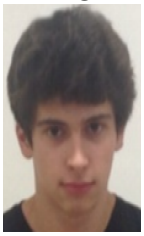
Exercício Prático Nº 1 - Grupo 3 - Sistemas de Representação de Conhecimento e Raciocínio

André Rodrigues Freitas



A74619

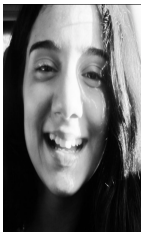
João Tiago Pereira Dias

A72095

Joel Tomás Morais

A70841

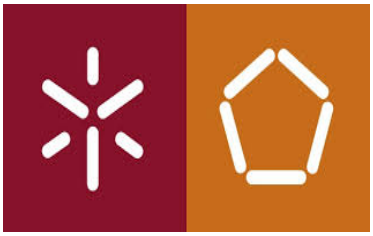
Sofia Manuela Gomes de Carvalho



A76658

19 de Março de 2017

Universidade do Minho



Resumo

Neste exercício prático, utilizamos a linguagem de programação em lógica PROLOG para desenvolver um sistema de representação de conhecimento e raciocínio que permite caraterizar o universo de discurso na área da prestação de cuidados de saúde.

Conteúdo

1 Introdução

No primeiro exercício prático, recorremos ao PROLOG para desenvolver um sistema de representação de conhecimento e raciocínio para caraterizar o universo de discurso na área da prestação de cuidados de saúde. Para tal, respeitamos o panorama sugerido no enunciado constituído por utentes, cuidados prestados e atos médicos e implementamos várias funcionalidades, funcionalidades essas, na nossa opinião, úteis para a realização e controlo de serviços médicos, que serão visitadas detalhadamente no decorrer deste mesmo documento.

2 Preliminares

Este trabalho prático surge no seguimento de um pedido do Serviço Nacional de Saúde (SNS) para armazenar e tratar dados relativos a várias aspetos integrantes do SNS através da implementação de várias funcionalidades cruciais que nos foram apresentadas. Alguns exemplos dessas funcionalidades são registar e remover utentes ou calcular os custo total dos atos médicos por instituição ou por data.

3 Descrição do Trabalho e Análise de Resultados

No desenvolvimento do código para este primeiro exercício prático, começamos por definir a quantidade de argumentos que define cada conhecimento, utilizando para isso o *dynamic*. Através do enunciado do exercício, definimos que o utente tem associado a si quatro parâmetros, os cuidados têm também quatro parâmetros que os definem e o mesmo acontece com atoMedico.

```
:- dynamic utente/4.
:- dynamic cuidados/4.
:- dynamic atoMedico/4.
:- op( 900,xfy,'::' ).
```

Figura 1: Definições inicias

Após essa definição, alteramos três *flags* do PROLOG para evitar ter *warnings* ou erros aquando da compilação do ficheiro usado para o desenvolvimento deste sistema, podendo assim ser executado corretamente e de acordo com o tema desta fase da disciplina.

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

Figura 2: Declarações inicias

De seguida, definimos invariantes estruturais e referenciais que se encontram dispostos nas figuras abaixo. Os invariantes estruturais dizem respeito apenas a operações de inserção de conhecimento no sistema, enquanto que os invariantes referenciais são usados quer em operações de inserção quer em operações de remoção.

```
% Invariantes Estruturais: não permitir a inserção de conhecimento repetido

% Utentes não repetidos
+utente( IU,N,ID,M ) :: (encontrar( (IU,N,ID,M),(utente(IU,N,ID,M)),S ),comprimento( S,C ), C == 1).

% Cuidados prestados não repetidos
+cuidados( IS,D,INST,C ) :: (encontrar( ( IS,D,INST,C ),(cuidados( IS,D,INST,C )),S ), comprimento( S,N ), N == 1).

% Atos médicos não repetidos
+atoMedico( D,IU,IS,C ) :: (encontrar( (D,IU,IS,C),(atoMedico( D,IU,IS,C )),S ), comprimento( S,N ), N == 1).
```

Figura 3: Invariantes Estruturais

```
% Invariantes Referenciais

% Impedir a remoção de utentes se houver conhecimento sobre eles nos atos médicos
-utente( IU,NO,ID,M ) :: (encontrar( (IU),(atoMedico(D,IU,IS,C)), S ), comprimento( S,N ), N == 0).

% Impedir a remoção de cuidados prestados se houver conhecimento sobre eles nos atos médicos
-cuidados( IS,DE,INST,CI ) :: (encontrar( (IS),(atoMedico(D,IU,IS,C)), S ), comprimento( S,N ), N == 0).

% Impedir a adição de utentes com um IdUt já existente
+utente( IU,NO,ID,M ) :: (encontrar( (NOs,IDs,Ms) ,(utente(IU,NOs,IDs,Ms)), S ), comprimento( S,C ), C == 1).

% Impedir a adição de cuidados com um IdServ já existente
+cuidados( IS,DES,INST,CI ) :: (encontrar( (DESs,INSTs,CIs),(cuidados(IS,DESs,INSTs,CIs)),S ), comprimento( S,C ), C == 1).

% Impedir a adição de atos médicos com uma combinação de IdUt e IdServ já existente
+atoMedico( D,IU,IS,C ) :: (encontrar( (Ds,Cs),(atoMedico(Ds,IU,IS,Cs)),S ), comprimento( S,N ), N == 1).

% Impedir a adição de atos médicos com um utente não existente (verificação do IdUt)
+atoMedico( D,IU,IS,C ) :: (encontrar( (IU),(utente(IU,NO,ID,M)),S ), comprimento( S,N ), N > 0).

% Impedir a adição de atos médicos com um cuidado prestado não existente (verificação do IdServ)
+atoMedico( D,IU,IS,C ) :: (encontrar( (IS),(cuidados( IS,DES,INST,CI )),S ), comprimento( S,N ), N > 0).
```

Figura 4: Invariantes Referenciais

Além disso, definimos os factos que são sempre verdadeiros, sendo eles:

- utente;
- cuidados;
- atoMedico.

```
%--------------------------------- - - - - - - - - - - - - - - - - - -
% Extensão do predicado utente: idUt, Nome, IDade, Morada -> {V,F}
utente(8, andre, 21, selva).
utente(2, joao, 21, braga).
utente(15, joel, 21, bragança).
utente(9, sofia, 20, porto).

%--------------------------------- - - - - - - - - - - - - - - - - - -
% Extensão do predicado cuidados: idServ, Descricao, Instituicao, Cidade -> {V,F}
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).

%--------------------------------- - - - - - - - - - - - - - - - - - -
% Extensão do predicado atoMedico: Data, idUt, idServ, Custo -> {V,F}
atoMedico(01-01-2001, 9, 1, 30).
atoMedico(14-10-2009, 2, 2, 18).
atoMedico(21-01-2014, 15, 4, 23).
atoMedico(30-07-2010, 8, 3, 50).
atoMedico(30-08-2015, 8, 1, 40).
atoMedico(20-11-2016, 9, 2, 20).
```

Figura 5: Factos

Por fim, foram definidos os predicados propriamente ditos, estando estes divididos em duas secções: predicados mais "gerais", que podem e são utilizados

por outros predicados, e os predicados que permitem resolver as necessidades de demonstração das funcionalidades apresentadas no enunciado do primeiro exercício prático.

Os predicados globais e definidos para o uso de outros predicados são:

- inserir;
- testar;
- retirar;
- comprimento;
- encontrar;
- concatenar;
- soma.

Já os predicados definidos para o conhecimento a tratar são:

- evolucao;
- identificarU;
- identificarI;
- identificarIC;
- identificarUIS
  - procuraIdut
  - procuraUtent
- identificarAMI
  - identificarAMIAux
- identificarAMS;
- identificarAMU;
- determinarInst
  - procuraInst
- determinarServ;
- calcularCINST
  - procuraCS
- calcularCD;
- calcularCIDU;
- calcularCIDS;
- involucao.

Os predicados indicados a seguir tratam-se de conhecimento estendido, incluindo novas funcionalidades ao sistema, quer ao nível das capacidades de representação de conhecimento, quer ao nível das faculdades de raciocínio.

- naoEstaoInst
  - apagarElems
  - apagarElemento
- numAtos.

3.1 Requisitos do enunciado

O enunciado deste exercício previa a existência dos seguintes requisitos, sendo que estes serão abordados na secção "Funcionalidades":

1) Registar utentes, cuidados prestados e atos médicos;
2) Identificar os utentes por critérios de seleção;
3) Identificar as instituições prestadoras de cuidados de saúde;
4) Identificar os cuidados prestados por instituição/cidade;
5) Identificar os utentes de uma instituição/serviço;
6) Identificar os atos médicos realizados, por utente/instituição/serviço;
7) Determinar todas as instituições/serviços a que um utente já recorreu;
8) Calcular o custo total dos atos médicos por utente/serviço/instituição/data;
9) Remover utentes, cuidados e atos médicos.

Para além destes, implementamos ainda mais dois requisitos, que são abordados na secção "Novas Funcionalidades":

10) Identificar os serviços que não se podem encontrar numa instituição;
11) Determinar o número de atos médicos de um utente.

3.2 Funcionalidades

3.2.1 Predicado inserir

Este predicado, como o seu nome sugere, permite inserir novo conhecimento no sistema. Para tal, recebe um F que corresponderá a esse conhecimento e ele é inserido caso tal seja possível. Em caso contrário, quando não é permitido inserir tal conhecimento pois desrespeita algum invariante, a inserção falha e o conhecimento não é inserido no sistema.

3.2.2 Predicado testar

O predicado apresentado na figura abaixo recebe uma lista de invariantes e permite testá-los um de cada vez quando invocado nos predicados evolucao e involucao.

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensao do predicado inserir : F -> {V,F}
inserir( F ) :- assert( F ).
inserir( F ) :- retract( F ), !,fail.
```

Figura 6: Predicado inserir

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensao do predicado testar : L -> {V,F}
testar( [] ).
testar( [I|Li] ) :- I,testar( Li ).
```

Figura 7: Predicado testar

3.2.3 Predicado retirar

Este predicado é o oposto do predicado inserir, já apresentado e explicado acima. Ou seja, como o seu nome sugere, permite retirar conhecimento já disponível no sistema. Para tal, recebe um F que corresponderá ao conhecimento que se pretende retirar, e ele é retirado caso tal seja possível. Caso contrário, quando não é permitido retirar tal conhecimento pois desrespeita algum invariante, a remoção falha e o conhecimento não é retirado do sistema.

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensao do predicado retirar : F -> {V,F}
retirar( F ) :- retract( F ).
retirar( F ) :- assert( F ), !,fail.
```

Figura 8: Predicado retirar

3.2.4 Predicado comprimento

O seguinte predicado permite calcular o comprimento de uma lista. Para tal, este recebe uma lista (L) e vai retornar um inteiro com o comprimento da lista (N). O cálculo do N é feito de forma recursiva, isto é, é adicionado 1 sempre que houver um elemento à cabeça da lista ao comprimento calculado até ao momento. No final, é retornado o valor obtido no fim da recursividade.

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensao do predicado comprimento : L,N -> {V,F}
comprimento( [],0 ).
comprimento( [X|L],N ) :- comprimento(L,N2), N is 1+N2.
```

Figura 9: Predicado comprimento

3.2.5 Predicado encontrar

Este predicado permite encontrar as soluções de uma questão (Q) que sejam da forma (F) e guarda-as numa lista (S) através da utilização do predicado predefinido do PROLOG designado por findall que implementa esta funcionalidade.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado encontrar : F, Q, S -> {V,F}
encontrar( F,Q,S ) :- findall( F,Q,S ).
```

Figura 10: Predicado encontrar

3.2.6 Predicado concatenar

Como o próprio nome indica, este predicado permite concatenar duas listas. Deste modo, ele recebe duas listas (L1 e L2), ou seja, as listas que vai concatenar, e devolve uma lista com o resultado dessa concatenação (L).

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado concatenar : L1, L2, L -> {V,F}
concatenar([],L,L).
concatenar([X|T],L1,[X|R]) :- concatenar(T,L1,R).
```

Figura 11: Predicado concatenar

3.2.7 Predicado soma

Por fim, este predicado permite somar todos os valores de uma lista. Para tal, recebe uma lista (L) com os elementos que se pretende somar e retorna um R que terá o valor da soma dos elementos da lista recebida. Se o L for uma lista vazia, a soma dos seus valores é igual a zero. Caso contrário, quando a lista recebida não é vazia, o predicado soma é chamado recursivamente para a cauda da lista e é somado ao valor resultante de somar os elementos da cauda o valor que estava à cabeça da lista, sendo retornado o valor final.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
%Extensao do predicado soma: L, R -> {V,F}
soma([],0).
soma([H|T],S):-soma(T,G),S is H+G.
```

Figura 12: Predicado soma

3.2.8 Predicado evolucao (alínea 1)

Este predicado é usado para registar utentes, cuidados prestados e atos médicos. Deste modo, sempre que se pretende registar algum deste tipo de conhecimento, os invariantes estruturais e referenciais são testados. Este teste aos invariantes estruturais permite não inserir nem utentes, nem cuidados prestados nem atos médicos que já estejam registados na base de conhecimento. O teste dos invariantes referenciais garante que não é inserido um utente com um IdUt já existente, nem que é inserido um cuidado com um IdServ já utilizado e que também não é adicionado um ato médico com uma combinação de IdUt e IdServ já existentes. Além disto, dois dos invariantes estruturais permitem garantir que não é adicionado nenhum ato médico com um utente (IdUt) inexistente na base de

conhecimento e que não é inserido um ato médico com um cuidado prestado (IdServ) não existente.

```
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensao do predicado evolucao : F -> {V,F}
evolucao( F ) :- encontrar( I, +F::I, Li ), inserir( F ), testar( Li ).
```

Figura 13: Predicado evolucao

3.2.9 Predicado identificarU (alínea 2)

Para identificar os utentes por critérios de seleção, criamos o predicado identificarU que recebe um IdUt, um nome, uma idade e uma morada, e devolve uma lista com os utentes encontrados na base de conhecimento com os parâmetros que foram passados a este predicado. Para procurar os utentes, recorremos ao predicado encontrar, que por sua vez utiliza o predicado utente, que retorna uma lista (LU) com os utentes identificados pelos critérios de seleção.

```
% 2-Identificar os utentes por critérios de seleção
%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensão do predicado identificarU: IU, N, I, M, LU -> {V,F}
identificarU(IU, N, I, M,LU) :- encontrar((IU, N, I, M),utente(IU, N, I, M),LU).
```

Figura 14: Predicado identificarU

3.2.10 Predicado identificarI (alínea 3)

Neste caso o nosso objetivo é identificarmos as instituições prestadoras de cuidados de saúde, para isso, foi preciso criarmos um predicado identificarI, que irá devolver uma lista das instituições (LI). Para conseguirmos encontrar essas devidas instituições foi necessário recorrermos a a outro predicado chamado encontra onde irá ser utilizado também outro predicado, o predicado cuidados, que irá retornar uma lista com as Instituições dentro dos Cuidados Prestados, tal como pedido.

```
% 3-Identificar as instituições prestadoras de cuidados de saúde
%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensão do predicado identificarI: LI -> {V,F}
identificarI(LI) :- encontrar((I),cuidados(IS, D, I, C),LI).
```

Figura 15: Predicado identificarI

3.2.11 Predicado identificarIC (alínea 4)

O predicado identificarIC permite identificar os cuidados prestados por instituição/cidade. Para tal, este recebe uma instituição (INST) e uma cidade (C), e devolve uma lista com os IdServ, descrição e a instituição (INST) e a cidade (C)

recebidas e, portanto, procuradas. Para obter estes valores, é usado o predicado encontrar que usa o predicado cuidados, predicados estes que devolvem uma lista (LI) com o conhecimento encontrado que iguala os parâmetros passados ao predicado original, o predicado identificarIC.

```
% 4-Identificar os cuidados prestados por instituição/cidade
%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado identificarIC: INST,C,LI -> {V,F}
identificarIC(INST,C,LI) :- encontrar((IS,D,INST,C),cuidados(IS, D, INST, C),LI).
```

Figura 16: Predicado identificarIC

3.2.12 Predicado identificarUIS (alínea 5)

Neste predicado irá ser recebido uma instituição e um serviço (INST / S), irá ser utilizado outro predicado chamado encontrar que por sua vez irá utilizar mais 3 predicados: cuidados, procuraIdut e procuraUtent, sendo que estes 2 serão explicados em baixo. O nosso objetivo será conseguirmos identificar os utentes de uma instituição/serviço. Como resultado iremos ter, tal como proposto, uma lista com os utentes.

```
% 5-Identificar os utentes de uma instituição/serviço
%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado identificarUIS: INST,S,L -> {V,F}
identificarUIS(INST,S,L) :- encontrar((S),cuidados(S, D, INST, C),L2), procuraIdut(L2,L1), procuraUtent(L1,L).
```

Figura 17: Predicado identificarUIS

Predicado procuraIdut

Este predicado pode ser dividido em três fases, a primeira, em que foi necessário recorrermos a outro predicado chamado 'encontrar', que por sua vez recorre a outro predicado chamado 'atoMedico', sendo o nosso objetivo ir procurar nos atos médicos os id de serviço, e devolver uma lista a informação relativa ao id de utente, numa segunda em que este processo e chamado recursivamente (criando varias listas) sendo que por fim as concatenamos, obtendo apenas uma lista como resultado final.

```
% Extensão do predicao procuraIdut: LIDS, LIDU -> {V,F}
procuraIdut([],[]).
procuraIdut([IDS|T],R) :- encontrar((IDU), atoMedico(D,IDU,IDS,C), L), procuraIdut(T,R2), concatenar(L,R2,R).
```

Figura 18: Predicado procuraIdut

Predicado procuraUtent

Este predicado pode ser agrupado em três fases, a primeira, em que recorrermos a outro predicado chamado 'encontrar', que por sua vez recorre a outro predicado chamado 'utente', sendo o nosso objetivo ir procurar nos utentes os id de utente, e devolver uma lista com toda a informação relativa a esses utentes em especifico, numa segunda em que este processo e chamado recursivamente (criando varias listas) sendo que por fim as concatenamos, obtendo apenas uma lista como resultado final.

```
% Extensão do predicado procuraUtent: LIDU, LU -> {V,F}
procuraUtent([],[]).
procuraUtent([IDU|T],R) :- encontrar((IDU, N, I, M), utente(IDU, N, I, M),L), procuraUtent(T,R2), concatenar(L,R2,R).
```

Figura 19: Predicado procuraUtent

3.2.13 Predicado identificarAMI (alínea 6)

Neste predicado o objetivo é conseguirmos identificar os atos médicos realizados por instituição, para isso necessitamos de receber uma instituição (INST) e devolver uma lista com os atos médicos correspondentes a essa mesma. Para realizarmos este feito recorremos a outro predicado chamado encontrar, que por sua vez irá recorrer a mais dois predicados: cuidados e identificarAMIAux, permitindo-nos assim obter a lista pretendida com os atos médicos realizados por instituição.

```
% 6-Identificar os atos médicos realizados, por utente/instituição/serviço
%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado identificarAMI: INST,L2 -> {V,F}
identificarAMI(INST, R) :- encontrar((IDS),cuidados(IDS,D,INST,C),X), identificarAMIAux(X,R).
```

Figura 20: Predicado identificarAMI

Predicado identificarAMIAux

Este predicado serve como predicado auxiliar para o identificarAMI. Podemos partir este predicado em três fases, a primeira, em que foi necessário recorrermos a outro predicado chamado encontrar, que por sua vez recorre a outro predicado chamado atoMedico, sendo o nosso objetivo ir procurar nos atos médicos os id de serviço, e devolver uma lista com toda a informação relativa aos atos médicos, numa segunda em que este processo e chamado recursivamente (criando varias listas) sendo que por fim as concatenamos, obtendo apenas uma lista como resultado final.

```
% Extensão do predicado identificarAMIAux: L1,L2 -> {V,F}
identificarAMIAux([],[]).
identificarAMIAux([IDS|T],R) :- encontrar((D,IDU,IDS,C),atoMedico(D,IDU,IDS,C),L),identificarAMIAux(T,R2),concatenar(L,R2,R).
```

Figura 21: Predicado identificarAMIAux

3.2.14 Predicado identificarAMS (alínea 6)

Este nosso predicado irá permitir-nos identificar os atos médicos realizados, por serviço, para isso, foi preciso criarmos um predicado identificarAMS, que irá receber um ID de serviço (IDS). Foi necessário recorrermos ao nosso predicado encontrar, que por sua vez irá usar o predicado atoMedico, de maneira a devolver uma lista com a informação dos atos médicos de um determinado serviço.

```
%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado identificarAMS: IDS,L2 -> {V,F}
identificarAMS(IDS,R) :- encontrar((D,U,IDS,CUS),atoMedico(D,U,IDS,CUS),R).
```

Figura 22: Predicado identificarAMS

3.2.15 Predicado identificarAMU (alínea 6)

Já neste caso foi necessário criar um predicado chamado identificarAMU, que vai permitir identificar os atos médicos realizados, por utente. Irá ser recebido um id do utente (IDU) e devolvida uma lista com os atos médicos desse utente (L2). Tal como no predicado anterior a estratégio irá ser a mesma, foi necessário recorrermos ao nosso predicado encontrar, que por sua vez irá usar o predicado 'atoMedico', de maneira a devolver uma lista com a informação dos atos médicos de um determinado utente.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado identificarAMU: IDU,L2 -> {V,F}
identificarAMU(IDU,R) :- encontrar((D,IDU,IDS,CUS),atoMedico(D,IDU,IDS,CUS),R).
```

Figura 23: Predicado identificarAMU

3.2.16 Predicado determinarInst (alínea 7)

Este predicado pretende determinar todas as instituições a que um utente já recorreu. Iremos receber um id de utente (IDU) e irá ser devolvido uma lista com as instituições (LINST). A nossa estratégia foi a seguinte: Utilizamos outro predicado, chamado encontrar, que por sua vez utiliza outro predicado chamado atoMedico que irá devolver uma lista com todos os id's de serviços. Após esta recolha iremos usar um predicado chamado procuraInst (explicado de seguida), podendo assim obter a lista pretendida (lista das instituições), tal como pedido no problema.

```
% 7-Determinar todas as instituições/serviços a que um utente já recorreu
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado determinarInst: IDU, LINST -> {V,F}
determinarInst(IDU, R) :- encontrar((IDS), atoMedico(D,IDU,IDS,C), LIDS), procuraInst(LIDS,R).
```

Figura 24: Predicado determinarInst

Predicado procuraInst O predicado procuraInst recebe uma lista de id's de serviço que são determinados na primeira parte do predicado determinarInst e encontra nos cuidados todas as instituições que correspondem a estes id's, armazenando-as numa lista.

```
% Extensão do predicado procuraInst: LIDS, L -> {V,F}
procuraInst([],[]).
procuraInst([IDS|T],R) :- encontrar((INST), cuidados(IDS,DES,INST,CID),L), procuraInst(T,R2), concatenar(L,R2,R).
```

Figura 25: Predicado procuraInst

3.2.17 Predicado determinarServ (alínea 7)

Este nosso predicado pretende determinar todos os serviços a que um utente já recorreu, para isso foi criado um predicado chamado determinarServ que irá receber um id de um utente (IDU) e devolver uma lista com os id's de serviço a que esse utente já recorreu (LIDS). Foi necessário recorrermos a outro predicado

chamado 'encontrar', que por sua vez recorre a outro predicado chamado 'atoMedico', sendo o nosso objetivo ir procurar nos atos médicos os id de serviço, e devolver uma lista com esses mesmos id's.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado determinarServ: IDU, LIDS -> {V,F}
determinarServ(IDU,R) :- encontrar((IDS),atoMedico(D,IDU,IDS,C),R).
```

Figura 26: Predicado determinarServ

3.2.18 Predicado calcularCINST (alínea 8)

Para este predicado será necessário calcular o custo total dos atos médicos por instituição. Vamos receber uma instituição (INST), e devolver o custo total (CUS). A nossa primeira tarefa foi recorrermos a outro predicado chamado encontrar, sendo devolvidos todos os id de serviço e armazenados em N, de seguida recorremos a outro predicado chamada procuraCS (será explicado em baixo), e a partir disto podemos calcular o nosso custo total utilizando um predicado auxiliar chamado soma.

```
% 8-Calcular o custo total dos atos médicos por utente/serviço/instituição/data
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado calcularCINST: INST, CUS -> {V,F}
calcularCINST(INST, R) :- encontrar((IDS),cuidados(IDS,D,INST,C),N), procuraCS(N,R2), soma(R2,R).
```

Figura 27: Predicado calcularCINST

Predicado procuraCS

```
% Extensão do predicado procuraCS: LIDS, L -> {V,F}
procuraCS([],[]).
procuraCS([IDS|T],R) :- encontrar((C), atoMedico(D,IDU,IDS,C),L), procuraCS(T,R2), concatenar(L,R2,R).
```

Figura 28: Predicado procuraCS

3.2.19 Predicado calcularCD (alínea 8)

Este predicado tem como objetivo calcular o custo total dos atos médicos por data. Para isto foi necessário recebermos uma data, e a partir dessa mesma, através de um outro predicado chamado encontrar, que irá usar outro predicado chamado atoMedico, permitindo assim ir buscar os custos por data, armazenando em N. E por fim, através de outro predicado auxiliar chamado soma, somar todos os custos ficando assim com um custo total, tal como pedido.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado calcularCD: D,CUS -> {V,F}
calcularCD(D,R) :- encontrar(CUS,atoMedico(D,U,IDS,CUS),N),soma(N,R).
```

Figura 29: Predicado calcularCD

3.2.20 Predicado calcularCIDU (alínea 8)

Neste predicado pretendemos calcular o custo total dos atos médicos por id de utente. Foi utilizado exatamente o mesmo raciocínio que no anterior, sendo a única diferença na utilização da variável que iremos receber, sendo neste caso o id de utente (IDU).

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado calcularCIDU: IDU,CUS -> {V,F}
calcularCIDU(IDU,R) :- encontrar(CUS,atoMedico(D,IDU,IDS,CUS),N),soma(N,R).
```

Figura 30: Predicado calcularCIDU

3.2.21 Predicado calcularCIDS (alínea 8)

Neste predicado pretendemos calcular o custo total dos atos médicos por id de serviço. Foi utilizado exatamente o mesmo raciocínio que no anterior, sendo a única diferença na utilização da variável que iremos receber, sendo neste caso o id de serviço (IDS).

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado calcularCIDS: IDS,CUS -> {V,F}
calcularCIDS(IDS,R) :- encontrar(CUS,atoMedico(D,U,IDS,CUS),N),soma(N,R).
```

Figura 31: Predicado calcularCIDS

3.2.22 Predicado involucao (alínea 9)

Este predicado é usado para remover utentes, cuidados e atos médicos. Deste modo, sempre que se pretende remover algum deste tipo de conhecimento, os invariantes referenciais de opções de remoção são testados. Este teste aos invariantes referenciais garante que não é removido um utente se houver conhecimento sobre ele nos atos médicos, nem que são removidos cuidados prestados se houver conhecimento sobre eles nos atos médicos.

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado involucao : F -> {V,F}
involucao( F ) :- encontrar( I, -F::I, Li ), retirar( F ), testar( Li ).
```

Figura 32: Predicado involucao

3.3 Novas funcionalidades

3.3.1 Predicado naoEstaoInst (alínea 10)

O objetivo deste predicado é identificar os serviços que não se podem encontrar numa instituição, recebendo, por isso, uma instituição, e utilizando os predicados apagarElems e encontrar para retornar uma lista com os id's dos serviços que não existem nessa instituição.

Predicado apagarElemento

Este predicado recebe um elemento e uma lista da qual o queremos apagar e retorna uma lista sem qualquer ocorrência esse elemento.

Predicado apagarElems

O predicado apagarElems recebe uma lista de elementos que queremos remover e a lista da qual os queremos remover e retorna uma lista sem qualquer ocorrência dos elementos da primeira lista recebida. Para isso, utiliza o predicado apagarElemento de modo a, recursivamente, apagar primeiro as ocorrências do primeiro elemento da lista de elementos a remover e só depois os seguintes.

```
% 10-Identificar os serviços que não se podem encontrar numa instituição
%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensão do predicado apagarElemento: X, LI, LF -> {V,F}
apagarElemento(X, [], []).
apagarElemento(X, [X|T], Y) :- apagarElemento(X, T, Y).
apagarElemento(X, [H|T], [H|Y]) :- apagarElemento(X, T, Y).

% Extensão do predicado apagarElems: L1, L2, LF -> {V,F}
apagarElems([],[X],[X]).
apagarElems([H|T],L,Z) :- apagarElemento(H,L,Y), apagarElems(T,Y,Z).
apagarElems([],Z,Z).

% Extensão do predicado naoEstaoInst: INST, LIDS -> {V,F}
naoEstaoInst(INST,LIDS) :- encontrar((IDS),cuidados(IDS,DES,INST,CID),L1), encontrar((I),cuidados(I,D,IN,C),L2), apagarElems(L1,L2,LIDS).
```

Figura 33: Predicados naoEstaoInst, apagarElemento e apagarElems

3.3.2 Predicado numAtos (alínea 11)

Este foi também um predicado que decidimos adicionar ao nosso exercicio, o objetivo é determinarmos o número de atos médicos de um utente. Por isso iremos receber o id de utente (IDUT) e devolver o número de atos médicos desse mesmo utente. Nós usámos um predicado já explicado anteriormente chamado 'encontrar' (que por sua vez utiliza um predicado chamado 'atoMedico') que coloca numa lista todos os id de utentes. Por ultimo, utilizamos o predicado 'comprimento' ficando assim com uma lista com o número de atos médicos.

```
% 11-Determinar o número de atos médicos de um utente
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado numAtos: IDUT, N -> {V,F}
numAtos(IU,N) :- encontrar((IU),atoMedico(D,IU,IS,C),L), comprimento(L,N).
```

Figura 34: Predicado numAtos

4 Conclusões e Sugestões

Na realização deste projeto deparamo-nos com vários precalços que nos impossibilitaram de fazer um percurso mais direto, mas, mesmo assim, não menos conciso que o realizado.

Em primeiro lugar, consideramos a liberdade de criação que nos foi dada a maior benesse e talvez o maior problema, dado que são essas as restrições nas quais o cerne deste projeto assenta e que essas escolhas serão repercutidas ao longo do mesmo.

Porém, julgamos ter sido capazes de apresentar explicações detalhadas para as escolhas que alicerçam o primeiro exercício prático e que garantem a presença de toda a informação essencial para o seu bom funcionamento. Foi ainda disponibilizada a informação necessária para entender o mecanismo de funcionamento do nosso sistema de representação de conhecimento e raciocínio, e, além disso, pautamo-nos também pela simplicidade, mantendo um "ambiente de trabalho"acessível de uma maneira que é possível a alguém estranho a PROLOG consiga ainda assim acompanhar e entender o que está a ser dito.

Em segundo lugar, num mundo idílico este projeto poderia ser dado simplesmente como concluído, porém, uma das noções básicas na representação de conhecimento é que nenhum projeto conseguirá realmente incluir todo o universo de informação existente, daí haver sempre espaço para uma melhoria que de alguma forma deixará o sistema mais completo. Isto foi uma das preocupações que tivemos: criar um projeto que permita uma fácil expansão tanto de funcionalidades como de novas ideias que porventura possam surgir. Um exemplo disso era a criação de um novo predicado médico, no qual era referido o local de trabalho (hospital ou centro de saúde), morada, especialidade, algo que é de valor óbvio para este tipo de exercício.

Em suma, fundamentamos as nossas escolhas, o que é notório na própria estrutura do trabalho, e estamos satisfeitos com as conclusões a que chegamos e com o plano de ação que elaboramos sendo que respondemos de forma simples e eficaz aos requisitos propostos e mais alguns. Foram ainda realizados testes para garantir o funcionamento correto dos invariantes que mantêm os dados consistentes nunca obtendo resultados não aceitáveis e redundantes.

5 Bibliografia

- "PROLOG: Programming for Artificial Intelligence", Ivan Bratko;
- "A Inteligência Artificial em 25 Lições", Hélder Coelho.

6 Anexos

6.1 Exemplos de execução da aplicação

Nesta secção, é possível encontrar exemplos da utilização dos predicados definidos para a resolução de cada uma das alíneas. Deve-se destacar a execução dos predicados evolucao e involucao por utilizarem os vários invariantes que foram definidos, sendo que tentamos mostrar ao máximo que estes predicados respeitam os pressupostos desses invariantes.

```
| ?- listing(utente).
utente(8, andre, 21, selva).
utente(2, joao, 21, braga).
utente(15, joel, 21, bragança).
utente(9, sofia, 20, porto).

yes
| ?- evolucao(utente(10,manuel,30,lisboa)).
yes
| ?- listing(utente).
utente(8, andre, 21, selva).
utente(2, joao, 21, braga).
utente(15, joel, 21, bragança).
utente(9, sofia, 20, porto).
utente(10, manuel, 30, lisboa).

yes
| ?- evolucao(utente(8,jose,25,faro)).
no
| ?- listing(utente).
utente(8, andre, 21, selva).
utente(2, joao, 21, braga).
utente(15, joel, 21, bragança).
utente(9, sofia, 20, porto).
utente(10, manuel, 30, lisboa).

yes
| ?- evolucao(utente(1,jose,25,faro)).
yes
| ?- listing(utente).
utente(8, andre, 21, selva).
utente(2, joao, 21, braga).
utente(15, joel, 21, bragança).
utente(9, sofia, 20, porto).
utente(10, manuel, 30, lisboa).
utente(1, jose, 25, faro).
```

Figura 35: Exemplo de registar utentes

```
| ?- listing(cuidados).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).

yes
| ?- evolucao(cuidados(5,medicinaGeral,hospitalBraga,braga)).
yes
| ?- listing(cuidados).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga)
cuidados(4, neurologia, hospitalGuimarães, guimarães).
cuidados(5, medicinaGeral, hospitalBraga, braga).

yes
| ?- evolucao(cuidados(1,estomatologia,hospitalBraga,braga)).
no
| ?- listing(cuidados).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).
cuidados(5, medicinaGeral, hospitalBraga, braga).

yes
| ?- evolucao(cuidados(6,estomatologia,hospitalBraga,braga)).
yes
| ?- listing(cuidados).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).
cuidados(5, medicinaGeral, hospitalBraga, braga).
cuidados(6, estomatologia, hospitalBraga, braga).
```

Figura 36: Exemplo de registar cuidados

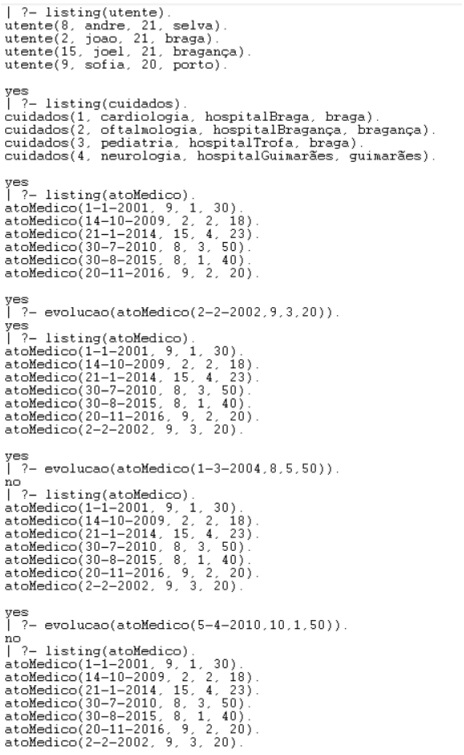


Figura 37: Exemplo de registar atos médicos

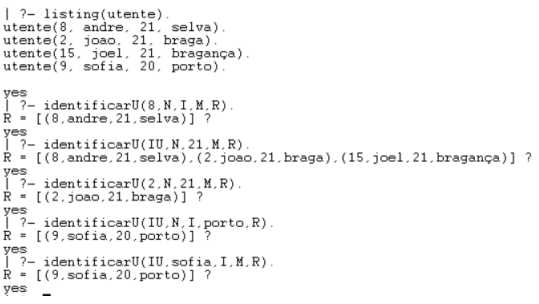


Figura 38: Exemplo de identificar os utentes por critérios de seleção

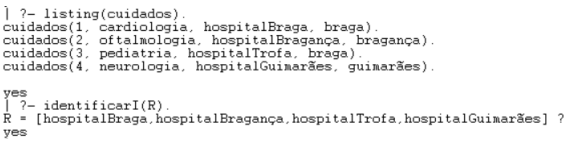


Figura 39: Exemplo de identificar as instituições prestadoras de cuidados de saúde

```
| ?- listing(cuidados).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).

yes
| ?- identificarIC(hospitalBraga,C,R).
R = [(1,cardiologia,hospitalBraga,braga)] ?
yes
| ?- identificarIC(I,braga,R).
R = [(1,cardiologia,hospitalBraga,braga),(3,pediatria,hospitalTrofa,braga)] ?
yes
| ?- identificarIC(hospitalTrofa,braga,R).
R = [(3,pediatria,hospitalTrofa,braga)] ?
yes
```

Figura 40: Exemplo de identificar os cuidados prestados por instituição ou cidade

```
| ?- listing(utente).
utente(8, andre, 21, selva).
utente(2, joao, 21, braga).
utente(15, joel, 21, bragança).
utente(9, sofia, 20, porto).

yes
| ?- listing(cuidados).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).

yes
| ?- listing(atoMedico).
atoMedico(1-1-2001, 9, 1, 30).
atoMedico(14-10-2009, 2, 2, 18).
atoMedico(21-1-2014, 15, 4, 23).
atoMedico(30-7-2010, 8, 3, 50).
atoMedico(30-8-2015, 8, 1, 40).
atoMedico(20-11-2016, 9, 2, 20).

yes
| ?- identificarUIS(hospitalBraga,IS,R).
R = [(9,sofia,20,porto),(8,andre,21,selva)] ?
yes
| ?- identificarUIS(I,2,R).
R = [(2,joao,21,braga),(9,sofia,20,porto)] ?
yes
| ?- identificarUIS(hospitalBraga,1,R).
R = [(9,sofia,20,porto),(8,andre,21,selva)] ?
yes
```

Figura 41: Exemplo de identificar os utentes de uma instituição ou serviço

```
| ?- listing(utente).
utente(8, andre, 21, selva).
utente(2, joao, 21, braga).
utente(15, joel, 21, bragança).
utente(9, sofia, 20, porto).

yes
| ?- listing(cuidados).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).

yes
| ?- listing(atoMedico).
atoMedico(1-1-2001, 9, 1, 30).
atoMedico(14-10-2009, 2, 2, 18).
atoMedico(21-1-2014, 15, 4, 23).
atoMedico(30-7-2010, 8, 3, 50).
atoMedico(30-8-2015, 8, 1, 40).
atoMedico(20-11-2016, 9, 2, 20).

yes
| ?- identificarAMI(hospitalBraga,R).
R = [(1-1-2001,9,1,30),(30-8-2015,8,1,40)] ?
yes
```

Figura 42: Exemplo de identificar os atos médicos realizados por instituição

Figura 43: Exemplo de identificar os atos médicos realizados por serviço

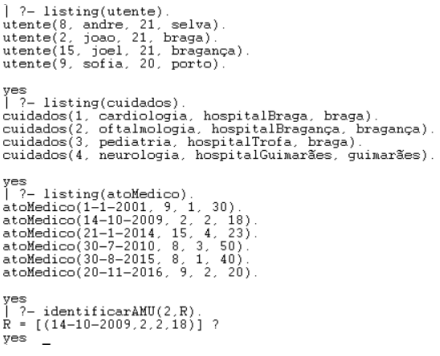


Figura 44: Exemplo de identificar os atos médicos realizados por utente

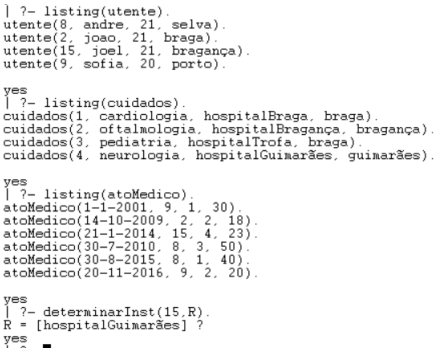


Figura 45: Exemplo de determinar as instituições a que um utente já recorreu

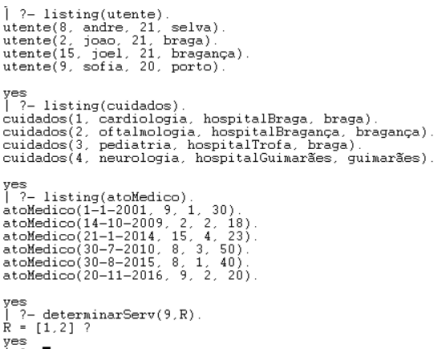


Figura 46: Exemplo de determinar os serviços a que um utente já recorreu

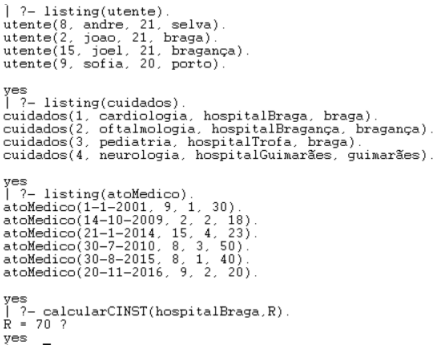


Figura 47: Exemplo de calcular o custo total dos atos médicos por instituição

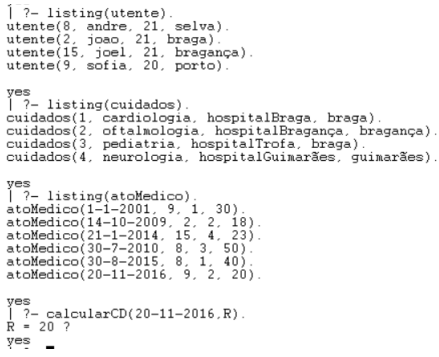


Figura 48: Exemplo de calcular o custo total dos atos médicos por data

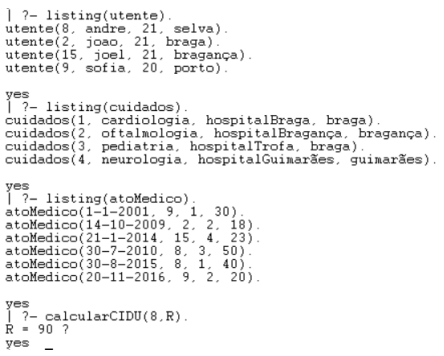


Figura 49: Exemplo de calcular o custo total dos atos médicos por utente

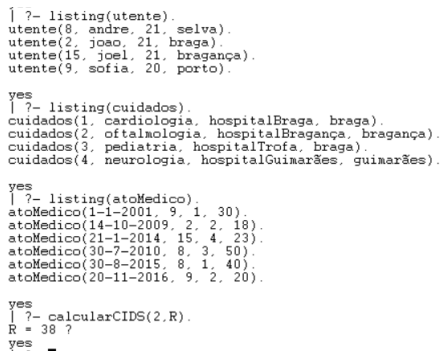


Figura 50: Exemplo de calcular o custo total dos atos médicos por serviço

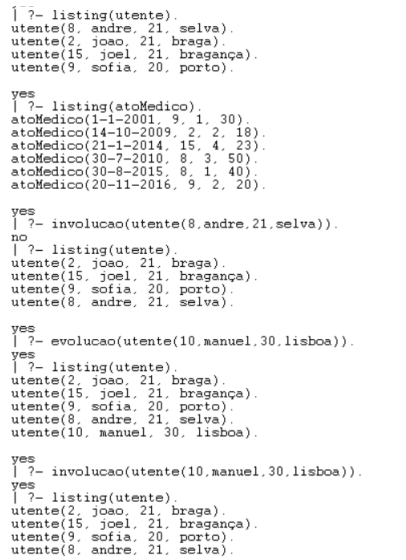


Figura 51: Exemplo de remover utentes

```
| ?- listing(cuidados).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).

yes
| ?- listing(atoMedico).
atoMedico(1-1-2001, 9, 1, 30).
atoMedico(14-10-2009, 2, 2, 18).
atoMedico(21-1-2014, 15, 4, 23).
atoMedico(30-7-2010, 8, 3, 50).
atoMedico(30-8-2015, 8, 1, 40).
atoMedico(20-11-2016, 9, 2, 20).

yes
| ?- involucao(cuidados(1,cardiologia,hospitalBraga,braga)).
no
| ?- listing(cuidados).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).
cuidados(1, cardiologia, hospitalBraga, braga).

yes
| ?- evolucao(cuidados(5,medicinaGeral,hospitalBraga,braga)).
yes
| ?- listing(cuidados).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(5, medicinaGeral, hospitalBraga, braga).

yes
| ?- involucao(cuidados(5,medicinaGeral,hospitalBraga,braga)).
yes
| ?- listing(cuidados).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).
cuidados(1, cardiologia, hospitalBraga, braga).
```

Figura 52: Exemplo de remover cuidados

```
| ?- listing(atoMedico).
atoMedico(1-1-2001, 9, 1, 30).
atoMedico(14-10-2009, 2, 2, 18).
atoMedico(21-1-2014, 15, 4, 23).
atoMedico(30-7-2010, 8, 3, 50).
atoMedico(30-8-2015, 8, 1, 40).
atoMedico(20-11-2016, 9, 2, 20).

yes
| ?- involucao(atoMedico(1-1-2001,9,1,30)).
yes
| ?- listing(atoMedico).
atoMedico(14-10-2009, 2, 2, 18).
atoMedico(21-1-2014, 15, 4, 23).
atoMedico(30-7-2010, 8, 3, 50).
atoMedico(30-8-2015, 8, 1, 40).
atoMedico(20-11-2016, 9, 2, 20).
```

Figura 53: Exemplo de remover atos médicos

```
| ?- listing(utente).
utente(8, andre, 21, selva).
utente(2, joao, 21, braga).
utente(15, joel, 21, bragança).
utente(9, sofia, 20, porto).

yes
| ?- listing(cuidados).
cuidados(1, cardiologia, hospitalBraga, braga).
cuidados(2, oftalmologia, hospitalBragança, bragança).
cuidados(3, pediatria, hospitalTrofa, braga).
cuidados(4, neurologia, hospitalGuimarães, guimarães).

yes
| ?- listing(atoMedico).
atoMedico(1-1-2001, 9, 1, 30).
atoMedico(14-10-2009, 2, 2, 18).
atoMedico(21-1-2014, 15, 4, 23).
atoMedico(30-7-2010, 8, 3, 50).
atoMedico(30-8-2015, 8, 1, 40).
atoMedico(20-11-2016, 9, 2, 20).

yes
| ?- naoEstaoInst(hospitalBraga,R).
R = [2,3,4] ?
yes
```

Figura 54: Exemplo de identificar os serviços que não se podem encontrar numa instituição

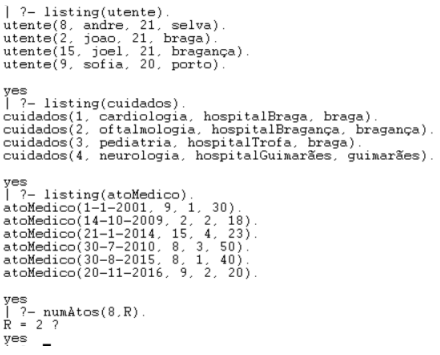


Figura 55: Exemplo de determinar o número de atos médicos de um utente