Universidade do Minho

Programação em Lógica e Invariantes Serviço Médico

Mestrado Integrado em Engenharia Informática

Sistema de Representação de Conhecimento e Raciocínio

(2ºSemestre/2017-2018)

A77278 Carlos Gonçalves

A78452 José Ferreira

A78587 Ricardo Peixoto

A78660 Jorge Oliveira

Grupo 3

Braga

18/03/2018

Resumo

O presente relatório pretende explicar a conceção, através de raciocínio e conhecimento, de um sistema de serviços relacionados com a saúde. Para auxiliar esta tarefa, usamos a linguagem de programação em lógica o PROLOG e o meio utilizado para testar todo o conhecimento foi a ferramenta SiCStus.

Como já referido, este relatório incide na explicação de todo o raciocínio na construção de todo o conhecimento necessário para manipulação de um serviço de cuidados de saúde. Para além disto, serão apresentados os resultados obtidos.

Índice

Índice de Figuras

1. Introdução

Hoje em dia a saúde é uma preocupação para todas as pessoas, sendo que as instituições têm uma forte influência no bem-estar da população. Como, este tema esta bastante ativo na sociedade, foi-nos proposto num modo simplista, desenvolver um sistema de representação de conhecimento e raciocínio com capacidade de qualificar a prestação de cuidados de saúde.

Para uma orientação correta no sentido de alcançar sucesso com a realização deste sistema, traçamos alguns objetivos:

O nosso principal objetivo era conseguir escrever todos os predicados básicos que estavam pré-definidos. Sem que os predicados básicos estarem todos construídos, não faria sentido implementar outros que pudessem ser mais complexos ou então que representassem outro conhecimento, daí que tenhamos definido este como principal objetivo.

Adicionar alguns predicados extras que pudessem fazer sentido na transposição para o mundo real. Para além dos predicados básicos, achávamos que faria sentido adicionar outros que tornasse mais rica a nossa base de conhecimento.

Tentar, de uma maneira simples, aproximar todos os predicados o mais possível do mundo real. Apesar, de o sistema ser bastante simples, tentámos sempre criar um conhecimento realista no contexto dos cuidados de saúde.

2. Preliminares

Para que o leitor possa entrar em sintonia com todos os conteúdos abordados neste relatório é importante que tenha em mente um conjunto de definições. É necessário saber o que é a programação em lógica e alguns dos seus conceitos essenciais, bem como a maneira correta de representar o conhecimento e raciocínio. Neste relatório, a linguagem utilizada é o PROLOG que faz jus da lógica para representar o conhecimento.

A programação em lógica apresenta bastantes diferenças em relação aos outros paradigmas de programação. A ideia, em lógica, não é escrever como as soluções se programam, mas sim escrever quais são as soluções, devemos determinar quais os resultados pretendidos.

Então o conhecimento pode ter por base conclusões que serão provadas por um conjunto de questões: P se Q. Ou então apenas factos, que é uma constatação de algo que se sabe que é verdadeiro, pode também ser entendido, como P se *True*. Para obter a informação na nossa base de conhecimento, teremos sempre de fazer questões à mesma. De notar, que a linguagem utilizada, PROLOG, apenas responde *yes* ou *no*, o que apesar de parecer bastante simples, apenas com estas duas possíveis respostas temos a capacidade de saber tudo o que está presente na base de conhecimento.

Também será necessário ter em mente alguma da terminologia do PROLOG e que é frequentemente apresentada no relatório:

. – O ponto significa o fim de uma declaração;

:- - signica “se”;

, - representa o “e”;

(+/-)P :: I – é a forma de se representar invariantes. Caso o primeiro carater seja um “+” então o invariante refere-se à inserção de conhecimento, se for “-“ então indica a remoção de conhecimento.

Para além de todo o conhecimento que o leitor deve possuir sobre o que foi abordado anteriormente, deve de ter a uma noção minimalista de como funciona um sistema de saúde. Um utente é alguém que necessita de algum cuidado, onde o pode encontrar em variadíssimas instituições (como hospitais por exemplo). Um prestador é um individuo que trata então do utente, pode ser um médico, enfermeiro, etc.

3. Descrição do Trabalho e Análise de Resultados

3.1. Predicado Base

3.1.1. Cuidado

Na nossa base de conhecimento o predicado utente representa um indivíduo ao qual é prestado um determinado cuidado, podemos então deduzir, que no mundo real o utente será um cidadão comum. Portanto tem de possuir um nome, mas como um nome não identifica inequivocamente uma pessoa numa atividade é necessário um identificador. Precisamos ainda de um campo com a idade dos cidadãos, pois podemos querer identifica-los por idades, e também da sua morada para que possamos identificar os utentes de diferentes localidades. Para caraterizar a morada de um utente decidimos criar um predicado morada, sendo que esta contém um campo relativo à rua da morada do mesmo, outra relativo à localidade e outro relativo à cidade.

Pelas razões referidas anteriormente, decidimos criar o predicado:

utente: IdUt, Nome, Idade, Morada -> {V,F}

utente(ID,Nome,Idade,morada(Rua,Localidade,Cidade))

De seguida apresentamos alguns exemplos relativos a utentes presentes na base de conhecimento:

utente(1,ana,14,morada('Rua do Louro','Caldas das Taipas','Guimaraes')).

utente(2,luis,13,morada('Rua do Azevinho','Braga','Braga')).

3.1.2. Morada

No predicado utente está presente um campo morada. Como a morada de um cidadão é algo complexo decidimos então criar um predicado que nos permitisse ter um maior controlo sobre a morada de um individuo. Assim neste predicado decidimos incluir a *rua*, local onde reside uma pessoa (neste campo rua já está incluído o número da porta), *localidade* e *cidade*.

morada(rua,localidade,cidade).

Alguns exemplos:

morada('Rua do Louro','Caldas das Taipas','Guimaraes').

morada('Rua do Azevinho','Braga','Braga') .

morada('Rua do Rual','São João de Ponte','Guimaraes').

3.1.3. Prestador

Na nossa base de conhecimento o predicado prestador representa um indivíduo que presta um determinado cuidado, podemos então deduzir, que no mundo real o prestador seria um profissional de saúde. Sendo assim tem de possuir um **nome**, e como o nome de uma pessoa não é suficiente para a identificar unicamente num serviço, foi necessário definir um **identificador** para cada prestador. Para além destes campos, um prestador deve de possuir uma **especialidade** e uma instituição onde pratica os cuidados. Todavia, um prestador pode trabalhar em **várias instituições**, caso assim deseje e seja permitido.

Posto isto usamos o predicado prestador para descrever a seguinte relação: **prestador(ID,Nome,Especialidade,L_Instituicoes)**. Segue alguns exemplos:

prestador(1,jose,'ortopedia',['Hospital de Braga']).

prestador(2,joao,'urologia',['Hospital de Guimaraes']).

prestador(3,jorge,'neurologia',['Hospital de Guimaraes']).

3.1.4. Instituição

Na nossa base de conhecimento o predicado instituicao representa uma instituição na qual serão prestados os diversos cuidados. Para a caracterizar é necessário indicar qual o seu nome, em que cidade se localiza, que tipo de instituição é (hospital, centro de saúde, etc) e a lista das especialidades dos seus serviços de saúde.

Pelos motivos referidos decidimos criar o seguinte predicado:

instituicao(Nome, Cidade, Tipo, ListaEspecialidade)

Exemplo de alguns cuidados registados na nossa base de conhecimento:

instituicao('Hospital de Braga','Braga','Hospital',['ortopedia']).

instituicao('Hospital de Guimaraes','Guimaraes','Hospital',['urologia','neurologia']).

3.1.5. Cuidado

Na nossa base de conhecimento o predicado cuidado representa um serviço médico prestado a um determinado utente. Esse serviço poderá ir de um simples curativo até uma intervenção cirúrgica. Para o caracterizarmos de uma forma completa precisamos de saber a quem foi prestado o serviço, quem foi o prestador do serviço, qual foi o tipo do serviço, quando foi prestado, em que instituição acorreu e qual foi o custo total que esse serviço implicou.

Pelos motivos referidos decidimos criar o seguinte predicado: **cuidado(Data,IdUtente,IdPrestador,Descrição,Custo,Instituição)**

Exemplo de alguns cuidados registados na nossa base de conhecimento:

cuidado(data(2018,3,14,20),1,1,'curativo',23,'Hospital de Braga').

cuidado(data(2018,3,10,20),2,3,'rotina',23,'Hospital de Guimaraes').

cuidado(data(2018,3,10,20),3,3,'rotina',23,'Hospital de Guimaraes').

3.1.6. Data

Para melhor manipularmos questões relacionadas com o tempo em que o cuidado foi realizado decidimos criar um predicado data. Este predicado contem todos os elementos essenciais de uma data como um ano, um mês, um dia. Adicionalmente decidimos acrecentar a hora, para que pudéssemos distinguir cuidados nas diferentes partes de um dia. O predicado construído é o seguinte: **data(Ano,Mes,Dia,Hora) .**

Alguns exemplos:

data(2018,03,17,10)

data(2018,02,01,22)

3.2. Povoamento da Base de Conhecimento

Para que pudéssemos testar todos os predicados e os invariantes construídos foi necessário proceder ao povoamento da base de conhecimento. Este, foi realizado, com a inserção manual dos predicados anteriormente referidos.

Tentámos simplificar o povoamento da base de conhecimento, para que a análise de resultados fosse mais percetível, quer para nós aquando do *debug* quer para o leitor. Desta forma era mais fácil ter o povoamento em mente, dedicando assim mais tempo sobre como os predicados, mais à frente referidos, funcionam.

```
%------------------------------- - - - - - - - - - - - - - - - -
% Extensao do predicado utente: IdUt, Nome, Idade, Morada -> {V,F}

utente( 1,ana,14,morada( 'Rua do Louro','Caldas das Taipas','Guimaraes' ) ).
utente( 2,luis,13,morada( 'Rua do Azevinho','Braga','Braga' ) ).
utente( 3,joao,13,morada( 'Rua do Rual','São João de Ponte','Guimaraes' ) ).

%------------------------------- - - - - - - - - - - - - - - - -
% Extensao do predicado prestador: IdPrest, Nome, Especialidade, Instituição -> {V,F}

prestador( 1,jose,'ortopedia',['Hospital de Braga'] ).
prestador( 2,joao,'urologia',['Hospital de Guimaraes'] ).
prestador( 3,jorge,'neurologia',['Hospital de Guimaraes'] ).

%------------------------------- - - - - - - - - - - - - - - - -
% Extensao do predicado cuidado: Data, IdUt, IdPrest, Descrição, Custo, Instituição -> {V,F}

cuidado( data( 2018,3,14,20 ),1,1,'curativo',23,'Hospital de Braga'  ).
cuidado( data( 2018,3,10,20 ),2,3,'rotina',23,'Hospital de Guimaraes' ).
cuidado( data( 2018,3,10,20 ),3,3,'rotina',23,'Hospital de Guimaraes' ).
cuidado( data( 2018,2,10,20 ),2,1,'emergência',23,'Hospital de Braga' ).
cuidado( data( 2018,3,15,20 ),3,3,'curativo',34,'Hospital de Guimaraes' ).

%------------------------------- - - - - - - - - - - - - - - - -
% Extensao do predicado instituição: Nome, Cidade, Tipo, ListaEspecialidade -> {V,F}

instituicao( 'Hospital de Braga','Braga','Hospital',['ortopedia'] ).
instituicao( 'Hospital de Guimaraes','Guimaraes','Hospital',['urologia','neurologia'] ).
```

Figura 1 - Povoamento da Base de Conhecimento

3.3. Meta-Predicados

3.3.1. Inserção

Para realizar os predicados de inserção, decidimos criar um predicado para cada um dos predicados referidos anteriormente.

Para isso necessitamos de criar um predicado geral que realiza uma evolução à nossa base de conhecimento, o predicado *evolução*. Este predicado tem como função, testar a possibilidade de adicionar o facto referido atendendo aos invariantes criados e caso seja possível, então insere-o na base de conhecimento.

Decidimos criar o predicado **evolução(T)**, que manipula um facto como conhecimento. Este predicado guarda todos os invariantes relativos à inserção do facto que vai inserir numa lista, inserindo o facto e testando cada um dos invariantes após a inserção. Caso um dos invariantes falhe e não posso evoluir o conhecimento, então retira o facto inserido.

A inserção é realizada com um predicado auxiliar (**inserir**), que tem a responsabilidade de remover o facto previamente inserido caso os invariantes falhem.

O predicado de teste (**testar**) manipula como conhecimento um facto e uma lista de invariantes, sendo que terá de testar cada um destes invariantes para o facto referido. Apenas será verdadeiro caso todos os invariantes sejam positivamente testados.

De seguida apresentamos os predicados referidos:

```
evolucao: Termo -> {V,F}
evolucao( Termo ) :- solucoes( Inv,+Termo::Inv,S ),
                              inserir( Termo ),
                              testar( S ).
inserir: Predicado -> {V,F}
inserir( P ) :- assert( P ).
inserir( P ) :- retract( P ), !, fail.

testar: ListaPredicado -> {V,F}
testar( [] ).
testar( [X|R] ) :- X,
                   testar( R ).
```

Para realizar a inserção de um utente criamos o predicado *adicionarUtente* que manipula exatamente o mesmo conhecimento que o predicado utente. Este predicado apoia-se no predicado evolução, passando como termo para evoluir o conhecimento o utente com o conhecimento recebido.

De seguida apresentamos o predicado referido:

```
adicionarUtente: Id,Nome,Idade,Morada -> {V,F}
adicionarUtente( IdUtente,Nome,Idade,Morada ) :-
                        evolucao( utente( IdUtente,Nome,Idade,Morada )
).
```

Os restantes predicados de inserção, para prestadores, cuidados e instituições seguem o mesmo raciocínio que este pelo que não aprofundamos a sua explicação, referindo-os em anexo.

3.3.2. Remoção

Para realizar os predicados de remoção utilizamos o mesmo raciocínio dos predicados de inserção. A única diferença é que neste caso queremos fazer uma involução à base de conhecimento, pelo que criamos o predicado involução. Para além disto, este terá de guardar uma lista com os invariantes relativos à remoção do facto que será removido e removerá este facto caso passe nos testes realizados a todos os invariantes.

Para isso utilizamos os seguintes predicados:

```
involucao: Termo -> {V,F}
involucao( Termo ) :- solucoes(Inv,-Termo::Inv,S),
                              remover(Termo),
                              testar(S).
```

```
remover: Predicado -> {V,F}
remover( P ) :- retract( P ).
remover( P ) :- assert( P ), !, fail.
```

Para realizar a inserção de um utente criamos o predicado retirararUtente que manipula como conhecimento o identificador de um utente, pois este será suficiente para o identificar e posteriormente remover. Este predicado apoia-se no predicado involução, passando como termo para involuir o conhecimento o utente com o conhecimento recebido, e com os restantes campos pertencentes ao predicado utente como quaisquer visto que o mecanismo de teste do *Prolog* irá encontrar o único utente com aquele id, caso exista.

Os restantes predicados de remoção, para prestadores e instituições seguem o mesmo raciocínio que este pelo que não aprofundamos a sua explicação, referindo-os em anexo.

Para remover um cuidado, e como consideramos que este apenas não poderia ser igual a outro existente na base de conhecimento, é necessário que o predicado de remoção do mesmo manipule toda a informação que o predicado cuidado manipula. Por isto, criamos o seguinte predicado:

```
retirarCuidado:   Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao
-> {V,F}
retirarCuidado( Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao )
:- inevolucao( cuidado(
Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao ) ).
```

3.4. Invariantes

3.4.1. Ids Repetidos

Como referido anteriormente, o ID (campo) identifica inequivocamente um utente. Por não poder ser possível, aquando da inserção de novos utentes, inserir um utente com um identificador que já esteja presente na base de conhecimento, decidimos criar um invariante para garantir esta questão. (Tanto é válido para o utente como para o prestador)

A manipulação do invariante foi feita da seguinte forma:

```
+utente( IdU,_,_,_ ) :: (solucoes( IdU,utente( IdU,_,_,_ ),L ),
                                    comprimento( L,X ),
                                    X =< 1).
```

Apenas temos de utilizar o predicado soluções (que nos dá todas as soluções para um predicado presentes na base de conhecimento), sendo que o predicado que procuramos são os utentes com o identificador igual ao do utente adicionado. Depois apenas temos de garantir que a lista de soluções que obtivemos tem um comprimento inferior ou igual a 1 (poderia ser só igual).

```
| ?- listing(utente).
utente(2, luis, 13, morada('Rua do Azevinho','Braga','Braga')).
utente(3, joao, 13, morada('Rua do Rual','SÃ£o JoÃ£o de Ponte','Guimaraes')).
utente(1, ana, 14, morada('Rua do Louro','Caldas das Taipas','Guimaraes')).

yes
| ?- adicionarUtente( 1,ana,14,morada( 'Rua do Louro','Caldas das Taipas','Guim
araes' ) ).
no
| ?-
```

Figura 2 - Resultado do Invariante IDs Repetidos

Tal como o esperado, ao tentar acrescentar um utente com um ID já existente, tal não foi possível.

3.4.2. Data Válida

Não é possível inserir a informação relativa a um cuidado se a data desse cuidado não foi válida.

Invariante: `+cuidado( Data,U,P,Descricao,C,I ) :: (dataValida(Data)).`

Neste caso apenas temos de utilizar o predicado *dataValida* que irá verificar se todos os valores presentes na data são válidos.

```
dataValida( data(A,M,D,H) ) :-  natural(A),
                                natural(M),
                                natural(D),
                                natural(H),
                                H < 25,
                                M < 13,
                                diasValidos( A,M,D ).
```

O predicado *dataValida* começa por verificar se os valores **indicados** na data são naturais (ano,mês,dia,hora). De seguida faz duas verificações simples onde verifica se o domínio das horas e do mês está correto (um dia tem 24 horas e um ano tem 12 meses). Por fim verifica se o dia indicado é compatível com o mês e com o ano em questão (é preciso verificar quantos dias tem o mês e se o ano é bissexto caso estejamos a manipular o mês de Fevereiro).

```
| ?-
| ?-
| ?-
| ?- adicionarCuidado( data( 2018,9,14,33 ),1,1,'curativo',23,'Hospital de Brag
ança'  ).
no
| ?-
```

Figura 3 - Resultado do Invariante Data Valida

Mais uma vez, o invariante deu o resultado pretendido, ao não deixar acrescentar à base de conhecimento um cuidado com uma data inválida.

3.4.3. Número de Cuidados num Determinado Período

Um prestador de serviços apenas está admitido a prestar 3 serviços por hora, logo temos de verificar senão existem já 3 cuidados daquele profissional à hora indicada.

Invariante*:*

```
+cuidado(D,U,P,Des,C,I)::(solucoes((D,P),cuidado(D,Ut,P,Descr,Cu
s,Ins),L), comprimento(L,X), X<4).
```

Utilizando o predicado soluções, colocamos numa lista pares que contêm informações sobre o prestador em questão e a data. De seguida verificamos se essa lista contém menos de 3 pares.

```
cuidado(data(2018,3,10,20), 2, 3, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,3,10,20), 3, 3, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,2,10,20), 2, 1, emergÃªncia, 23, 'Hospital de Braga').
cuidado(data(2018,3,15,20), 3, 3, curativo, 34, 'Hospital de Guimaraes').
cuidado(data(2018,3,14,20), 1, 1, exame, 25, 'Hospital de Braga').
cuidado(data(2018,3,14,20), 1, 1, curativo, 23, 'Hospital de Braga').
cuidado(data(2018,3,14,20), 2, 1, exame, 50, 'Hospital de Braga').

yes
| ?- adicionarCuidado( data( 2018,3,14,20),3,1,exame,50,'Hospital de Braga').
no
```

Figura 4 - Resultado do predicado Numero de Cuidados

Como o prestador com o ID 1 já tem 3 cuidados à mesma hora na mesma data, não foi possível adicionar um quarto cuidado nessa hora.

3.4.4. Cuidados com utentes/prestadores Inválidos

Cuidado tem utente válido: Não podemos adicionar cuidados para os quais não existam utentes registados, logo temos de verificar se o utente presente no cuidado é válido.

Invariante: `+cuidado( _,IdU,_,_,_,_ ) :: (solucoes( IdU,utente( IdU,_,_,_ ),L ),comprimento( L,X ),X == 1).`

Utilizando o predicado soluções, construímos uma lista com o ID do utente presente no cuidado e verificamos se essa lista contem um e um só ID.

Para o prestador o predicado é idêntico, incidindo no id do prestador.

```
| ?- listing(prestador).
prestador(1, jose, ortopedia, ['Hospital de Braga']).
prestador(2, joao, urologia, ['Hospital de Guimaraes']).
prestador(3, jorge, neurologia, ['Hospital de Guimaraes']).

yes
| ?- adicionarCuidado( data( 2018,9,12,9 ),1,5,'curativo',23,'Hospital de Braga
nça'  ).
no
| ?-
```

Figura 5 - Resultados do Predicado Cuidados Inválidos

Ao tentar adicionar um cuidado no qual não existia um utente tal não foi possível acrescentar à base de conhecimento.

3.4.5. Cuidado com Prestador Inválido na Instituição

Como os prestadores não trabalham em todas as instituições temos de confirmar que ele faz parta da instituição indicada no cuidado.

Invariante: `+cuidado( _,_,IdP,_,_,I ) :: (solucoes( LI,( prestador( IdP,_,_,LI ),pertence( I,LI ) ),L ), comprimento( L,X ),  X == 1).`

Conjugando os predicados prestador e pertence, recorremos uma vez mais ao predicado solucoes. Primeiramente colocamos numa lista as instituições onde o prestador em questão trabalha e de seguida verificamos se a instituição indicada no cuidado pertence a essa mesma lista.

```
| ?- listing(prestador).
prestador(1, jose, ortopedia, ['Hospital de Braga']).
prestador(2, joao, urologia, ['Hospital de Guimaraes']).
prestador(3, jorge, neurologia, ['Hospital de Guimaraes']).

yes
| ?-  adicionarCuidado( data( 2018,4,10,20 ),2,1,'rotina',23,'Hospital de Guimaraes'
).
no
| ?-
```

Figura 6 - Resultado do Invariante Prestador Inválido na Instituição

Como o prestador 1 pertence à instituição ‘Hospital de Braga’, não foi possível adicionar o cuidado indicado já que tornaria a base de dados inconsistente.

3.4.6. Custo Inváldio

Não é possível que um cuidado prestado tenha um valor negativo. Assim sendo, temos de verificar se o custo indicado no cuidado é positivo.

Invariante: `+cuidado( _,_,_,_,C,_ ) :: (C >= 0).`

```
| ?- listing(cuidado).
cuidado(data(2018,3,14,20), 1, 1, curativo, 23, 'Hospital de Braga').
cuidado(data(2018,3,10,20), 2, 3, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,3,10,20), 3, 3, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,2,10,20), 2, 1, emergÃªncia, 23, 'Hospital de Braga').
cuidado(data(2018,3,15,20), 3, 3, curativo, 34, 'Hospital de Guimaraes').

yes
| ?- adicionarCuidado(  data( 2018,3,14,20 ),1,1,'curativo',-23,'Hospital de Br
aga'    ).
no
| ?-
```

Figura 7 - Resultado Invariante Custo Inválido

O resultado esperado para este invariante foi alcançado, já que não permite acrescentar um cuidado com valores de custos inválidos.

3.4.7. Remoção de Utente/Prestador

Não é possível remover um utente da base de conhecimento para os quais ainda existem cuidados, visto que iria retirar consistência à mesma. Para garantir o referido, criamos um invariante (tanto é válido para o utente como para o prestador).

A manipulação do invariante foi feita da seguinte forma:

```
-utente( IdU,_,_,_ ) :: (solucoes( D,cuidado(D,IdU,_,_,_,_ ),L ),
                                    comprimento( L,X ),
                                    X == 0).
```

Utilizamos o predicado soluções para obter todas as datas dos cuidados para o utente com o identificador do utente que queremos retirar. Depois apenas podemos remover se o comprimento dessa lista for zero ou seja, caso não existam cuidados.

```
| ?- listing(cuidado).
cuidado(data(2018,3,10,20), 2, 3, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,3,10,20), 3, 2, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,3,10,20), 3, 2, exame, 23, 'Hospital de Guimaraes').
cuidado(data(2018,3,10,20), 3, 1, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,2,10,20), 2, 1, emergÃªncia, 23, 'Hospital de Braga').
cuidado(data(2018,3,15,20), 3, 3, curativo, 34, 'Hospital de Guimaraes').
cuidado(data(2018,3,14,20), 1, 1, exame, 25, 'Hospital de Braga').
cuidado(data(2018,3,14,20), 1, 1, curativo, 23, 'Hospital de Braga').
cuidado(data(2018,3,14,20), 2, 1, exame, 50, 'Hospital de Braga').

yes
| ?- involucao(   utente( 3,joao,13,morada( 'Rua do Rual','São João de Ponte','Guimaraes' ) )).
no
| ?-
```

Figura 8 - Resultado do Invariante Remover

O resultado foi o esperado, já que, como para o utente com o ID 3 ainda existem cuidados na base de conhecimento, então não pode ser removido.

3.4.8. Especialidade Válida de um Prestador

Não é possível um prestador ser contratado por uma instituição onde, nessa, não tenha disponível a especialidade que o prestador possui. Desta forma, decidimos definir um invariante, aquando a inserção de um novo predicado *prestador* que impedisse que este fosse adicionado caso alguma das instituições indicadas não possuísse a respetiva especialidade.

A manipulação do invariante foi feita da seguinte forma:

```
+prestador(IDp,_,Esp,LInst)::(validaEspecialidadeInstituicao( Esp,LInst
)).

validaEspecialidadeInstituicao( Esp,[] ).

validaEspecialidadeInstituicao( Esp,[C|L] ) :-
                                instituicao( C,_,_,E ),
                                pertence( Esp,E ),
                                validaEspecialidadeInstituicao( Esp,L ).
```

O predicado *validaEspecialidadeInstituicao* apenas verifica se uma especialidade pertence a todos os elementos de uma lista de instituições. Para tal, selecionamos a instituição que está à cabeça da lista e fazemos match com o predicado instituição cujo o nome unifica com a cabeça, desta forma conseguimos obter as listas de especialidades. Posteriormente é necessário verificar se a Especialidade pertence à lista obtida, algo que é feito com o predicado pertence. Por fim, é só preciso verificar para o resto da lista.

Caso alguma lista não possua a Especialidade indicada a resposta deste predicado será *no* e consequentemente a resposta do invariante será *no* pelo que não deixará acrescentar à base de conhecimento o novo prestador. Se, se obtiver a lista vazia então, a resposta do predicado será *yes* e consequentemente será também *yes* pelo que é permitido adicionar o novo prestador à base de conhecimento, havendo uma evolução desta.

```
| ?- listing(prestador).
prestador(1, jose, ortopedia, ['Hospital de Braga']).
prestador(2, joao, urologia, ['Hospital de Guimaraes']).
prestador(3, jorge, neurologia, ['Hospital de Guimaraes']).

yes
| ?- listing(instituicao).
instituicao('Hospital de Braga', 'Braga', 'Hospital', [ortopedia]).
instituicao('Hospital de Guimaraes', 'Guimaraes', 'Hospital', [urologia,neurolo
gia]).

yes
| ?- adicionarPrestador(4,antonio,cardiologia,['Hospital de Braga']).
no
| ?-
```

Figura 9 - Resultado do Invariante Especialidade Inválida

Como a especialidade de cardiologia não existe na instituição indicada, não deverá ser possível adicionar então o prestador.

3.4.9. Instituições Válidas do Prestador

Não deve ser possível adicionar um predicado prestador onde alguma das instituições indicadas não pertença à base de conhecimento.

```
+prestador(ID, _, _, LI) :: validaInstituicao(LI).

validaInstituicao([]).

validaInstituicao([C|L]) :-
      instituicao(C, _, _, _),
      validaInstituicao(L).
```

Para fazer esta validação apenas se tornou necessário de verificar se cada elemento da lista do predicado prestador unifica com um nome de um predicado instituição que está na base de conhecimento. Isso foi feito com o auxilio de um predicado *validaInstituição*. Caso o predicado

```
| ?-
| ?- listing(instituicao).
instituicao('Hospital de Braga', 'Braga', 'Hospital', [ortopedia]).
instituicao('Hospital de Guimaraes', 'Guimaraes', 'Hospital', [urologia,neurologia]).


yes
| ?- adicionarPrestador( 4,marilia,'neurologia',['Hospital da Povoa'] ).
no
| ?-
```

Figura 10 - Resultado do Invariante de Instituição Inválida

dê a resposta *yes* então vai ser possível adicionar um novo predicado de prestador caso a resposta seja *no* então tal predicado não será válido para adicionar à base de conhecimento.

Como a instituição 'Hospital da Póvoa' não existe na base de conhecimento, tal prestador não foi possível adicionar.

3.4.10. Cuidado Igual

Não é possível inserir dois cuidados com exatamente a mesma informação uma vez que consideramos que se estavam a referir ao mesmo.

```
+cuidado(    D,IdU,IdP,Desc,C,I    )    ::    (solucoes(    LI,cuidado(
D,IdU,IdP,Desc,C,I ),L ),

                                        comprimento( L,X ),

                                        X == 1).
```

Para esta confirmação, inserimos o cuidado e verificamos se apenas existe um cuidado na lista resultante.

```
| ?-
| ?- listing(cuidado).
cuidado(data(2018,3,14,20), 1, 1, curativo, 23, 'Hospital de Braga').
cuidado(data(2018,3,10,20), 2, 3, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,3,10,20), 3, 3, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,2,10,20), 2, 1, emergÃªncia, 23, 'Hospital de Braga').
cuidado(data(2018,3,15,20), 3, 3, curativo, 34, 'Hospital de Guimaraes').

yes
| ?- adicionarCuidado( data( 2018,3,14,20 ),1,1,'curativo',23,'Hospital de Brag
a'  ).
no
| ?-
```

Figura 11 - Resultado de Adicionar Cuidado Repetido

Mais uma vez não foi possível adicionar um cuidado onde todos os campos são iguais a um cuidado que já está presente na base de conhecimento.

3.4.11. Remoção de instituição

Não é possível remover uma instituição da base de conhecimento para a qual ainda existem cuidados ou prestadores, visto que a iria tornar inconsistente. Para garantir este ponto, criamos dois invariantes.

Relativo aos prestadores:

```
-instituicao( I,C,T,LE ) :: (solucoes( P,( prestador( P,_,_,LI
),pertence( I,LI ) ),L),
                                        comprimento( L,X ),
                                        X==0).
```

Relativo aos cuidados:

```
-instituicao( I,C,T,LE ) :: (solucoes( P,prestador( _,P,_,_,_,I ),L),
                                        comprimento( L,X ),
                                        X==0).
```

```
| ?- listing(cuidado).
cuidado(data(2018,3,10,20), 2, 3, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,3,10,20), 3, 2, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,3,10,20), 3, 2, exame, 23, 'Hospital de Guimaraes').
cuidado(data(2018,3,10,20), 3, 1, rotina, 23, 'Hospital de Guimaraes').
cuidado(data(2018,2,10,20), 2, 1, emergÃªncia, 23, 'Hospital de Braga').
cuidado(data(2018,3,15,20), 3, 3, curativo, 34, 'Hospital de Guimaraes').
cuidado(data(2018,3,14,20), 1, 1, exame, 25, 'Hospital de Braga').
cuidado(data(2018,3,14,20), 1, 1, curativo, 23, 'Hospital de Braga').
cuidado(data(2018,3,14,20), 2, 1, exame, 50, 'Hospital de Braga').

yes
| ?- involucao( instituicao( 'Hospital de Braga','Braga','Hospital',['ortopedia
'] )).
no
| ?-
```

Figura 12 - Resultados Invariante Remover Instituição

Os resultados mostram então que não é possível remover uma instituição para a qual existam cuidados associados.

3.5. Predicados Propostos

3.5.1. Identificar Utentes

Para identificar os utentes utilizamos quatro critérios de seleção: Id, Nome, Idade e Morada. Dividimos posteriormente a idade em cinco subcritérios de seleção: “igual a”, “menor que”, “maior que”, “igual ou menor que” e “igual ou maior que”, e a morada em três subcritérios: rua, localidade e cidade. Para representar cada um casos de seleção descritos acima recorremos aos seguintes predicados.

```
utenteID( IdU,R ) :- solucoes( (IdU,N,I,M) , utente( IdU,N,I,M ) , R
).
```

De modo a representar o conjunto de utentes com determinadas caraterísticas, utilizamos um predicado auxiliar *solucoes* que, dados os parâmetros de seleção, agrupa um conjunto de parâmetros, indicados no primeiro argumento do solucoes, sempre que uma instância da relação *utente* satisfaça os predicados especificados no segundo argumento do *solucoes*.

```
| ?- utenteID(1,R).
R = [(1,ana,14,morada('Rua do Louro','Caldas das Taipas','Guimaraes'))] ?
yes
| ?- utenteID(2,R).
R = [(2,luis,13,morada('Rua do Azevinho','Braga','Braga'))] ?
yes
| ?- utenteID(3,R).
R = [(3,joao,13,morada('Rua do Rual','SÃ£o JoÃ£o de Ponte','Guimaraes'))] ?
yes
```

Figura 13 - Resultados do Predicado utenteID

3.5.2. Identificar as Instituições Prestadoras de Cuidados de Saúde

Para apresentar as instituições prestadoras de cuidados de saúde, recorremos também ao predicado *solucoes*, mencionado na secção anterior, que nos dá o conjunto de todas as instituições representadas na base de conhecimento. Ainda utilizamos um predicado auxiliar, *multiConjunto* onde a sua explicação será dada mais adiante.

```
instituicoesPrestadoresCuidados( R ) :- solucoes( I,cuidado(
_,_,_,_,_,I ),Aux ),
      multiConjunto( Aux,R ).
```

```
| ?- instituicoesPrestadoresCuidados(R).
R = [('Hospital de Guimaraes',3),('Hospital de Braga',2)] ?
yes
```

Figura 14 - Resultados do Predicado instituicoesPrestadoresCuidados

3.5.3. Identificar Cuidados de Saúde prestados por instituição/cidade/datas

Para identificar os cuidados de saúde prestados por uma instituição ou cidade ou por datas , utilizamos o predicado *solucoes* uma vez mais.

```
cuidadosSaudeCidade(Cid, R) :-
      solucoes((Data, IdU, IdP, Desc, Custo, Inst), ( cuidado(Data,
IdU, IdP, Desc, Custo, Inst),instituicao( Inst,Cid,_,_ ) ), R)
```

Dada a semelhança entre todas as resoluções apenas iremos explicar o raciocínio utilizado na identificação de cuidados de saúde numa dada cidade. Assim sendo, para identificação por cidade, precisamos de verificar quais as instituições que se localizam na cidade dada e quais cuidados de saúde realizados nessas instituições e agrupar os parâmetros desses cuidados (Data, ID do Utente, ID do Prestador, Descrição do serviço, Custo, Instituição).

```
| ?- cuidadosSaudeCidade('Guimaraes',R).
R = [(data(2018,3,10,20),2,3,rotina,23,'Hospital de Guimaraes'),(data(2018,3,10,20),3,3,rotina,23,'Hospital de Guimaraes'),
(data(2018,3,15,20),3,3,curativo,34,'Hospital de Guimaraes')] ?
yes
| ?- cuidadosSaudeCidade('Braga',R).
R = [(data(2018,3,14,20),1,1,curativo,23,'Hospital de Braga'),(data(2018,2,10,20),2,1,emergÃªncia,23,'Hospital de Braga')]
?
yes
```

Figura 15 - Resultados do Predicado cuidadosSaudeCidade

3.5.4. Identificar os Utentes de um prestador/especialidade/instituição

Para a identificação dos utentes de um prestador, especialidade ou instituição recorremos novamente ao predicado *solucoes*. As resoluções dos casos desta secção apresentam um raciocínio semelhante ao das secções anteriores. Contudo, nesta secção, há a adição do predicado *multiConjunto* (que também já foi utilizado num predicado anterior).

```
utentesDaEspecialidade( Esp,R ) :- solucoes( ( IdU,N ), (
prestador(IdP,_,Esp,_), cuidado(_,IdU,IdP,_,_,_), utente(IdU,N,_,_) ),
L ),
                                        multiConjunto( L,R ).
```

Como todos os casos desta secção são muito semelhantes, explicamos apenas o raciocínio para identificação de todos os utentes que efetuaram serviços de saúde de uma determinada especialidade por ser o caso mais complexo. Visto que a especialidade é um parâmetro do prestador, selecionamos todos os prestadores que possuam essa especialidade. De seguida verificamos que cuidados de saúde foram efetuados por esses prestadores através dos seus IDs para obter os IDs dos utentes. Por último selecionamos todos os clientes através dos IDs obtidos anteriormente e agrupamos os parâmetros (ID e Nome do Utente). Como o conjunto de soluções pode conter elementos repetidos, aplicamos o predicado multiConjunto para contar quantos repetidos ocorrem para cada utente. Deste modo obtemos o conjunto de todos os utentes que receberam cuidados de saúde de uma dada especialidade, sem elementos repetidos, e quantos cuidados de saúde dessa especialidade cada cliente recebeu.

```
| ?- utentesDaEspecialidade('neurologia', R).
R = [((3,joao),2),((2,luis),1)] ?
yes
% 1
| ?- utentesDaEspecialidade('urologia', R).
R = [] ?
yes
% 1
| ?- utentesDaEspecialidade('ortopedia', R).
R = [((2,luis),1),((1,ana),1)] ?
yes
```

Figura 16 - Resultados do Predicado utentesDaEspecialidade

De acordo com a nossa base de conhecimento, o predicado referido apresenta os resultados esperados.

3.5.5. Identificar cuidados de saúde realizados por utente/instituição/prestador

Para a identificação dos cuidados de saúde realizados por utente, instituição ou prestador aplicamos outra vez o predicado *solucoes*. O raciocínio e resolução destes três casos é muito semelhante aos casos abordados nas secções anteriores, iremos apresentar apenas os predicados em anexo.

```
| ?- cuidadosUtente(1,R).
R = [(data(2018,3,14,20),1,1,curativo,23,'Hospital de Braga')] ?
yes
% 1
| ?- cuidadosUtente(2,R).
R = [(data(2018,3,10,20),2,3,rotina,23,'Hospital de Guimaraes'),(data(2018,2,10,20),2,1,emergÃªncia,23,'Hospital de Braga')] ?
yes
% 1
| ?- cuidadosUtente(3,R).
R = [(data(2018,3,10,20),3,3,rotina,23,'Hospital de Guimaraes'),(data(2018,3,15,20),3,3,curativo,34,'Hospital de Guimaraes')] ?
yes
```

Figura 17 - Resultados do predicado cuidadosUtente

Mais uma vez, o conhecimento apresentado pelo predicado está de acordo com o que seria espectado.

3.5.6. Determinar Todas as instituições/prestadores a que um Utente já Recorreu

Para determinar todos os prestadores a que um utente recorreu decidimos criar um predicado, que manipula o conhecimento de um identificador do utente e a lista com os prestadores recorridos. Neste predicado auxiliamo-nos no predicado *solucoes*, sendo que queríamos uma lista com termos: identificador do prestador, nome do prestador. A *questão* que colocávamos no *solucoes* teria de ser relativa aos cuidados do utente, sendo que também teríamos de identificar o nome do prestador, daí necessitarmos do predicado *prestador.* Após obter esta lista, auxiliamo-nos ainda no predicado *multiConjunto* para obter o resultado, por forma a eliminar os repetidos e obter uma lista com pares de elementos e número de ocorrências desses elementos.

```
prestadoresRecorridosUtente: IdUtente, ListaPrestador -> {V,F}

prestadoresRecorridosUtente( IdU,R ) :- solucoes( ( P,NP ),( cuidado(
_,IdU,P,_,_,_ ),prestador( P,NP,_,_ ) ),L ),
                                multiConjunto( L,R ).
```

Para as instituições o raciocínio é semelhante e por isso apenas apresentamos em anexo.

```
| ?-
| ?- prestadoresRecorridosUtente(2, R).
R = [((1,jose),1),((3,jorge),1)] ?
yes
% 1
| ?- prestadoresRecorridosUtente(1, R).
R = [((1,jose),1)] ?
yes
% 1
| ?- prestadoresRecorridosUtente(3, R).
R = [((3,jorge),2)] ?
yes
```

Figura 18 - Resultados do Predicado prestadoresRecorridosUtentes

O predicado mencionado, tal como se pode observar na figura, apresenta os resultados pretendidos, em relação à nossa base de conhecimento.

3.5.7. Calcular o Custo Total dos Cuidados de Saúde por utente/especialidade/prestador/datas

Para determinar o total de custos de cuidados de saúde por especialidade decidimos criar um predicado, que manipula o conhecimento de uma especialidade e o total de custos. Neste auxiliamo-nos no predicado *solucoes*, sendo que queríamos uma lista com termos: custo do cuidado. A *questão* que colocávamos no *solucoes* teria de ser relativa aos cuidados de saúde, sendo que também teríamos de identificar a especialidade do cuidado, daí necessitarmos do predicado *prestador*. Após obter esta lista, auxiliamo-nos ainda no predicado *somaC* para obter o resultado, por forma a soma total dos custos.

```
Extensao do predicado totalCustoEspecialidade: Especialidade, Custo ->
{V,F}

totalCustoEspecialidade( Esp,C ) :- solucoes( Custo,(
cuidado(_,_,P,_,Custo,_),prestador(P,_,Esp,_) ),L ),
                                    somaC( L,C ).
```

```
| ?- totalCustoEspecialidade('neurologia',R).
R = 80 ?
yes
% 1
| ?- totalCustoEspecialidade('ortopedia',R).
R = 46 ?
yes
```

Figura 19 - Resultados do predicado totalCustoEspecialidade

Como podemos observar pela figura anterior, o predicado apresenta o conhecimento pretendido.

3.6. Predicado Extra

Após terminar todos os predicados indicados, decidimos criar uns predicados extras que poderiam ter uma resposta interessante sobre a base de conhecimento.

3.6.1. Utentes com Mais Custos

Decidimos implementar um predicado que nos desse o top de clientes com mais custos: utentesMaisCusto(N , R), onde N é o número de utentes que queremos visualizar (caso seja 10, então daria o top10 de clientes que mais gastam) e R seria a lista ordenada por ordem decrescente dos utentes com os seus respetivos custos.

```
utentesMaisCusto(N , R) :-
      solucoes( ( C,ID ),( utente(ID, _, _, _),totalCustoUtente(ID,C)
),L ),
      ordenaPar(L, Rs),
      take(Rs, N, R).
```

Para isso, precisávamos de uma lista completa com todos os utentes e o valor total gasto. Através do predicado *soluções* obtivemos esta lista – L. Em seguida, tornou-se necessário ordená-la por forma decrescente. Como cada elemento da lista é um par, foi preciso escrever um novo predicado que fosse capaz de ordenar uma lista de pares. O predicado ordena, como o nome indica, ordena uma lista. Manipula como conhecimento a lista desordenada e a lista ordenada. Este predicado apenas é chamado recursivamente e depois auxilia se no predicado insereOrdenado, inserindo o elemento na lista ordenada recursivamente. O predicado insereOrdenado manipula como conhecimento um elemento a inserir, a lista e a lista com o elemento inserido. Este predicado apenas verifica se o elemento é maior do que o elemento à cabeça da lista e cajo seja insere na cabeça da lista, caso seja menor invoca recursivamente na cauda da lista. De referir que para comparação de elementos utilizamos o @ visto que o SICStus interpreta este símbolo como comparador de diferentes "tipos" (apesar destes não existirem em Prolog) podendo assim ordenar listas de pares ou de elementos. Depois da lista estar ordenada, foi só aplicar um predicado, *take*, que mantém apenas os primeiros N elementos de uma lista.

```
| ?- utentesMaisCusto(0, R).
R = [] ?
yes
| ?- utentesMaisCusto(1, R).
R = [(57,3,joao)] ?
yes
| ?- utentesMaisCusto(3, R).
R = [(57,3,joao),(46,2,luis),(23,1,ana)] ?
yes
| ?-
```

Figura 20 - Resultados do Predicado utentesMaisCusto

De acordo com a nossa base de conhecimento, os resultados apresentados são o que eram pretendidos.

3.6.2. Identificar Instituições

Na nossa base de conhecimento, como já foi referido anteriormente, optamos por acrescentar um predicado de instituição, pelo que nos pareceu necessário que fosse possível

identificar instituições de diversas formas. Desta forma decidimos criar três predicados adicionais que permitissem identificar as instituições por tipo, cidade e especialidade.

```
instituicoesPorEspecialidade( Esp,R ) :- solucoes( (Inst,Cid,Tipo) , ( instituicao(Inst,Cid,Tipo,E) , pertence(Esp,E) ) , R ).
```

Para identificar as instituições por especialidade apenas foi preciso encontrar todas as instituições onde a especialidade passada como parâmetro pertencesse à lista de especialidades presentes então na instituição.

Os outros dois predicados seguem um raciocínio parecido com o da especialidade, pelo que só o apresentamos em anexo.

```
| ?- instituicoesPorEspecialidade('urologia',R).
R = [('Hospital de Guimaraes','Guimaraes','Hospital')] ?
yes
| ?- instituicoesPorEspecialidade('neurologia',R).
R = [('Hospital de Guimaraes','Guimaraes','Hospital')] ?
yes
| ?- instituicoesPorEspecialidade('ortopedia',R).
R = [('Hospital de Braga','Braga','Hospital')] ?
yes
```

Figura 21 - Resultados do Predicado instituicoesPorEspecialidade

O conhecimento adquirido do predicado anterior é consistente de acordo com a nossa base.

3.6.3. Identificar Cuidados por Tipo de Instituição

É habitual, numa área de serviços de saúde, que os cuidados prestados estejam organizados. Há variadíssimas formas de o fazer, e uma delas (bastante usual) é por tipo de instituição. Desta forma é possível efetuar estudos sobre que tipos de instituições as pessoas mais procuram, podendo assim dar uma melhor resposta à necessidade da população. Posto isto, criamos então o predicado pretendido.

```
cuidadosPorTipoDeInstituicao( Tipo,R ) :- solucoes( (D,IdU,IdP,Desc,Custo,Inst) , (instituicao(Inst,_,Tipo,_) , cuidado(D,IdU,IdP,Desc,Custo,Inst)) , R ).
```

A sua construção é bastante simples apenas foi necessário encontrar, através do predicado *soluções*, a instituição do Tipo passado como parâmetro e em que cuidados está presente essa mesma instituição. Após encontrar o pretendido foi só preciso apresentar os campos do cuidado.

```
| ?- cuidadosPorTipoDeInstituicao('Privado',R).
R = [] ?
yes
| ?- cuidadosPorTipoDeInstituicao('Hospital',R).
R = [(data(2018,3,14,20),1,1,curativo,23,'Hospital de Braga'),(data(2018,2,10,20),2,1,emergÃªncia,23,'Hospital de Braga'),(data(2018,3,10,20),2,3,rotina,23,'Hospital de Guimaraes'),(data(2018,3,10,20),3,3,rotina,23,'Hospital de Guimaraes'),(data(2018,3,15,20),3,3,curativo,34,'Hospital de Guimaraes')] ?
yes
```

Figura 22 - Resultados do Predicado cuidadosPorTipoDeInstituicao

Como na nossa base de conhecimento apenas apresentamos instituições com o tipo de hospitais públicos, então era espectável que apresentasse todos os cuidados presentes na base de conhecimento

4. Conclusões e Sugestões

Em suma, podemos afirmar, que todos os objetivos traçados pelo grupo aquando o inicio da realização deste sistema foram cumpridos. Todas as funcionalidades básicas, que eram pedidas, foram corretamente implementadas.

Quanto ao aspeto de realizar extras no trabalho, também foi cumprido com sucesso e em número considerável. Colocámos os invariantes que nos pareciam essenciais para retratar a realidade na inserção de cada predicado, para que a nossa base de conhecimento não se tornasse inconsistente. Todas as funcionalidades extras foram escritas apresentando todos os resultados previstos.

No final, o nosso sistema, apesar de toda a complexidade de um sistema de saúde, parece-nos que retrata simplificadamente a prestação de serviços na área de saúde, cumprindo assim o nosso último objetivo traçado. Um exemplo desta simplicidade foi o facto de considerarmos a possibilidade de existirem 3 cuidados por hora. Numa perspetiva mais real poderíamos considerar os minutos na data e o tempo de duração do cuidado, no entanto isso complicaria bastante o processo e esse não era o nosso objetivo.

A maior dificuldade sentida pelo grupo foi, por vezes, aplicar o nosso raciocínio corretamente de acordo com os princípios da programação em lógica e da representação do conhecimento. Este trabalho, foi sem dúvida, essencial para melhorarmos o modo de pensar para resolver problemas que envolvam a representação de raciocínio e conhecimento. De um posto de vista geral, ficamos agradados com o trabalho proposto e os resultados que obtivemos.

5. Anexos

```
%------------------------------- - - - - - - - - - - - - - - - -

% SIST. REPR. CONHECIMENTO E RACIOCINIO - MiEI/3


%------------------------------- - - - - - - - - - - - - - - - -

% TP1 - Programacao em logica e invariantes

%------------------------------- - - - - - - - - - - - - - - - -

% SICStus PROLOG: Declaracoes iniciais


:- set_prolog_flag( discontiguous_warnings,off ).

:- set_prolog_flag( single_var_warnings,off ).

:- set_prolog_flag( unknown,fail ).



%------------------------------- - - - - - - - - - - - - - - - -

% SICStus PROLOG: definicoes iniciais


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

:- dynamic utente/4.

:- dynamic prestador/4.

:- dynamic cuidado/6.

:- dynamic instituicao/4.



%------------------------------- - - - - - - - - - - - - - - - -

% Extensao do predicado utente: IdUt, Nome, Idade, Morada -> {V,F}


utente( 1,ana,14,morada( 'Rua do Louro','Caldas das Taipas','Guimaraes'
) ).

utente( 2,luis,13,morada( 'Rua do Azevinho','Braga','Braga' ) ).

utente( 3,joao,13,morada( 'Rua do Rual','São João de Ponte','Guimaraes'
) ).


%------------------------------- - - - - - - - - - - - - - - - -

% Extensao do predicado prestador: IdPrest, Nome, Especialidade,
Instituição -> {V,F}
```

```
prestador( 1,jose,'ortopedia',['Hospital de Braga'] ).

prestador( 2,joao,'urologia',['Hospital de Guimaraes'] ).

prestador( 3,jorge,'neurologia',['Hospital de Guimaraes'] ).


%-------------------------------- - - - - - - - - - -  -  -  -  -  -

% Extensao do predicado cuidado: Data, IdUt, IdPrest, Descrição, Custo, Instituição -> {V,F}


cuidado( data( 2018,3,14,20 ),1,1,'curativo',23,'Hospital de Braga'  ).

cuidado( data( 2018,3,10,20 ),2,3,'rotina',23,'Hospital de Guimaraes' ).

cuidado( data( 2018,3,10,20 ),3,3,'rotina',23,'Hospital de Guimaraes' ).

cuidado( data( 2018,2,10,20 ),2,1,'emergência',23,'Hospital de Braga' ).

cuidado( data( 2018,3,15,20 ),3,3,'curativo',34,'Hospital de Guimaraes' ).


%-------------------------------- - - - - - - - - - -  -  -  -  -  -

% Extensao do predicado instituição: Nome, Cidade, Tipo, ListaEspecialidade -> {V,F}


instituicao( 'Hospital de Braga','Braga','Hospital',['ortopedia'] ).

instituicao(                         'Hospital                         de Guimaraes','Guimaraes','Hospital',['urologia','neurologia'] ).


%----------------------------------Data do cuidado é válida-------------------------------


+cuidado( Data,U,P,Descricao,C,I ) :: (dataValida(Data)).



%----------------- ----- --- -- -

% Extensao do predicado dataValida: Data -> {V,F}

dataValida( data(A,M,D,H) ) :-      natural(A),
                                          natural(M),
                                          natural(D),
                                          natural(H),
```

```
                                        H < 25,
                                        M < 13,
                                        diasValidos( A,M,D ).

diasValidos( A,M,D ) :- R is M mod 2,
                                        R \= 0,
                                        D < 32,
                                        D > 0.
diasValidos( A,M,D ) :- R is M mod 2,
                                        R == 0,
                                        M \= 2,
                                        D < 30,
                                        D > 0.
diasValidos( A,2,D ) :- R is A mod 4,
                                        R \= 0,
                                        D < 29,
                                        D > 0.
diasValidos( A,2,D ) :- R is A mod 4,
                                        R == 0,
                                        D < 30,
                                        D > 0.

%----------------------------------Não pode haver mais do que 3 cuidados
à mesma hora tanto para o utente como o profissional-----------------

+cuidado(D,U,P,Des,C,I)                                                 ::
(solucoes((D,P),cuidado(D,Ut,P,Descr,Cus,Ins),L),
                                        comprimento(L,X),
                                        X=<3
                                        ).

+cuidado(D,U,P,Des,C,I)                                                 ::
(solucoes((D,U),cuidado(D,U,Pr,Descr,Cus,Ins),L),
                                        comprimento(L,X),
                                        X=<3
                                        ).
```

```
%-------- Não podem haver ids repetidos -------------

+utente( IdU,_,_,_ ) :: (solucoes( IdU,utente( IdU,_,_,_ ),L ),
                                    comprimento( L,X ),
                                    X =< 1).

+prestador( IdP,_,_,_ ) :: (solucoes( IdP,prestador( IdP,_,_,_ ),L ),
                                    comprimento( L,X ),
                                    X =< 1).

%--------- Não pode ter uma idade inválida ----------
+utente( IdU,_,I,_ ) :: ( validadeIdade( I ) ).

%--------------- --- -- -
% Extensao do predicado validaIdade: Idade -> {V,F}
validaIdade( A ) :- R is A+1,
                            natural(R).

%--------------- Não se podem adicionar cuidados para os quais não existam utentes/prestadores -------------------

+cuidado( _,IdU,_,_,_,_ ) :: (solucoes( IdU,utente( IdU,_,_,_ ),L ),
                                    comprimento( L,X ),
                                    X == 1).

+cuidado( _,_,IdP,_,_,_ ) :: (solucoes( IdP,prestador( IdP,_,_,_ ),L ),
                                    comprimento( L,X ),
                                    X == 1).

%--------------- Não se pode adicionar um cuidado prestado numa instituição da qual o prestador não faça parte ---------------

+cuidado( _,_,IdP,_,_,I ) :: (solucoes( LI,( prestador( IdP,_,_,LI ),pertence( I,LI ) ),L ),
                                    comprimento( L,X ),
                                    X == 1).
```

```
%--------------- Não se pode adicionar um cuidado cujo custo seja
negativo ------------------------


+cuidado( _,_,_,_,C,_ ) :: (C >= 0).


%----------------- Não se pode adicionar um cuidado com os campos todos
iguais --------------------

+cuidado(    D,IdU,IdP,Desc,C,I    )    ::    (solucoes(    I,cuidado(
D,IdU,IdP,Desc,C,I ),L ),
                                    comprimento( L,X ),
                                    X == 1).


%--------------- Não se pode remover um utente/prestador para o qual
existam cuidados relativos ---------------


-utente( IdU,_,_,_ ) :: (solucoes( D,cuidado(D,IdU,_,_,_,_ ),L ),
                                    comprimento( L,X ),
                                    X == 0).



-prestador( IdP,_,_,_ ) :: (solucoes( D,cuidado(D,_,IdP,_,_,_ ),L ),
                                    comprimento( L,X ),
                                    X == 0).


%--------------- Garantir que o prestador tem a especialidade que
pertença à lista de especialidades da instituição a que pertença -----


+prestador(  IDp,_,Esp,LInst  )  ::  (validaEspecialidadeInstituicao(
Esp,LInst )).


%------------------------ ---- -- -
% Extensao do predicado validaEspecialidadeInstituicao: Especialidade,
ListaInstituicao -> {V,F}
validaEspecialidadeInstituicao( Esp,[] ).
validaEspecialidadeInstituicao( Esp,[C|L] ) :- instituicao( C,_,_,E ),
      pertence( Esp,E ),
      validaEspecialidadeInstituicao( Esp,L ).
```

```
%----------------- Não se pode adicionar prestadores que prestem serviços em instituições que não estejam registadas no sistema ----------

+prestador(ID, _, _, LI) :: validaInstituicao(LI).


%------------------------- ---- -- -

% Extensao do predicado validaInstituicao: ListaInstituicao -> {V,F}

validaInstituicao([]).

validaInstituicao([C|L]) :-

      instituicao(C, _, _, _),

      validaInstituicao(L).


%--------------- Não se pode remover uma instituicao para a qual existam prestadores ----------------------

-instituicao( I,C,T,LE ) :: (solucoes( P,( prestador( P,_,_,LI ),pertence( I,LI ) ),L),

                                          comprimento( L,X ),

                                          X==0).


%--------------- Não se pode remover uma instituicao para a qual existam cuidados ----------------------

-instituicao( I,C,T,LE ) :: (solucoes( P,prestador( _,P,_,_,_,I ),L),

                                          comprimento( L,X ),

                                          X==0).


%------------------------- Predicados de inserção ----------------------------------

%---------------- Adicionar utente ----------------------------------

% Extensao do predicado adicionarUtente: Id,Nome,Idade,Morada -> {V,F}


adicionarUtente( IdUtente,Nome,Idade,Morada ) :-

      evolucao( utente( IdUtente,Nome,Idade,Morada ) ).



%---------------- Adicionar prestador ----------------------------------
```

```
%       Extensao       do       predicado       adicionarPrestador:
Id,Nome,Especialidade,ListaInstituicao -> {V,F}


adicionarPrestador( IdPrestador,Nome,Especialidade,ListaI ) :-

      evolucao( prestador( IdPrestador,Nome,Especialidade,ListaI ) ).


%---------------- Adicionar cuidado ----------------------------------
-

%        Extensao        do        predicado        adicionarCuidado:
Data,IdUtente,IdPrestador,Descricao,Custo,Instituicao -> {V,F}


adicionarCuidado( Data,IdUtente,IdPrestador,Descricao,Custo,Instituicao
) :-

      evolucao(                                                cuidado(
Data,IdUtente,IdPrestador,Descricao,Custo,Instituicao ) ).


%---------------- Adicionar instituicao ------------------------------
-----

%       Extensao       do       predicado       adicionarInstituicao:
Nome,Cidade,Tipo,ListaEspecialidade -> {V,F}


adicionarInstituicao( Nome,Cidade,Tipo,ListaEspecialidade ) :-

      evolucao( instituicao( Nome,Cidade,Tipo,ListaEspecialidade ) ).


%-------------------------- Predicados de remocao ---------------------
-------------


%---------------- Remover utente -----------------------------------

% Extensao do predicado retirarUtente: Id -> {V,F}


retirarUtente( IdUtente ) :-
                              inevolucao( utente( IdUtente,_,_,_ ) ).



%---------------- Remover prestador ----------------------------------
-

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

```
retirarPrestador( IdPrestador ) :-
                                        inevolucao(         prestador(
IdPrestador,_,_,_ ) ).


%---------------- Remover cuidado -----------------------------------

%         Extensao         do         predicado         retirarCuidado:
Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao -> {V,F}

retirarCuidado(  Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao  )
:-

      inevolucao(                                                cuidado(
Data,IdUtente,IdPrestador,Descricao,Custo,Instituiao ) ).


%---------------- Remover instituicao -------------------------------
---

% Extensao do predicado retirarInstituicao: Nome -> {V,F}

retirarInstituicao( Instituicao ) :-
                                        inevolucao(       instituicao(
Instituicao,_,_,_ ) ).



%--------------------------Identificar utentes ------------------------
---------------


%---------- Por Id ---------------------

% Extensao do predicado utenteId: IdU,Resposta(Utente) -> {V,F}


utenteID( IdU,R ) :- solucoes( (IdU,N,I,M) , utente( IdU,N,I,M ) , R ).


%------------- Por Nome -----------

% Extensao do predicado utenteNome: Nome,Resposta(ListaUtente) -> {V,F}

utenteNome( N,R ) :- solucoes( (IdU,N,I,M) , utente( IdU,N,I,M ) , R ).


%----------- Por Morada --------------

% Extensao do predicado utenteMorada: Tipo(Rua,Localidade ou Cidade),
Morada, Resposta(ListaUtente) -> {V,F}


utenteMorada( rua,Rua,R ) :- solucoes( ( IdU,N,I,morada( Rua,Loc,Cid )
) , utente( IdU,N,I,morada( Rua,Loc,Cid ) ) , R ).

utenteMorada(  localidade,Loc,R  )  :-  solucoes(  (  IdU,N,I,morada(
Rua,Loc,Cid ) ) , utente( IdU,N,I,morada( Rua,Loc,Cid ) ) , R ).
```

```
utenteMorada( cidade,Cid,R ) :- solucoes( ( IdU,N,I,morada( Rua,Loc,Cid ) ) , utente( IdU,N,I,morada( Rua,Loc,Cid ) ) , R ).


%------------ Por idade --------------------

% Extensao do predicado utentePorIdade: Relacao(>,<,=,=<,>=), Idade, Resposta(ListaUtente) -> {V,F}


utentePorIdade( =,I,X ) :- solucoes( (Id,N),utente(Id,N,I,M),X ).

utentePorIdade( <,I,X ) :- solucoes( (Id,N),(utente(Id,N,Idade,M), Idade < I),X ).

utentePorIdade( >,I,X ) :- solucoes( (Id,N),(utente(Id,N,Idade,M), Idade > I),X ).

utentePorIdade( =<,I,X ) :- solucoes( (Id,N),(utente(Id,N,Idade,M), Idade =< I),X ).

utentePorIdade( >=,I,X ) :- solucoes( (Id,N),(utente(Id,N,Idade,M), Idade >= I),X ).



%------------------------------ Identificar instituicoes prestadoras de cuidados ----------------------------

% Extensao do predicado instituicoesPrestadoresCuidados: Resposta(ListaInstituicao) -> {V,F}


instituicoesPrestadoresCuidados( R ) :- solucoes( I,cuidado( _,_,_,_,_,I ),Aux ),

      multiConjunto( Aux,R ).



%-------------------------- Cuidados de saude prestados ---------------
----------------------------------


%%----------------------------- Por instituicao

% Extensao do predicado cuidadosSaudeInstituicao: Instituicao, ListaCuidado -> {V,F}


cuidadosSaudeInstituicao(Inst, R) :-

      solucoes((Data, IdU, IdP, Desc, Custo), cuidado(Data, IdU, IdP, Desc, Custo, Inst), R).


%%---------------------------------------- Por cidade
```

```
% Extensao do predicado cuidadosSaudeCidade: Cidade, ListaCuidado ->
{V,F}

cuidadosSaudeCidade(Cid, R) :-

      solucoes((Data, IdU, IdP, Desc, Custo, Inst), ( cuidado(Data, IdU,
IdP, Desc, Custo, Inst),instituicao( Inst,Cid,_,_ ) ), R).


%%---------------------------------------------- Por Data

% Extensao do predicado cuidadosSaudeData: Criterio(maior,igual,menor),
Data, ListaCuidado -> {V,F}

cuidadosSaudeData(igual, Data, R) :-

      solucoes((D, IdU, IdP, Desc, Custo, Inst), (cuidado(D, IdU, IdP,
Desc, Custo, Inst) , (comparaData(igual,D, Data, D))), R).

cuidadosSaudeData(maior, Data, R) :-

      solucoes((D, IdU, IdP, Desc, Custo, Inst), (cuidado(D, IdU, IdP,
Desc, Custo, Inst) , (comparaData(maior,D, Data, D))), R).

cuidadosSaudeData(menor, Data, R) :-

      solucoes((D, IdU, IdP, Desc, Custo, Inst), (cuidado(D, IdU, IdP,
Desc, Custo, Inst) , (comparaData(menor,D, Data, D))), R).


%------------

% Extensao do predicado cuidadosSaudeData: Criterio(entre), Data, Data,
ListaCuidado -> {V,F}

cuidadosSaudeData(entre, Data1, Data2, R) :-

      solucoes((D, IdU, IdP, Desc, Custo, Inst), (cuidado(D, IdU, IdP,
Desc,     Custo,     Inst)     ,     (comparaData(maior,D,     Data1,
D)),(comparaData(menor,D, Data2, D))), R).


%-------------------------------------------------------------

% Extensao do predicado comparaData: Criterio(maior,menor), Data, Data
-> {V,F}


comparaData(igual,data(A1, M1, D1, H1), data(A2, M2, D2, H2), data(A1,
M1, D1, H1)) :-

      A1 == A2,

      M1 == M2,

      D1 == D2,

      H1 == H2.

comparaData(maior,data(A1, M1, D1, H1), data(A2, M2, D2, H2), data(A1,
M1, D1, H1)) :-

      A1 > A2.

comparaData(maior,data(A1, M1, D1, H1), data(A2, M2, D2, H2), data(A1,
M1, D1, H1)) :-
```

```
      A1 == A2,
      M1 > M2.
comparaData(maior,data(A1, M1, D1, H1), data(A2, M2, D2, H2), data(A1,
M1, D1, H1)) :-
      A1 == A2,
      M1 == M2,
      D1 > D2.
comparaData(maior,data(A1, M1, D1, H1), data(A2, M2, D2, H2), data(A1,
M1, D1, H1)) :-
      A1 == A2,
      M1 == M2,
      D1 == D2,
      H1 > H2.
comparaData(menor,data(A1, M1, D1, H1), data(A2, M2, D2, H2), data(A1,
M1, D1, H1)) :-
      A1 < A2.
comparaData(menor,data(A1, M1, D1, H1), data(A2, M2, D2, H2), data(A1,
M1, D1, H1)) :-
      A1 == A2,
      M1 < M2.
comparaData(menor,data(A1, M1, D1, H1), data(A2, M2, D2, H2), data(A1,
M1, D1, H1)) :-
      A1 == A2,
      M1 == M2,
      D1 < D2.
comparaData(menor,data(A1, M1, D1, H1), data(A2, M2, D2, H2), data(A1,
M1, D1, H1)) :-
      A1 == A2,
      M1 == M2,
      D1 == D2,
      H1 < H2.



% ------------------------------- Identificar os utentes de um
prestador/especialidade/instituição ---------------------------------

%-------------------------------------------------

% Extensao do predicado utentesDoPrestador: IdPrestador, ListaUtente ->
{V,F}

utentesDoPrestador(   IdP,R   )   :-   solucoes(   (   IdU,N   ),   (
cuidado(_,IdU,IdP,_,_,_), utente(IdU,N,_,_) ), L ),
```

```
                                multiConjunto( L,R ).

%-----------------------------------------------------

% Extensao do predicado utentesDaEspecialidade: Especialidade, ListaUtente -> {V,F}

utentesDaEspecialidade( Esp,R ) :- solucoes( ( IdU,N ), ( prestador(IdP,_,Esp,_), cuidado(_,IdU,IdP,_,_,_), utente(IdU,N,_,_) ), L ),

                                    multiConjunto( L,R ).


%-----------------------------------------------------

% Extensao do predicado utentesDaInstituicao: Instituicao, ListaUtente -> {V,F}

utentesDaInstituicao( Inst,R ) :- solucoes( ( IdU,N ), ( cuidado(_,IdU,_,_,_,Inst), utente(IdU,N,_,_) ), L ),

                                    multiConjunto( L,R ).




%------------------------------ Identificar cuidados de utente/prestador/instituicao ----------------------------------------------


%-----------------------------------------------------

% Extensao do predicado cuidadosUtente: IdUtente, ListaCuidado -> {V,F}

cuidadosUtente(IdU,R) :- solucoes( (D,IdU,IdP,Desc,C,Inst) , cuidado(D,IdU,IdP,Desc,C,Inst) , R ).


%-----------------------------------------------------

% Extensao do predicado cuidadosInstituicao: Instituicao, ListaCuidado -> {V,F}

cuidadosInstituicao(Inst,R) :- solucoes( (D,IdU,IdP,Desc,C) , cuidado(D,IdU,IdP,Desc,C,Inst), R ).


%-----------------------------------------------------

% Extensao do predicado cuidadosPrestador: IdPrestador, ListaCuidado -> {V,F}

cuidadosPrestador(IdP,R) :- solucoes( (D,IdU,IdP,Desc,C,Inst) , cuidado(D,IdU,IdP,Desc,C,Inst) , R ) .
```

```
%--------------------------------       Determinar       todas       as
instituicoes/prestadores a que um utente já recorreu-------------------
---


%-------------------------------------------------

% Extensao do predicado instituicoesRecorridasUtente: IdUtente,
ListaInstituicao -> {V,F}

instituicoesRecorridasUtente(      IdU,R      )      :-      solucoes(
I,cuidado(_,IdU,_,_,_,I),L ),

      multiConjunto( L,R ).


%-------------------------------------------------

% Extensao do predicado prestadoresRecorridosUtente: IdUtente,
ListaPrestador -> {V,F}

prestadoresRecorridosUtente( IdU,R ) :- solucoes( ( P,NP ),( cuidado(
_,IdU,P,_,_,_ ),prestador( P,NP,_,_ ) ),L ),

      multiConjunto( L,R ).



% ---------------------------------Total Custo -------------------------
----------------------


%-------------------------------------------------

% Extensao do predicado totalCustoUtente: IdUtente, Custo -> {V,F}

totalCustoUtente(         IdU,C         )         :-         solucoes(
Custo,cuidado(_,IdU,_,_,Custo,_),L ),

                                        somaC( L,C ).


%-------------------------------------------------

% Extensao do predicado totalCustoEspecialidade: Especialidade, Custo -
> {V,F}

totalCustoEspecialidade(    Esp,C    )    :-    solucoes(    Custo,(
cuidado(_,_,P,_,Custo,_),prestador(P,_,Esp,_) ),L ),

                                        somaC( L,C ).


%-------------------------------------------------

% Extensao do predicado totalCustoPrestador: IdPrestador, Custo -> {V,F}

totalCustoPrestador(    IdP,C    )    :-    solucoes(    Custo    ,
cuidado(_,_,IdP,_,Custo,_) , L ),

                                        somaC( L,C ).
```

```
%-------------------------------------------------

% Extensao do predicado totalCustoPorDatas: Criterio(maior,igual,menor),
Data, ListaCuidado -> {V,F}

totalCustoPorDatas(   igual,Data,C   )   :-   solucoes(   Custo   ,
(cuidado(D,_,_,_,Custo,_) , comparaData(igual,D, Data, D)), L), somaC(
L,C ).

totalCustoPorDatas(   maior,Data,C   )   :-   solucoes(   Custo   ,
(cuidado(D,_,_,_,Custo,_) , comparaData(maior,D, Data, D)), L), somaC(
L,C ).

totalCustoPorDatas(   menor,Data,C   )   :-   solucoes(   Custo   ,
(cuidado(D,_,_,_,Custo,_) , comparaData(menor,D, Data, D)), L), somaC(
L,C ).


% Extensao do predicado totalCustoPorDatas: Criterio(entre), Data, Data,
ListaCuidado -> {V,F}

totalCustoPorDatas(  entre,Data1,Data2,C  )  :-  solucoes(  Custo  ,
(cuidado(D,_,_,_,Custo,_)   ,   comparaData(maior,D,   Data1,   D)   ,
comparaData(menor,D, Data2, D)), L), somaC( L,C ).




% ------------------------------------------ PREDICADOS EXTRA ---------
-----------------------------------------


% ------------------------------------------ Utente Com Mais Custos ---
-----------------------------------------


%-------------------------------------------------

% Extensao do predicado utentesMaisCusto: Numero, ListaUtentes -> {V,F}

utentesMaisCusto( N,R) :-

      solucoes(     (     C,ID,No     ),(     utente(ID,     No,     _,
_),totalCustoUtente(ID,C) ),L ),

      ordena(L, Rs),

      take(Rs, N, R).



%-------------------------------------- Identificar instituicao ------
-------------------------------------


%-------------------------------------------------
```

```
% Extensao do predicado instituicoesPorTipo: Tipo, ListaInstituicoes -> {V,F}

instituicoesPorTipo( Tipo,R ) :- solucoes( (Inst,Cid,Tipo,Esp) , instituicao(Inst,Cid,Tipo,Esp) , R ).


%-----------------------------------------------

% Extensao do predicado instituicoesPorCidade: Cidade, ListaInstituicoes -> {V,F}

instituicoesPorCidade( Cid,R ) :- solucoes( (Inst,Cid,Tipo,Esp) , instituicao(Inst,Cid,Tipo,Esp) , R ).


%-----------------------------------------------

% Extensao do predicado instituicoesPorEspecialidade: Especialidade, ListaInstituicoes -> {V,F}

instituicoesPorEspecialidade( Esp,R ) :- solucoes( (Inst,Cid,Tipo) , ( instituicao(Inst,Cid,Tipo,E) , pertence(Esp,E) ) , R ).



%---------------------------- Identificar cuidados por tipo de instituicao ----------------------------------


%-----------------------------------------------

% Extensao do predicado cuidadosPorTipoDeInstituicao: Tipo, ListaCuidados -> {V,F}

cuidadosPorTipoDeInstituicao( Tipo,R ) :- solucoes( (D,IdU,IdP,Desc,Custo,Inst) , (instituicao(Inst,_,Tipo,_) , cuidado(D,IdU,IdP,Desc,Custo,Inst)) , R ).



% ------------------------------------------ PREDICADOS AUXILIARES ----------------------------------------


%-----------------------------------------------

% Extensao do predicado natural: Numero -> {V,F}

natural( 1 ).

natural( X ) :- X < 1, !, fail.

natural( N ) :- R is N-1,

            natural(R).


%-----------------------------------------------
```

```
% Extensao do predicado somaC: Lista,Soma -> {V,F}
somaC( [],0 ).
somaC( [C|L],R ) :- somaC( L,A ),
                    R is C+A.

%--------------------------------------------------
% Extensao do predicado multiConjunto: Lista, Lista com pares (ocorrencia,nrOcorrencia) -> {V,F}
multiConjunto( [],[] ).
multiConjunto( [C|L], R ) :- multiConjunto(L,A),
                                        insereMultiConjunto(C,A,R).

%--------------------------------------------------
% Extensao do predicado insereMultiConjunto: Elemento, Lista, Lista com pares (ocorrencia,nrOcorrencia) -> {V,F}
insereMultiConjunto( E,[],[(E,1)] ).
insereMultiConjunto( E,[(E,N)|L],[(E,M)|L] ) :- M is N+1.
insereMultiConjunto( E,[(C,N)|L],[(C,N)|R] ) :- E \= C,
      insereMultiConjunto(E,L,R).

%--------------------------------------------------
% Extensao do predicado solucoes: Termo,Questao,Solucao(ListaQuestao) -> {V,F}
solucoes(X,Y,Z) :- findall(X,Y,Z).
%--------------------------------------------------
% Extensao do predicado construir: ListaInicial,ListaFinal -> {V,F}
construir( L,[F|R] ) :- retract( tmp(F) ),
                                    construir( L,R ).
construir( S,S ).


%% --------------------------
%% Ordena uma lista de pares por ordem decrescente
% Extensao do predicado ordena: Lista,ListaOrdenada -> {V,F}
ordena([],[]).
ordena([H|T],L) :- ordena(T,T1), insereOrdenado(H,T1,L).
```

```
%------------ Insere um elemento ordenado numa lista

% Extensao do predicado insereOrdenado: Elemento,Lista,ListaComElemento
-> {V,F}

insereOrdenado(X,[],[X]).

insereOrdenado(X,[Y|Ys],[X | [Y|Ys] ]) :- X @>= Y.

insereOrdenado(X,[Y|Ys],[Y|Zs]) :- X @< Y, insereOrdenado(X,Ys,Zs).


%% --------------------------

%% Fica os primeiros n elementos de uma lista

% Extensao do predicado take: Lista,NumeroElementos,ListaComOsNElementos
-> {V,F}

take( [],_,[] ).

take( C,0,[] ).

take( [C|L],N,[C|R] ) :-

      M is N-1,

      take( L,M,R ).



%-------------------------------- - - - - - - - - - -  -  -  -  -   -

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

pertence( X,[X|L] ).

pertence( X,[C|L] ) :- pertence( X,L ).


%-------------------------------- - - - - - - - - - -  -  -  -  -   -

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

comprimento( [],0 ).

comprimento( [C|L],R ) :- comprimento( L,A ),

                          R is A+1.




%---------------------------- Predicados de Evolucao ------------------
-


%--------------------------------------------------

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

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

```
                              inserir( Termo ),
                              testar( S ).

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


%-----------------------------------------------------
% Extensao do predicado involucao: Termo -> {V,F}
inevolucao( Termo ) :- solucoes(Inv,-Termo::Inv,S),
                              remover(Termo),
                              testar(S).


%-----------------------------------------------------
% Extensao do predicado remover: Predicado -> {V,F}
remover( P ) :- retract( P ).
remover( P ) :- assert( P ), !, fail.

%-----------------------------------------------------
% Extensao do predicado testar: ListaPredicado -> {V,F}
testar( [] ).
testar( [X|R] ) :- X,
                        testar( R ).

%-------------------------- Predicado não - negacao por falha na prova
----------------
% Extensao do predicado nao: Termo -> {V,F}

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