido unicamente através da matéria lecionada na unidade curricular.

No entanto, para a resolução de algumas funcionalidades extra, foi necessário a utilização de algumas funções pré-definidas no PROLOG, como por exemplo o "*sort*".

3. DESCRIÇÃO DO TRABALHO E ANÁLISE DE RESULTADOS

Neste trabalho existem 3 tipos de predicados que representam o conhecimento, sendo, eles os **utentes**, os **prestadores** e os **cuidados de saúde**. Os utentes possuem um identificador, um nome, uma idade e a morada. Os prestadores contêm um identificador, um nome, uma especialidade e uma instituição. E por fim, os cuidados de saúde possuem uma data, o identificador do utente, o identificar do prestador, uma descrição e um custo.

- utente: #IdUt, Nome, Idade, Morada ⇝ { 𝕍, 𝔽 }
- prestador: #IdPrest, Nome, Especialidade, Instituição ⇝ { 𝕍, 𝔽 }
- cuidado: Data, #IdUt, #IdPrest, Descrição, Custo ⇝ { 𝕍, 𝔽 }

Para além disto, também foi adicionado nos utentes o tipo "género" que distingue os utentes do sexo masculino dos utentes do sexo feminino.

3.1 Representação do conhecimento

Para representar o conhecimento foi então necessário povoar a base de dados com exemplos de utentes, prestadores e cuidados de saúde.

Primeiro começamos pelos utentes. Estes possuem identificador, nome, idade, morada e género:

```
%utente(Id,Nome,Idade,Morada,Genero)
utente(1,'Raul',20,'Campos','Masculino').
utente(2,'Francisco',20,'Joane','Masculino').
utente(3,'Vitor',20,'Vermoim','Masculino').
utente(4,'Carlos',7,'Campos','Masculino').
utente(5,'Bruno',20,'Campos','Masculino').
utente(6,'Ana',3,'Cerveira','Feminino').
utente(7,'Susana',20,'Cerveira','Feminino').
utente(8,'Cristina',40,'Cerveira','Feminino').
utente(9,'Fatima',77,'Braga','Feminino').
utente(10,'Filipe',33,'Braga','Masculino').
utente(11,'Carla',11,'Porto','Feminino').
utente(12,'Fabio',88,'Famalicao','Masculino').
```

Figura 1 - Base de conhecimento dos utentes

Depois seguem-se os prestadores, caraterizados pelo identificador, nome, especialidade e instituição:

```
%prestador(IDPrestador,Nome,Especialidade,Instituição)
prestador(1,'Tiago','Ortopedia','Hospital de Braga').
prestador(2,'Guilherme','Urologia','Hospital de Braga').
prestador(3,'Renato','Radiologia','Hospital de Santa Maria').
prestador(4,'Filipe','Psiquiatria','Hospital de Santa Maria').
prestador(5,'Tiago','Cirurgia','Hospital de Braga').
prestador(6,'Vitor','Pediatria','Hospital de Santo Antonio').
prestador(7,'Gil','Cirurgia','Hospital de Braga').
prestador(8,'Joao','Ortopedia','Hospital de Braga').
prestador(9,'Diana','Psiquiatria','Hospital de Braga').
```

Figura 2 - Base do conhecimento dos prestadores

Por fim, temos os cuidados de saúde que possuem uma data, a identificação do utente e prestador, uma descrição e um custo:

```
%cuidado(Data,IDU,IDP,Descrição,Custo)
cuidado('01-01-2018', 3, 1,'Dor de barriga', 15).
cuidado('04-04-2018', 7, 2,'Braco partido',20).
cuidado('23-01-2018', 4, 2,'Perna partida', 30).
cuidado('01-05-2018', 2, 3,'Reacao alergica', 5).
cuidado('03-03-2018', 12,8, 'Febre', 4).
cuidado('31-12-2018', 10,5, 'Febre', 5).
cuidado('30-03-2018', 7, 9,'Analises', 5).
cuidado('31-12-2018', 7, 6,'Urgencia', 200).
cuidado('31-12-2018', 1, 6,'Braco partido', 1).
cuidado('07-08-2018', 11, 9,'Reacao alergica', 5).
cuidado('01-04-2018', 1, 4,'Dor de cabeca', 30).
```

Figura 3 - Base do conhecimento dos cuidados de saúde

3.2 Registar utentes, prestadores e cuidados de Saúde

O primeiro ponto do enunciado pede para criar uma sintaxe que consiga registar utentes, prestadores e cuidados de saúde. Para além disso, esta inserção tem de respeitar certos aspetos, como por exemplo, não haver a possibilidade de inserir dois utentes com o mesmo identificador. Por esse motivo, também se criou invariantes que não permitam que isso aconteça.

Para isso utilizou-se a função evolução criada nas aulas práticas que permite a inserção de conhecimento na base de dados, com a ajuda da função "*assert*" do PROLOG, mas também permite ver se a inserção respeita todos os invariantes existentes. Caso o invariante não seja respeitado, o conhecimento, que foi anteriormente adicionado, é removido.

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

insere(P) :- assert(P).
insere(P) :- retract(P),!,fail.

evolucao(Termo) :- solucoes(Inv,+Termo::Inv,S),
                   insere(Termo),
                   teste(S).
```

Figura 4 - Código que permite a inserção de conhecimento

Logo, como não é possível que haja dois utentes com o mesmo identificador criou-se o seguinte algoritmo que dado um utente com um certo ID vai procurar na base de dados se há algum utente com esse mesmo ID, caso haja, coloca esse ID numa lista. Depois, é só analisar o tamanho da lista e verificar se a condição é respeitada.

Visto que, o conhecimento do utente em causa é adicionado antes dos invariantes serem verificados, o tamanho mínimo da lista será 1 que representará o utente que queremos adicionar. No entanto, se o tamanho for igual a 2, isto significa que já existia na base de conhecimento um utente com esse identificador e por esse motivo, não podemos adicionar o utente à base, sendo por isso removido.

```
% Não pode haver mais do que uma ocorrência de um utente
+utente(ID,No,I,M,G) :: (solucoes(ID,utente(ID,X,Y,Z,W),S ),
                         comprimento(S,N),
                         N == 1).
```

Figura 5 – Invariante que não permite inserção de conhecimento de utentes repetidos

Para comprovar que o invariante estava correto, criou-se um exemplo em que se utilizou um identificador já usado na base de conhecimento (ID=1) e verificou-se que a sua inserção não acontecia. Depois, usando o mesmo exemplo, apenas alterando o identificador para um que não existia na base de conhecimento (ID = 13) verificou-se que o conhecimento foi inserido.

```
| ?- evolucao(utente(1,'Rafael',10,'Braga','Masculino')).
no
| ?- evolucao(utente(13,'Rafael',10,'Braga','Masculino')).
yes
| ?- listing(utente).
utente(1, 'Raul', 20, 'Campos', 'Masculino').
utente(2, 'Francisco', 20, 'Joane', 'Masculino').
utente(3, 'Vitor', 20, 'Vermoim', 'Masculino').
utente(4, 'Carlos', 7, 'Campos', 'Masculino').
utente(5, 'Bruno', 20, 'Campos', 'Masculino').
utente(6, 'Ana', 3, 'Cerveira', 'Feminino').
utente(7, 'Susana', 20, 'Cerveira', 'Feminino').
utente(8, 'Cristina', 40, 'Cerveira', 'Feminino').
utente(9, 'Fatima', 77, 'Braga', 'Feminino').
utente(10, 'Filipe', 33, 'Braga', 'Masculino').
utente(11, 'Carla', 11, 'Porto', 'Feminino').
utente(12, 'Fabio', 88, 'Famalicao', 'Masculino').
utente(13, 'Rafael', 10, 'Braga', 'Masculino').
```

Figura 6 - Resultado do invariante

Para além deste, também se criou outro invariante que não permite a inserção de um utente com uma idade inválida (que não pertença aos números naturais com o zero inclusive) e outro um que não permite que um utente seja de outro sexo para além de masculino ou feminino.

```
% O utente a ser inserido não pode ter uma idade inválida
+utente(ID,N,I,M,G) :: naturais(I).

naturais(0).
naturais(X) :- N is X-1, N >= 0, naturais(N).

% O utente a ser inserido apenas pode ter dois géneros, masculino ou femininos
+utente(ID,N,I,M,G) :: genero(G).

genero(X) :- X == 'Masculino'; X == 'Feminino'.
```

Figura 8 - Invariante para a idade e para o género

```
| ?- evolucao(utente(13,'Rafael',-5,'Braga','Alien')).
no
| ?- listing(utente).
utente(1, 'Raul', 20, 'Campos', 'Masculino').
utente(2, 'Francisco', 20, 'Joane', 'Masculino').
utente(3, 'Vitor', 20, 'Vermoim', 'Masculino').
utente(4, 'Carlos', 7, 'Campos', 'Masculino').
utente(5, 'Bruno', 20, 'Campos', 'Masculino').
utente(6, 'Ana', 3, 'Cerveira', 'Feminino').
utente(7, 'Susana', 20, 'Cerveira', 'Feminino').
utente(8, 'Cristina', 40, 'Cerveira', 'Feminino').
utente(9, 'Fatima', 77, 'Braga', 'Feminino').
utente(10, 'Filipe', 33, 'Braga', 'Masculino').
utente(11, 'Carla', 11, 'Porto', 'Feminino').
utente(12, 'Fabio', 88, 'Famalicao', 'Masculino').
```

Figura 7 - Resultado do invariante

Para o prestador, tal como para o utente, criou-se um invariante que não permite a inserção de um prestador com um identificador igual a um prestador já existente. A estrutura deste invariante é muito semelhante à do invariante anterior.

```
% Não pode haver mais do que uma ocorrência de um prestador
+prestador(ID,No,E,I) :: (solucoes(ID,prestador(ID,X,Y,Z),S),
                          comprimento(S,N),
                          N == 1).
```

Figura 9 - Invariante dos prestadores

Por último, para os cuidados criou-se três invariantes: O primeiro não permite a inserção de cuidados, caso o identificador do utente a ser inserido não exista; O segundo não permite a inserção caso o identificador do prestador não exista na base de conhecimento; O terceiro invariante não permite que o custo do cuidado a ser inserido seja inválido, isto é, menor que zero.

3.3 Remover utentes, prestadores e cuidados de saúde

O seguinte ponto pede para remover utentes, prestadores e cuidados de saúde. Para isso criou-se então a seguinte função 'involucao' que permite que isso aconteça. Assim como a 'evolucao', a remoção do conhecimento também tem que se sujeitar a alguns invariantes que impedem que certas situações ocorram. Para além disto, também é importante referir que não é possível remover conhecimento que não exista na base de conhecimento. Para isso não acontecer, o argumento que a função 'involucao' recebe tem de ser verdadeiro, em que, apenas nesse caso o resto do predicado se irá realizar.

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

involucao( Termo ) :-  Termo,
                       solucoes(Inv,-Termo::Inv,S),
                       remove(Termo),
                       teste(S).
```

Figura 10 - Código para a remoção de conhecimento

```
| ?- involucao(utente(8,'Cristina',40,'Cerveira','Feminino')).
yes
| ?- listing(utente).
utente(1, 'Raul', 20, 'Campos', 'Masculino').
utente(2, 'Francisco', 20, 'Joane', 'Masculino').
utente(3, 'Vitor', 20, 'Vermoim', 'Masculino').
utente(4, 'Carlos', 7, 'Campos', 'Masculino').
utente(5, 'Bruno', 20, 'Campos', 'Masculino').
utente(6, 'Ana', 3, 'Cerveira', 'Feminino').
utente(7, 'Susana', 20, 'Cerveira', 'Feminino').
utente(9, 'Fatima', 77, 'Braga', 'Feminino').
utente(10, 'Filipe', 33, 'Braga', 'Masculino').
utente(11, 'Carla', 11, 'Porto', 'Feminino').
utente(12, 'Fabio', 88, 'Famalicao', 'Masculino').
```

Figura 11 - Resultado da remoção de conhecimento

Um dos casos em que se verificou que uma remoção seria inválida, era quando se tentaria remover um utente ou um prestador, quando existem cuidados de saúde com eles. Pois isso depois implicaria que haveria cuidados com utentes ou prestadores que não existiam na base de conhecimento e caso se quisesse saber a informações deles, seria impossível. Logo, criou-se dois invariantes que não permitem que se remova tanto um utente como um prestador, caso existam cuidados de saúdes com eles. Para isso, por exemplo, ao ser fornecido o utente para remover vamos procurar nos cuidados se há algum com o identificador do utente e caso haja é colocado numa lista. Depois, após calcular o comprimento da lista, retira-se as conclusões, isto é, caso o tamanho da lista seja maior ou igual a 1, quer dizer que existem cuidados com esse utente, caso o tamanho da lista seja 0, quer dizer que não existem cuidados com esse utente, e por isso, podemos removê-lo da base de conhecimento.

```
% Invariante que não permite a remoção de utentes caso exista cuidados de saúde com eles
-utente(ID,No,I,M,G) :: (solucoes(ID,cuidado(_,ID,_,_,_),L),
                         comprimento(L,X),
                         X == 0).

% Invariante que não permite a remoção de prestadores caso exista cuidados de saúde com eles
-prestador(ID,No,E,I) :: (solucoes(ID,cuidado(_,_,ID,_,_),L),
                          comprimento(L,X),
                          X == 0).
```

Figura 12 - Invariantes de remoção

Visto que, como a Susana (utente com ID=7) está presente em dois cuidados de saúde não pode ser removida. No entanto, a sua posição no 'listing' é alterada porque o algoritmo primeiro remove e só depois de verificar se os invariantes foram todos respeitados é que volta a adicionar, caso isto não se verifique a inserção não é efetuada.

```
| ?- involucao(utente(7,'Susana',20,'Cerveira','Feminino')).
no
| ?- listing(utente).
utente(1, 'Raul', 20, 'Campos', 'Masculino').
utente(2, 'Francisco', 20, 'Joane', 'Masculino').
utente(3, 'Vitor', 20, 'Vermoim', 'Masculino').
utente(4, 'Carlos', 7, 'Campos', 'Masculino').
utente(5, 'Bruno', 20, 'Campos', 'Masculino').
utente(6, 'Ana', 3, 'Cerveira', 'Feminino').
utente(8, 'Cristina', 40, 'Cerveira', 'Feminino').
utente(9, 'Fatima', 77, 'Braga', 'Feminino').
utente(10, 'Filipe', 33, 'Braga', 'Masculino').
utente(11, 'Carla', 11, 'Porto', 'Feminino').
utente(12, 'Fabio', 88, 'Famalicao', 'Masculino').
utente(7, 'Susana', 20, 'Cerveira', 'Feminino').
```

Figura 13 - Exemplo dos invariantes

3.4 Identificar utentes por critérios de seleção

Neste ponto, o exercício propunha a identificação dos diversos utentes pelos seus critérios de seleção como por exemplo, o nome, a idade, a morada e o género, sendo este último criado extra.

Para a seleção pelo nome criou-se o predicado 'utentePorNome' que recebe como argumento um nome e vai retornar uma lista com todas as informações de todos os utentes com esse nome. Para isso, a função vai comparar o nome recebido com o nome de todos os utentes e os utentes que tiverem o nome igual ao pretendido guarda-se numa lista que posteriormente é retornada como o resultado.

```
% Extensao do predicado utentesPorNome: Nome,R -> {V,F}
utentePorNome(Nome,R):- solucoes(utente(ID,Nome,Idade,Morada,Genero),utente(ID,Nome,Idade,Morada,Genero),R).
```

Figura 14 – Função que seleciona por nome

Para o caso da idade e da morada, a estrutura do algoritmo é a mesma, apenas alterando o argumento recebido que será agora uma idade ou uma morada:

```
% Extensao do predicado utentesPorIdade: Idade,R -> {V,F}
utentePorIdade(Idade,R) :- solucoes(utente(ID,Nome,Idade,Morada,Genero),utente(ID,Nome,Idade,Morada,Genero),R).

% Extensao do predicado utentesPorMorada: Morada,R -> {V,F}
utentePorMorada(Morada,R) :- solucoes(utente(ID,Nome,Idade,Morada,Genero),utente(ID,Nome,Idade,Morada,Genero),R).
```

Figura 15 - Funções que selecionam por idade e morada

Podemos assim usar, por exemplo, a função 'utentePorIdade' e dar o argumento 20 e descobrir todos os utentes que têm vinte anos de idade:

```
| ?- utentePorIdade(20,R).
R = [utente(1,'Raul',20,'Campos','Masculino'),utente(2,'Francisco',20,'Joane','
Masculino'),utente(3,'Vitor',20,'Vermoim','Masculino'),utente(5,'Bruno',20,'Cam
pos','Masculino'),utente(7,'Susana',20,'Cerveira','Feminino')] ? ;
```

Figura 16 - Resultado obtido pela função 'utentePorIdade' para utentes com 20 anos.

Também se criou um predicado para o género, em que retorna todos os utentes de um determinado género:

```
% Extensao do predicado utentesPorGenero: R -> {V,F}
% Retorna as informações de todos os utentes de um determinado genero
utentesPorGenero(Genero,R) :- solucoes(utente(IDU,Nome,Idade,Morada,Genero),utente(IDU,Nome,Idade,Morada,Genero),R).
```

Figura 17 - Função que seleciona por género

Por último, criou-se um predicado que retornava as informações dos utentes ordenados crescentemente por idade, 'utentes_orderIdade'. Para isso utiliza-se uma função auxiliar, a 'ordenarPorIdade' que por usa vez utiliza outra função, denominada 'insertPorIdade'. Com a combinação destas duas funções conseguimos com a lista de todas as informações de utentes ordena-los por idade, retornando assim o resultado pretendido.

```
% Extensao do predicado utentes_orderIdade: R -> {V,F}
% Ordenar utentes por idade
utentes_orderIdade(R) :- solucoes(utente(ID,Nome,Idade,Morada,Genero),utente(ID,Nome,Idade,Morada,Genero),L), ordenarPorIdade(L,R).
```

Figura 19 - Função que retorna os utentes ordenados por idade

```
% Extensao do predicado ordenarPorIdade: L,Resultado -> {V,F}
ordenarPorIdade([X],[X]).
ordenarPorIdade([X|Y],T):-
    ordenarPorIdade(Y,R),insertPorIdade(X,R,T).

% Extensao do predicado insertPorIdade: X,L,Resultado  -> {V,F}
insertPorIdade((X,Y,Z,W), [], [(X,Y,Z,W)]).
insertPorIdade((X1,Y1,Z1,W1), [(X2,Y2,Z2,W2)|T], [(X1,Y1,Z1,W1)|[(X2,Y2,Z2,W2)|T]]) :- Z1=<Z2.
insertPorIdade((X1,Y1,Z1,W1), [(X2,Y2,Z2,W2)|T], [(X2,Y2,Z2,W2)|R]) :- Z1>Z2,
    insertPorIdade((X1,Y1,Z1,W1),T,R).
```

Figura 18 - Funções auxiliares que inserem utentes ordenados por idade.

```
| ?- utentes_orderIdade(X).
X = [(6,'Ana',3,'Cerveira','Feminino'),(4,'Carlos',7,'Campos','Masculino'),(11,
'Carla',11,'Porto','Feminino'),(1,'Raul',20,'Campos','Masculino'),(2,'Francisco
',20,'Joane','Masculino'),(3,'Vitor',20,'Vermoim','Masculino'),(5,'Bruno',20,..
.,...),(7,'Susana',...,...),(10,...,...),(...,...)|...] ?
```

Figura 20 - Resultado da 'utentes_orderIdade'.

3.5 Identificar as instituições prestadores de cuidados de saúde

Neste exercício, é pedido para identificar as instituições que aparecem nos cuidados de saúde. Como não é possível fazer uma ligação direta entre eles é necessário colocar mais argumentos no 'solucoes', isto é, ele vai primeiro descobrir todos os identificadores de prestadores que tem cuidados de saúde e retornar as instituições desses prestadores. No entanto, como ao fazer isto vai haver uma lista com as instituições repetidas usa-se a função 'removeDups' para remover as que estão repetidas. Obtendo assim, as instituições prestadoras de cuidados de saúde.

```
% Extensao do predicado inst_cuidados: R -> {V,F}
% Identificar as instituições prestadoras de cuidados de saúde, sem repetidos
inst_cuidados(R):- solucoes(Insti,(cuidado(_,_,IDP,_,_),prestador(IDP,_,_,Insti)),L),
                   removeDups(L,R).
```

Figura 22 - Identifica as instituições que prestam cuidados.

```
| ?- inst_cuidados(R).
R = ['Hospital de Santo Antonio','Hospital de Braga','Hospital de Santa Maria']
```

Figura 21 - Resultado do predicado 'inst_cuidados'.

3.6 Identificar cuidados de saúde prestados por instituição/cidade/data.

Neste ponto, o objetivo é identificar os cuidados de saúde por instituição, cidade e data.
Primeiro começamos por identificar todos os cuidados prestados por uma determinada instituição. Para isso, usou-se o 'solucoes' que procurou todos os identificadores dos prestadores da instituição dada como argumento e depois procurou todos os cuidados que tinham esses mesmos prestadores, retornando assim todos os cuidados em que os seus prestadores trabalham nessa instituição.

```
% Extensao do predicado cuidados_insti: Inst,R -> {V,F}
% Cuidados realizados por uma determinada Instituicao
cuidados_insti(Inst,R) :- solucoes(cuidado(D,IDU,IDP,Desc,C),(prestador(IDP,_,_,Inst),cuidado(D,IDU,IDP,Desc,C)),R).
```

Figura 24 - Retorna os cuidados realizados por uma instituição.

```
| ?- cuidados_insti('Hospital de Santa Maria',R).
R = [cuidado('01-05-2018',2,3,'Reacao alergica',5),cuidado('01-04-2018',1,4,'Do
r de cabeca',30)] ? ;
```

Figura 23 - Resultado da 'cuidados_insti' aplicada ao 'Hospital de Santa Maria'.

Depois segue-se a identificação dos cuidados por cidade, no entanto, visto que nenhum possuía um atributo 'Cidade', utilizou-se a morada. A abordagem utilizada foi muito semelhante a anterior, pois procurou-se todos os utentes que possuíam aquela morada e retornou-se todos os cuidados que possuem esses utentes encontrados. Permitindo assim, que o resultado apresentado fosse todos os cuidados de uma morada especifica.

```
% Extensao do predicado cuidados_morada: Morada,R -> {V,F}
% Retorna todos os cuidados de uma certa morada
cuidados_morada(Morada,R):- solucoes(cuidado(D,IDU,IDP,Desc,C),(utente(IDU,_,_,Morada,_),cuidado(D,IDU,IDP,Desc,C)),R).
```

Figura 26 - Cuidados obtidos de uma morada.

```
| ?- cuidados_morada('Campos',R).
R = [cuidado('31-12-2018',1,6,'Braco partido',1),cuidado('01-04-2018',1,4,'Dor
de cabeca',30),cuidado('23-01-2018',4,2,'Perna partida',30)] ?
```

Figura 25 - Resultado da 'cuidados_morada' aplicada a 'Campos'.

Por último, temos a função que identifica os cuidados prestados numa determianda data. Para isso, simplesmente usamos o 'solucoes' com a data pretendida e ele vai retornar a lista de todos os cuidados que foram realizados nessa determinada data, fornecida como argumento.

```
% Extensao do predicado cuidados_data: Data,R -> {V,F}
% Retorna todos os cuidados numa determinada data
cuidados_data(Data,R) :- solucoes(cuidado(Data,X,Y,Z,W),cuidado(Data,X,Y,Z,W),R).
```

Figura 28 - Cuidados realizados numa determinada data.

```
| ?- cuidados_data('31-12-2018',R).
R = [cuidado('31-12-2018',10,5,'Febre',5),cuidado('31-12-2018',7,6,'Urgencia',2
00),cuidado('31-12-2018',1,6,'Braco partido',1)] ?
```

Figura 27 - Resultado da 'cuidados_data' aplicada ao dia 31-12-2018.

3.7 Identificar os utentes de um prestador/especialidade/instituição

Para identificar todos os utentes de um certo prestador é necessário pesquisar nos cuidados e encontrar todos os identificadores de utentes que tinham sido tratados pelo prestador dado como argumento. Depois, apenas é necessário procurar os utentes que cumprem este requisito e retorná-los numa lista. Utilizando o 'solucoes', tal como apresentado em baixo, vai retornar todos os utentes de um prestador especifico.

```
% Extensao do predicado utentes_prest: IDP,R -> {V,F}
% Retorna os utentes de um certo prestador
utentes_prest(IDP,R) :- solucoes(utente(IDU,N,I,M,G), (cuidado(Data,IDU,IDP,Descricao,Custo),utente(IDU,N,I,M,G)),R).
```

Figura 29 - Retorna os utentes de um prestador específico.

```
| ?- utentes_prest(6,R).
R = [utente(7,'Susana',20,'Cerveira','Feminino'),utente(1,'Raul',20,'Campos','M
asculino')] ? ;
```

Figura 30 - Resultado da 'utentes_prest' aplicado ao prestador com ID=6.

Para os dois outros casos, a estrutura é a mesma, apenas alterando em que, em vez de se procurar nos cuidados e nos utentes, também tem de se procurar nos prestadores pela especialidade ou instituição pedida.

```
% Extensao do predicado utentes_espe: Especialidade,R -> {V,F}
% Retorna os utentes de uma certa especialidade
utentes_espe(Espe,R) :- solucoes(utente(IDU,N,I,M,G), (cuidado(_,IDU,IDP,_,_),prestador(IDP,_,Espe,_),utente(IDU,N,I,M,G)),R).


% Extensao do predicado utentes_insti: Instituicao,R -> {V,F}
% Retorna os utentes  de uma determinada instituição
utentes_inst(Inst,R) :- solucoes(utente(IDU,N,I,M,G), (cuidado(_,IDU,IDP,_,_),prestador(IDP,_,_,Inst),utente(IDU,N,I,M,G)),R).
```

Figura 31 – Predicados que retornam os utentes de uma certa especialidade e de uma certa instituição.

3.8 Identificar cuidados de saúde realizados por utente/prestador

Para identificar os cuidados de saúde realizados por um utente basta procurar em todos os cuidados por esse utente e retornar apenas os cuidados que o incluem.

Para identificar os cuidados de um prestador a ideia é a mesma, excetuando que, em vez de procurar pelos utentes, procura-se antes pelo prestador, fornecido como argumento.

```
% Extensao do predicado cuidados_utente: Utente,R -> {V,F}
% Cuidados realizados por um utente especifico
cuidados_utente(Utente,R) :- solucoes(cuidado(X,Utente,Y,Z,W),cuidado(X,Utente,Y,Z,W),R).

% Extensao do predicado cuidados_prest: Prestador,R -> {V,F}
% Cuidados realizados por um prestador
cuidados_prest(Prestador,R) :- solucoes(cuidado(X,Y,Prestador,Z,W),cuidado(X,Y,Prestador,Z,W),R).
```

Figura 33 – Funções que descobrem os cuidados realizados por um utente e por um prestador

```
| ?- cuidados_utente(7,R).
R = [cuidado('04-04-2018',7,2,'Braco partido',20),cuidado('30-03-2018',7,9,'Ana
lises',5),cuidado('31-12-2018',7,6,'Urgencia',200)] ?
yes
| ?- cuidados_prest(6,R).
R = [cuidado('31-12-2018',7,6,'Urgencia',200),cuidado('31-12-2018',1,6,'Braco p
artido',1)] ?
yes
```

Figura 32 - Resultado das funções

3.9 Determinar todas as instituições/prestadores a que um utente já recorreu

Para determinar todas as instituições que um utente já frequentou, é necessário ligar todos os conhecimentos, pois as instituições estão nos prestadores e para ligar prestadores a utentes é necessário usar os cuidados. Para isso, nos 'solucoes' forneceu-se os seguintes argumentos, nos utentes vamos buscar o utente dado com argumento, nos cuidados vamos buscar todos os cuidados que tenham o utente em causa e obtemos o identificador dos prestadores que cuidaram desse utente, e por fim, vamos aos prestadores obtidos e vamos retornar todas as instituições desses prestadores. Obtendo assim todas as instituições a que um utente recorreu, no entanto é necessário remover os repetidos, pois um utente pode ter realizado vários cuidados na mesma instituição que iram aparecer repetidamente.

```
% Extensao do predicado insti_utente: Utente,R -> {V,F}
% Retorna a lista de instituições frequentadas por um utente sem repetições
insti_utente(IDU,R) :- solucoes(Insti,(cuidado(_,IDU,IDP,_,_),prestador(IDP,_,_,Insti),utente(IDU,_,_,_,_)),L),
                       removeDups(L,R).
```

Figura 35 - Retorna todas as instituições que um utente frequentou.

```
| ?- insti_utente(1,R).
R = ['Hospital de Santo Antonio','Hospital de Santa Maria'] ?
```

Figura 34 - Resultado de 'insti_utente(1, R)'.

Para determinar todos os prestadores a que utente recorreu é necessário primeiro descobrir todos os identificadores de prestadores que lhe prestaram serviço, para isso, procura-se nos cuidados os que apresentam o identificador do utente dado e depois com o identificador do prestador obtido retorna-se as suas informações.

```
% Extensao do predicado prest_utente: Utente,R -> {V,F}
%pega num utente e retorna todos os prestadores que cuidaram dele
prest_utente(IDU,R) :- solucoes(prestador(IDP,N,E,I),(cuidado(_,IDU,IDP,_,_),prestador(IDP,N,E,I)),L),
                       removeDups(L,R).
```

Figura 37 - Retorna todos os prestadores de um utente

```
| ?- prest_utente(1,X).
X = [prestador(6,'Vitor','Pediatria','Hospital de Santo Antonio'),prestador(4,'
Filipe','Psiquiatria','Hospital de Santa Maria')] ? ;
```

Figura 36 - Resultado do prest_utente(1, X)

3.10 Calcular o custo total dos cuidados de saúde por utente/especialidade/prestador/datas

Para calcular o custo total que um utente gastou é necessário procurar todos os cuidados em que esse utente aparece e guardar os seus respetivos custos numa tabela. Depois, com ajuda da função auxiliar 'somaL', que soma todos os elementos de uma lista, é só somar a lista de custos obtida e o resultado da função vai irá o resultado pretendido.

```
% Extensao do predicado custo_utente : Utente,R -> {V,F}
% Calcular o custo total dos cuidados de saúde de um utente
custo_utente(Utente,R) :- solucoes(Custo,cuidado(_,Utente,_,_,Custo),L),
                          somaL(L,R).
```

Figura 39 - Determina o custo total de um utente

```
| ?- listing(cuidado).
cuidado('01-01-2018', 3, 1, 'Dor de barriga', 15).
cuidado('04-04-2018', 7, 2, 'Braco partido', 20).
cuidado('23-01-2018', 4, 2, 'Perna partida', 30).
cuidado('01-05-2018', 2, 3, 'Reacao alergica', 5).
cuidado('03-03-2018', 12, 8, 'Febre', 4).
cuidado('31-12-2018', 10, 5, 'Febre', 5).
cuidado('30-03-2018', 7, 9, 'Analises', 5).
cuidado('31-12-2018', 7, 6, 'Urgencia', 200).
cuidado('31-12-2018', 1, 6, 'Braco partido', 1).
cuidado('07-08-2018', 11, 9, 'Reacao alergica', 5)
cuidado('01-04-2018', 1, 4, 'Dor de cabeca', 30).

yes
| ?- custo_utente(7,X).
X = 225 ?
```

Figura 38 - Resutado do custo_utente(7, X)

Para calcular o custo total de uma especialidade especifica é preciso ligar os prestadores e os cuidados, pois é neles que as informações estão contidas. Por isso, é necessário procurar pelos prestadores que apresentam a especialidade em causa e depois retornar os custos de todos os cuidados em que esses prestadores aparecem. Por fim, é só usar a função auxiliar 'somaL' para somar todos os valores da lista de custos e obtém-se o valor pretendido.

```
% Extensao do predicado custo_espe : E,R -> {V,F}
% Calcular o custo total dos cuidados de saúde de uma especialidade
custo_espe(E,R) :- solucoes(Custo,(cuidado(_,_,IDP,_,Custo),prestador(IDP,_,E,_)),L),
                   somaL(L,R).
```

Figura 41 - Obtém o custo total de uma certa especialidade.

```
| ?- custo_espe('Ortopedia',R).
R = 19 ?
```

Figura 40 - Resultado obtido

Por último também se calculou os custos totais para um prestador e para uma determinada data.

```
% Extensao do predicado custo_prestador: Prestador,R -> {V,F}
% Calcular o custo total dos cuidados de saúde de um prestador
custo_prestador(Prestador,R) :- solucoes(Custo,cuidado(_,_,Prestador,_,Custo),L),
                                somaL(L,R).

% Extensao do predicado custo_data: Data,R -> {V,F}
%Calcular o custo total dos cuidados de saúde por data
custo_data(Data,R) :- solucoes(Custo,cuidado(Data,_,_,_,Custo),L),
                      somaL(L,R).
```

Figura 43 – Predicados que calculam o custo total para um prestador e para uma data.

```
| ?- custo_prestador(6,R).
R = 201 ?
yes
| ?- custo_data('31-12-2018',X).
X = 206 ?
```

Figura 42 - Resultados obtidos.

4. EXTRAS

Para além dos exercícios propostos no enunciado ainda foram feitos alguns adicionalmente. Como por exemplo, o predicado capaz de calcular o número de utentes do sexo feminino e do sexo masculino. Para isso, procura todos os utentes masculinos ou femininos e depois é só retornar o tamanho dessa lista.

```
% Extensao do predicado num_ut_fem:: R -> {V,F}
% Retorna o número de utentes do sexo feminino
num_ut_fem(R) :- solucoes(IDU,utente(IDU,N,I,M,'Feminino'),L),
                 comprimento(L,R).

% Extensao do predicado num_ut_mas:: R -> {V,F}
% Retorna o número de utentes do sexo masculino
num_ut_mas(R) :-  solucoes(IDU,utente(IDU,N,I,M,'Masculino'),L),
                  comprimento(L,R).
```

Figura 45 - Predicados que calculam o número de utentes do sexo feminino e masculino.

```
| ?- num_ut_mas(R).
R = 7 ? ;
no
| ?- num_ut_fem(R).
R = 5 ? ;
```

Figura 44 - Resultado dos predicados.

Também se criou um predicado que descobre todos os cuidados dos utentes do sexo masculino. Para isso, procura os identificadores dos utentes do sexo masculino e depois procura nos cuidados todos aqueles que tem este identificador retornando esses.

```
% Extensao do predicado utentes_masc : R -> {V,F}
%cuidados de todos os utentes do sexo masculino
utentes_masc(R) :- solucoes(cuidados(D,IDU,IDP,Desc,C),(utente(IDU,_,_,_,'Masculino'),cuidado(D,IDU,IDP,Desc,C)),R).
```

Figura 47 - Cuidados de todos os utentes do sexo masculino.

```
| ?- utentes_masc(R).
R = [cuidados('31-12-2018',1,6,'Braco partido',1),cuidados('01-04-2018',1,4,'Do
r de cabeca',30),cuidados('01-05-2018',2,3,'Reacao alergica',5),cuidados('01-01
-2018',3,1,'Dor de barriga',15),cuidados('23-01-2018',4,2,'Perna partida',30),c
uidados('31-12-2018',10,5,'Febre',5),cuidados('03-03-2018',12,8,'Febre',4)] ?
```

Figura 46 - Resultado da 'utentes_masc'.

Visto que, nenhum dos exercícios propostos misturava todos os conhecimentos, decidiu-se fazer uma função que retorna os cuidados de todos os utentes de uma certa idade e que foram tratados por prestadores de uma instituição especifica. Para isso, procura-se todos os utentes com a idade dada como argumento e procura-se também todos os prestadores da instituição, dada também como

argumento da função, e depois é só preciso retornar todos os cuidados em que estes dois identificadores, tanto do utente como do prestador, estejam presentes.

```
% Extensao do predicado utenteInstAnos : N,Inst,R -> {V,F}
% Retorna os cuidados dos utentes de uma certa idade e com os prestadores de uma certa instituicao
utenteInstNAnos(N,Inst,L) :-
    solucoes(cuidado(D,IdU,IdP,Desc,C),(utente(IdU,_,N,_,_),prestador(IdP,_,_,Inst),cuidado(D,IdU,IdP,Desc,C)),L).
```

Figura 49 - Descobre todos os cuidados dos utentes de uma certa idade e dos prestadores de uma certa instituição.

```
| ?- utenteInstNAnos(20,'Hospital de Braga',R).
R = [cuidado('01-01-2018',3,1,'Dor de barriga',15),cuidado('04-04-2018',7,2,'Br
aco partido',20),cuidado('30-03-2018',7,9,'Analises',5)] ?
```

Figura 48 - Resultado da 'utenteInstNanos(20, Hospital de Braga', R)'.

Por último, criou-se uma função capaz de descobrir o top 5 dos utentes que gastaram mais com cuidados de saúde. Para isso descobria-se a lista de custos totais ordenados por utentes, isto com ajuda da função 'custo_utentes', depois ordenou-se crescentemente com a função 'sort', reverteu-se essa lista e por fim limitou-se a lista a 5 elementos. Retornado assim os cinco utentes que gastaram mais.

```
% Extensao do predicado custo_utentes: lista,R -> {V,F}
%retorna a lista de custos totais por utente
custo_utentes([],[]).
custo_utentes([IDU|T],R) :- solucoes(Custo,cuidado(_,IDU,_,_,Custo),L1),
                            somaL(L1,X),
                            custo_utentes(T,L2),
                            concat1([X],L2,R).

% Extensao do predicado top5Custo : R -> {V,F}
%top 5 dos utentes que gastam mais
top5Custo(R) :- solucoes(IDU,utente(IDU,_,_,_,_),L1),
                custo_utentes(L1,L2),
                concatPair(L2,L1,L3),
                sort(L3,L4),
                reverse(L4,L5),
                limite(L5,5,R).
```

Figura 51 - Função que retorna os 5 utentes que gastaram mais.

```
| ?- top5Custo(R).
R = [(225,7),(31,1),(30,4),(15,3),(5,11)] ? ;
no
```

Figura 50 - Resultado da top5Custo(R).

5. FUNÇÕES AUXILIARES

Neste capítulo mostramos todas as funções auxiliares que foram usadas para a resolução de alguns algoritmos:

```
% Extensao do meta-predicado nao: Questao -> {V,F}
nao(Questao) :-
    Questao, !, fail.
nao(Questao).

comprimento(S,N) :- length(S,N).

% Extensão do predicado solucoes: X,Y,Z -> {V,F}
solucoes(X,Y,Z) :-
    findall(X,Y,Z).

%verifica se um elemento pertence a uma lista
pertence(X,[X|_]).
pertence(X,[H|T]) :- X \= H , pertence(X,T).

%verifica se um elemento não pertence a uma lista
n_pertence(X,L):- nao(pertence(X,L)).

%remove os valores duplicados de uma lista
removeDups([],[]).
removeDups([X|L],[H|R]) :- pertence(X,L), removeDups(L,[H|R]).
removeDups([X|L],[X|R]) :- n_pertence(X,L),removeDups(L,R).

% Extensao do predicado concat1(L1,L2,L3)->{V,F}
concat1([],L2,L2).
concat1([X|L1],L2,[X|L3]) :- concat1(L1,L2,L3).

% Extensao do predicado concatPair(L1,L2,L3)->{V,F}
concatPair([],[],[]).
concatPair([X|L1],[Y|L2],[(X,Y)|R]) :- concatPair(L1,L2,R).

%soma todos os valores de uma lista
somaL([],0).
somaL([H|T],R) :- somaL(T,R1), R is R1+H.


% Extensao do predicado ordenarPorIdade: L,Resultado -> {V,F}
ordenarPorIdade([X],[X]).
ordenarPorIdade([X|Y],T):-
    ordenarPorIdade(Y,R),insertPorIdade(X,R,T).

% Extensao do predicado insertPorIdade: X,L,Resultado  -> {V,F}
insertPorIdade((X,Y,Z,W), [], [(X,Y,Z,W)]).
insertPorIdade((X1,Y1,Z1,W1), [(X2,Y2,Z2,W2)|T], [(X1,Y1,Z1,W1)|[(X2,Y2,Z2,W2)|T]]) :- Z1=<Z2.
insertPorIdade((X1,Y1,Z1,W1), [(X2,Y2,Z2,W2)|T], [(X2,Y2,Z2,W2)|R]) :- Z1>Z2,
    insertPorIdade((X1,Y1,Z1,W1),T,R).

% Extensao do predicado reverse(L1,L2)->{V,F}
reverse([],[]).
reverse([H|T],L) :- reverse(T,P), concat1(P,[H],L).

% Extensao do predicado limite L,N,L ->{V,F}
limite(X,0,[]).
limite([X|T],N,[X|R]) :- N1 is N - 1, limite(T,N1,R).
```

Figura 52 - Funções auxiliares usadas.

6. CONCLUSÕES E SUGESTÕES

Após concluir todos os exercícios propostos, podemos passar a uma análise final do trabalho realizado.

Podemos observar positivamente a realização deste trabalho, uma vez que concluímos todos os exercícios propostos corretamente, permitindo ainda solidificar conhecimentos aprendidos nas aulas e também adquirir novos conceitos acerca da programação lógica com o PROLOG.

Num projeto deste tipo, há quase sempre melhorias que podem ser implementadas, para permitir uma melhor manipulação do conhecimento guardado. Esse é sem dúvida um trabalho a desenvolver futuramente, que seria capaz de melhorar este sistema.