Universidade do Minho
Escola de Engenharia - Departamento de Informática

Relatório do Trabalho Prático - 1ª Fase

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

A81451 – Alexandre Rzepecki Rodrigues

A82145 - Filipa Correia Parente

A81896 – Nuno Afonso Gonçalves Solha Moreira Valente

A81403 – Pedro Henrique de Passos Ferreira

Braga, março 2019

Resumo

O seguinte relatório visa elucidar os procedimentos efetuados para construir um sistema de representação de conhecimento e raciocínio, proposto nesta primeira fase.

Em primeiro lugar, começa-se por apresentar uma pequena fundamentação teórica com os conceitos na qual nos baseamos para a realização das funcionalidades propostas.

Seguidamente, será apresentada o modo de resolução de cada funcionalidade pedida, com recurso a imagens ilustrativas dos predicados utilizados.

Finalmente, na secção Conclusões e Sugestões, será feita uma pequena análise do trabalho realizado, num computo geral, bem como as dificuldades sentidas durante a sua realização.

É de referir também que na secção Anexos se encontram todos os predicados auxiliares utilizados para a realização das funcionalidades bem como a exemplificação dos resultados no interpretador.

Índice

Introdução **6**

Preliminares **6**

Descrição do Trabalho e Análise de Resultados **7**

Fundamentação teórica 7

Inserção de Conhecimento 9

Resolução das funcionalidades propostas 10

Funcionalidade extra 19

Conclusões e Sugestões **20**

Referências **21**

Anexos **22**

Predicados auxiliares utilizados 22

Exemplificação dos resultados 24

Índice de figuras

Figura 1 - Exemplo de um invariante 8
Figura 2 - Inserção do conhecimento relativo aos utentes 9
Figura 3 - Inserção do conhecimento relativo aos serviços 9
Figura 4 - Inserção do conhecimento relativo às consultas 9
Figura 5 - Invariantes referenciais associados à inserção 10
Figura 6 - Invariantes referenciais relativo ao predicado utente 11
Figura 7 - Invariante referencial relativo ao predicado consulta 11
Figura 8 - Invariante referencial relativo ao predicado servico 11
Figura 9 - Invariante referencial relativo ao predicado consulta 12
Figura 10 - Invariante referencial relativo ao predicado servico 12
Figura 11 - Invariante referencial relativo ao predicado servico 12
Figura 12 - Invariantes estruturais associados à remoção de conhecimento 13
Figura 13 - Invariante referencial relativo à remoção do utente 14
Figura 14 - Invariante referencial relativo à remoção do serviço 14
Figura 15 - Extensão do predicado involucao 14
Figura 16 - Extensão do predicado prestaservico 15
Figura 17 - Extensão do predicado listUtentes 15
Figura 18 - Extensão do predicado listServicos 16
Figura 19 - Extensão do predicado listConsultas 16
Figura 20 - Extensão do predicado servicosPrestados 16
Figura 21 - Extensão do predicado identificaUtentes 17
Figura 22 - Extensão do predicado servicosRealizados 17
Figura 23 - Extensão do predicado custoSaude 18
Figura 24 - Extensão do listInstituicoesCidade 19
Figura 25 - predicados auxiliares utilizados para definição dos invariantes e dos predicados "evolucao" e "involucao" 22
Figura 26 - predicados auxiliares utilizados para definição dos predicados "evolucao" e "involucao" 22
Figura 27 - predicados auxiliares "pertence", "elimina" e "remove_duplicates" 23
Figura 28 - predicados auxiliares "concat" e "servicos" 23
Figura 29 - predicado auxiliar "utentes" 23
Figura 30 - Inserção de consultas 24
Figura 31 - Inserção de serviços 24
Figura 32 - Inserção de utentes 25
Figura 33 – Remoção 25
Figura 34 - Instituições prestadoras de serviços 26
Figura 35 - Listar consultas dado um argumento 26
Figura 36 - Listar serviços dado um argumento 26
Figura 37 - Listar utentes dado um argumento 26
Figura 38 - Identificar serviços dado um argumento 27
Figura 39 - Identificar utentes dado uma instituição 27
Figura 40 - Identificar utentes dado um serviço 27
Figura 41 - Identificar serviços dado uma instituição 27
Figura 42 - Identificar serviços dado um utente 27
Figura 43 - Identificar serviços dado uma cidade 27
Figura 44 - Calcular o custo total dado uma instituição 28

Figura 45 - Calcular o custo total dado uma data 28
Figura 46 - Calcular o custo total dado um utente 28
Figura 47 - Calcular o custo total dado um serviço 28
Figura 48 - determinar instituições existentes numa dada cidade 28

Introdução

Com este trabalho pretende-se consolidar conhecimento da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio. O trabalho consiste na utilização da linguagem de programação em lógica PROLOG para representar o conhecimento e raciocínio na área de cuidados de saúde.

Obtendo o conhecimento através do enunciado onde começamos com os predicados: utente, serviço e consulta. Podemos depois aumentar a nossa representação de conhecimento de acordo com o objetivo do trabalho.

Preliminares

Para a realização do trabalho proposto foi necessário conhecimento da linguagem de programação em lógica PROLOG. Tendo esta sido aprendida pelos elementos que compõem o grupo no decorrer das aulas de Sistemas de Representação de Conhecimento e Raciocínio.

Analisando o enunciado meticulosamente de modo a ter um entendimento profundo do problema proposto, obtemos a compreensão necessária para conseguirmos detetar o melhor método de resolução.

Após identificarmos uma solução adequada para o problema este foi resolvido através da linguagem de programação em lógica PROLOG.

Descrição do trabalho e Análise de resultados

Antes de entrar diretamente na resolução das alíneas propostas irá ser feita uma pequena fundamentação teórica usada para a elaboração deste trabalho prático.

Fundamentação teórica

Para a realização deste trabalho recorremos aos seguintes conceitos associados à linguagem de programação PROLOG:

Fato:

Um fato, em PROLOG, é um predicado verdadeiro que é inserido na base de conhecimento.

Regra:

Uma regra, ou cláusula, é também um predicado que faz uso de variáveis para provar a veracidade de um fato. São baseadas nas cláusulas de horn.

Pressuposto do mundo fechado:

De acordo com este princípio, todo o predicado numa consulta é avaliado como falso no caso de não estar presente nenhuma regra positiva ou fato que dê suporte ao termo proposto. Ou seja, assume-se que tudo o que é importante saber está na base de conhecimento.

Inserção e remoção da base de conhecimento

A inserção e remoção da base de conhecimento, em PROLOG, é feita recorrendo ao uso dos predicados assert e retract. Contudo, estes não garantem a consistência e a preservação de certas restrições que se pretenda fazer (ex: não fazer a inserção de fatos repetidos ou remover fatos que não existem na base de conhecimento).
No intuito de conferir essa consistência, recorreu-se ao uso de invariantes.

Invariante

Um invariante é algo que não se altera ao aplicar-se um conjunto de transformações. Mais precisamente, é uma condição que se deve manter inalterada ao longo das transformações (operações de inserção e remoção) existentes na base de conhecimento.

Estrutura de um invariante

Um invariante, em PROLOG, é estruturado da seguinte forma:

```
+filho( F,P ) :: (solucoes( (F,P),(filho( F,P )),S ),
                  comprimento( S,N ),
                                N == 1
                  ).
```

Figura 1 - Exemplo de um invariante

Em que:

- + -> símbolo que representa a inserção de conhecimento (o sinal "-" é utilizado para a remoção);
- filho(F,P) -> predicado a ser aplicado o invariante (Termo);
- A vermelho encontra-se o invariante a ser respeitado a cada transformação.

Dentro dos invariantes existe ainda a distinção entre invariante estrutural e referencial. O invariante estrutural, é um invariante que condiciona a base de conhecimento a nível estrutural. O invariante referencial, condiciona a base de conhecimento tendo como base o significado do predicado.

Após a retenção destes conceitos, partimos para a inserção de conhecimento útil para a resolução das funcionalidades propostas.

Inserção de conhecimento

De forma a inserir o conhecimento necessário, decidimos construir os predicados que a seguir se apresentam:

Utentes

```
%   utente: #IdUt, Nome, Idade, Cidade ~ {V,F}
utente(0,joao,19,porto).
utente(1,maria,45,viana).
utente(2,pedro,56,lisboa).
utente(3,jose,67,covilha).
utente(4,albertina,23,viseu).
utente(5,joao,21,viana).
```

Figura 2 - Inserção do conhecimento relativo aos utentes

Serviços

```
%   servico: #IdServ, Descricao, Instituicao, Cidade ~ {V,F}
servico(0, 'consulta medicina interna', 'Hospital Particular de Viana', viana).
servico(1, 'consulta medicina dentaria', 'Hospital Particular de Braga', braga).
servico(2, 'consulta dermatologia', 'Hospital Lusiadas', lisboa).
servico(3, 'consulta oftalmologia', 'Hospital de Santo Antonio', porto).
servico(4, 'Consulta de Otorrinolaringologia', 'Hospital Particular de Viana', viana).
servico(5, 'Consulta de Otorrinolaringologia', 'Centro Hospitalar de Viana', viana).
```

Figura 3 - Inserção do conhecimento relativo aos serviços

Consultas

```
%   consulta: #IdCons, Data, #IdUt, #IdServ, Custo ~ {V,F}
consulta(0, 2012, 0, 0, 10).
consulta(1, 2014, 0, 0, 12).
consulta(2, 2012, 2, 2, 13).
consulta(3, 2019, 0, 3, 20).
consulta(4, 2019, 4, 0, 12).

consulta(5, 2019, 1, 1, 12).
consulta(6, 2019, 1, 1, 12).
consulta(7, 2019, 2, 2, 12).
consulta(8, 2019, 3, 3, 12).
```

Figura 4 - Inserção do conhecimento relativo às consultas

Como já referido anteriormente, esta inserção de conhecimento, já caraterizado, como referido no enunciado, foi útil para a resolução das funcionalidades propostas.

Resolução das funcionalidades propostas

1. Registar utentes, serviços e consultas

Para a corresponder a esta funcionalidade recorreu-se aos seguintes invariantes:

Invariantes Estruturais

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Invariante Estrutural: nao permitir a adicao de utentes com o mesmo id na BD

+utente(Id,_,_,_) :: (solucoes(Id,(utente(Id,_,_,_)),S),
                              comprimento( S,N ),
                              N==1).

%------------------------------ - - - - - - - - - - - - - - - - - -
% Invariante Estrutural: nao permitir a adicao de consultas com o mesmo id na BD

+consulta( Id,_,_,_,_ ) :: (solucoes(Id,(consulta( Id,_,_,_,_ )),S ),
                                        comprimento( S,N ),
                                        N == 1).

%------------------------------ - - - - - - - - - - - - - - - - - -
% Invariante Estrutural: nao permitir a adicao de servicos com o mesmo id na BD

+servico( Id,_,_,_ ) :: (solucoes(I,(servico( Id,_,_,_ )),S ),
                                      comprimento( S,N ),
                                      N == 1
                                      ).
```

Figura 5 - Invariantes referenciais associados à inserção

Os invariantes estruturais apresentados na figura 5 garantem que não é inserido nenhum utente, serviço ou consulta com o mesmo ID. O grupo decidiu que o ID é o argumento de distinção entre serviços, consultas e utentes diferentes. Assim acrescenta-se a possibilidade de acrescentar utentes com o mesmo, nome, idade e a morar na mesma cidade, uma vez que podem ser entidades diferentes.

Invariantes Referenciais

```
%------------------------------ - - - - - - - - - - - -
% Invariante Referencial: os ids e as idades sao inteiros

+utente(Id,_,_,_) :: (
    integer(Id)
).

+utente(_,_,Id,_) :: (
    integer(Id)
).
```

Figura 6 - Invariantes referenciais relativo ao predicado utente

```
%------------------------------ - - - - - - - - - - - - - - - -
% Invariante Referencial: os ids sao inteiros

+consulta( Id,_,_,_,_ ) :: (
    integer(Id)
).
```

Figura 7 - Invariante referencial relativo ao predicado consulta

```
%------------------------------ - - - - - - - - - - - - - - - -
% Invariante Referencial: os ids sao inteiros

+servico(Id,_,_,_) :: (
    integer(Id)
    ).
```

Figura 8 - Invariante referencial relativo ao predicado servico

Os invariantes referenciais apresentados nas figuras 6, 7 e 8, garantem que os ID's de cada um são inteiros, não aceitando argumentos de outro tipo. Na figura 6 ainda existe a condição de que a idade é um inteiro.

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Invariante Referencial: nao admitir consultas que nao tenham um servico e um utente incorporado na BD

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

Figura 9 - Invariante referencial relativo ao predicado consulta

O invariante apresentado na figura 9, acrescenta a condição de que nenhuma consulta é adicionada sem a existência de um serviço, e de um utente previamente associado na base de conhecimento.

Finalmente, na figura 10, apresenta-se mais um invariante referencial relativo ao serviço, que apenas permite acrescentar serviços iguais em instituições diferentes.

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Invariante Referencial: nao admitir varios servicos iguais no mesmo hospital

+servico(_,D,Instituicao,_) :: (solucoes( (D,Instituicao),(servico(_,D,Instituicao,_)),S ),
                                comprimento( S,N ),
                                N==1
                                ).
```

Figura 10 - Invariante referencial relativo ao predicado servico

Tendo como base estes invariantes apresentados, podemos constatar o seguinte predicado de inserção de conhecimento:

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensao do predicado que permite a evolucao do conhecimento

evolucao( Termo ) :-
        solucoes(Invariante,+Termo::Invariante,Lista),
        insercao(Termo),
        teste(Lista).
```

Figura 11 - Invariante referencial relativo ao predicado servico

É de realçar que os predicados auxiliares utilizados na definição deste predicado encontram-se nos Anexos, para posterior consulta.

2. Remover utentes, serviços e consultas

Para resolver esta alínea, também recorremos ao uso de invariantes, como se pode apresentar em seguida:

Invariantes Estruturais

```
%------------------------------ - - - - - - - - - - -  -  -  -  -  -
% Invariante Estrutural: nao remover consultas que nao estejam na BD

-consulta(Id,_,_,_,_) :: (solucoes( (Id,_,_,_,_),(consulta(Id,_,_,_,_)),S ),
                  comprimento( S,N ),
                  N == 0).

%------------------------------ - - - - - - - - - - -  -  -  -  -  -
% Invariante Estrutural: nao remover utentes que nao estejam na BD

-utente( Id,_,_,_ ) :: (
                  solucoes((Id,_,_,_),(utente(Id,_,_,_)),S),
                  comprimento( S,N ),
                  N == 0
                  ).

%------------------------------ - - - - - - - - - - -  -  -  -  -  -
% Invariante Estrutural: nao remover servicos que nao estejam na BD

-servico( Id,_,_,_ ) :: (solucoes( (Id,_,_,_),(servico( Id,_,_,_ )),S ),
                  comprimento( S,N ),
                  N == 0).
```

Figura 12 - Invariantes estruturais associados à remoção de conhecimento

Tal como se pode observar na figura 11, não pode ser removido nenhum facto que não esteja presente na base de conhecimento.

Invariantes Referenciais

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Invariante Referencial: nao remover utentes que tenham consultas com o id do
% utente associado

-utente( Id,_,_,_ ) :: (
                solucoes( Id,(consulta(_,_,Id,_,_)),S ),
                comprimento( S,N ),
                N > 0).
```

Figura 13 - Invariante referencial relativo à remoção do utente

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Invariante Referencial: nao remover servicos que tenham consultas com o id do
% servico associado

-servico( Id,_,_,_ ) :: (
                solucoes( (Id),(consulta(_,_,_,Id,_)),S ),
                comprimento( S,N ),
                N > 0).
```

Figura 14 - Invariante referencial relativo à remoção do serviço

Tendo em conta estes invariantes, construímos assim, a seguinte extensão do predicado chamado involução:

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensao do predicado que permite a involucao do conhecimento

involucao( Termo ) :-
        Termo,
        solucoes(Invariante,-Termo::Invariante,Lista),
        remocao(Termo),
        teste(Lista).
```

Figura 15 - Extensão do predicado involucao

3. Identificar as instituições prestadoras de serviços

Para a realização desta alínea recorremos ao predicado soluções, que determina a lista de soluções que seguem um determinado formato numa dada relação, ambos dados como argumentos do mesmo. Nota: Nos anexos encontram-se todas as definições das extensões dos predicados auxiliares à resolução.

Assim:

```
prestaservico(H) :- solucoes(Instituicao,servico(_,_,Instituicao,_),X),
                    remove_duplicates(X,H).
```

Figura 16 - Extensão do predicado prestaservico

Como, neste caso estamos à procura das instituições que prestam serviços, é necessário inserir na lista de soluções, apenas estas. O predicado remove_duplicates remove todas as ocorrências repetidas da mesma instituição na lista de soluções.

4. Identificar utentes/serviços/consultas por critérios de seleção

Antes de apresentar a resolução, é importante referir que os critérios utilizados para a apresentação das soluções foram cada um dos parâmetros de cada um, como por exemplo o id, nome, idade e cidade no caso do utente, e o id, a descrição, a instituição e a cidade, no caso do serviço.

Tendo em conta estes critérios podemos apresentar a seguinte resolução:

```
%% para os utentes
listUtentes(id,Id,LS) :- solucoes((Id,Nome,idade,Cidade),(utente(Id,Nome,idade,Cidade)),LS).
listUtentes(nome,Nome,LS) :- solucoes((Id,Nome,idade,Cidade),(utente(Id,Nome,idade,Cidade)),LS).
listUtentes(idade,Idade,LS) :- solucoes((Id,Nome,idade,Cidade),(utente(Id,Nome,idade,Cidade)),LS).
listUtentes(cidade,Cidade,LS) :- solucoes((Id,Nome,idade,Cidade),(utente(Id,Nome,idade,Cidade)),LS).
```

Figura 17 - Extensão do predicado listUtentes

```
%% para os servicos
listServicos(id,Id,LS) :- solucoes((Id,Descricao,Instituicao,Cidade),servico(Id,Descricao,Instituicao,Cidade),LS).
listServicos(descricao,Descricao,LS) :- solucoes((Id,Descricao,Instituicao,Cidade),servico(Id,Descricao,Instituicao,Cidade),LS).
listServicos(instituicao,Instituicao,LS) :- solucoes((Id,Descricao,Instituicao,Cidade),servico(Id,Descricao,Instituicao,Cidade),LS).
listServicos(cidade,Cidade,LS) :- solucoes((Id,Descricao,Instituicao,Cidade),servico(Id,Descricao,Instituicao,Cidade),LS).
```

Figura 18 - Extensão do predicado listServicos

```
%% para as consultas
listConsultas(id,IdCons,LS) :- solucoes((IdCons,Data,IdUt,IdServ,Custo),(consulta(IdCons,Data,IdUt,IdServ,Custo)),LS).
listConsultas(data,Data,LS) :- solucoes((IdCons,Data,IdUt,IdServ,Custo),(consulta(IdCons,Data,IdUt,IdServ,Custo)),LS).
listConsultas(utente,IdUt,LS) :- solucoes((IdCons,Data,IdUt,IdServ,Custo),consulta(IdCons,Data,IdUt,IdServ,Custo),LS).
listConsultas(servico,IdServ,LS) :- solucoes((IdCons,Data,IdUt,IdServ,Custo),consulta(IdCons,Data,IdUt,IdServ,Custo),LS).
listConsultas(custo,Custo,LS) :- solucoes((IdCons,Data,IdUt,IdServ,Custo),consulta(IdCons,Data,IdUt,IdServ,Custo),LS).
```

Figura 19 - Extensão do predicado listConsultas

5. Identificar serviços prestados por instituição/cidade/datas/custo

```
servicosPrestados(custo,Custo,LS) :- solucoes(IdServ,consulta(IdCons,Data,IdUt,IdServ,Custo),X),
                                     servicos(X,XS),
                                     remove_duplicates(XS,LS).

servicosPrestados(instituicao,Instituicao,LS) :- listServicos(instituicao,Instituicao,LS).

servicosPrestados(cidade,Cidade,LS) :- listServicos(cidade,Cidade,LS).

servicosPrestados(data,Data,LS) :- solucoes(IdServ,consulta(IdCons,Data,IdUt,IdServ,Custo),X),
                                   servicos(X,XS),
                                   remove_duplicates(XS,LS).
```

Figura 20 - Extensão do predicado servicosPrestados

Para identificar os serviços prestados recorremos aos predicados realizados no ponto 4 (“listServicos”), exceto para os custos e para as datas, uma vez que os mesmos se encontram nas consultas.

Neste caso, foi necessário,

- determinar o id do serviço correspondente à consulta com esse custo, ou data;
- inseri-lo numa lista de ids;
- usar um predicado que devolvesse a lista com os serviços correspondentes(”servicos(X,XS)”);
- remover os serviços que estavam repetidos, usando o “remove_duplicates(XS,LS)”.

6. Identificar os utentes de um serviço/instituição

```
identificautentes(servico,Ser,Y) :-
                        solucoes(Uti,consulta(ID,Data,Uti,Ser,Custo),X),
                        utentes(X,XS),
                        remove_duplicates(XS,Y).

identificautentes(instituicao,Ins,Y) :-
                                    solucoes(Uti,
                                        (consulta(IDconsulta,Data,Uti,Ser,Custo),
                                        servico(Ser,Desc,Ins,Cidade)),
                                        X),
                                    utentes(X,XS),
                                    remove_duplicates(XS,Y).
```

Figura 21 - Extensão do predicado identificaUtentes

À semelhança do realizado no ponto 5, para identificar tanto os utentes de um determinado serviço, como de uma instituição, fez-se o seguinte:

- Determinou-se a lista de ids de cada utente que usufruiu do serviço;
- A partir dessa lista de ids obtiveram-se a informação integral dos utentes respetivos, através do predicado (“utentes”);
- Retiraram-se os elementos repetidos.

Estes apenas diferem no formato do predicado “solucoes”.

7. Serviços realizados por utente/instituição/cidade

```
%------------------------------ - - - - - - - - - - - - - - - -
% VII - Servicos realizados por utente/instituicao/Cidade

servicosRealizados(utente,IdUt,Result) :- solucoes(IdServ,consulta(_,_,IdUt,IdServ,_),List),
                                          servicos(List,L1),
                                          remove_duplicates(L1,Result).
servicosRealizados(instituicao,Instituicao,Result) :- solucoes(IdServ,(consulta(_,_,_,IdServ,_),servico(IdServ,_,Instituicao,_)),List)
                                                      servicos(List,L1),
                                                      remove_duplicates(L1,Result).
servicosRealizados(cidade,Cidade,Result) :- solucoes(IdServ,(consulta(_,_,_,IdServ,_),servico(IdServ,_,_,Cidade)),List),
                                            servicos(List,L1),
                                            remove_duplicates(L1,Result).
```

Figura 22 - Extensão do predicado servicosRealizados

Para a realização deste requisito, recorremos ao registo das consultas existente, uma vez que só há garantia de o serviço ter sido prestado ao utente, ou na Instituição, ou na cidade referida, no caso de haver o registo de uma consulta com o id do utente associado.

Sendo assim apenas foi necessário, recolher os ids dos serviços associados, e inserir numa lista de serviços tal como efetuado nas alíneas anteriores.

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

```
custoSaude(utente,Utente,C) :- solucoes((Id,Data,Utente,Servico,Custo),
                               consulta(Id,Data,Utente,Servico,Custo),R),
                               somaCusto(R, C).

custoSaude(servico,Servico,C) :- solucoes((Id,Data,Utente,Servico,Custo),
                                 consulta(Id,Data,Utente,Servico,Custo),R),
                                 somaCusto(R, C).

custoSaude(instituicao,Instituicao,C) :- solucoes((Id,Data,Utente,Servico,Custo),
                                         consulta(Id,Data,Utente,Servico,Custo),R),
                                         somaCusto(R, C).

custoSaude(data,Data,C) :- solucoes((Id,Data,InstiUtentetuicao,Servico,Custo),
                           consulta(Id,Data,Utente,Servico,Custo),R),
                           somaCusto(R, C).
```

Figura 23 - Extensão do predicado custoSaude

Para responder a esta alínea, também se recorreu ao predicado “solucoes” para filtrar as consultas de determinada instituição, utente, serviço ou data.

Numa última fase, utilizou-se o predicado auxiliar “somaCusto” para calcular o somatório de todos os custos, utilizando a lista previamente filtrada.

Funcionalidade extra

1. Determinar as instituições existentes numa dada cidade

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% IX - Instituicoes de uma dada cidade

listInstituicoesCidade(Cidade,LS):-
                    solucoes(Instituicao,servico(_,_,Instituicao,Cidade),X),
                    remove_duplicates(X,LS).
```

Figura 24 - Extensão do listInstituicoesCidade

Com o intuito de ajudar o utente a escolher uma instituição mais próxima da sua área de residência, acrescentou-se esta funcionalidade de determinar as instituições de prestação de cuidados de saúde presentes numa dada cidade.

Para o efeito, criou-se um predicado que determina as instituições através do predicado "solucoes". Posteriormente, é-se retirado da lista de soluções os elementos repetidos.

Conclusões e sugestões

Com a realização deste trabalho, foi possível adquirir uma maior prática na utilização da linguagem de programação PROLOG e obter um maior conhecimento sobre os conceitos que esta apresenta. Isto surgiu através de um processo de melhoria do código face à análise das questões propostas.

Inicialmente cada elemento do grupo tinha as suas tarefas, mas, analisando melhor as interrogações requeridas, foi observado que seria possível facilitar as respostas criando predicados que fossem auxiliares aos que respondem às questões, já que parte dos seus procedimentos se repetem frequentemente em cada uma.

O predicado auxiliar “solucoes” foi utilizada em todos os pontos propostos, pois agrupa todos os resultados que se encontram de acordo com o argumento fornecido numa lista fácil de percorrer, o que ajudou bastante a completar as interrogações.

Apesar disso, houve uma dificuldade sentida pelo grupo de trabalho no que conta à compreensão do enunciado, especialmente na diferenciação entre algumas perguntas, que davam a entender que o que estava a ser pedido já tinha sido realizado numa questão anterior.

Finalmente, considerámos que os objetivos do trabalho foram concluídos resultando num sistema com capacidade de caracterizar um ambiente na área da prestação de cuidados de saúde.

Referências

[Analide, 2001] ANALIDE, Cesar, NOVAIS, Paulo, NEVES, José,

"Sugestões para a Elaboração de Relatórios", Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2001.

[Bratko, 2000] BRATKO, Ivan,

"PROLOG: Programming for Artificial Intelligence", 3rd Edition, Addison-Wesley Longman Publishing Co., Inc., 2000

Anexos

Predicados auxiliares utilizados

```
%------------------------------- - - - - - - - - - - - - - - - - - -
% Extensao do predicado comprimento: ListaSolucoes,Valor -> {V,F}

comprimento([],0).
comprimento([Head],1).
comprimento([Head|Tail],G):-
    comprimento(Tail,T),
    G is T+1.

%------------------------------- - - - - - - - - - - - - - - - - - -
% Extensao do predicado teste: Lista -> {V,F}

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

%------------------------------- - - - - - - - - - - - - - - - - - -
% Extensao do predicado insercao : Termo -> {V,F}

insercao( Termo ) :- assert( Termo ).

insercao( Termo ) :- retract( Termo ),!,fail.
```

Figura 25 - predicados auxiliares utilizados para definição dos invariantes e dos predicados "evolucao" e "involucao"

```
%------------------------------- - - - - - - - - - - - - - - - - - -
% Extensao do predicado remocao : Termo -> {V,F}

remocao( Termo ) :- retract( Termo ).

remocao( Termo ) :- assert( Termo ),!,fail.

%------------------------------- - - - - - - - - - - - - - - - - - -
% Extensao do predicado solucoes: Formato,Relacao,ListaSolucoes -> {V,F}

solucoes(F,R,LS) :-
            R,assert(tmp(F)),fail.

solucoes(F,R,LS) :-
            construir([],LS).

%------------------------------- - - - - - - - - - - - - - - - - - -
% Extensao do predicado construir: ListaSolucoes,Solucao -> {V,F}

construir(LS,S):-
        retract(tmp(X)),
        !,
        construir([X|LS],S).

construir(S,S).
```

Figura 26 - predicados auxiliares utilizados para definição dos predicados "evolucao" e "involucao"

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

elimina(_,[],[]).
elimina(X,[X|T],R):- elimina(X,T,R).
elimina(X,[H|T],[H|R]) :- elimina(X,T,R).

remove_duplicates([],[]).
remove_duplicates([H|T], [H|R]) :-
                                elimina(H,T,R1),
                                remove_duplicates(R1,R).
```

Figura 27 - predicados auxiliares "pertence", "elimina" e "remove_duplicates"

```
concat([],L,L).
concat([X|T1],L2,[X|T2]):-concat(T1,L2,T2).

servicos([],[]).
servicos([H],L):- listServicos(id,H,L).
servicos([H|T],LS) :- listServicos(id,H,X),
                      servicos(T,L1),
                      concat(X,L1,LS).
```

Figura 28 - predicados auxiliares "concat" e "servicos"

```
utentes([],[]).
utentes([H],L):- listUtentes(id,H,L).
utentes([H|T],LS) :- listUtentes(id,H,X),
                     utentes(T,L1),
                     concat(X,L1,LS).
```

Figura 29 - predicado auxiliar "utentes"

Exemplificação dos resultados

1.

```
| ?- listing(consulta).
consulta(0, 2012, 0, 0, 10).
consulta(1, 2014, 0, 0, 12).
consulta(2, 2012, 2, 2, 13).
consulta(3, 2019, 0, 3, 20).
consulta(4, 2019, 4, 0, 12).
consulta(5, 2019, 1, 1, 12).
consulta(6, 2019, 1, 1, 12).
consulta(7, 2019, 2, 2, 12).
consulta(8, 2019, 3, 3, 12).

yes
| ?- evolucao(consulta(9, 2019, 3, 10, 20)).
no
| ?- evolucao(consulta(9, 2019, 10, 2, 20)).
no
| ?- evolucao(consulta(9, 2019, 3, 2, 20)).
yes
| ?- listing(consulta).
consulta(0, 2012, 0, 0, 10).
consulta(1, 2014, 0, 0, 12).
consulta(2, 2012, 2, 2, 13).
consulta(3, 2019, 0, 3, 20).
consulta(4, 2019, 4, 0, 12).
consulta(5, 2019, 1, 1, 12).
consulta(6, 2019, 1, 1, 12).
consulta(7, 2019, 2, 2, 12).
consulta(8, 2019, 3, 3, 12).
consulta(9, 2019, 3, 2, 20).

yes
```

Figura 30 - Inserção de consultas

```
| ?- listing(servico).
servico(0, 'consulta medicina interna', 'Hospital Particular de Viana', viana).
servico(1, 'consulta medicina dentaria', 'Hospital Particular de Braga', braga).
servico(2, 'consulta dermatologia', 'Hospital Lusiadas', lisboa).
servico(3, 'consulta oftalmologia', 'Hospital de Santo Antonio', porto).
servico(4, 'Consulta de Otorrinolaringologia', 'Hospital Particular de Viana', viana).

yes
| ?- evolucao(servico(3, 'consulta dentista', 'Hospital Particular de Lisboa', lisboa)).
no
| ?- evolucao(servico(5, 'consulta dentista', 'Hospital Particular de Lisboa', lisboa)).
yes
| ?- evolucao(servico(5, 'consulta dentista', 'Hospital Particular de Lisboa', lisboa)).
no
```

Figura 31 - Inserção de serviços

```
| ?- listing(utente).
utente(0, joao, 19, porto).
utente(1, maria, 45, viana).
utente(2, pedro, 56, lisboa).
utente(3, jose, 67, covilha).
utente(4, albertina, 23, viseu).
utente(5, joao, 21, viana).

yes
| ?- evolucao(utente(4, antonio, 18, lisboa)).
no
| ?- evolucao(utente(6, antonio, naointeiro, lisboa)).
no
| ?- evolucao(utente(6, antonio, 18, lisboa)).
yes
| ?- evolucao(utente(6, antonio, 18, lisboa)).
no
| ?- listing(utente).
utente(0, joao, 19, porto).
utente(1, maria, 45, viana).
utente(2, pedro, 56, lisboa).
utente(3, jose, 67, covilha).
utente(4, albertina, 23, viseu).
utente(5, joao, 21, viana).
utente(6, antonio, 18, lisboa).

yes
```

Figura 32 - Inserção de utentes

2.

```
| ?- listing(utente).
utente(0, joao, 19, porto).
utente(1, maria, 45, viana).
utente(2, pedro, 56, lisboa).
utente(3, jose, 67, covilha).
utente(4, albertina, 23, viseu).
utente(5, joao, 21, viana).

yes
| ?- listing(consulta).
consulta(0, 2012, 0, 0, 10).
consulta(1, 2014, 0, 0, 12).
consulta(2, 2012, 2, 2, 13).
consulta(3, 2019, 0, 3, 20).
consulta(4, 2019, 4, 0, 12).
consulta(5, 2019, 1, 1, 12).
consulta(6, 2019, 1, 1, 12).
consulta(7, 2019, 2, 2, 12).
consulta(8, 2019, 3, 3, 12).

yes
| ?- %Nao da para retirar o utilizador 3 pois tem a consulta 8 associada a si
| ?- involucao(utente(3,_,_,_)).
no
| ?- %Remove-se a consulta 8
| ?- involucao(consulta(8,_,_,_,_)).
yes
| ?- %Remove-se o utilizador 3
| ?- involucao(utente(3,_,_,_)).
yes
| ?- listing(utente).
utente(0, joao, 19, porto).
utente(1, maria, 45, viana).
utente(2, pedro, 56, lisboa).
utente(4, albertina, 23, viseu).
utente(5, joao, 21, viana).

yes
| ?- listing(consulta).
consulta(0, 2012, 0, 0, 10).
consulta(1, 2014, 0, 0, 12).
consulta(2, 2012, 2, 2, 13).
consulta(3, 2019, 0, 3, 20).
consulta(4, 2019, 4, 0, 12).
consulta(5, 2019, 1, 1, 12).
consulta(6, 2019, 1, 1, 12).
consulta(7, 2019, 2, 2, 12).
```

Figura 33 – Remoção

3.

```
| ?- prestaservico(H).
H = ['Hospital Particular de Lisboa','Hospital Particular de Viana','Hospital de Santo Anton
io','Hospital Lusiadas','Hospital Particular de Braga'] ?
yes
```

Figura 34 - Instituições prestadoras de serviços

4.

```
consulta(0, 2012, 0, 0, 10).
consulta(1, 2014, 0, 0, 12).
consulta(2, 2012, 2, 2, 13).
consulta(3, 2019, 0, 3, 20).
consulta(4, 2019, 4, 0, 12).
consulta(5, 2019, 1, 1, 12).
consulta(6, 2019, 1, 1, 12).
consulta(7, 2019, 2, 2, 12).
consulta(8, 2019, 3, 3, 12).

yes
| ?- listConsultas(data, 2019, R).
R = [(8,2019,3,3,12),(7,2019,2,2,12),(6,2019,1,1,12),(5,2019,1,1,12),(4,2019,4,0,12),(3,2019
,0,3,20)] ?
yes
| ?- listConsultas(custo, 12, R).
R = [(8,2019,3,3,12),(7,2019,2,2,12),(6,2019,1,1,12),(5,2019,1,1,12),(4,2019,4,0,12),(1,2014
,0,0,12)] ?
yes
```

Figura 35 - Listar consultas dado um argumento

```
| ?- listing(servico).
servico(0, 'consulta medicina interna', 'Hospital Particular de Viana', viana).
servico(1, 'consulta medicina dentaria', 'Hospital Particular de Braga', braga).
servico(2, 'consulta dermatologia', 'Hospital Lusiadas', lisboa).
servico(3, 'consulta oftalmologia', 'Hospital de Santo Antonio', porto).
servico(4, 'Consulta de Otorrinolaringologia', 'Hospital Particular de Viana', viana).
servico(5, 'consulta dentista', 'Hospital Particular de Lisboa', lisboa).

yes
| ?- listServicos(cidade, viana, R).
R = [(4,'Consulta de Otorrinolaringologia','Hospital Particular de Viana',viana),(0,'consult
a medicina interna','Hospital Particular de Viana',viana)] ?
yes
| ?- listServicos(cidade, braga, R).
R = [(1,'consulta medicina dentaria','Hospital Particular de Braga',braga)] ?
yes
| ?- listServicos(instituicao, 'Hospital Particular de Viana', R).
R = [(4,'Consulta de Otorrinolaringologia','Hospital Particular de Viana',viana),(0,'consult
a medicina interna','Hospital Particular de Viana',viana)] ?
yes
```

Figura 36 - Listar serviços dado um argumento

```
| ?- listing(utente).
utente(0, joao, 19, porto).
utente(1, maria, 45, viana).
utente(2, pedro, 56, lisboa).
utente(3, jose, 67, covilha).
utente(4, albertina, 23, viseu).
utente(5, joao, 21, viana).
utente(6, antonio, 18, lisboa).

yes
| ?- listUtentes(id, 0, R).
R = [(0,joao,19,porto)] ?
yes
| ?- listUtentes(nome, joao, R).
R = [(5,joao,21,viana),(0,joao,19,porto)] ?
yes
| ?- listUtentes(idade, 20, R).
R = [] ?
yes
```

Figura 37 - Listar utentes dado um argumento

5.

```
| ?- listing(servico).
servico(0, 'consulta medicina interna', 'Hospital Particular de Viana', viana).
servico(1, 'consulta medicina dentaria', 'Hospital Particular de Braga', braga).
servico(2, 'consulta dermatologia', 'Hospital Lusiadas', lisboa).
servico(3, 'consulta oftalmologia', 'Hospital de Santo Antonio', porto).
servico(4, 'Consulta de Otorrinolaringologia', 'Hospital Particular de Viana', viana).

yes
| ?- listing(consulta).
consulta(0, 2012, 0, 0, 10).
consulta(1, 2014, 0, 0, 12).
consulta(2, 2012, 2, 2, 13).
consulta(3, 2019, 0, 3, 20).
consulta(4, 2019, 4, 0, 12).
consulta(5, 2019, 1, 1, 12).
consulta(6, 2019, 1, 1, 12).
consulta(7, 2019, 2, 2, 12).
consulta(8, 2019, 3, 3, 12).

yes
| ?- servicosPrestados(custo, 12, R).
R = [(3,'consulta oftalmologia','Hospital de Santo Antonio',porto),(2,'consulta dermatologia
','Hospital Lusiadas',lisboa),(1,'consulta medicina dentaria','Hospital Particular de Braga'
,braga),(0,'consulta medicina interna','Hospital Particular de Viana',viana)] ?
yes
| ?- servicosPrestados(cidade, viana, R).
R = [(4,'Consulta de Otorrinolaringologia','Hospital Particular de Viana',viana),(0,'consult
a medicina interna','Hospital Particular de Viana',viana)] ?
yes
| ?- servicosPrestados(cidade, braga, R).
R = [(1,'consulta medicina dentaria','Hospital Particular de Braga',braga)] ?
yes
| ?- servicosPrestados(data, 2019, R).
R = [(3,'consulta oftalmologia','Hospital de Santo Antonio',porto),(2,'consulta dermatologia
','Hospital Lusiadas',lisboa),(1,'consulta medicina dentaria','Hospital Particular de Braga'
,braga),(0,'consulta medicina interna','Hospital Particular de Viana',viana)] ?
yes
| ?- servicosPrestados(data, 2012, R).
R = [(2,'consulta dermatologia','Hospital Lusiadas',lisboa),(0,'consulta medicina interna','
Hospital Particular de Viana',viana)] ?
yes
```

Figura 38 - Identificar serviços dado um argumento

6.

```
| ?- identificautentes(instituicao,'Hospital Particular de Viana',R).
R = [(4,albertina,23,viseu),(0,joao,19,porto)] ?
```

Figura 39 - Identificar utentes dado uma instituição

```
| ?- identificautentes(servico,0,Y).
Y = [(4,albertina,23,viseu),(0,joao,19,porto)] ?
```

Figura 40 - Identificar utentes dado um serviço

7.

```
| ?- servicosRealizados(instituicao,'Hospital Particular de Viana',R).
R = [0] ?
```

Figura 41 - Identificar serviços dado uma instituição

```
| ?- servicosRealizados(utente,0,R).
R = [3,0] ?
```

Figura 42 - Identificar serviços dado um utente

```
| ?- servicosRealizados(cidade,'porto',R).
R = [3] ?
```

Figura 43 - Identificar serviços dado uma cidade

8.

```
| ?- custoSaude(instituicao,'Hospital Particular de Viana',C).
C = 115 ?
```

Figura 44 - Calcular o custo total dado uma instituição

```
| ?- custoSaude(data,2019,C).
C = 80 ?
```

Figura 45 - Calcular o custo total dado uma data

```
| ?- custoSaude(utente,0,C).
C = 42 ?
```

Figura 46 - Calcular o custo total dado um utente

```
| ?- custoSaude(servico,0,C).
C = 34 ?
```

Figura 47 - Calcular o custo total dado um serviço

```
| ?- listInstituicoesCidade(viana,LS).
LS = ['Centro Hospitalar de Viana','Hospital Particular de Viana'] ?
yes
```

Figura 48 - Determinar instituições existentes numa dada cidade