

UNIVERSIDADE DO MINHO

Exercício nº 1

Programação em lógica e Invariantes

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e
Raciocínio

Ano Letivo 2016/2017

2º Semestre

A73674 – Alexandre Lopes Mandim da Silva

A74219 – Hugo Alves Carvalho

A74260 – Luís Miguel da Cunha Lima

Braga,

19 de março de 2017


Resumo


O trabalho apresentado neste relatório foi desenvolvido no âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio com o objetivo de desenvolver competências na utilização da linguagem de programação em lógica PROLOG.

O exercício consiste num sistema de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde pela realização de serviços de atos médicos.

Ao longo deste relatório iremos explicar todo o processo de desenvolvimento e as decisões tomadas para a realização do trabalho.

Índice

1.	Introdução	5
2.	Preliminares	6
3.	Descrição do Trabalho e Análise de Resultados	7
 3.1.	Representação de Conhecimento	7
3.1.1.	Utente	7
3.1.2.	Cuidado Prestado	7
3.1.3.	Ato Médico	8
3.2.	Registar Utente, Cuidado Prestado e Ato Médico	8
3.3.	Utentes por Critérios de Seleção	9
3.3.1.	Utentes por Nome	10
3.3.2.	Utentes por Idade	10
3.3.3.	Utentes por Morada	10
3.3.4.	Utentes que gastaram mais do que um valor X	10
3.3.5.	Utentes mais frequentes	11
3.4.	Instituições prestadoras de cuidados de saúde	11
3.5.	Cuidados prestados por instituição/cidade	12
3.5.1.	Instituição	12
3.5.2.	Cidade	12
3.6.	Utentes de uma instituição/serviço	12
3.6.1.	Instituição	13
3.6.2.	Serviço (Cuidado Prestado)	13
3.7.	Atos médicos realizados por utente/instituição/serviço	14
3.7.1.	Serviço	14
3.7.2.	Instituição	14
3.7.3.	Utente	14
3.8.	Instituições/serviços que um utente já recorreu	15
3.9.	Custo total dos atos médicos por utente/serviço/ instituição/data	15
3.9.1.	Utente	16
3.9.2.	Serviço (Cuidado Prestado)	16
3.9.3.	Data	16
3.9.4.	Instituição	16

3.10.	Remover utentes, cuidados e atos médicos	17
 3.11.	Predicados auxiliares	18
3.11.1.	Remover duplicados de uma lista	18
3.11.2.	Ordenar decrescentemente	19
3.11.3.	Número de vezes que o elemento X aparece numa lista.....	19
3.11.4.	Soluções	19
3.11.5.	Testar	19
3.11.6.	Somatório.....	20
3.11.7.	Predicado não.....	20
3.11.8.	Pertence.....	20
3.11.9.	Insere ordenado	21
3.11.10.	Concatena.....	21
4.	Conclusões e Sugestões	22
5.	Bibliografia.....	23

1. Introdução

Este relatório foi realizado no âmbito do trabalho nº1, proposto na unidade curricular de SRCR (Sistemas de Representação de Conhecimento e Raciocínio) do curso MIEI (Mestrado Integrado em Engenharia Informática) da Universidade do Minho.

Este trabalho consiste na realização de um conjunto de exercícios, propostos pelo professor, na linguagem de programação PROLOG. O presente relatório relata a descrição destes mesmos exercícios, a maneira de como foram abordados e finalmente, como o grupo os resolveu.

O tema dos exercícios propostos é a prestação de cuidados de saúde pela realização de serviços de atos médicos. Posto isto, é necessário representar o conhecimento sobre utentes (identificação, nome, idade e uma morada), cuidados prestados (identificação do serviço, uma descrição, instituição e cidade) e, por fim, representar conhecimento que relaciona os utentes com os cuidados prestados denominado por atos médicos, em que cada ato é caracterizado por data, utente, serviço e o custo do ato.

Com o conhecimento que queremos retratar bem definido, foram criados predicados para representar esse mesmo conhecimento:

- utente: IdU, Nome, Idade, Morada -> {V,F}
- cuidadoPrestado: IdC, Descricao, Instituicao, Cidade -> {V,F}
- atoMedico: Data, IdU, IdC, Custo -> {V,F}

Por fim, o professor forneceu um conjunto de funcionalidades em que a nossa solução em PROLOG fosse capaz de demonstrar a sua correta execução. Esse conjunto de funcionalidades e respetiva solução dos mesmos é ilustrada no capítulo 3.

2. Preliminares

Para o desenvolvimento deste trabalho não foi necessário o estudo de algum tipo de matéria ou tópico a não ser a matéria lecionada até então, na unidade curricular de SRCR. Saber programação em lógica, com invariantes, conhecer a sintaxe do *PROLOG* (*Sicstus*), saber representar conhecimento e construir mecanismos de raciocínio para dar solução aos problemas propostos, são alguns aspetos importantes que facilitam a leitura e compreensão deste relatório.

3. Descrição do Trabalho e Análise de Resultados

Como já explicado anteriormente, existem três predicados que representam o conhecimento relativo aos utentes, cuidados prestados e atos médicos que faz a associação entre estes últimos dois:

- utente: IdU, Nome, Idade, Morada -> {V,F}
- cuidadoPrestado: IdC, Descricao, Instituicao, Cidade -> {V,F}
- atoMedico: Data, IdU, IdC, Custo -> {V,F}

Posto isto, é necessário dar solução a um conjunto de funcionalidades propostas no enunciado. As respetivas funcionalidades são descritas em seguida.

3.1. Representação de Conhecimento

Em seguida serão apresentados todos os predicados usados para representar o conhecimento, bem como exemplos que foram usados para podermos testar todas as funcionalidades.

3.1.1. Utente

Um utente é caracterizado pelo seu id, por um nome, idade e morada:

```
% Extensao do predicado utente: IdU, Nome, Idade, Morada -> {V,F}
utente(1, hugo, 20, cerveira).
utente(2, alexandre, 21, povoadevarzim).
utente(3, luis, 20, pontedelima).
utente(4, francisco, 33, braga).
utente(5, antonio, 5, guimaraes).
utente(6, anarita, 25, pontedelima).
utente(7, marta, 65, guimaraes).
utente(8, paulo, 15, barcelos).
utente(9, josefina, 42, barcelos).
utente(10, luis, 52, vizela).
```

Figura 1 - Predicado utente

3.1.2. Cuidado Prestado

Um cuidado prestado é caracterizado por um id, descrição do cuidado, instituição e cidade:

```
% Extensao do predicado cuidadoPrestado: IdC, Descricao, Instituicao, Cidade -> {V,F}
cuidadoPrestado(1, pediatria, cserveira, vncerveira).
cuidadoPrestado(2, cardiologia, hospitalbraga, braga).
cuidadoPrestado(3, cirurgia, hospitalbraga, braga).
cuidadoPrestado(4, ginecologia, hospitalbraga, braga).
cuidadoPrestado(5, neurologia, hsmm, barcelos).
cuidadoPrestado(6, psiquiatria, hsog, guimaraes).
cuidadoPrestado(7, oftamologia, hospitaldaluz, povoadevarzim).
```

Figura 2 - Predicado cuidadoPrestado

3.1.3. Ato Médico

Um ato médico faz associação entre um utente e um cuidado prestado, deste modo é composto pelos id's do utente e respetivo cuidado, bem como a data e custo:

```
% Extensao do predicado atoMedico: Data,IdU,IdC,Custo -> {V,F}
atoMedico(01-01-2017, 1, 6, 15).
atoMedico(13-01-2017, 3, 4, 30).
atoMedico(13-01-2017, 2, 5, 30).
atoMedico(14-01-2017, 2, 7, 8).
atoMedico(20-01-2017, 4, 2, 20).
atoMedico(02-02-2017, 7, 4, 5).
atoMedico(03-02-2017, 3, 5, 24).
atoMedico(20-02-2017, 6, 7, 37).
atoMedico(22-02-2017, 6, 2, 55).
atoMedico(04-03-2017, 2, 2, 98).
atoMedico(15-03-2017, 1, 1, 5).
```

Figura 3 - Predicado atoMedico

3.2. Registrar Utente, Cuidado Prestado e Ato Médico

Neste ponto do enunciado, foi pedido uma implementação de um predicado que permitisse o registo de utentes, cuidados prestados e atos médicos.

Para isso, o grupo criou dois predicados (registar e inserção) que tratam da inserção na base de conhecimento.

O predicado registar verifica o predicado testar, que será explicado no tópico dos predicados auxiliares.

```
% -----
% Extensão do predicado registar: Termo -> {V,F}

registar(Termo) :-
    findall(Invariante, +Termo::Invariante, Lista),
    insercao(Termo),
    testar(Lista).

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

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

Figura 4 - Predicados registar e insercao

No entanto, esta inserção de dados deve obedecer a alguns princípios que o grupo entendeu como fundamentais. Por exemplo, não deve ser permitido inserir um utente quando o id de utente que estamos a inserir já existe na base de conhecimento. A mesma situação acontece no caso dos cuidados prestados.

Com esse objetivo de melhorar a inserção de conhecimento, foi necessário implementar invariantes, que são utilizados no predicado “findall”.

Em primeiro lugar, para podermos implementar invariantes, foi necessário recorrer a definições do SICStus PROLOG que nos permitem posteriormente realizarmos a nossa implementação:


```
% SICStus PROLOG: definicoes iniciais

:- op( 900,xfy,'::' ).

% Para os invariantes:

:- dynamic utente/4.
:- dynamic cuidadoPrestado/4.
:- dynamic atoMedico/4.
```

Figura 5 - Definições iniciais para podermos desenvolver os invariantes

De seguida, o grupo implementou os invariantes que considerou essenciais para uma correta resolução do problema em questão.

```
% Invariante que não permite a inserção de conhecimento de um utente com um id já existente

+utente(Id, N, I, M) :: (solucoes((Id, N), utente(Id, X, Y, Z), S),
    comprimento(S, L),
    L == 1).
```

Figura 6 - Invariante que não permite a inserção de um utente com um id já existente na base de conhecimento

```
% Invariante que não permite a inserção de conhecimento de um cuidadoPrestado com um id já existente

+cuidadoPrestado(Id,D,I,C) :: (solucoes((Id, D), cuidadoPrestado(Id, X, Y, Z), S),
    comprimento(S, L),
    L == 1).
```

Figura 7 - Invariante que não permite a inserção de um cuidado com o id já existente na base de conhecimento

Estes invariantes procuram as soluções que obedecem a um determinado termo, e coloca-as numa lista. Assim, ao procurar todas as soluções para um determinado id utente/cuidado prestado, só deve ser possível inserir conhecimento se o tamanho da lista (já com esse termo) for 1.

Por outro lado, apenas é possível inserir conhecimento sobre atos médicos quando estes correspondem a um utente e a um cuidado médico existentes na base de conhecimento.

```
% Invariante que não permite a inserção de conhecimento de um atoMedico quando o id do utente/cuidadoPrestado
% não existem na base de conhecimento

+atoMedico(D,U,S,C) :: (solucoes((S, Xs), cuidadoPrestado(S, Xs, Ys, Zs), Servs),
    comprimento(Servs, Ls),
    Ls == 1,
    solucoes((U, Xu), utente(U, Xu, Yu, Zu), Uts),
    comprimento(Uts, Lu),
    Lu == 1).
```

Figura 8 - Invariante que não permite a inserção de conhecimento de um atoMedico quando o id utente/cuidadoPrestado não existem na base de conhecimento.

3.3. Utentes por Critérios de Seleção

Neste exercício era proposto a identificação de utentes por critérios de seleção.

O grupo decidiu identificar utentes por um dado nome, idade ou morada, bem como a identificação através do valor pago em atos médicos, ou seja, utentes que gastaram mais do que um valor X nesses atos. Por fim criamos um predicado que lista, em ordem decrescente, os utentes mais frequentes, ou seja, com mais atos médicos.

3.3.1. Utentes por Nome

Neste predicado é fornecido um nome na variável Nome e o predicado “findall” vai procurar as soluções no predicado utente que tenham esse dado nome. As soluções são guardadas na variável Resultado na forma de uma lista em que cada elemento dessa lista é composto por 4 valores, ou seja, os dados dos utentes.

```
% Extensao do predicado utentesPNome: Nome,Resultado -> {V,F}
utentesPNome(Nome,Resultado)
:- findall( (IdU,Nome,Idade,Morada), utente(IdU,Nome,Idade,Morada), Resultado ).
```

Figura 9 - Predicado utentesPNome

3.3.2. Utentes por Idade

Este predicado segue o mesmo raciocínio do predicado “utentesPNome”, apenas é fornecido a idade ao invés do nome.

```
% Extensao do predicado utentesPIdade: Idade,Resultado -> {V,F}
utentesPIdade(Idade,Resultado)
:- findall( (IdU,Nome,Idade,Morada), utente(IdU,Nome,Idade,Morada), Resultado ).
```

Figura 10 - Predicado utentesPIdade

3.3.3. Utentes por Morada

Este predicado, á semelhança dos dois anteriores, procura todas as soluções no predicado utente que tenham uma determinada morada.

```
% Extensao do predicado utentesPMorada: Morada,Resultado -> {V,F}
utentesPMorada(Morada,Resultado)
:- findall( (IdU,Nome,Idade,Morada), utente(IdU,Nome,Idade,Morada), Resultado ).
```

Figura 11 - Predicado utentesPMorada

3.3.4. Utentes que gastaram mais do que um valor X

Neste predicado, pretendemos devolver uma lista com os utentes (id's) que gastaram mais do que um valor X em atos médicos. Neste caso, a lista poderia devolver os nomes e não os id's, no entanto, não existem id's repetidos, enquanto que nomes podem existir, deste modo, poderiam existir ambiguidades e não sabermos qual utente é que gastou mais que um determinado valor.

Para dar solução a este problema, inicialmente guardamos uma lista com os pares (id do utente, custo) de todos os atos médicos. Em seguida, usando um predicado auxiliar denominado “utentesQGastaramMaisQXAux”, usamos a lista gerada anteriormente e o valor que pretendemos comparar (X) para obtermos uma lista com os id's dos utentes que tinham custos superior a esse valor X.

Por fim, para não mostrarmos utilizadores repetidos, pois um utilizador poderia ter gasto mais que X em atos médicos várias vezes, usamos um predicado “removeDup” que remove

os duplicados de uma lista. Este predicado “removeDup” é explicado no ponto dos predicados auxiliares deste relatório.

```
% Extensao do predicado utentesQGastaramMaisQX: Valor,Resultado -> {V,F}
utoresQGastaramMaisQX(Valor,R1)
:- findall( (IdU,Custo), atoMedico(Data,IdU,IdC,Custo), Resultado),
   utentesQGastaramMaisQXAux(Resultado,Valor,R),removeDup(R,R1).

% Extensao do predicado utentesQGastaramMaisQXAux: Lista,Valor,Resultado -> {V,F}
utoresQGastaramMaisQXAux([],Valor,[]).
utoresQGastaramMaisQXAux([(IdU,Custo)|T],Valor,L):- Custo <= Valor,utoresQGastaramMaisQXAux(T,Valor,L).
utoresQGastaramMaisQXAux([(IdU,Custo)|T],Valor,[IdU|L]):- Custo > Valor,utoresQGastaramMaisQXAux(T,Valor,L).
```

Figura 12 - Predicados *utoresQGastaramMaisQX* e *utoresQGastaramMaisQXAux*

3.3.5. Utentes mais frequentes

Para obtermos a lista com os utentes mais frequentes usamos a seguinte abordagem: através do predicado “findall” obtemos duas listas, uma com todos os id’s dos utentes do predicado utente e uma segunda lista com os id’s dos utentes no predicado “atoMedico”. Desta forma, usando um predicado auxiliar denominado “listarUtentesMaisFreqAux”, usamos cada elemento da primeira lista e verificamos quantas vezes aparece na segunda, ou seja, quantos atos médicos cada utente realizou. Este predicado auxiliar devolve uma lista com pares, em que cada par tem o id de utente e o respetivo numero de vezes que realizou um ato médico. Este predicado usa um predicado chamado “quantosTem” que dado um elemento verifica quantas vezes aparece numa lista. Este predicado é explicado no ponto 3.10 deste relatório.

Finalmente, como queremos apresentar a lista ordenado usamos um predicado “ordenarDecresc” que ordenou de forma decrescente os pares da lista, baseado no segundo elemento do par, ou seja, o número de atos médicos. Este predicado também é explicado no subtópico dos predicados auxiliares.

```
% Extensao do predicado ListarUtentesMaisFreq: Resultado -> {V,F}
listarUtentesMaisFreq(Resultado):- findall( IdU, utente(IdU,Nome,Idade,Morada), R ),
                                   findall( IdU, atoMedico(Data,IdU,IdC,Custo), R2 ),
                                   listarUtentesMaisFreqAux(R,R2,R3),
                                   ordenarDecresc(R3,Resultado).

% Extensao do predicado listarUtentesMaisFreqAux: Utentes,Resultado -> {V,F}
% Esta funcao pega em cada elemento da 1ª lista e verifica quantas vezes aparece na segunda, n
% o fim devolve um par com cada elemento e o nr de vezes q apareceu.

listarUtentesMaisFreqAux([],L,[]).
listarUtentesMaisFreqAux([H|T], L, [(H,Q)|R]):- quantosTem(H,L,Q), listarUtentesMaisFreqAux(T,L,R).
```

Figura 13 - *listarUtentesMaisFreq* e *listarUtentesMaisFreqAux*

3.4. Instituições prestadoras de cuidados de saúde

Neste exercício é pedido para identificar todas as instituições que prestam cuidados de saúde, ou seja, todas as instituições representadas no predicado “cuidadoPrestado”. Novamente é usado o “findall” que encontra todas as soluções no predicado “cuidadoPrestado” e seleciona apenas as instituições. De seguida, o predicado

“removeDup”, já mencionado anteriormente neste relatório, remove os duplicados dessa lista para não apresentar instituições repetidas.

```
% Extensao do predicado instituicoes: Valor,Resultado -> {V,F}
instituicoes(Resultado)
:- findall( Instituicao, cuidadoPrestado(IdC,Descricao,Instituicao,Cidade), R ), removeDup(R,Resultado).
```

Figura 14 - Predicado instituicoes

3.5. Cuidados prestados por instituição/cidade

Neste exercício o objetivo é identificar todos os cuidados prestados por instituição ou cidade. Deste modo, o grupo desenvolveu dois predicados que respondem ao que é pedido: “getCuidadosbyInstituição” e “getCuidadosbyCidade”. De seguida, será explicado o raciocínio que o grupo teve para a resolução destes dois predicados.

3.5.1. Instituição

Neste predicado é fornecido o nome da instituição na variável “I”, e o predicado “findall” vai pesquisar todas as soluções de cuidados prestados que incluam este “I” de instituição associada, criando assim uma lista com todos os tipos de cuidados prestados pela instituição dada em que cada elemento é composto por 4 valores, ou seja, os dados dos vários cuidados prestados.

```
% Extensao do predicado getCuidadosbyInstituicao: I,R -> {V,F}
getCuidadosbyInstituicao(I,S) :-
    findall((Ss,Desc,I,Cid), cuidadoPrestado(Ss,Desc,I,Cid), S).
```

Figura 15 - Predicado getCuidadosbyInstituicao

3.5.2. Cidade

Este predicado segue o mesmo raciocínio do predicado anterior (“getCuidadosbyInstituição”), somente é fornecido uma cidade ao invés da instituição.

```
% Extensao do predicado getCuidadosbyCidade: C,R -> {V,F}
getCuidadosbyCidade(C,S) :-
    findall((Ss,Desc,Ins,C), cuidadoPrestado(Ss,Desc,Ins,C), S).
```

Figura 16 - Predicado getCuidadosbyCidade

3.6. Utentes de uma instituição/serviço

Neste exercício é pedido para apresentar todos os utentes de uma dada instituição ou serviço. Deste modo, o grupo criou dois predicados: “getUtByIns” e “getUtBySer” para resolver o problema proposto.

3.6.1. Instituição

Para responder a esta funcionalidade foi criado o predicado “getUtByIns” que dado o nome de uma instituição, coloca numa lista todos os utentes que a frequentaram. Para isso, recorremos ao predicado “findall” que recolhe todos os serviços que façam *match* com os códigos dos serviços associados com a instituição “I”. De seguida, são guardados todos os códigos dos utentes associados aos serviços anteriormente recolhidos, através do predicado auxiliar “getUtByInsAux”.

Para finalizar, utilizamos o predicado getUt que verifica se uma lista tem todos os dados relativos aos códigos dos utentes recolhidos.

```
% -----
% Extensao do predicado getUtByIns: I,R -> {V,F}

getUtByIns(I,R):- findall(Ss ,cuidadoPrestado(Ss,_,I,_), R1),
                  getUtByInsAux(R1,R2),
                  getUt(R2,R).

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

getUtByInsAux([],[]).
getUtByInsAux([IdC|Y],R) :- findall(IdU, atoMedico(_,IdU,IdC,_), R1),
                             getUtByInsAux(Y,R2), concatena(R1,R2,R).

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

getUt([],[]).
getUt([IdU|Y],R) :- findall((IdU,Nome,Idade,Morada), utente(IdU,Nome,Idade,Morada), R1),
                    getUt(Y,R2), concatena(R1,R2,R).
```

Figura 17 - Predicados getUtByIns, getUtByInsAux e getUt

3.6.2. Serviço (Cuidado Prestado)

Este requisito foi respondido com a utilização do predicado “getUtBySer” que opera de forma análoga ao predicado descrito anteriormente, em que a única e simples diferença é que agora queremos saber quais os utentes que frequentam um determinado serviço.

```
% -----
% Extensao do predicado getUtBySer: D,R -> {V,F}

getUtBySer(D,R) :-
    findall(Ss , cuidadoPrestado(Ss,D,_,_), R1 ),
    getUtBySerAux(R1,R2),
    getUt(R2,R).

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

getUtBySerAux([],[]).
getUtBySerAux([IdC|Y],R) :-
    findall(IdU, atoMedico(_,IdU,IdC,_), R1),
    getUtBySerAux(Y,R2),
    concatena(R1,R2,R).
```

Figura 18 - Predicados getUtBySer e getUtBySerAux

3.7. Atos médicos realizados por utente/instituição/serviço

Neste predicado o objetivo é identificar todos os atos médicos realizados por um determinado utente, instituição ou por um serviço. Deste modo o grupo revolveu criar três predicados para satisfazer o pedido: “getAtosBySer”, “getAtosByIns” e “getAtosByUt”.

Nestes três predicados foi utilizado o predicado auxiliar “concatena” que irá fazer a junção das duas listas numa só, tal como explicado no tópico dos predicados auxiliares.

3.7.1.Serviço

Este predicado tem como objetivo para que dado um serviço, coloca numa lista todos os atos médicos relativos a esse tipo de serviço. Para isso recorremos ao predicado “findall” que irá recolher todos os códigos de serviço que façam *match* com o nome do serviço “S”. Para finalizar, é criada uma lista através do predicado auxiliar “getAtosBySerAux” com todos os atos médicos que foram realizados pelo tipo de serviço pretendido, em que cada elemento da lista construída será composto por quatro valores, ou seja, os dados dos vários atos médicos.

```
% -----
% Extensao do predicado getAtosBySer: S,R -> {V,F}

getAtosBySer(S,R) :- findall(Ss ,cuidadoPrestado(Ss,S,_,_), R1), getAtosBySerAux(R1,R).

% Extensao do predicado getAtosBySerAux: L,L -> {V,F}
getAtosBySerAux([],[]).
getAtosBySerAux([IdC|Y],R) :- findall((Data,IdU,IdC,Custo), atoMedico(Data,IdU,IdC,Custo), R1),
                             getAtosBySerAux(Y,R2), concatena(R1,R2,R).
```

Figura 19 - Predicados getAtosBySer e getAtosBySerAux

3.7.2. Instituição

Esta funcionalidade foi respondida através do predicado “getAtosByIns” que segue o mesmo raciocínio do tópico anteriormente descrito, em que a única diferença é que recebemos o nome de uma instituição, ou seja, o objetivo é identificar os atos médicos relativos a esta instituição.

```
% -----
% Extensao do predicado getAtosByIns: I,R -> {V,F}

getAtosByIns(I,R):- findall(Ss ,cuidadoPrestado(Ss,_,I,_), R1), getAtosByInsAux(R1,R).

% Extensao do predicado getAtosBySer: L,L -> {V,F}

getAtosByInsAux([],[]).
getAtosByInsAux([IdC|Y],R) :- findall((Data,IdU,IdC,Custo), atoMedico(Data,IdU,IdC,Custo), R1),
                             getAtosByInsAux(Y,R2), concatena(R1,R2,R).
```

Figura 20 - Predicados getAtosByIns e getAtosByInsAux

3.7.3.Utente

Neste predicado pensamos em duas formas de resolver em que nos podia ser passado como parâmetro o nome do utente ou o código desse mesmo utente. Para uma melhor gestão

do domínio de conhecimento optamos por escolher receber como parâmetro o código do utente (id) pois poderá haver mais do que um utente com o mesmo nome o que tornaria ambígua a procura pelos atos médicos pretendidos.

Desta forma apenas precisamos recorrer ao predicado “findall” de maneira a recolher todos os atos médicos que façam *matching* com o código do utente em questão.

```
% -----
% Extensao do predicado getAtosByUt: U,R -> {V,F}

getAtosByUt(U,R) :-
    forall((D,U,S,C),atoMedico(D,U,S,C),R).
```

Figura 21 - Predicado getAtosByUt

3.8. Instituições/serviços que um utente já recorreu

Neste predicado, temos que devolver todas as instituições e serviços que um utente já recorreu. Existem duas possíveis soluções para este problema. Uma é criar um predicado que calculasse as instituições que um dado utente já recorreu e criar um segundo predicado que devolvesse os serviços que esse utente já utilizou. Esta abordagem foi utilizada em funcionalidades anteriores.

De modo a variar as soluções apresentadas às diferentes funcionalidades, neste exercício seguimos outra abordagem, em que criamos apenas um predicado que devolve uma lista de pares (Instituição, Serviço) que cada utente já recorreu.

Para implementar o predicado que dá solução a este problema (“getInstSerByUt”) inicialmente calculamos todos os id’s de cuidados médicos que o utente já recorreu. Com essa lista, removemos os id’s repetidos e através do predicado auxiliar “getInstSerByUtAux”, obtemos os pares (Instituição, Serviço) da lista de id’s obtida anteriormente

```
% Extensao do predicado getInstSerByUt: Id,Resultado -> {V,F}

getInstSerByUt(IdU,Resultado2) :-
    forall(IdC , atoMedico(Data,IdU,IdC,Custo), R ),removeDup(R,Resultado), getInstSerByUtAux(Resultado,Resultado2).

% Extensao do predicado getInstByUtAux: L,Resultado -> {V,F}

getInstSerByUtAux([],[]).
getInstSerByUtAux([IdC|T],[([Instituicao,Descricao]|Resto)] :- cuidadoPrestado(IdC,Descricao,Instituicao,Cidade),
    getInstSerByUtAux(T,Resto).
```

Figura 22 - Predicados getInstSerByUt e getInstSerByUtAux

3.9. Custo total dos atos médicos por utente/serviço/instituição/data

Neste exercício, o objetivo é calcular o custo total de um utente, serviço, instituição ou data. Para isso, o grupo desenvolveu quatro predicados que respondem ao que é pedido: “getTotalByUtente”, “getTotalByServico”, “getTotalByData” e “getTotalByInstituicao”. Os três primeiros predicados invocam um predicado auxiliar “somatorio” que será descrito no tópico de predicados auxiliares.

De seguida, será explicado o raciocínio que o grupo teve para a resolução deste problema.

3.9.1. Utente

Neste predicado é fornecido o id do utente na variável Id, e o predicado “findall” vai procurar todas as soluções de atos médicos que contenham este id de cliente associado, criando assim a lista com os custos de cada ato médico (“TotUtente”).

Seguidamente, temos que verificar se o somatório desta lista de custos corresponde a “T”, através do predicado auxiliar “somatorio”.

```
% Extensao do predicado getTotalByUtente: IdUtente, Total -> {V,F}

getTotalByUtente(Id,T) :-
    solucoes(C, atoMedico(D, Id, S, C), TotUtente),
    somatorio(TotUtente,T).
```

Figura 23 - Predicado getTotalByUtente

3.9.2. Serviço (Cuidado Prestado)

Este predicado segue o mesmo raciocínio do predicado “getTotalByUtente”, apenas é fornecido o Id de cuidado prestado ao invés do id de utente.

```
% Extensao do predicado getTotalByServico: IdServico, Total -> {V,F}

getTotalByServico(Id,T) :-
    solucoes(C, atoMedico(D, U, Id, C), TotServico),
    somatorio(TotServico,T).
```

Figura 24 - Predicado getTotalByServico

3.9.3. Data

Este predicado segue o mesmo raciocínio dos pontos anteriores, apenas é fornecido a data prestado ao invés do id de utente ou id de cuidado prestado.

```
% Extensao do predicado getTotalByData: Data, Total -> {V,F}

getTotalByData(D,T) :-
    solucoes(C, atoMedico(D, U, S, C), TotData),
    somatorio(TotData,T).
```

Figura 25 - Predicado getTotalByData

3.9.4. Instituição

Este predicado, apesar de ser semelhante aos anteriores, tem apenas uma diferença pois em primeiro lugar precisamos de procurar quais os serviços existentes na instituição “I”. De seguida, é necessário verificar se a lista com todos os id’s de cuidados prestados (“Servs”) corresponde a “T”.

Com esse propósito foi implementado um predicado auxiliar “getTotalListServs” que calcula o total de cada um dos serviços da lista, através do predicado anterior “getTotalByServiço”, fazendo a respectiva soma de todos os custos.

```
% Extensao do predicado getTotalByInstituicao: Instituicao, Total -> {V,F}

getTotalByInstituicao(I,T) :-
    solucoes(Id, cuidadoPrestado(Id, D, I, C), Servs),
    getTotalListServs(Servs,T).

getTotalListServs([],T) :-
    T is 0.
getTotalListServs([X|Y], T) :-
    getTotalListServs(Y,Z),
    getTotalByServico(X,R),
    T is Z+R.
```

Figura 26 - Predicados getTotalByInstituicao e getTotalListServs

3.10. Remover utentes, cuidados e atos médicos

Neste predicado, o objetivo é permitir a remoção de informação sobre utentes, cuidados prestados e atos médicos.

Deste modo, o grupo criou dois predicados (remover e remoção) que tratam da remoção na base de conhecimento.

O predicado remover verifica o predicado testar, que tal como já explicado anteriormente será explicado no tópico dos predicados auxiliares.

```
% -----
% Extensão do predicado remover: Termo -> {V,F}

remover(Termo) :-
    findall(Invariante, -Termo::Invariante, Lista),
    testar(Lista),
    remocao(Termo).

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

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

Figura 27 - Predicados remover e remocao

Esta remoção de dados deve ter em consideração alguns cuidados, como por exemplo não permitir a remoção de um utente ou serviço que estejam associados a atos médicos, uma vez que resulta numa perda de conhecimento.

Por outro lado, só deve ser permitido remover conhecimento se este de facto existir. Deste modo, uma alteração do predicado “remover” para o predicado “registar” descrito anteriormente neste relatório, consiste no facto de invocar primeiro o predicado testar, uma vez que se fosse ao contrário, ao tentar remover conhecimento inexistente, este seria sempre adicionado à base de conhecimento.

Com o objetivo de implementar um correto predicado de remoção de conhecimento, foi necessário implementar invariantes, que são utilizados no predicado “findall”.

```
%-----
% Invariante que não permite a remocao de conhecimento de um utente não presente na base de conhecimento
% e com id associado a atoMedico

-utente(Id,N,I,M) :: (solucoes((Id), utente(Id,N,I,M), Uts),
                    comprimento(Uts, Lu),
                    Lu == 1,
                    solucoes((Id), atoMedico(X, Id, Y, Z), R),
                    comprimento(R, L),
                    L == 0).
```

Figura 28 - Invariante que não permite a remoção de conhecimento de um utente não presente na base de conhecimento ou com o id associado atos médicos

```
%-----
% Invariante que não permite a remocao de conhecimento de um cuidadoPrestado não presente na base de
% conhecimento e com o id associado a atoMedico

-cuidadoPrestado(Id, D, I, C) :: (solucoes((Id), cuidadoPrestado(Id,D,I,C), Servs),
                                comprimento(Servs, Lc),
                                Lc == 1,
                                solucoes((Id), atoMedico(X, Y, Id, Z), R),
                                comprimento(R, L),
                                L == 0).
```

Figura 29 - Invariante que não permite a remoção de conhecimento de um cuidado prestado não presente na base de conhecimento ou com o id associado atos médicos

Por outro lado, é também necessário implementar um invariante que não permita que seja removido conhecimento sobre um ato médico que não exista, pela mesma situação anteriormente referida.

```
%-----
% Invariante que não permite a remocao de conhecimento de um atoMedico não presente na base de conhecimento

-atoMedico(D,U,S,C) :: (solucoes((D,U,S,C), atoMedico(D,U,S,C), A),
                       comprimento(A, L),
                       L == 1).
```

Figura 30 - Invariante que não permite a remoção de conhecimento de um ato médico não existente na base de conhecimento

3.11. Predicados auxiliares

3.11.1. Remover duplicados de uma lista

Este predicado recebe uma lista e verifica se existem elementos repetidos. Caso existam são removidos. No fim de execução deste predicado temos uma segunda lista sem elementos repetidos.

```
% Extensao do predicado removeDup: L,R -> {V,F}

removeDup([], []).
removeDup([X|T], R) :- pertence(X,T),
                        removeDup(T, R).
removeDup([X|T], [X|R]) :- nao(pertence(X, T)),
                           removeDup(T, R).
```

Figura 31 - Predicado removeDup

3.11.2. Ordenar decrescentemente

Este predicado é usado para ordenar de forma decrescente uma lista. Isto é feito através da inserção numa segunda lista, elemento a elemento, de forma ordenada.

```
% Extensao do predicado ordenarDecresc: L,Resultado -> {V,F}

ordenarDecresc([X], [X]).
ordenarDecresc([X|Y], T) :-
    ordenarDecresc(Y, R), insereOrdenado(X, R, T).
```

Figura 32 - Predicado ordenarDecresc

3.11.3. Número de vezes que o elemento X aparece numa lista

Este predicado calcula quantas vezes um dado elemento aparece numa dada lista.

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

quantosTem(A, [], 0).
quantosTem(A, [H|T], Resultado) :- (A == H), quantosTem(A, T, R), Resultado is R+1.
quantosTem(A, [H|T], Resultado) :- (A \= H), quantosTem(A, T, Resultado).
```

Figura 33 - Predicado quantosTem

3.11.4. Soluções

O predicado soluções encontra todas as possibilidades de prova de um teorema, sendo verdadeira sempre que o predicado “findall” também for.

```
% -----
% Extensão do predicado solucoes: X,Y,Z -> {V,F}

solucoes(X,Y,Z) :-
    findall(X,Y,Z).
```

Figura 34 - Predicado solucoes

3.11.5. Testar

Este predicado recebe uma lista e verifica se esta é válida.

```
% -----
% Extensão do predicado testar: Lista -> {V,F}

testar([]).
testar([I|L]) :-
    I,
    testar(L).
```

Figura 35 - Predicado testar

3.11.6. Somatório

O predicado “somatorio” calcula a soma de um conjunto de valores. É considerado neste caso que quando uma lista está vazia, o seu somatório é 0.

```
% -----
% Extensao do predicado somatorio: lista, resultado -> {V,F}

somatorio([], 0).
somatorio([X|Y], R) :-
    somatorio(Y,G),
    R is X+G.
```

Figura 36 - Predicado somatorio

3.11.7. Predicado não

Este predicado calcula o valor de verdade contrário à resposta a uma determinada questão.

```
% Extensao do predicado nao: Q -> {V,F}

nao(Q):- Q, !, fail.
nao(Q).
```

Figura 37 - Predicado nao

3.11.8. Pertence

Este predicado recebe um elemento e uma lista e verifica se este existe na lista. O algoritmo consiste em verificar se o elemento a procurar é a cabeça da lista. Se for, então o elemento foi encontrado e estamos perante uma verdade (“yes”). Se não encontrar, procura na cauda da lista, até esta ser vazia. Quando for vazia, uma vez que uma lista vazia não contém elementos, a operação falha e estamos perante o valor de verdade “no”.

```
% Extensao do predicado pertence: X,L -> {V,F}

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

Figura 38 - Predicado pertence

3.11.9. Insere ordenado

Este predicado insere um elemento numa lista ordenada. No final da execução deste predicado, temos uma segunda lista já com o elemento inserido.

```
% Extensao do predicado insereOrdenado: X,L,Resultado -> {V,F}
|
insereOrdenado(X1,Y1), [], [(X1,Y1)].
insereOrdenado(X1,Y1), [(X2,Y2)|Z], [(X1,Y1)|[(X2,Y2)|Z]] :- Y1>Y2.
insereOrdenado(X1,Y1), [(X2,Y2)|Z], [(X2,Y2)|R2] :- Y1<Y2,
    insereOrdenado(X1,Y1),Z,R2).
```

Figura 39 - Predicado insereOrdenado

3.11.10. Concatena

O predicado “concatena” resulta de concatenar os elementos da lista L1 com os elementos da lista L2.

```
% Extensao do predicado concatena(L1,L2,L3)->{V,F}

concatena([],L2,L2).
concatena([X|L1],L2,[X|L]) :- concatena(L1,L2,L).|
```

Figura 40 - Predicado concatena

4. Conclusões e Sugestões

Este trabalho prático foi realizado com a intenção de solidificar os conhecimentos da unidade curricular de sistemas de representação de conhecimento e raciocínio. Concluído este trabalho, acreditamos ter desenvolvido um sistema de representação de conhecimento e raciocínio que responde a todos os requisitos propostos.

Como trabalho futuro alguns aspetos podiam ser revistos e melhorados, como por exemplo a implementação de um maior conjunto de funcionalidades adicionais para uma melhor gestão do sistema.

Foram aplicados os conhecimentos da linguagem de PROLOG que adquirimos ao longo das aulas, sendo estes desenvolvidos à medida que o trabalho ia sendo realizado, uma vez que foi necessário resolver um conjunto de novas situações e criar mecanismos de raciocínio para a resolução de problemas.

De uma forma geral, os resultados produzidos foram bastante satisfatórios e enriquecedores para todos os elementos do grupo pois permitiu consolidar conhecimentos.

5. Bibliografia

- [Analide, 2011] ANALIDE, César, NOVAIS, Paulo, NEVES, José,
“Sugestões para a Redacção de Relatórios Técnicos”,
Relatório Técnico, Departamento de Informática, Universidade
do Minho, Portugal, 2011
- [Bratko, 1986] BRATKO, Ivan,
“PROLOG: Programming for Artificial Intelligence”,
British Library of Congress, Grã Bretanha, 2011.