Sistemas de Representação de Conhecimento e Raciocínio: Exercício Prático Nº1

Universidade do Minho

Mestrado Integrado em Engenharia Informática
Grupo 14

Duarte Freitas
A63129

Afonso Sousa
A74196

José Lopes Ramos
A73855

Rafael Silva
A74264

31 de Março de 2019

Resumo

Este documento é um relatório técnico sobre o trabalho desenvolvido para o primeiro trabalho prático da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio.
Nesta primeira de três fases foi-nos pedido para fazer o exercício de uma forma mais próxima da linguagem (de acordo com as aulas práticas), sem grandes extensões. Para cada exercício enunciado apresentaremos a forma escolhida para representar o conhecimento, o raciocínio feito para chegar à solução, a própria solução por nós encontrada, o conhecimento usado pelos casos exemplo e o resultado desses exemplos. Ao longo da exposição serão usados predicados auxiliares que só serão analisados numa secção mais à frente. Todo o código aqui apresentado está na íntegra no ficheiro em anexo.

Conteúdo

Lista de Figuras

Introdução

Para o primeiro exercício prático foi-nos pedido para desenvolver um projeto de programação em lógica com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde.
Com uma base de conhecimento de complexidade média, este trabalho terá escala suficiente para mostrar o conhecimento adquirido durante esta fase, destacando-se a representação de conhecimento, desenvolvimento de predicados e invariantes.

Preliminares

Este trabalho prático foi resolvido com recurso ao conhecimento adquirido nas aulas da UC, material bibliográfico digital sobre *Prolog* e aos nossos apontamentos. *Prolog* é uma linguagem declarativa de lógica de primeira-ordem, com uma sintaxe base semelhante a outras linguagens de programação populares, mas tem algumas diferenças chave que a diferenciam. As mais relevantes para este trabalho podem ser descritas sumariamente como:

1. Objetos:

Facto	Constata algo que se reconhece e se sabe verdadeiro;
Predicado	Implementa uma relação;
Regra	Utilizada para definir um novo predicado;
Invariante	Regra inviolável ao longo da execução do programa.

2. Pontuação:

.	Utilizado para terminar uma declaração;
,	Utilizado para representar a conjunção lógica;
:-	Utilizado para representar a implicação conversa lógica ($\leftarrow$);
;	Utilizado para representar a disjunção inclusiva lógica;
//	Utilizado para representar a unificação de dois conjuntos.

3. Predicados: *findall* Constrói a lista com os termos que respeitam condições argumento.

Todos os resultados apresentados são obtidos com o conhecimento mostrado na próxima secção, evitando quaisquer alterações causadas pelas evoluções ou involuções resultantes das perguntas 1 e 2.
Por indicação do docente foram usadas três *flags* específicas da linguagem *PROLOG*, que evitam *warnings* ou erros de carregamento/consulta do ficheiro usado para desenvolver o projeto, podendo ser assim executado corretamente e de acordo com os parâmetros definidos para o conhecimento.

```
1 :- set_prolog_flag( discontiguous_warnings,off ).
2 :- set_prolog_flag( single_var_warnings,off ).
3 :- set_prolog_flag( unknown,fail ).
```

Descrição do Trabalho e Análise de Resultado

Base de Conhecimento

Para começarmos a resolução dos problemas enunciados temos primeiro de fazer algumas declarações iniciais e definir a quantidade de argumentos relativos a cada conhecimento, sendo que neste caso optamos por usar *dynamic*. Teremos ***utente***, ***servico*** e ***consulta*** com quatro parâmetros, mais um predicado *solucoes(T,Q,S)* que servirá de *alias* para o *findall(T,Q,S)*. Esta renomeação foi feita para abstrair a sintaxe do *PROLOG*.

```
:- op( 900,xfy,'::' ).
:- dynamic utente/4.
:- dynamic servico/4.
:- dynamic consulta/4.
solucoes(T,Q,S) :- findall(T,Q,S).
```

Para incorporar significado nas fórmulas de uma linguagem que não o faz naturalmente recorremos a invariantes estruturais, referentes à inserção (+) e remoção (-) de conhecimento, que serão usados em predicados das alínea 1 e 2 (evolução e involução) a ser explicados mais à frente em Predicados Auxiliares.

```
% Invariante Estrutural: não permitir a inserção de numero de utente repetido
+utente( ID,_,_,_ ) :: (solucoes( (ID), (utente( ID,_,_,_ )),S ),
                        comprimento( S,N ),
                          N == 1 ).
% Invariante Estrutural: não permitir a remoção de utente caso exista consulta
 ↪ associada
-utente( ID,_,_,_ ) :: (solucoes( (ID), consulta( _,ID,_,_ ),S ),
                        comprimento( S,N ),
                           N == 0 ).
% Invariante Estrutural: não permitir a inserção de número de serviço repetido
+servico( ID,_,_,_ ) :: (solucoes( (ID), (servico( ID,_,_,_ )),S ),
                        comprimento( S,N ),
                          N == 1 ).
% Invariante Estrutural: não permitir a remoção de serviço caso exista consulta
 ↪ associada
-servico( ID,_,_,_ ) :: (solucoes( (ID), consulta( _,_,ID,_ ),S ),
                         comprimento( S,N ),
                           N == 0 ).
% Invariante Estrutural: não permitir a inserção de consulta repetidoa
+consulta( X,Y,Z,W ) :: (solucoes( (X,Y,Z,W),(consulta( X,Y,Z,W )),S ),
                          comprimento( S,N ),
                            N == 1 ).
% Invariante Estrutural: não permitir a inserção de custo negativo em consulta
+consulta( _,_,_,C ) :: (solucoes( (C),consulta( _,_,_,C ),S ),
                            naoNegativo( S )).
% Invariante Estrutural: não permitir a inserção de consulta com utente inexistente
+consulta( _,ID,_,_,_ ) :: (solucoes( (ID),(utente(ID,_,_,_ )),S ),
```

```
                                contem( ID,S )).
% Invariante Estrutural: não permitir a inserção de consulta com servico inexistente
+consulta( _,_,ID,_,_ ) :: (solucoes( (ID),(servico( ID,_,_,_ )),S ),
                                contem( ID,S )).
```

Definimos também os factos da base de conhecimento que são pedidos no enunciado, que por definição serão sempre verdadeiros, sendo estes:

utente - Identificado por um Nome do utente, Idade e Cidade;

servico - Identificado por Descrição do serviço , Instituição prestadora e Cidade;

consulta - Identificado por Data da consulta, Custo do serviço (cobrado) e referências para o serviço prestado e o utente em questão.

Os identificadores que precedem em *utente* e *servico* são os mesmos usados como referência em *consulta*.

```
% Utente: #idUt, Nome, Idade, Cidade -> {V,F}
utente(1, jose, 22, braga).
utente(2, rafael, 22, chaves).
utente(3, afonso, 22, braga).
utente(4, duarte, 23, braga).
utente(5, luis, 34, porto).
utente(6, manuela, 25, faro).
utente(7, maria, 34, algarve).
utente(8, sofia, 43, lisboa).
utente(9, daniel, 56, aveiro).
utente(10, rui, 21, barcelos).

% Serviço: #idServ, Descrição, Instituição, Cidade -> {V,F}
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
servico(5, oftalmologia, hospitalPorto, porto).
servico(6, cardiologia, hospitalPorto, porto).
servico(7, ortopedia, hospitalPorto, porto).
servico(8, otorrinolaringologia, hospitalPorto, porto).
servico(9, cirugia, hospitalLisboa, lisboa).
servico(10, podologia, hospitalLisboa, lisboa).
servico(11, ortopedia, hospitalLisboa, lisboa).
servico(12, cardiologia, hospitalLisboa, lisboa).

% Consulta: Data, #idUt, #idServ, Custo -> {V,F}
consulta(23-02-2016, 1, 3, 23).
consulta(23-02-2016, 2, 4, 42).
consulta(20-01-2017, 4, 5, 54).
consulta(02-09-2014, 9, 2, 21).
```

```
consulta(31-01-2012, 10, 9, 34).
consulta(29-11-2014, 2, 8, 12).
consulta(20-05-2018, 5, 12, 55).
consulta(19-03-2017, 4, 7, 33).
consulta(21-02-2012, 5, 8, 12).
consulta(13-03-2018, 4, 10, 90).
consulta(31-03-2014, 9, 12, 39).
consulta(05-08-2014, 6, 2, 54).
consulta(09-10-2013, 3, 3, 54).
consulta(21-04-2018, 2, 11, 35).
consulta(26-04-2010, 9, 2, 29).
consulta(30-01-2012, 3, 7, 43).
consulta(18-02-2016, 5, 6, 67).
consulta(15-05-2016, 8, 5, 65).
consulta(03-12-2016, 9, 8, 24).
consulta(07-11-2016, 10, 9, 42).
consulta(25-12-2016, 4, 1, 25).
```

Durante a resolução foram usados predicados que, apesar de não responderem diretamente a qualquer problema proposto, serviram de suporte para os predicados que os resolvem e para algumas funcionalidades adicionais. Entre estes há alguns que foram usados com mais regularidade para manipular e fazer diferentes cálculos sobre listas, agrupados no final do programa. Todos estes serão analisados mais à frente em 1. Registar utentes, serviços e consultas e em 2. Remover utentes, serviços e consultas.

```
% remove os elementos repetidos de uma lista
% Extensao do predicado diferentes: L1, L2 -> {V,F}
diferentes( [],[] ).
diferentes( [X|L],[X|NL] ) :- removerElemento( L,X,TL ), diferentes( TL,NL ).
%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% remove um elemento de uma lista
% Extensao do predicado removerElemento: L1, Y, L2 -> {V,F}
removerElemento( [],_,[] ).
removerElemento( [X|L],X,NL ) :- removerElemento( L,X,NL ).
removerElemento( [X|L],Y,[X|NL] ) :- X \== Y, removerElemento( L,Y,NL ).
%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% Soma os elementos de uma lista
% Extensao do predicado somaC : LN,R -> {V,F}
somaC([X],X).
somaC([X|L],R):- somaC(L,RL), R is X+RL.
%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% concatena duas listas
% Extensao do predicado concatenar: L1,L2,L3 -> {V,F}
concatenar( [],L,L ).
concatenar( [H|T],L2,[H|L] ) :- concatenar(T,L2,L).
%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% Calcula o comprimento de uma lista
% Extensao do predicado comprimento: L,R -> {V,F}
comprimento([],0).
```

```
25 comprimento([H|T],R) :- comprimento(T,N), R is N+1.
26 %-------------------------------- - - - - - - - - - - - - - - - -
27 % Testa todos os elementos da lista
28 % Extensão do predicado teste: [R|LR] -> {V,F}
29 teste([]).
30 teste([I|L]) :- I, teste(L).
31 %-------------------------------- - - - - - - - - - - - - - - - -
32 % Verifica se todos os elementos da lista são não negativos
33 % Extensao do predicado naoNegativo: L -> {V,F}
34 naoNegativo([]).
35 naoNegativo([H|T]) :- H>=0, naoNegativo(T).
36 %-------------------------------- - - - - - - - - - - - - - - - -
37 % Verifica se contem um elemento numa dado lista
38 % Extensão do predicado contem: H,[H|T] -> {V, F}
39 contem(H, [H|T]).
40 contem(X, [H|T]) :- contem(X, T).
41 %-------------------------------- - - - - - - - - - - - - - - - -
42 % Faz a media de uma lista
43 % Extensao do predicado media: L,R -> {V,F}
44 media([H|T],S) :- somaC([H|T],S1), comprimento([H|T],S2), S is S1/S2.
```

Funcionalidades

No enunciado deste exercício prático são pedidos predicados para as seguintes funcionalidades:

1. Registar utentes, serviços e consultas;
2. Remover utentes, serviços e consultas;
3. Identificar as instituições prestadoras de serviços;
4. Identificar utentes/serviços/consultas por critérios de seleção;
5. Identificar serviços prestados por instituição/cidade/datas/custo;
6. Identificar os utentes de um serviço/instituição;
7. Identificar serviços realizados por utente/instituição/cidade;
8. Calcular o custo total dos cuidados de saúde por utente/serviço/instituição/data.

1. Registar utentes, serviços e consultas

Objetivo: adicionar registos de utentes, serviços e consultas à base de conhecimento.
Com este predicado todas as tentativas de inserção de conhecimento passam primeiro por um teste aos invariantes estruturas, permitindo apenas as alterações à base de conhecimento que os preservam. Um teste importante é verificar se o conhecimento a ser inserido já existe e, se for esse o caso, não permitir a inserção, ou seja, quando um utilizador (idUt) ou serviço (idServ) já estão associados a conhecimento de *utente* e *servico* respectivamente, esse identificador não pode voltar a ser inserido. O mesmo acontece com os pares de idUt e idServ em *consulta* iguais.
Com os três últimos invariantes estruturais apresentados na subsecção Base de Conhecimento conseguimos garantir que uma consulta só é inserida caso o utente (idUt) e o serviço (idServ) existam na base de conhecimento, e que o preço da consulta seja um valor não negativo.

```
1  %--------------------------------- - - - - - - - - - -  -  -  -  -  -
2  %---------- 1. Registar utentes, serviços e consultas   -  -  -  -  -
3  %--------------------------------- - - - - - - - - - -  -  -  -  -  -
4  % Faz a insercao de conhecimento
5  % Extensão do predicado que permite a evolucao do conhecimento: Termo -> {V,F}
6  evolucao( Termo ) :- solucoes( Invariante,+Termo::Invariante,Lista ), insercao( Termo
     ↪  ), teste( Lista ).
7  insercao( Termo) :- assert( Termo ).
8  insercao( Termo) :- retract( Termo ),!,fail.
9  %Recordemos o predicado auxiliar teste(L) em anexo.
10 %teste([]).
11 %teste([I|L]) :- I, teste(L).
```

A base de conhecimento foi evoluída para o seguinte:

```
utente(1, jose, 22, braga).
utente(2, rafael, 22, chaves).
utente(3, afonso, 22, braga).
utente(4, duarte, 23, braga).
utente(5, luis, 34, porto).
utente(6, manuela, 25, faro).
utente(7, maria, 34, algarve).
utente(8, sofia, 43, lisboa).
utente(9, daniel, 56, aveiro).
utente(10, rui, 21, barcelos).
utente(11,daniel,33,trofa).
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
servico(5, oftalmologia, hospitalPorto, porto).
servico(6, cardiologia, hospitalPorto, porto).
servico(7, ortopedia, hospitalPorto, porto).
servico(8, otorrinolaringologia, hospitalPorto, porto).
servico(9, cirugia, hospitalLisboa, lisboa).
servico(10, podologia, hospitalLisboa, lisboa).
servico(11, ortopedia, hospitalLisboa, lisboa).
servico(12, cardiologia, hospitalLisboa, lisboa).
servico(13,cardiologia,hospitalTrofa,trofa).
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
consulta(23-02-2016, 1, 3, 23).
consulta(23-02-2016, 2, 4, 42).
consulta(20-01-2017, 4, 5, 54).
consulta(02-09-2014, 9, 2, 21).
consulta(31-01-2012, 10, 9, 34).
consulta(29-11-2014, 2, 8, 12).
consulta(20-05-2018, 5, 12, 55).
consulta(19-03-2017, 4, 7, 33).
consulta(21-02-2012, 5, 8, 12).
consulta(13-03-2018, 4, 10, 90).
consulta(31-03-2014, 9, 12, 39).
consulta(05-08-2014, 6, 2, 54).
consulta(09-10-2013, 3, 3, 54).
consulta(21-04-2018, 2, 11, 35).
consulta(26-04-2010, 9, 2, 29).
consulta(30-01-2012, 3, 7, 43).
consulta(18-02-2016, 5, 6, 67).
consulta(15-05-2016, 8, 5, 65).
consulta(03-12-2016, 9, 8, 24).
consulta(07-11-2016, 10, 9, 42).
consulta(25-12-2016, 4, 1, 25).
consulta(25-12-2018,4,5,10).
```

```
| ?- evolucao(utente(11,daniel,33,trofa)).
yes
| ?- evolucao(servico(13,cardiologia,hospitalTrofa,trofa)).
yes
| ?- evolucao(consulta(25-12-2018,4,5,20)).
yes
```

Figura 1: Resultados de evolução de conhecimento

2. Remover utentes, serviços e consultas

Objetivo: remover registos de utentes, serviços e consultas da base de conhecimento.
Este predicado é o oposto do apresentado anteriormente, tratando agora da remoção de conhecimento. De forma análoga à evolução, a involução da base de conhecimento terá invariantes estruturais a garantir que, para a remoção de consultas poder ser feita, tanto o utente (idUt) como o serviço (idServ) devem estar na base de conhecimento.

```
%-------------------------------- - - - - - - - - - - - - - - - - -
%--------- 2. Remover utentes, serviços e consultas - - - - - - -
%-------------------------------- - - - - - - - - - - - - - - - - -
% Faz a removoção de conhecimento
% Extensão do predicado que permite a involucao do conhecimento: Termo -> {V,F}
involucao( Termo ) :- solucoes( Invariante, -Termo::Invariante, Lista ), remocao(
 ↪  Termo ), teste( Lista ).
remocao( Termo ):- retract( Termo ).
remocao( Termo ):- assert( Termo ),!,fail.
%Recordemos o predicado auxiliar teste(L) em anexo.
%teste([]).
%teste([I|L]) :- I, teste(L).
```

Para mostrar as diferentes possibilidades entre presença e ausência fomos escolher casos que o evidenciem. Tanto o utente e o serviço não estão na base de conhecimento evoluída, mas a consulta é uma da base de conhecimento original. Os predicados foram testados com a base de conhecimento resultante das evoluções anteriores, tendo sido involuída para:

```
utente(1, jose, 22, braga).
utente(2, rafael, 22, chaves).
utente(3, afonso, 22, braga).
utente(4, duarte, 23, braga).
utente(5, luis, 34, porto).
utente(6, manuela, 25, faro).
utente(7, maria, 34, algarve).
utente(8, sofia, 43, lisboa).
utente(9, daniel, 56, aveiro).
utente(10, rui, 21, barcelos).
utente(11,daniel,33,trofa).
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
servico(5, oftalmologia, hospitalPorto, porto).
servico(6, cardiologia, hospitalPorto, porto).
servico(7, ortopedia, hospitalPorto, porto).
servico(8, otorrinolaringologia, hospitalPorto, porto).
servico(9, cirugia, hospitalLisboa, lisboa).
servico(10, podologia, hospitalLisboa, lisboa).
servico(11, ortopedia, hospitalLisboa, lisboa).
servico(12, cardiologia, hospitalLisboa, lisboa).
servico(13,cardiologia,hospitalTrofa,trofa).
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
consulta(23-02-2016, 2, 4, 42).
consulta(20-01-2017, 4, 5, 54).
consulta(02-09-2014, 9, 2, 21).
consulta(31-01-2012, 10, 9, 34).
consulta(29-11-2014, 2, 8, 12).
consulta(20-05-2018, 5, 12, 55).
consulta(19-03-2017, 4, 7, 33).
consulta(21-02-2012, 5, 8, 12).
consulta(13-03-2018, 4, 10, 90).
consulta(31-03-2014, 9, 12, 39).
consulta(05-08-2014, 6, 2, 54).
consulta(09-10-2013, 3, 3, 54).
consulta(21-04-2018, 2, 11, 35).
consulta(26-04-2010, 9, 2, 29).
consulta(30-01-2012, 3, 7, 43).
consulta(18-02-2016, 5, 6, 67).
consulta(15-05-2016, 8, 5, 65).
consulta(03-12-2016, 9, 8, 24).
consulta(07-11-2016, 10, 9, 42).
consulta(25-12-2016, 4, 1, 25).
```

```
| ?- involucao(utente(11,joel,44,braga)).
yes
| ?- involucao(servico(14,cardiologia,hospitalTrofa,trofa)).
yes
| ?- involucao(consulta(23-02-2016,1,3,23)).
yes
```

Figura 2: Resultados de involução de conhecimento

3. Identificar as instituições prestadoras serviços

Para fazer a listagem foi preciso criar um predicado listarInstituicoes, que irá apresentar a lista de todas as instituições. Para conseguirmos encontrar estas instituições recorrermos a outro predicado chamado solucoes que procura em servicos e recolhe uma lista de todas as Instituicoes, aonde depois procedemos à transformação desta numa lista sem repetições com o predicado diferentes.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
%----- 3. Identificar as instituicoes prestadoras de servicos - - -
%--------------------------------- - - - - - - - - - - - - - - - - -
% Identifica instituições prestadoras de Serviços
% Extensao do predicado listarInstituicoes: ListaDeResultados -> {V,F}
listarInstituicoes( L ) :- solucoes( Instituicao,servico(X,Y,Instituicao,W),R ),
 ↪  diferentes(R,L).
%ver diferentes(R,L) em anexo.
```

Os exemplos foram feitos para usar o seguinte conhecimento e na respetiva ordem:

```
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
servico(5, oftalmologia, hospitalPorto, porto).
servico(6, cardiologia, hospitalPorto, porto).
```

```
servico(7, ortopedia, hospitalPorto, porto).
servico(8, otorrinolaringologia, hospitalPorto, porto).
servico(9, cirugia, hospitalLisboa, lisboa).
servico(10, podologia, hospitalLisboa, lisboa).
servico(11, ortopedia, hospitalLisboa, lisboa).
servico(12, cardiologia, hospitalLisboa, lisboa).
```

```
| ?- listarInstituicoes(L).
L = [hospitalBraga,hospitalPorto,hospitalLisboa] ?
yes
```

Figura 3: Resultados da listagem de instituições

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

Cada tipo de conhecimento tem predicados próprios para os diferentes critérios de seleção Para identificar os utentes por critérios de seleção decidimos usar predicados que recebem para cada caso de procura a característica correspondente, isto é, o identificador pelo qual desejamos encontrar os utentes (idUtente, Nome, Idade, Cidade) recorrendo ao predicado solucoes para procurar os utentes cuja informação corresponda à informação requerida.

```
%-------------------------------- - - - - - - - - - - - - - - - - -
%----- 4. Identificar utentes por critérios de seleção - - - - - -
%-------------------------------- - - - - - - - - - - - - - - - - -
% Identifica utentes pelo Nome
% Extensao do predicado listarUtenteNome: Nome, ListaDeResultados -> {V,F}
listarUtenteNome(Nome,R):- solucoes((X,Nome,Z,W),utente(X,Nome,Z,W),R).
%-------------------------------- - - - - - - - - - - - - - - - - -
% Identifica utentes pela Idade
% Extensao do predicado listarUtenteIdade: Idade, ListaDeResultados -> {V,F}
```

```
listarUtenteIdade(Idade,R):- solucoes((X,Y,Idade,W),utente(X,Y,Idade,W),R).
%------------------------------- - - - - - - - - - - - - - - - -
% Identifica utentes pela Cidade
% Extensao do predicado listarUtenteCidade: Cidade, ListaDeResultados -> {V,F}
listarUtenteCidade(Cidade,R):- solucoes((X,Y,Z,Cidade),utente(X,Y,Z,Cidade),R).
%------------------------------- - - - - - - - - - - - - - - - -
% Identifica utentes por Nome e Idade
% Extensao do predicado listarUtenteNomeIdade: Nome, Idade, ListaDeResultados -> {V,F}
listarUtenteNomeIdade(Nome,Idade,R):-
 ↪ solucoes((X,Nome,Idade,W),utente(X,Nome,Idade,W),R).
%------------------------------- - - - - - - - - - - - - - - - -
% Identifica utentes por Nome e Cidade
% Extensao do predicado listarUtenteNomeCidade: Nome, Cidade, ListaDeResultados->{V,F}
listarUtenteNomeCidade(Nome,Cidade,R):-
 ↪ solucoes((X,Nome,Z,Cidade),utente(X,Nome,Z,Cidade),R).
%------------------------------- - - - - - - - - - - - - - - - -
% Identifica utentes por idade e cidade
% Extensao do predicado listarUtenteIdadeCidade: Idade, Cidade,
 ↪ ListaDeResultados->{V,F}
listarUtenteIdadeCidade(Idade,Cidade,R):-
 ↪ solucoes((X,Y,Idade,Cidade),utente(X,Y,Idade,Cidade),R).
%------------------------------- - - - - - - - - - - - - - - - -
% Identifica utentes por nome, idade e cidade
% Extensao do predicado listarUtenteNomeIdadeCidade: Nome, Idade, Cidade,
 ↪ ListaDeResultados->{V,F}
listarUtenteNomeIdadeCidade(Nome,Idade,Cidade,R):-
 ↪ solucoes((X,Nome,Idade,Cidade),utente(X,Nome,Idade,Cidade),R).
%------------------------------- - - - - - - - - - - - - - - - -
% Identifica utentes por id
% Extensao do predicado listarUtenteID: IdUtente,ListaDeResultados -> {V,F}
listarUtenteIdUtente(IdUtente,R):-
 ↪ solucoes((IdUtente,N,I,C),utente(IdUtente,N,I,C),R).
```

Os exemplos foram feitos para usar o seguinte conhecimento e na respetiva ordem:

```
utente(2, rafael, 22, chaves).
%------------------------------- - - - - - - - - - - - - - - - -
utente(3, afonso, 22, braga).
%------------------------------- - - - - - - - - - - - - - - - -
utente(1, jose, 22, braga).
%------------------------------- - - - - - - - - - - - - - - - -
utente(9, daniel, 56, aveiro).
```

```
| ?- listarUtenteNome(rafael,R).
R = [(2,rafael,22,chaves)] ?
yes
| ?- listarUtenteNomeIdade(afonso,22,R).
R = [(3,afonso,22,braga)] ?
yes
| ?- listarUtenteNomeIdadeCidade(jose,22,braga,R).
R = [(1,jose,22,braga)] ?
yes
| ?- listarUtenteIdUtente(9,R).
R = [(9,daniel,56,aveiro)] ?
yes
```

Figura 4: Resultados das diferentes formas de procurar utentes

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

Para diferenciar um serviço prestado do realizado, nós assumimos um serviço prestado como um disponível aos utentes, enquanto que um realizado foi um que teve consulta. No caso da procura por data assumimos que em datas passadas um serviço só foi prestado se também teve consulta. Os serviços podem ser identificados pelos diferentes critérios com os seguintes predicados.

```
%--------------------------------- - - - - - - - - - - - - - - - - - - - -
%-- 5. Identificar serviços realizados por critérios de seleção
%--------------------------------- - - - - - - - - - - - - - - - - - - - -
% Lista os servicos realizados por instituição
% Extensao do predicado servicosPorInstituicao: Instituicao, ListaDeResultados ->
 ↪  {V,F}
servicosPorInstituicao( I,R ):- solucoes(( ID,D,I,C ), servico( ID,D,I,C ), R).
%--------------------------------- - - - - - - - - - - - - - - - - - - - -
% Lista os servicos realizados por cidade
% Extensao do predicado servicosPorCidade: Cidade, ListaDeResultados -> {V,F}
servicosPorCidade( C,R ):- solucoes(( ID,D,I,C ), servico( ID,D,I,C ), R).
%--------------------------------- - - - - - - - - - - - - - - - - - - - -
% Lista os servicos realizados por data
% Extensao do predicado servicosPorData: Data, ListaDeResultados -> {V,F}
servicosPorData( D,R ):- solucoes(IdS, consulta(D,IdU,IdS,C ), S),
 ↪  servicosPorDataRec(S,R).
servicosPorDataRec([],[]).
servicosPorDataRec([H|T],R):- solucoes(( H,D,I,C ),servico( H,D,I,C ), L1),
 ↪  servicosPorDataRec(T,L2), concatenar(L1,L2,R).
```

```
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Lista os servicos realizados por custo
% Extensao do predicado servicosPorCusto: Custo, ListaDeResultados -> {V,F}
servicosPorCusto( C,R ):- solucoes(IdS, consulta(D,IdU,IdS,C ), S),
servicosPorCustoRec(S,R).
servicosPorCustoRec([],[]).
servicosPorCustoRec([H|T],R):- solucoes(( H,D,I,Cid ),servico( H,D,I,Cid ), L1),
 ↪  servicosPorCustoRec(T,L2), concatenar(L1,L2,R).
```

Os exemplos foram feitos para usar o seguinte conhecimento e na respetiva ordem:

```
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
consulta(23-02-2016, 1, 3, 23).
consulta(23-02-2016, 2, 4, 42).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
consulta(03-12-2016, 9, 8, 24).
servico(8, otorrinolaringologia, hospitalPorto, porto).
```

```
| ?- servicosPorInstituicao(hospitalBraga,R).
R = [(1,ortopedia,hospitalBraga,braga),(2,cardiologia,hospitalBraga,braga),(3
    ,clinicaGeral,hospitalBraga,braga),(4,psiquiatria,hospitalBraga,braga)] ?
yes
| ?- servicosPorCidade(braga,R).
R = [(1,ortopedia,hospitalBraga,braga),(2,cardiologia,hospitalBraga,braga),(3
    ,clinicaGeral,hospitalBraga,braga),(4,psiquiatria,hospitalBraga,braga)] ?
yes
| ?- servicosPorData(23-02-2016,R).
R = [(3,clinicaGeral,hospitalBraga,braga),(4,psiquiatria,hospitalBraga,braga)]
    ?
yes
| ?- servicosPorCusto(24,R).
R = [(8,otorrinolaringologia,hospitalPorto,porto)] ?
yes
| ?-
```

Figura 5: Resultados das diferentes formas de procurar serviços

6. Identificar os utentes de um serviço/instituição

Os utentes podem ser identificados com os seguintes predicados.

```
%--------------------------------- - - - - - - - - - - - - - - - - - - - - - -
%-- 6. Identificar os utentes de um serviço/instituição
%--------------------------------- - - - - - - - - - - - - - - - - - - - - - -
%---------------------------------- - - - - - - - - - - - - - - - - - -
% Fornece os utentes de uma instituição
% Extensao do predicado utentesDeInstituicao: P,U -> {V,F}
utentesDeInstituicao(P,U) :- solucoes(ID,servico(ID,DE,P,CI),NL), diferentes(NL,OUT),
 ↪  utInsAux(OUT,U).
utInsAux([],[]).
utInsAux([H|T],S) :- solucoes((H,Y,W,Z),utente(H,Y,W,Z),NL), concatenar(N,NL,S),
 ↪  utInsAux(T,N).
%---------------------------------- - - - - - - - - - - - - - - - - - -
% Fornece os Ids das instituições que disponibilizam um dado serviço
% Extensao do predicado instituicaoServico: Serviço,Instituição -> {V,F}
instituicaoServico(S,I) :- solucoes(ID, servico(ID,S,INS,CI), I).
%---------------------------------- - - - - - - - - - - - - - - - - - -
% Fornece os Ids dos utentes de um serviço
% Extensao do predicado utentesEspecialidade: I,S -> {V,F}
utentesServico(S,U) :- instituicaoServico(S,P), utentesHospital(P,UT),
 ↪  diferentes(UT,OUT), utenServAux(OUT,U).
utenServAux([],[]).
```

```
19 utenServAux([ID|T],S) :- solucoes((ID,NO,I,CI),utente(ID,NO,I,CI),NL),
    ↪ concatenar(N,NL,S), utenServAux(T,N).
20 %-------------------------------- - - - - - - - - - - - - - - - - -
21 % Fornece a lista dos Ids dos utentes que receberam cuidados dos hospitais cujos Ids
    ↪ estão contidos no Input
22 % Extensao do predicado utentesHospital: ListaInstituição,ListaUtentes -> {V,F}
23 utentesHospital([],[]).
24 utentesHospital([H|T],S) :- solucoes(X,consulta(A,X,H,D),NL), concatenar(N,NL,S),
    ↪ utentesHospital(T,N).
```

Os exemplos foram feitos para usar o seguinte conhecimento e na respetiva ordem:

```
1  servico(1, ortopedia, hospitalBraga, braga).
2  servico(2, cardiologia, hospitalBraga, braga).
3  servico(3, clinicaGeral, hospitalBraga, braga).
4  servico(4, psiquiatria, hospitalBraga, braga).
5  consulta(23-02-2016, 1, 3, 23).
6  consulta(23-02-2016, 2, 4, 42).
7  consulta(09-10-2013, 3, 3, 54).
8  consulta(25-12-2016, 4, 1, 25).
9  utente(1, jose, 22, braga).
10 utente(2, rafael, 22, chaves).
11 utente(3, afonso, 22, braga).
12 utente(4, duarte, 23, braga).
13 %-------------------------------- - - - - - - - - - - - - - - - - -
14 servico(2, cardiologia, hospitalBraga, braga).
15 servico(6, cardiologia, hospitalPorto, porto).
16 servico(12, cardiologia, hospitalLisboa, lisboa).
17 consulta(02-09-2014, 9, 2, 21).
18 consulta(20-05-2018, 5, 12, 55).
19 consulta(31-03-2014, 9, 12, 39).
20 consulta(05-08-2014, 6, 2, 54).
21 consulta(18-02-2016, 5, 6, 67).
22 utente(5, luis, 34, porto).
23 utente(6, manuela, 25, faro).
24 utente(9, daniel, 56, aveiro).
```

```
| ?- utentesDeInstituicao(hospitalBraga,R).
R = [(4,duarte,23,braga),(3,afonso,22,braga),(2,rafael,22,chaves),(1,jose,22
    ,braga)] ?
yes
| ?-
| ?- utentesServico(cardiologia,R).
R = [(6,manuela,25,faro),(9,daniel,56,aveiro),(5,luis,34,porto)] ?
yes
| ?-
```

Figura 6: Resultados das diferentes formas de procurar utentes de um dado serviço ou instituição

7. Identificar serviços realizados por utente/instituição/cidade

Os serviços podem ser identificados com os seguintes predicados.

```
%--------------------------------- - - - - - - - - - - - - - - - - - - - - -
%-- 7. Identificar serviços realizados por utente/instituição/cidade}
%--------------------------------- - - - - - - - - - - - - - - - - - - - - -
% Dado um ID de utente, indica a lista dos serviços que já realizou
% Extensao do predicado servicosRealizadosUtente : Utente,Servicos -> {V,F}
servicosRealizadosUtente( Utente,Servico ):- solucoes(IdS,
 ↪ consulta(Data,Utente,IdS,Custo), Consultas),
 ↪ servicosRUtenteAux(Consultas,Servico).
servicosRUtenteAux( [],[] ).
servicosRUtenteAux([ID|T],R):- solucoes((ID,Descricao,Instituicao,Cidade),
 ↪ servico(ID,Descricao,Instituicao,Cidade), L1), servicosRUtenteAux(T,L2),
 ↪ concatenar(L1,L2,R).
%--------------------------------- - - - - - - - - - - - - - - - - - - - - -
% Fornece a lista dos serviços realizados por instituição
% Extensao do predicado servicosRealizadosInstituicao : Instituicao ,R -> {V,F}
servicosRealizadosInstituicao( Instituicao,R ):- solucoes(ID,
 ↪ servico(ID,Descricao,Instituicao,Cidade), S),
 ↪ servicosRealizadosInstituicaoAux(S,R1), servicosRealizadosInstituicaoAux2(R1,R).
servicosRealizadosInstituicaoAuxConsulta( [],[] ).
servicosRealizadosInstituicaoAuxConsulta([H|T],R):- solucoes(H, consulta(D,I,H,C),
 ↪ L1), servicosRealizadosInstituicaoAuxConsulta(T,L2), concatenar(L1,L2,R).
servicosRealizadosInstituicaoAuxServico([],[]).
```

```
servicosRealizadosInstituicaoAuxServico([H|T],R):- solucoes((H,D,I,C),
 ↪ servico(H,D,I,C), L1), servicosRealizadosInstituicaoAuxServico(T,L2),
 ↪ concatenar(L1,L2,R).
%-------------------------------- - - - - - - - - - - - - - - - -
% Fornece a lista dos serviços realizados por cidade
% Extensao do predicado servicosRealizadosCidade : Cidade ,R -> {V,F}
servicosRealizadosCidade( C,R ):- solucoes(IdS, consulta(D,U,IdS,Custo), S),
 ↪ diferentes(S,S1), servicosRCidadeAux(C,S1,R).
servicosRCidadeAux( C,[],[] ).
servicosRCidadeAux( C, [H|T], R):- solucoes((H,D,I,C), servico(H,D,I,C), L1),
 ↪ servicosRCidadeAux(C,T,L2), concatenar(L1,L2,R).
```

Os exemplos foram feitos para usar o seguinte conhecimento e na respetiva ordem:

```
consulta(23-02-2016, 2, 4, 42).
consulta(29-11-2014, 2, 8, 12).
consulta(21-04-2018, 2, 11, 35).
servico(4, psiquiatria, hospitalBraga, braga).
servico(8, otorrinolaringologia, hospitalPorto, porto).
servico(11, ortopedia, hospitalLisboa, lisboa).
%-------------------------------- - - - - - - - - - - - - - - - -
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
consulta(23-02-2016, 1, 3, 23).
consulta(23-02-2016, 2, 4, 42).
consulta(02-09-2014, 9, 2, 21).
consulta(05-08-2014, 6, 2, 54).
consulta(09-10-2013, 3, 3, 54).
consulta(26-04-2010, 9, 2, 29).
consulta(25-12-2016, 4, 1, 25).
%-------------------------------- - - - - - - - - - - - - - - - -
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
```

```
| ?- servicosRealizadosUtente(2,R).
R = [(4,psiquiatria,hospitalBraga,braga),(8
    ,otorrinolaringologia,hospitalPorto,porto),(11
    ,ortopedia,hospitalLisboa,lisboa)] ?
yes
| ?- servicosRealizadosInstituicao(hospitalBraga,R).
R = [(1,ortopedia,hospitalBraga,braga),(2,cardiologia,hospitalBraga,braga),(2
    ,cardiologia,hospitalBraga,braga),(2,cardiologia,hospitalBraga,braga),(3
    ,clinicaGeral,hospitalBraga,braga),(3,clinicaGeral,hospitalBraga,braga),(4
    ,psiquiatria,hospitalBraga,braga)] ?
yes
| ?- servicosRealizadosCidade(braga,R).
R = [(3,clinicaGeral,hospitalBraga,braga),(4,psiquiatria,hospitalBraga,braga),(
    2,cardiologia,hospitalBraga,braga),(1,ortopedia,hospitalBraga,braga)] ?
yes
| ?-
```

Figura 7: Resultados das diferentes formas de identificar serviços

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

Os custos podem ser calculados com os seguintes predicados.

```
%---------------------------------- - - - - - - - - - - - - - - - - - - - - - -
%-- 8. Calcular o custo total dos cuidados de saúde por
 ↪  utente/serviço/instituição/data
%---------------------------------- - - - - - - - - - - - - - - - - - - - - - -
% Custo total associado a um utente
% Extensao do predicado custoUtente: IdUtente,R -> {V,F}
custoUtente(I,R) :- solucoes(C,consulta(D,I,IdS,C),L), somaC(L,R).
%---------------------------------- - - - - - - - - - - - - - - - - - - - - - -
% Custo total associado a um serviço
% Extensao do predicado custoServico: Servico,R -> {V,F}
custoServico( S,R ):- solucoes(C,consulta(D,I,S,C),L), somaC(L,R).
%---------------------------------- - - - - - - - - - - - - - - - - - - - - - -
% Custo total associado a uma instituição
% Extensao do predicado custoInstituicao: Instituicao,R -> {V,F}
custoInstituicao(I,R) :- solucoes(ID,servico(ID,D,I,Cid),L), custoInstituicaoAux(L,Y),
 ↪  somaC(Y,R).
custoInstituicaoAux( [],[] ).
custoInstituicaoAux([H|T],R):- solucoes(C,consulta(D,I,H,C),L1),
 ↪  custoInstituicaoAux(T,L2), concatenar(L1,L2,R).
%---------------------------------- - - - - - - - - - - - - - - - - - - - - - -
% Custo total associado a uma data
```

```
% Extensao do predicado custoData: Data,R -> {V,F}
custoData(D,R):- solucoes(C,consulta(D,I,IdS,C),L), somaC(L,R).
```

Os exemplos foram feitos para usar o seguinte conhecimento e na respetiva ordem:

```
utente(2, rafael, 22, chaves).
consulta(23-02-2016, 2, 4, 42).
consulta(29-11-2014, 2, 8, 12).
consulta(21-04-2018, 2, 11, 35).
%-------------------------------- - - - - - - - - - - - - - - - - -
servico(1, ortopedia, hospitalBraga, braga).
consulta(25-12-2016, 4, 1, 25).
%-------------------------------- - - - - - - - - - - - - - - - - -
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
consulta(23-02-2016, 1, 3, 23).
consulta(23-02-2016, 2, 4, 42).
consulta(02-09-2014, 9, 2, 21).
consulta(05-08-2014, 6, 2, 54).
consulta(09-10-2013, 3, 3, 54).
consulta(26-04-2010, 9, 2, 29).
consulta(25-12-2016, 4, 1, 25).
%-------------------------------- - - - - - - - - - - - - - - - - -
consulta(23-02-2016, 1, 3, 23).
consulta(23-02-2016, 2, 4, 42).
```

```
| ?- custoUtente(2,R).
R = 89 ?
yes
| ?- custoServico(1,R).
R = 25 ?
yes
| ?- custoInstituicao(hospitalBraga,R).
R = 248 ?
yes
| ?- custoData(23-02-2016,R).
R = 65 ?
yes
```

Figura 8: Resultados de calcular os custos com cuidados de saúde segundo os diferentes critérios

Predicados Auxiliares

Predicado Comprimento

O calculo é feito de forma recursiva, usando um "acumulador"para aumentar o resultado em uma unidade cada vez que existir um elemento à cabeça da lista, sendo esse valor o resultado pretendido.

```
1 % Calcula o comprimento de uma lista
2 % Extensao do predicado comprimento: L,R -> {V,F}
3 comprimento([],0).
4 comprimento([H|T],R) :- comprimento(T,N), R is N+1.
```

Predicado Teste

O predicado de teste verifica se os elementos de uma dada lista são todos verdadeiros, útil para podermos usar nos predicados de evolucao e involucao, para garantir que não há inserção nem remoção de conhecimento que coloque a base de conhecimento num estado inconsistente, apensas permite inserções e remoções que validam os invariantes estruturais.

```
1 % Testa todos os elementos da lista
2 % Extensão do predicado teste: [R|LR] -> {V,F}
3 teste([]).
4 teste([I|L]) :- I, teste(L).
```

Predicado contem

O predicado irá verificar de uma dada lista irá conter um certo elemento, útil para a criação dos invariantes estruturais.

```
% Verifica se contem um elemento numa dado lista
% Extensão do predicado contem: H,[H|T] -> {V, F}
contem(H, [H|T]).
contem(X, [H|T]) :- contem(X, T).
```

Predicado Não negativo

Através deste predicado podemos verificar se todos os elementos de uma dada lista serão não negativos, ou seja, todos eles maiores ou iguais a zero.

```
% Verifica se todos os elementos da lista são não negativos
% Extensao do predicado naoNegativo: L -> {V,F}
naoNegativo([]).
naoNegativo([H|T]) :- H>=0, naoNegativo(T).
```

Diferentes

Este predicado tem como função remover elementos repetidos de uma lista, para isso irá usar recursividade e um predicado auxiliar denotado removerElemnto, a ser exposto imediatamente abaixo.

```
% remove os elementos repetidos de uma lista
% Extensao do predicado diferentes: L1, L2 -> {V,F}
diferentes( [],[] ).
diferentes( [X|L],[X|NL] ) :- removerElemento( L,X,TL ), diferentes( TL,NL ).
```

Remover elemento

Remove todas as instâncias de um elemento dado de uma lista se a lista o tiver.

```
% remove um elemento de uma lista
% Extensao do predicado removerElemento: L1, Y, L2 -> {V,F}
removerElemento( [],_,[] ).
removerElemento( [X|L],X,NL ) :- removerElemento( L,X,NL ).
removerElemento( [X|L],Y,[X|NL] ) :- X \== Y, removerElemento( L,Y,NL ).
```

Somatório de uma lista

Cálculo recursivo do valor da soma de todos os elementos dessa lista.

```
% Soma os elementos de uma lista
% Extensao do predicado somaC : LN,R -> {V,F}
somaC([X],X).
somaC([X|L],R):- somaC(L,RL), R is X+RL.
```

Concatenar duas listas

Este predicado é responsável por concatenar recursivamente duas listas numa única lista, mantendo todos os argumentos.

```
% concatena duas listas
% Extensao do predicado concatenar: L1,L2,L3 -> {V,F}
concatenar( [],L,L ).
concatenar( [H|T],L2,[H|L] ) :- concatenar(T,L2,L).
```

Média de uma lista

O predicado calcula o valor da média de uma dada lista, usando predicados auxiliares comprimento e somaC já explicados anteriormente.

```
% Faz a media de uma lista
% Extensao do predicado media: L,R -> {V,F}
media([H|T],S) :- somaC([H|T],S1), comprimento([H|T],S2), S is S1/S2.
```

Novas Funcionalidades

Nesta secção iremos explicar e demonstrar as funcionalidades novas que acrescentamos.

Média de idades dos utentes

Para a construção deste predicado usamos outros dois já definidos: o primeiro denotado solucoes que irá devolver uma lista com os parâmetros definidos e o outro predicado é chamado media, que nos calcula a media de uma lista. Sendo assim é nos possível saber qual a idade media que os utentes possuem na nossa base de conhecimento.

```
1 % Média de idade de todos os utentes
2 % Extensao do predicado mediaIdadesUtentes: R -> {V,F}
3 mediaIdadeUtentes( R ) :- solucoes( I,utente(_,_,I,_),S ), media( S,R ).
```

```
| ?- mediaIdadeUtentes( R ).
R = 30.2 ?
yes
```

Figura 9: Resultado do predicado extra que calcula a média de idade dos utentes

Média de idades dos utentes por morada

Este problema pode ser dividido em duas fases distintas: primeiro é preciso procurar e construir a lista de utentes cuja morada corresponda à argumento, e por fim calcular a média com o predicado media que nos indicará a média de idade dos utentes agrupados por morada.

```
1 % Dá a media de idades correspondentes aos utentes com a morada dada
2 % Extensao do predicado mediaIdades_morada: Morada,R -> {V,F}
3 mediaIdadePorMorada( M,R ) :- solucoes( ID, utente(_,_,ID,M),S ), media( S,R ).
```

```
| ?- listing(utente).
utente(1, jose, 22, braga).
utente(2, rafael, 22, chaves).
utente(3, afonso, 22, braga).
utente(4, duarte, 23, braga).
utente(5, luis, 34, porto).
utente(6, manuela, 25, faro).
utente(7, maria, 34, algarve).
utente(8, sofia, 43, lisboa).
utente(9, daniel, 56, aveiro).
utente(10, rui, 21, barcelos).

yes
| ?- mediaIdadePorMorada( braga,R ).
R = 22.333333333333332 ?
yes
```

Figura 10: Resultado do predicado extra que calcula a média de idade dos utentes numa mesma morada

Número de serviços correspondentes a uma instituição

Este predicado irá usar dois predicados para a sua construção: o solucoes para construir uma lista com os serviços de uma dada instituição, e depois o predicado comprimento para calcular o comprimento dessa lista, dando assim o número total de serviços que a instituitição oferece.

```
1 % Dá o numero de servicos correspondente a uma instituiçao
2 % Extensao do predicado servicosNporInstituicao: Instituicao,R -> {V,F}
3 servicosNPorInstituicao( I,R ) :- solucoes( I, servico(_,_,I,_),S ), comprimento( S,R
    ↪ ).
```

```
| ?- listing(servico).
servico(1, ortopedia, hospitalBraga, braga).
servico(2, cardiologia, hospitalBraga, braga).
servico(3, clinicaGeral, hospitalBraga, braga).
servico(4, psiquiatria, hospitalBraga, braga).
servico(5, oftalmologia, hospitalPorto, porto).
servico(6, cardiologia, hospitalPorto, porto).
servico(7, ortopedia, hospitalPorto, porto).
servico(8, otorrinolaringologia, hospitalPorto, porto).
servico(9, cirugia, hospitalLisboa, lisboa).
servico(10, podologia, hospitalLisboa, lisboa).
servico(11, ortopedia, hospitalLisboa, lisboa).
servico(12, cardiologia, hospitalLisboa, lisboa).

yes
| ?- servicosNPorInstituicao( hospitalBraga,R ).
R = 4 ?
yes
```

Figura 11: Resultado do predicado que que calcula quantos serviços existem na instituição

Número de utentes por instituição

Para mostrar o número de Utentes que uma Instituição possui, este predicado irá utilizar o predicado já definido utentesDeInstituicao, que nos constrói uma lista com o conjunto dos utentes de uma dada instituição. Calculando o comprimento da lista obtemos o número de utentes.

```
1 % Fornece o número de utentes por instituição
2 % Extensao do predicado nUtentesInstituicao: Instituicao,R -> {V,F}
3 utentesNPorInstituicao( I,R ) :- utentesDeInstituicao( I,L ), comprimento( L,R ).
```

```
| ?- utentesNPorInstituicao( hospitalPorto,R ).
R = 4 ?
yes
```

Figura 12: Resultado do predicado extra que calcula quantos utentes foram à instituição

Número de consultas por instituição

O cálculo pode ser feito com um predicado parecido aos dois predicados anteriores, diferindo apenas num predicado auxiliar. Depois de construir uma lista de identificadores de serviços de uma dada especialidade com o predicado *solucoes*, usamos o predicado consultasPorIDsServico que nos irá construir uma lista de consultas de uma dado serviço. Calculando o comprimento dessa lista temos o número de consultas da especialidade.

```
% Fornece o número de consultas por instituicao
% Extensao do predicado consultasNPorInstituicao Instituicao,R -> {V,F}
consultasNPorInstituicao( I,R ) :- solucoes( ID, servico(ID,_,I,_),S ),
 ↪  consultasPorServico( S,L ), comprimento( L,R ).
consultasPorIDsServico( [],[] ).
consultasPorIDsServico( [H|T],R ) :- solucoes((D,B,H,C), consulta(D,B,H,C),L1 ),
 ↪  consultasPorIDsServico( T,L2 ), concatenar( L1,L2,R ).
```

```
| ?- consultasNPorInstituicao( hospitalBraga,R ).
R = 7 ?
yes
```

Figura 13: Resultado do predicado extra que calcula quantas consultas foram prestadas na instituição

Número de consultas por especialidade

De forma análoga ao predicado anterior, este predicado irá verificar quantas consultas de uma determinada especialidade foram efetuadas, usando para isso vários predicados auxiliares para nos ajudar com essa tarefa. Depois de construir uma lista de identificadores de serviços de uma dada especialidade com o predicado *solucoes*, usamos novamente o predicado consultasPorIDsServico da funcionalidade anterior (constrói uma lista de consultas de uma dado serviço). Calculando o comprimento dessa lista temos o número de consultas da especialidade.

```
% Fornece o número de consultas por especialidade
% Extensao do predicado consultasNPorEspecialidade Especialidade,Resultado -> {V,F}
% Serviço: #idServ, Descrição, Instituição, Cidade -> {V,F}
% Consulta: Data, #idUt, #idServ, Custo -> {V,F}
consultasNPorEspecialidade( E,R ) :- solucoes( ID, servico(ID,E,I,C),S ),
 ↪  consultasPorIDsServico( S,L ), comprimento( L,R ).
```

```
| ?- consultasNPorEspecialidade( cardiologia,R ).
R = 6 ?
yes
```

Figura 14: Resultado do predicado extra que calcula quantas consultas foram prestadas de uma dada especialidade

Conclusão e Trabalho Futuro

Por natureza daquilo que nos foi pedido, nesta primeira fase desenvolvemos a base de conhecimento com algumas limitações devidas aos pressupostos e leis tomadas, tais como o pressuposto de mundo fechado e a lei do terceiro excluído, que nos impedem de representar de forma mais fiel um sistema de cuidados de saúde. Porém, o trabalho feito pode servir de base para uma versão melhorada, que permita representar o desconhecido no contra-domínio através de extensões à programação à lógica usada, podendo depois representar coisas como faltas às consultas e não apenas 'não-presenças' graças à capacidade de colocar provas de que algo é falso.