UNIVERSIDADE DO MINHO

Programação em Lógica e Invariantes

Mestrado integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

(2º semestre, ano letivo 2016/17)

A75135 Bruno Pereira
A73580 Maria Ana de Brito
A27748 Gustavo Andrez
A74634 Rogério Moreira
A76507 Samuel Ferreira

Braga,

2017-03- 19

Índice

Resumo

O trabalho aqui apresentado consiste no desenvolvimento de um sistema de representação e raciocínio de conhecimento que permita caracterizar um universo de discurso com o qual se pretende descrever um determinado cenário.

O cenário está enquadrado na área da prestação de cuidados de saúde pela realização de atos médicos. Assim, recorrendo a conhecimentos já adquiridos, pretende-se que se desenvolva certas operações que sejam úteis dentro do universo hospitalar, nomeadamente para médicos e utentes do serviço.

A realização do exercício permitiu ao grupo conhecer e aplicar algumas funcionalidades da linguagem de programação em lógica PROLOG, permitindo melhorar a capacidade de utilização da ferramenta *SICStus Prolog*.

9. Introdução

9.1. Motivação e objetivos

Este projeto proposto na Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio surge como uma oportunidade para colocar em prática os conhecimentos aí adquiridos.

Assim, o principal objetivo torna-se em desenvolver, através do conhecimento já adquiridos, um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso.

Através deste exemplo, mostramos como é que o PROLOG poderá ser útil para ser utilizado na resolução de problemas mais complexos.

Este projeto serve também para aplicar os conhecimentos adquiridos na unidade curricular a um problema real.

9.2. Estrutura

O relatório é constituído por oito partes, que se enunciarão de seguida.

A primeira parte, menciona os principais objetivos e a motivação, bem como uma pequena introdução ao projeto desenvolvido. A segunda contextualiza-se o projeto, na terceira parte explicamos o que foi definido como base de conhecimento durante a elaboração do projeto e como garantimos a sua integrabilidade.

Na quarta são apresentadas e justificadas todas as decisões feitas pelo grupo na escolha dos predicados, obrigatórios, adicionais e auxiliares, a serem desenvolvidos, na quinta parte são mostrados alguns exemplos práticos.

Por fim, são feitas as conclusões do exercício realizado, indicadas as referências bibliográficas e, em anexo, encontra-se todos os predicados definidos.

9.3. Introdução

O projeto apresentado, que faz parte da componente prática da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio é o primeiro de vários exercícios. Um dos principais objetivos deste primeiro exercício é a utilização da linguagem de programação PROLOG, para representar conhecimento e construir mecanismos de raciocínio para a resolução de problemas.

O universo de discurso para o qual se pretende que se desenvolva este sistema de representação de conhecimento é o universo hospitalar. Assim, este projeto consiste em construir uma base de conhecimento sobre atos médicos, utentes e cuidados prestados. Para além destes, estendemos o conhecimento adicionando informação sobre médicos e especialidades médicas.

10. Preliminares

10.1. Estudos Anteriores

Para a realização deste projeto foram necessários alguns conhecimentos prévios sobre programação em lógica e o uso da linguagem PROLOG.

Este conhecimento foi adquirido ao longo das aulas teóricas (programação em lógica) e aulas teórico-práticas (programação com PROLOG) de SRCR.

Sobre estes conhecimentos destacamos todos os conceitos que foram aprendidos, tais como o que são predicados, o que são cláusulas, o que é uma base de conhecimento, entre outros.

Depois de adquiridos estes conhecimentos, o projeto tem como objetivo coloca-los em prática numa situação que simula um problema enquadrado no mundo real.

10.2. Programação em Lógica e Prolog

Apresentamos de seguida alguns conceitos fundamentais para a realização e compreensão deste trabalho:

- Factos: declaram coisas incondicionalmente verdadeiras;
- Regras: declaram coisas que podem ou não ser verdadeiras, mediante certas condições;
- Conectivos: permitem a estruturação modular do conhecimento;
- Teoremas: deduzidos através de regras e factos para provar algo.

Este são apenas alguns dos exemplos de conhecimento base que permitem perceber a programação em lógica. De seguida são descritos como estes conhecimentos são descritos na linguagem de programação utilizada.

- “Se” - representa-se por: “:-”;
- “E” - representa-se por: “,”;

Foi graças a este tipo de conhecimentos, entre outros descritos quando pertinentes, que foi possível a elaboração deste trabalho prático.

11. Base de Conhecimento

Nesta secção será apresentada a forma como o grupo decidiu organizar o conhecimento do exercício proposto, partindo da estrutura mais elementar (os factos e os seus relacionamentos) até à integridade da base de conhecimento em causa (invariantes definidos e sua justificação).

11.1. Conhecimento Obrigatório

- **Cuidados**

Os cuidados prestados são caracterizados por um número de identificação único, que associa uma determinada especialidade a uma instituição (por exemplo, Hospital São João) que se situa numa cidade (também parte integrante dos parâmetros dos cuidados). Assim, pode-se definir concretamente quais os cuidados que estão a ser prestados numa instituição e qual a sua localização.

O número de identificação é representado como sendo um código, isto é, associa a letra “s” a um certo número inteiro, que ficará à escolha do utilizador que efetuar o seu registo. Denote-se que a sua estrutura não é verificada, ou seja, recai no utilizador a responsabilidade de introduzir o código com a seguinte forma: sX, sendo X um número inteiro.

De seguida, a especialidade do cuidado é, igualmente, dada por um código que associa a letra “e” e um número inteiro, como é explicado posteriormente nesta secção.

Por fim, tanto a instituição e a sua localização são *strings*, uma vez que surgiu a necessidade de introduzir nomes que continham espaços, tal como Hospital da Luz, que de outra forma não era possível ser representado.

Exemplo:

```
cuidado(s1,e5,'hospital sao joao','porto').
```

- **Ato Médico**

Como o ato médico não é um termo geral como o cuidado, que se distingue por ser uma entidade universal a uma instituição, foi preciso introduzir características mais específicas para o descrever.

Primeiramente, temos a hora e a data em que o ato médico (ou consulta) irá ser (ou foi) realizado. A hora encontra-se na forma HH:MM, em que HH indica as horas e MM corresponde aos minutos. Por outro lado, a estrutura da data é DD-MM-AAAA, em que DD representa o dia, MM o mês e AAAA o ano.

Em seguida, recorre-se ao número de identificação de utente, que se encontra na forma uX, em que X é um número inteiro único, ao código do cuidado prestado e ao *ID* do médico que realizou a consulta. Desta forma, caracteriza-se a entidade que usufruiu do ato médico e a que prestou este serviço.

Por fim, o custo da consulta é expresso na forma EE.CC, em que EE representa os euros do custo e CC a parcela de cêntimos.

Exemplo:

```
atomedico('10:20',28-03-2017, u5, s6, m4,  54.32).
```

- **Utentes**

O utente é a entidade que vai usufruir dos atos médicos prestados numa determinada instituição. Assim, o código de identificação é único e relaciona a letra "u" a um número inteiro, tal como uX. O nome e a morada do mesmo são *strings*, pois devem permitir que os nomes inseridos contenham espaços. Além disso, a caracterização do utente contém, ainda, a sua idade.

Exemplo:

```
utente(u2,'Maria Brito',21,'Braga').
```

11.2. Conhecimento Complementar

- **Especialidade**

A especialidade foi introduzida, pois tornou-se necessário definir quais são as vertentes dos cuidados prestados pelas instituições. Este termo é geral às várias instituições, ou seja, uma especialidade pode ser parte de cuidados em organizações distintas.

Desta forma, a especialidade é caracterizada pelo seu código (associação entre a letra "e" e um número inteiro) e a sua designação (*string*).

Exemplo:

```
especialidade(e2, 'pediatria').
```

- **Médico**

O médico é a entidade que efetua um ato médico dentro da sua especialidade. É caracterizado pelo seu número identificador, que associa a letra “m” a um número inteiro, nome (que é uma *string*), idade e código da sua especialidade.

Exemplo:

```
medico(m5, 'Carlos Costa', 45, e5).
```

11.3. Integridade da Base de Conhecimento

De forma a manter o conhecimento coerente e de acordo com a realidade, foi necessário garantir que as inserções não pudessem introduzir conhecimento repetido ou incorreto e que as remoções não pudessem retirar conhecimento que estaria associado a outros. Desta maneira, foram definidos invariantes que garantissem a integridade da base de conhecimento em questão.

- **Invariantes de Inserção**

  - **Utente**

```
+utente(Idu,Nome,Idade,Morada)::
(solucoes(Idu,utente(Idu,_,_,_),S),
comprimento(S,N), N == 1).
```

Este invariante não permite que sejam inseridos utentes com o mesmo ID, isto é, se já existir um utente na base de conhecimento com o número de identificação u1, não poderá ser inserido um novo utente com o mesmo ID. Só é feita a verificação para ID, uma vez que podem existir, tal como se verifica no mundo real, utentes que tenham o mesmo nome, idade ou morada.

  - **Médico**

```
+medico(IdMed,Nome,Idade,IdEsp)::
(solucoes(IdMed,medico(IdMed,_,_,_),S),
                        comprimento(S,N), N == 1).
```

Neste caso, em semelhança ao anterior, não é permitido fazer inserções de médicos com o mesmo código identificador, de forma a garantir a sua unicidade. Por isso, na base de conhecimento não é possível existirem dois médicos que possuam a mesma identificação.

```
+medico(IdMed,Nome,Idade,IdEsp) :: (especialidade(IdEsp,_)).
```

Este invariante garante que nenhum médico pode ser inserido se a especialidade indicada não existir na base de conhecimento. Assim, garantimos que nenhum médico tem uma especialidade inexistente.

- **Especialidade**

```
+especialidade(IdEsp, Desig)::
(solucoes(IdEsp,especialidade(IdEsp,_),S),
                                    comprimento(S,N),
                                    N == 1).
```

Novamente, visando a coerência e integridade da base conhecimento, é preciso garantir que não são inseridas duas especialidades com o mesmo número de identificação. Assim, duas especialidades distintas não podem ter o mesmo código.

- **Cuidado**

```
+cuidado(Ids,Desc,Inst,Cidade) ::
(solucoes(Ids,cuidado(Ids,_,_,_),S),
                                    comprimento(S,N),
                                    N == 1,

solucoes(Ids,cuidado(_,Desc,Inst,Cidade),S),
                                    comprimento(S,N),
                                    N == 1).
```

Neste invariante são garantidos dois aspetos muito importantes: não são inseridos cuidados com o mesmo número de identificação nem podem existir cuidados com a mesma descrição, instituição e cidade (pois, neste caso, ocorreria uma duplicação de cuidados, com *IDs* diferentes). Assim, é impossível que existam cuidados com o mesmo código, o que visa a unicidade da sua identificação e a mesma organização não pode ter cuidados iguais, mesmo que tenham *IDs* diferentes.

- **Ato médico**

```
+atomedico(Hora,Data,IdU,IdS,IdMed,Custo) :: Custo > 0.
```

Este invariante garante que não sejam inseridos atos médicos com custo nulo ou negativo, algo que é imprescindível se pretendemos aproximarmo-nos da realidade.

```
+atomedico(Hora,Data,IdU,IdS,IdMed,Custo) ::
(medico(IdMed,_,_,IdEsp),
cuidado(IdS,IdEsp,_,_)).
```

De modo a garantir a coerência do relacionamento entre os cuidados prestados, os atos médicos realizados e os médicos que os efetuam, não é possível que o ato médico seja inserido com uma identificação do serviço cuja especialidade seja diferente da do médico. Este aspeto é importante, uma vez que não faria sentido que o médico que efetuasse um determinado ato médico num utente não fosse especialista do cuidado prestado.

```
+atomedico(Hora,Data,IdU,IdS,IdMed,Custo) ::
(solucoes(IdU, utente(IdU,_,_,_),S),
comprimento(S,N),
N == 1,
solucoes(IdMed, medico(IdMed,_,_,_),L),
comprimento(L,M),
M == 1,
solucoes(IdS, cuidado(IdS,_,_,_),R),
comprimento(R,Z),
Z == 1).
```

O invariante acima garante que não sejam inseridos atos médicos em que o utente, o cuidado ou o médico não existam na base de conhecimento, o que é fundamental de modo a garantir a sua integridade.

```
+atomedico(Hora,Data,IdU,IdS,IdMed,Custo) :: (solucoes(IdMed,
atomedico(Hora,Data,_,_,IdMed,_),S),
comprimento(S,N),
N==1).
```

Analisando o que acontece no mundo real, o grupo decidiu que não seria possível que um mesmo utente tivesse mais do que uma consulta numa determinada data a uma certa hora. Assim, este invariante garante que não sejam inseridos atos médicos se o utente já tiver marcado uma consulta nessa mesma data e hora.

- **Invariantes de Remoção**

  - **Utentes**

```
-utente(Id, Nome, Idade, Morada) :: (solucoes(Id,
atomedico(_,_,Id,_,_,_), S),
comprimento(S,N),
N==0).
```

Este invariante não permite que um utente seja removido da base de conhecimento se possuir atos médicos a si associados, uma vez que se fosse eliminado e existissem, ainda, atos médicos relacionados com ele, a base de conhecimento ficaria incoerente.

  - **Médico**

```
-medico(Id,Nome,Idade,IdEsp)::
(solucoes(Id,atomedico(_,_,_,_,Id,_),S),
comprimento(S,N),
N==0).
```

À semelhança do invariante anterior, este também não permite que um médico seja removido se possuir atos médicos a si associados.

  - **Especialidade**

```
-especialidade(IdEsp, Desc) ::
(solucoes(IdEsp,medico(_,_,_,IdEsp),S),
comprimento(S,N),
N==0).
```

A especialidade não pode ser eliminada da base de conhecimento se um médico ainda a possuir, pois, caso fosse removida, o médico estaria associado a uma especialidade que não existia.

```
-especialidade(IdEsp, Desc) ::
(solucoes(IdEsp,cuidado(_,IdEsp,_,_),S),
 comprimento(S,N),
 N==0).
```

Uma especialidade não é possível de ser eliminada se um cuidado ainda estiver a si associado, pois, como no exemplo anterior, se a especialidade fosse removida e ainda estivesse associada a um cuidado, este teria um *ID* inexistente, o que causaria uma incoerência na base de conhecimento.

12. Funcionalidades

Nesta secção serão apresentadas as soluções e explicações relativas aos predicados desenvolvidos. Serão inicialmente explorados os predicados relacionados com os problemas apresentados no enunciado do trabalho prático, referenciados como requisitos base, e posteriormente outros predicados relacionados com o conhecimento envolvido.

12.1. Funcionalidades Obrigatórias

- **Registar Utentes**

O predicado `registoUtente` recebe como parâmetros o *ID* do utente, o seu nome, a sua idade e a morada. Este predicado utiliza `insere`, que recebe um termo (utente) e os seus argumentos (neste caso, os parâmetros recebidos pelos predicado `registoUtente`,inserindo este conhecimento da base de conhecimento. A este predicado está sujeito um invariante que não permite a inserção de utentes com o mesmo id.

```
registoUtente(Id,Nome,Idade,Morada) :-
      insere(utente(Id,Nome,Idade,Morada)).
```

- **Registar Cuidados**

O predicado `registoCuidado` recebe como parâmetros o *ID* do cuidado, o *ID* da especialidade, a instituição e a cidade. Semelhantemente ao predicado anterior, é usado o predicado `insere`, de modo a inserir conhecimento, sendo o termo neste caso cuidado. A este predicado está associado um invariante que não permite inserção de cuidados com *IDs* repetidos, nem cuidados com a mesma especialidade, instituição e cidade.

```
registoCuidado(Id,Descricao,Instituicao,Cidade) :-
            insere(cuidado(Id,Descricao,Instituicao,Cidade)).
```

O facto de existir o *ID* da especialidade, contrariamente à designação, vem do facto de, posteriormente, ter sido criado um predicado especialidade.

- ## Registar Atos Médicos

O predicado `registoAtoMedico` recebe como parâmetros a *hora*, *data*, *ID do utente*, *ID do cuidado*, *ID do médico* e *custo do ato médico*. Tal como nos dois predicados prévios, `insere` irá adicionar o conhecimento à base de conhecimento, utilizado `atomedico` como termo. O parâmetro *ID* da hora foi adicionado, pois considerou-se necessário, tal como o *ID* do médico. Estes dois parâmetros impedem que existam conflitos em termos de atos médicos, não sendo possível médicos prestarem a mesma consulta à mesma hora. A este predicado estão associados vários invariantes, que têm as seguintes funcionalidades:

- Não permite a inserção de custos negativos;
- Não permite a inserção de atos médicos em que a especialidade do cuidado seja diferente da do ato médico;
- Não permite a inserção de um ato médico em que o utente, o serviço ou o médico não existam;
- Não permite a inserção de um ato médico em que o médico já tenha consulta à hora estipulada;
- Não permite a inserção de um ato médico à mesma hora, na mesma data com o mesmo utente.

```
registoAtoMedico(Hora,Data,IdU,IdS,IdMed,Custo) :-
      insere(atomedico(Hora,Data,IdU,IdS,IdMed,Custo)).
```

- ## Identificar Utentes por Idade

O predicado listarUtentesIdade recebe como parâmetros uma idade e uma lista em que serão colocados os dados do utente. Assim, o predicado `solucoes` é utilizado para obter o id, o nome e a morada de um utente, cuja idade seja igual à dada como parâmetro, organizando-os num tuplo que será colocado na lista resultado.

```
listarUtentesIdade(C,L) :-
    solucoes({I1,I2,I3},utente(I1,I2,C,I3),L).
```

- **Identificar Utentes por Morada**

O predicado `listarUtentesCidade` recebe como parâmetros uma morada e uma lista em que serão colocados os dados do utente. Assim, o predicado `solucoes` é utilizado para obter o id, o nome e a idade de um utente, cuja morada seja igual à dada como parâmetro, organizando-os num tuplo que será colocado na lista resultado.

```
listarUtentesCidade(C,L) :-
    solucoes({I1,I2,I3},utente(I1,I2,I3,C),L).
```

- **Identificar Utentes por Nome**

O predicado `listarUtentesNome` recebe como parâmetros um nome e uma lista em que serão colocados os dados do utente. Tal como nos dois predicados referidos anteriormente, o predicado `solucoes` colocará o tuplo {id, nome, idade, morada} numa lista que será depois apresentada.

```
listarUtentesNome(C,L) :-
    solucoes({I1,C,I2,I3},utente(I1,C,I2,I3),L).
```

- **Identificar as Instituições Prestadoras de Cuidados de Saúde**

O predicado `listarInstituicoes` recebe uma lista como parâmetro que será utilizada como lista resultado. A estratégia passou por utilizar o predicado `solucoes` para percorrer todos os cuidados, armazenando a instituição numa lista, e terminou eliminando todos os elementos repetidos dessa lista utilizando o predicado `removerRepetidos`.

```
listarInstituicoes(L) :-
    solucoes(C,cuidado(I1,I2,C,I3),L1),
    removerRepetidos(L1,L).
```

- **Identificar os Cuidados Prestados por Instituição**

O predicado `listarCuidadosInstituicao` recebe como parâmetros uma instituição e uma lista que será utilizada para apresentar o resultado. Através do predicado `solucoes`, os triplos {id da especialidade, cidade, *ID* do cuidado} serão armazenados numa lista, à qual os elementos não repetidos serão colocados numa

nova lista. O último passo passa por aplicar o predicado `mostrarServicoLista` que substitui o *ID* da especialidade pela designação da especialidade.

```
listarCuidadosInstituicao(C,L) :-
  solucoes({I2,I3,I1},cuidado(I1,I2,C,I3),L1),
  removerRepetidos(L1,L2),
  mostrarServicoLista(L2,L).
```

- **Identificar os Cuidados Prestados por Cidade**

O predicado `listarCuidadosCidade` recebe como parâmetros uma cidade e uma lista que será utilizada para apresentar o resultado. Através do predicado `solucoes`, os tuplos{id da especialidade, instituição, *ID* do cuidado} serão armazenados numa lista, à qual os elementos não repetidos serão colocados numa nova lista. O último passo passa por aplicar o predicado `mostrarServicoLista` que substitui o *ID* da especialidade pela designação da especialidade.

```
listarCuidadosCidade(C,L) :-
   solucoes({I2,I3},cuidado(I1,I2,I3,C),L1),
   removerRepetidos(L1,L2),
   mostrarServicoLista(L2,L).
```

- **Listar os Utentes por Instituição**

O predicado `listarUtentesInstituicao` recebe como parâmetros uma instituição e uma lista que armazenará o resultado. A estratégia passou por utilizar o predicado `solucoes` para armazenar os *IDs* dos cuidados numa lista. Através dessa lista, usando o predicado `listarUtentesPorCuidados`, obtém-se uma lista que contém todos os *IDs* dos utentes. De seguida, é utilizado o predicado `listarUtentesPorIds` que determina uma lista de quádruplos com as informações dos utentes. Finalmente, utiliza-se o predicado `removerRepetidos` para que o resultado apresentado não possua repetições.

```
listarUtentesInstituicao(Inst,S) :-
    solucoes(I1,cuidado(I1,I2,Inst,I3),S1),
    listarUtentesPorCuidados(S1,L),
    listarUtentesPorIds(L,S2),
    removerRepetidos(S2,S).
```

- **Listar os Utentes por Cuidado**

O predicado `listarUtentesCuidado` recebe um *ID* do cuidado e uma lista onde será armazenado o resultado. A solução passou por utilizar o predicado `solucoes`, onde os *IDs* dos utentes serão armazenados numa lista se o utente estiver presente num ato médico, seguido do predicado `listarUtentesPorIds` que coloca numa lista as informações relativas a um utente.

```
listarUtentesCuidado(C,L) :-
    solucoes(Idu, atomedico(_,_,Idu,C,_,_),AUX),
    listarUtentesPorIds(AUX,L).
```

- **Identificar Atos Médicos por Utente**

O predicado `atosMedicosPorUtente` recebe como parâmetro um *ID* do utente e uma lista que irá armazenar o resultado. A estratégia passou por utilizar o predicado `solucoes` que armazenará {hora, data, *ID* do cuidado, *ID* do médico, custo} na lista resultado.

```
atosMedicosPorUtente(Idu,L) :-
    solucoes({Hora,Data,Ids,IdMed,Custo},
           atomedico(Hora,Data,Idu,Ids, IdMed,Custo),
                                L).
```

- **Identificar Atos Médicos por Instituição**

O predicado `atosMedicosPorInstituicao` recebe como parâmetro uma instituição e uma lista que irá armazenar o resultado. A estratégia passou por utilizar o predicado `solucoes` para armazenar os *IDs* dos cuidados numa lista.

De seguida, utilizou-se o predicado `listarAtosMedicosPorIdServico` que recebe uma lista de *IDs* de cuidados e coloca os dados referentes ao ato médico que contém esse *ID* do cuidado numa lista.

```
atosMedicosPorInstituicao(Inst,L) :-
    solucoes(Ids,cuidado(Ids,Desc,Inst,Cidade),L1),
    listarAtosMedicosPorIdServico(L1,L).
```

- **Identificar Atos Médicos por Cuidado**

O predicado `atosMedicosPorServico` recebe como parâmetro um *ID* do cuidado e uma lista que irá armazenar o resultado. A estratégia passou por utilizar o predicado `solucoes` que armazenará {hora, data, *ID* do utente, *ID* do médico, custo} na lista resultado.

```
atosMedicosPorServico(Ids,L) :-
          solucoes({Hora,Data,Idu,IdMed,Custo},
                atomedico(Hora,Data,Idu,Ids,IdMed,Custo),L).
```

- **Determinar Utentes de uma Instituição**

O predicado `listarUtentesInstituicao` recebe como parâmetro uma instituição e uma lista que irá armazenar o resultado. A estratégia passou por utilizar o predicado `solucoes` para armazenar os *IDs* dos cuidados numa lista. De seguida, utilizou-se essa lista como parâmetro do predicado `listarUtentesPorCuidados` que coloca numa lista os *IDs* dos utentes que estiveram num ato médico corresponde ao *ID* do cuidado. Após este passo, com a lista dos *IDs* dos utentes utilizou-se o predicado `listarUtentesPorIds` para colocar numa lista as informações relativas aos *IDs* dos utentes e o predicado `removerRepetidos` para remover ocorrências repetidas.

```
listarUtentesInstituicao(Inst,S) :-
    solucoes(I1,cuidado(I1,I2,Inst,I3),S1),
    listarUtentesPorCuidados(S1,L),
    listarUtentesPorIds(L,S2),
    removerRepetidos(S2,S).
```

- **Determinar Utentes Por Cuidados**

O predicado `listarUtentesPorCuidados` recebe como parâmetro uma lista dos *IDs* dos cuidados e uma lista resultado. Primeiramente, serão armazenados os *IDs* dos utentes numa lista através do predicado `solucoes`. De seguida, chama-se o predicado novamente sobre a cauda da lista, adicionado o resultado do predicado `solucoes` à lista resultado, através do predicado `acrescenta`.

```
listarUtentesPorCuidados([],[]).
listarUtentesPorCuidados([H|T],L) :-
    solucoes(Idu,atomedico(_,_,Idu,H,_,_),AUX),
    listarUtentesPorCuidados(T,X),
    acrescenta(AUX,X,L).
```

- **Calcular o Custo Total dos Atos Médicos por Utente**

O predicado `calculoCustoTotalAtosUtente` recebe como parâmetro o *ID* do utente e uma variável onde será armazenado o resultado. Para o *ID* passado como parâmetro, o predicado `solucoes` irá armazenar numa lista todos os custos dos atos médicos desse utente. Em último lugar, utiliza-se o predicado `somatorio` que soma todos os elementos de uma lista.

```
calculoCustoTotalAtosUtente(Idu,R) :-
    solucoes(C, atomedico(_,_,Idu,_,_,C),X),
    somatorio(X,R).
```

- **Calcular o Custo Total dos Atos Médicos por Cuidado**

O predicado `calculoCustoTotalServico` recebe como parâmetro o *ID* do cuidado e uma variável onde será armazenado o resultado. Para o *ID* passado como parâmetro, o predicado `solucoes` irá armazenar numa lista todos os custos dos atos médicos desse cuidado. Em último lugar, utiliza-se o predicado `somatorio` que soma todos os elementos de uma lista.

```
calculoCustoTotalServico(C,R) :-
    solucoes(P,atomedico(_,_,_,C,_,P),X),
    somatorio(X,R).
```

- **Calcular o Custo Total dos Atos Médicos por Instituição**

O predicado `calculoCustoTotalInstituicao` recebe como parâmetro uma instituição e uma variável onde será armazenado o resultado. Para a instituição passada como parâmetro, o predicado `solucoes` irá armazenar numa lista todos os *IDs* dos cuidados dessa instituição. De seguida, utiliza-se o predicado `listaCustosServicos` que recebe uma lista de *IDs* de cuidado e uma lista resultado que irá conter todos os custos referentes a esses *IDs* de cuidado. Em último lugar, utiliza-se o predicado `somatorio` que soma todos os elementos de uma lista.

```
calculoCustoTotalInstituicao(IdI, L):-
  solucoes(IdS,cuidado(IdS,_,IdI,_),P),
  listaCustosServicos(P,L1),
  somatorio(L1,L).
```

- **Calcular o Custo Total dos Atos Médicos por Data**

O predicado `calculoCustoTotalData` recebe como parâmetro uma data e uma variável onde será armazenado o resultado. Para a data passada como parâmetro, o predicado `solucoes` irá armazenar numa lista todos os custos dos atos médicos nessa data. Em último lugar, utiliza-se o predicado `somatorio` que soma todos os elementos de uma lista.

```
calculoCustoTotalData(C,R) :-
    solucoes(P,atomedico(_,C,_,_,_,P),X),
    somatorio(X,R).
```

- **Remover Utentes**

O predicado `removerUtente` recebe como parâmetros o *ID* do utente, o seu nome, a sua idade e a morada. Este predicado utiliza `remove,` que recebe um termo (utente) e os seus argumentos (neste caso, os parâmetros recebidos pelos predicado `removeUtente`, removendo este conhecimento da base de conhecimento. A este predicado está sujeito um invariante que não permite a remoção de um utente com atos médicos registados.

```
removerUtente(Id,Nome,Idade,Morada) :-
      remocao(utente(Id,Nome,Idade,Morada )).
```

- **Remover Cuidados**

O predicado `removerCuidado` recebe como parâmetros o *ID* do cuidado, o *ID* da especialidade, a instituição e a cidade. Semelhantemente ao predicado anterior, é usado o predicado `remove`, de modo a remover conhecimento, sendo o termo neste caso cuidado. Na eventualidade de remoção de um cuidado, é necessário remover todos os atos médicos associados a esse cuidado. Exemplo: se uma ala hospitalar dedicada a um determinado cuidado fecha, todas as consultas serão canceladas. Assim, utiliza-se o predicado `solucoes` para determinar os dados referentes aos atos médicos que devem ser removidos e o predicado `removerTodosAtosMedicosPorIdS` que remove todos os atos médicos que possuam o *ID* do cuidado passado como parâmetro.

```
removerCuidado(Id,Descricao,Instituicao,Cidade) :-
        retract(cuidado(Id,Descricao,Instituicao,Cidade)),
        solucoes(Id, atomedico(_, _, _, Id, _, _), L),
        removerTodosAtosMedicosPorIdS(Id,L).
```

- **Remover Atos Médicos**

O predicado `removerAtoMedico` recebe como parâmetros a hora, data, *ID* do utente, *ID* do cuidado, *ID* do médico e custo do ato médico. Tal como nos dois predicados prévios, `remove` irá remover o conhecimento da base de conhecimento, utilizado atomedico como termo.

```
removerAtoMedico(Hora,Data,IdU,IdS,IdMed,Custo) :-
       remocao(atomedico(Hora,Data,IdU,IdS,IdMed,Custo)).
```

12.2. Funcionalidades Adicionais

- **Identificar os Utentes de um Médico**

O predicado `utentesPorMedico` recebe como parâmetros um *ID* de médico e uma lista que será utilizada para apresentar o resultado. Através do predicado `solucoes`, o *ID* do utente que tenha um ato médico com o médico recebido como parâmetro será armazenado numa lista, à qual os elementos não repetidos serão colocados numa nova lista. O último passo passa por aplicar o predicado `listarUtentesPorIds` que acrescenta num tuplo a seguinte informação de cada utente {id, nome, idade, morada}.

```
utentesPorMedico(C,R) :-
   solucoes(U,atomedico(_,_,U,_,C,_),R1),
   removerRepetidos(R1,R2),
   listarUtentesPorIds(R2,R).
```

- **Calcular o Custo Total dos Atos médicos**

O predicado `calculoCustoTotalAtos` recebe como parâmetro uma lista que será utilizada para apresentar o resultado. Através do predicado `solucoes`, o preço associado a cada ato médico será armazenado numa lista. Em seguida é aplicado o predicado `somatorio` à lista de valores por forma a ser armazenado na lista passada como parâmetro o valor correspondente à soma de todos os valores da lista.

```
calculoCustoTotalAtos(R) :-
      solucoes(P,atomedico(_,_,_,_,_,P),X),
      somatorio(X,R).
```

- **Determinar os Médicos com Atos Médicos numa Instituição**

O predicado `idMedicosPorInstituicao` recebe como parâmetros a designação de uma instituição e uma lista que será utilizada para apresentar o resultado. Através do predicado `solucoes`, os *IDs* dos cuidados associados à instituição fornecida como parâmetro são armazenados numa lista. Em seguida é aplicado o predicado `listarIdMedicosPorIdServico` à lista de valores de modo a serem armazenados na lista passada como parâmetro os *IDs* dos médicos que prestaram esses cuidados.

```
idMedicosPorInstituicao(Inst,L) :-
      solucoes(Ids,cuidado(Ids,Desc,Inst,Cidade),L1),
      listarIdMedicosPorIdServico(L1,L).
```

- **Determinar Todos os Médicos a que um Utente Já Recorreu**

O predicado `medicoPorUtente` recebe como parâmetros o *ID* de um utente e uma lista que será utilizada para apresentar o resultado. Através do predicado `solucoes`, os *IDs* dos médicos associados aos atos médicos cujo *ID* do utente é o fornecido como parâmetro, são armazenados numa lista. Em seguida é aplicado o predicado `removerRepetidos` que permite criar uma nova lista com os mesmos elementos, sem repetições. Por fim é aplicado o predicado `listarMedicosPorIds` por forma a obter uma lista final com tuplos com a seguinte informação associada a cada médico {id, nome, idade, especialidade}.

```
medicoPorUtente(IdU,R) :-
      solucoes(IdM,atomedico(_,_,IdU,_,IdM,_),R1),
      removerRepetidos(R1,R2),
      listarMedicosPorIds(R2,R).
```

- **Identificar os Utentes por ID**

O predicado `listarUtentesPorIds` recebe como parâmetro uma lista de *IDs* de utentes e uma lista em que serão colocados todos os dados do utente. Para cada um dos elementos da lista, a restante informação associada a esse *ID* do utente é recolhida através de um *matching* do *ID* com o conhecimento de utente. A informação de cada utente é organizada num tuplo com a seguinte informação {id, nome, idade, morada}, sendo segunda lista passada como parâmetro, preenchidas com vários tuplos dos utentes.

```
listarUtentesPorIds([],[]).
listarUtentesPorIds([H|T],L) :-
      utente(H,I1,I2,I3),
      listarUtentesPorIds(T,X),
      acrescenta([{H,I1,I2,I3}],X,L).
```

- **Registar um Médico**

O predicado `registoMedico` recebe como parâmetros o *ID* do médico, o seu nome, a sua idade e a especialidade. Este predicado utiliza `insere`, que recebe um termo (medico) e os seus argumentos (neste caso, os parâmetros recebidos pelo predicado `registoMedico`, inserindo este conhecimento da base de conhecimento. A este predicado está sujeito um invariante que não permite a inserção de médicos com o mesmo id.

```
registoMedico(IdMed,Nome,Idade,IdEsp) :-
            insere(medico(IdMed,Nome,Idade,IdEsp)).
```

- **Registar uma Especialidade**

O predicado `registoEspecialidade` recebe como parâmetros o *ID* da especialidade e a sua designação. Este predicado utiliza `insere`, que recebe um termo (especialidade) e os seus argumentos (neste caso, os parâmetros recebidos pelo predicado `registoEspecialidade`, inserindo este conhecimento na base de conhecimento. A este predicado está sujeito invariante que não permite a inserção de especialidades com o mesmo id.

```
registoEspecialidade(IdEsp,Desig) :-
    insere(especialidade(IdEsp,Desig)).
```

- **Remover um Médico**

O predicado `removerMedico` recebe como parâmetros o *ID* do médico, o seu nome, a sua idade e o *ID* da especialidade. Este predicado utiliza `remover`, que recebe um termo (medico) e os seus argumentos (neste caso, os parâmetros recebidos pelo predicado `removerMedico`, removendo este conhecimento da base de conhecimento. A este predicado está sujeito um invariante que não permite a inserção de médicos com o mesmo *ID* nem de médicos que tenham *IDs* de especialidades que não existam. Este predicado tem associado um invariante que não permite a remoção de um médico com atos médicos marcados.

```
removerMedico(IdMed, Nome, Idade, IdEsp) :-
        remocao(medico(IdMed, Nome, Idade, IdEsp)).
```

- **Remover uma Especialidade**

O predicado `removerEspecialidade` recebe como parâmetros o *ID* da especialidade e a sua designação. Este predicado utiliza `remover`, que recebe um termo (especialidade) e os seus argumentos (neste caso, os parâmetros recebidos pelo predicado `removerEspecialidade`, removendo este conhecimento da base de conhecimento. A este predicado está associado o invariante que não permite a remoção de uma especialidade se um médico a possuir.

```
removerEspecialidade(IdEsp, Desig) :-
        remocao(especialidade(IdEsp, Desig)).
```

12.3. Predicados Auxiliares

O predicado `solucoes` serve para renomear o predicado `findall`.

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

O predicado `comprimento` serve para renomear o predicado `length`.

```
comprimento(S,N):- length(S,N).
```

O predicado `acrescenta` serve para renomear o predicado `append`.

```
acrescenta(L1,L2,L):- append(L1,L2,L).
```

O predicado `removerElemento` tem 3 parâmetros, sendo eles uma lista, um elemento e uma outra lista. É utilizado para remover todas as ocorrências do elemento na primeira lista, ficando armazenado o resultado pretendido da segunda lista fornecida como parâmetro.

```
removerElemento( [],_,[] ).
removerElemento( [X|L],X,NL ) :-
    removerElemento( L,X,NL ).
removerElemento( [X|L],Y,[X|NL] ) :-
    X \== Y, removerElemento( L,Y,NL ).
```

O predicado `removerRepetidos` tem duas listas como parâmetros. É utilizado para remover todos os elementos repetidos da primeira lista, ficando armazenado o resultado pretendido da segunda lista fornecida como parâmetro.

```
removerRepetidos( [],[] ).
removerRepetidos( [X|L],[X|NL] ) :-
    removerElemento( L,X,TL ), removerRepetidos( TL,NL ).
```

O predicado `insere` tem apenas um termo como parâmetro. É utilizado para inserir conhecimento, verificando primeiro se o invariante do termo em questão se mantém. Inicialmente o termo é sempre adicionado, sendo removido novamente se não for possível a inserção.

```
insere( Termo ) :- solucoes( Inv, +Termo::Inv, Linv),
                   assert(Termo),
                   testa(Linv).
insere( Termo ) :- retract(Termo).
```

O predicado `remocao` tem apenas um termo como parâmetro. É utilizado para remover conhecimento, verificando a lista de invariantes para o termo recebido como parâmetro. Inicialmente o termo é sempre removido, sendo adicionado novamente se não for possível a remoção.

```
remocao( Termo ) :- solucoes(Inv, -Termo::Inv, Linv),
                    retract(Termo),
                    testa(Linv).
remocao( Termo ) :- assert(Termo).
```

O predicado `contem` tem 2 parâmetros: um elemento e uma lista. É usado para determinar se o elemento existe na lista.

```
contem(H, [H|T]).
contem(X, [H|T]) :-
    contem(X, T).
```

O predicado `contem` tem duas listas como parâmetros. É usado para determinar se todos os elementos da primeira são também elementos da segunda lista.

```
contemTodos([], _).
contemTodos([H|T], L) :-
    contem(H, L), contemTodos(T, L).
```

O predicado `testa` recebe uma lista como parâmetro. É usado para calcular a conjunção lógica de todos dos valores numa lista.

```
testa([]).
testa([H|T]):- H, testa(T).
```

O predicado `somatorio` recebe uma lista e um número inteiro como parâmetros. É usado para calcular o somatório dos números de uma lista.

```
somatorio([],0).
somatorio([H|T],Sum):-
  somatorio(T,Rest),
  Sum is H+Rest.
```

O predicado `removerTodosAtosMedicosPorId` tem um número (id do serviço) e uma lista como argumentos. É usado para remover todos os atos médicos de um serviço, usando a lista apenas para controlo do número de vezes em que o predicado é testado recursivamente.

```
removerTodosAtosMedicosPorIdS(Id, []).
removerTodosAtosMedicosPorIdS(Id, [H|T]) :-
      retract(atomedico(_,_,_,Id,_,_)),
      removerTodosAtosMedicosPorId(Id,T).
```

O predicado `listarInfoServicos` tem duas listas como parâmetros e é usado para obter informação sobre todos os parâmetros que definem os cuidados cujos *IDs* se encontram na primeira lista.

```
listarInfoServicos([],[]).
listarInfoServicos([H|T],L) :-

solucoes({Desc,Inst,Cidade},cuidado(H,Desc,Inst,Cidade),AUX),
    listarInfoServicos(T,X),
    acrescenta(AUX,X,L1),
    mostrarServicoLista(L1,L).
```

O predicado `mostrarServicoLista` tem duas listas como parâmetros. É usado para obter informação sobre todos os parâmetros que definem os cuidados cujos *IDs* se encontram na primeira lista.

```
mostrarServicoLista([],[]).
mostrarServicoLista([H|T],Y):-
    mostrarServico(H,X),
    mostrarServicoLista(T,Z),
    acrescenta(X,Z,Y).
```

O predicado listarUtentesPorIds tem duas listas como parâmetros e é usado para obter informação sobre todos os parâmetros que definem os utentes cujos *IDs* se encontram na primeira lista.

```
listarUtentesPorIds([],[]).
listarUtentesPorIds([H|T],L) :-
    utente(H,I1,I2,I3),
    listarUtentesPorIds(T,X),
    acrescenta([{H,I1,I2,I3}],X,L).
```

O predicado listarUtentesPorCuidados tem, como parâmetros, uma lista com *IDs* de cuidados médicos e uma lista com *IDs* de utentes. É usado para obter a lista dos *IDs* de utentes que já recorreram a algum dos serviços presentes na primeira lista.

```
listarUtentesPorCuidados([],[]).
listarUtentesPorCuidados([H|T],L) :-
    solucoes(Idu,atomedico(_,_,Idu,H,_,_),AUX),
    listarUtentesPorCuidados(T,X),
    acrescenta(AUX,X,L).
```

O predicado `listarInstituicoesVariosServicos` tem uma lista de *IDs* de serviços e uma lista de nomes de instituições como parâmetros. É usado para determinar os nomes das instituições onde se realizam os cuidados presentes na primeira lista.

```
listarInstituicoesVariosServicos([], []).
listarInstituicoesVariosServicos([H|T], P) :-
    listarInstituicoesServico(H,R),
    listarInstituicoesVariosServicos(T,L),
    acrescenta(R,L,P).
```

O predicado listarAtosMedicosPorIdServico tem duas listas como parâmetros e é usado para determinar todos os atos médicos associados a uma lista de *IDs* de serviços.

```
listarAtosMedicosPorIdServico([],[]).
listarAtosMedicosPorIdServico([H|T],L) :-
      solucoes({Hora,Data,Idu,IdMed,Custo},atomedico(Hora,Data,
Idu,H,IdMed,Custo),AUX),
    listarAtosMedicosPorIdServico(T,X),
    acrescenta(AUX,X,L).
```

O predicado listaCustosServicos recebe como parâmetro uma lista de *IDs* de cuidados e uma variável onde será armazenado o resultado. O predicado é usado para calcular o valor gasto nos atos médicos com os *IDs* de cuidados contidos na lista.

```
listaCustosServicos([],[]).
listaCustosServicos([H|T],R):-
  listaCustosServicos(T,R1),
  solucoes(C, atomedico(_,_,_,H,_,C),X),
  somatorio(X,Y),
  acrescenta([Y],R1,R).
```

O predicado listarMedicosPorIds tem duas listas como parâmetros. É usado para obter todas as informações sobre os médicos presentes na primeira lista (identificados pelos seus ids), colocando-as na segunda lista.

```
listarMedicosPorIds([],[]).
listarMedicosPorIds([H|T],L) :-
    medico(H,I1,I2,I3),
    listarMedicosPorIds(T,X),
    acrescenta([{H,I1,I2,I3}],X,L).
```

13. Exemplos Práticos

De seguida, serão apresentados exemplos práticos das funcionalidades obrigatórias e adicionais definidas pelo grupo.

13.1. Funcionalidades Obrigatórias

- **Registar um Utente**

```
| ?-
| ?- registoUtente(u10,'Rogerio Lopes',24,braga).
yes
| ?-
| ?-
| ?-
```

- **Registar um Cuidado**

```
| ?- registoCuidado(s10,e10,'hospital de guimaraes',guimaraes).
yes
| ?-
| ?-
| ?-
```

- **Registar um Ato Médico**

```
| ?- registoAtoMedico(14:56,19-03-2017,u10,s7,m8,3.44).
yes
| ?-
| ?-
| ?-
```

- **Remover um Utente**

```
| ?- removerUtente(u10,'Rogerio Lopes',24,braga).
yes
| ?-
| ?-
| ?-
| ?-
```

- **Remover um Cuidado**

```
| ?-
| ?- removerCuidado(s10,e10,'hospital de guimaraes',guimaraes).
yes
| ?-
| ?-
```

- **Remover um Ato Médico**

```
| ?-
| ?- removerAtoMedico(14:56,19-03-2017,u10,s7,m8,3.44).
yes
| ?-
| ?-
```

- **Identificar Utentes por Vários Critérios**
    - **Idade**

```
| ?- listarUtentesIdade(20,L).
L = [{u4,'Rogerio Moreira','Braga'},{u5,'Samuel Ferreira','Sao Joao da Madeira'}] ?
yes
| ?-
| ?-
```

- **Morada**

```
| ?-
| ?- listarUtentesCidade(braga,L).
L = [{u10,'Rogerio Lopes',24}] ?
yes
| ?-
| ?-
```

- **Nome**

```
| ?-
| ?-
| ?- listarUtentesNome('Rogerio Moreira',R).
R = [{u4,'Rogerio Moreira',20,'Braga'}] ?
yes
| ?-
| ?-
```

- **Identificar as instituições prestadoras de cuidados de saúde**

```
| ?-
| ?-
| ?- listarInstituicoes(L).
L = ['hospital sao joao','hospital sao marcos','hospital da luz','centro de saude do caranda','maternidade alfredo da costa'] ?
yes
| ?-
```

- **Identificar os cuidados prestados por instituição**

```
| ?-
| ?- listarCuidadosInstituicao('hospital sao marcos',L).
L = [{ginecologia,braga,s2}] ?
yes
| ?-
| ?-
```

- **Identificar os cuidados prestados por cidade**

```
| ?-
| ?- listarCuidadosCidade('braga',L).
L = [{ginecologia,'hospital sao marcos'},{geral,'centro de saude do caranda'}] ?
yes
| ?-
| ?-
```

- **Identificar os utentes por instituição**

```
| ?-
| ?- listarUtentesInstituicao('hospital sao marcos',L).
L = [{u5,'Samuel Ferreira',20,'Sao Joao da Madeira'}] ?
yes
| ?-
| ?-
```

- **Identificar os utentes por cuidado prestado**

```
| ?-
| ?- listarUtentesCuidado(s1,L).
L = [{u4,'Rogerio Moreira',20,'Braga'},{u5,'Samuel Ferreira',20,'Sao Joao da Madeira'}] ?
yes
| ?-
| ?-
| ?-
```

- **Identificar atos médicos por utente**

```
| ?-
| ?- atosMedicosPorUtente(u2,L).
L = [{'21:00',2-3-2017,s4,m2,16.5},{'11:20',28-2-2017,s6,m3,12.2},{'00:00',25-12-2016,s7,m8,891.24}] ?
yes
| ?-
| ?-
```

- **Identificar atos médicos por instituição**

```
| ?-
| ?-
| ?- atosMedicosPorInstituicao('hospital sao marcos',L).
L = [{'17:20',12-3-2017,u5,m4,14.8},{'10:20',28-3-2017,u5,m4,54.32}] ?
yes
| ?-
| ?-
| ?-
```

- **Identificar atos médicos por cuidado**

```
| ?-
| ?-
| ?- atosMedicosPorServico(s2,L).
L = [{'17:20',12-3-2017,u5,m4,14.8},{'10:20',28-3-2017,u5,m4,54.32}] ?
yes
| ?-
| ?-
| ?-
```

- **Determinar cuidados de um utente**

```
| ?-
| ?-
| ?- listarUtentesPorCuidados([s3,s4],L).
L = [u5,u2,u3] ?
yes
| ?-
| ?-
```

- **Calcular o custo total dos atos médicos por utente**

```
| ?-
| ?-
| ?- calculoCustoTotalAtosUtente(u3,L).
L = 33.2 ?
yes
| ?-
| ?-
```

- **Calcular o custo total dos atos médicos por cuidado**

```
| ?- calculoCustoTotalServico(s1,R).
R = 30.4 ?
yes
```

- **Calcular o custo total dos atos médicos por instituição**

```
| ?- calculoCustoTotalInstituicao('hospital sao joao',R).
R = 92.30000000000001 ?
yes
| ?-
```

- **Calcular o custo total dos atos médicos por data**

```
| ?-
| ?-
| ?-
| ?- calculoCustoTotalData(15-03-2017,R).
R = 36.599999999999994 ?
yes
| ?-
| ?-
```

13.2. Funcionalidades Adicionais

- **Identificar os utentes de um Médico**

```
| ?-
| ?-
| ?- utentesPorMedico(m2,R).
R = [{u2,'Maria Brito',21,'Braga'},{u3,'Gustavo Andrez',36,'Guimaraes'}] ?
yes
| ?-
| ?-
```

- **Calcular o custo total dos atos médicos**

```
| ?-
| ?-
| ?-
| ?-
| ?- calculoCustoTotalAtos(R).
R = 1131.3800000000003 ?
yes
| ?-
| ?-
```

- **Determinar os médicos com atos médicos numa instituição**

```
| ?-
| ?-
| ?- idMedicosPorInstituicao('hospital sao marcos',L).
L = [m4] ?
yes
| ?-
| ?-
| ?-
```

- **Determinar todos os médicos a que um utente já recorreu**

```
| ?-
| ?-
| ?- medicoPorUtente(u4,R).
R = [{m5,'Carlos Costa',45,e5}] ?
yes
| ?-
| ?-
```

- **Identificar os utentes por ID**

```
| ?-
| ?- listarUtentesPorIds([u1,u2],L).
L = [{u1,'Bruno Pereira',21,'Braga'},{u2,'Maria Brito',21,'Braga'}] ?
yes
| ?-
| ?-
```

- **Registar um Médico**

```
| ?- registoMedico(m10,'Alberto Joao',56,e1).
yes
| ?-
| ?-
| ?-
```

- **Registar uma Especialidade**

```
| ?- registoEspecialidade(e10,'otorrinolaringologia').
yes
| ?-
| ?-
| ?-
```

- **Remover um Médico**

```
| ?- removerMedico(m10,'Alberto Joao',56,e1).
yes
| ?-
| ?-
| ?-
```

- **Remover uma Especialidade**

```
| ?- removerEspecialidade(e10,'otorrinolaringologia').
yes
| ?-
| ?-
| ?-
| ?-
```

14. Conclusão

Em primeiro lugar, é de realçar a importância deste tipo de trabalho para a compreensão e aperfeiçoamento dos conhecimentos obtidos, até este momento, na disciplina de Sistemas de Representação de Conhecimento e Raciocínio. Para a resolução dos diversos problemas que foram surgindo à medida que o trabalho foi desenvolvido, tivemos de procurar a informação nas mais variadas fontes, seja o manual do PROLOG, a *Internet* ou os docentes, permitindo assim alargar os conhecimentos na linguagem, para além daqueles já adquiridos durante as aulas.

Uma vez concluído o projeto, podemos dizer que os objetivos básicos propostos foram inteiramente cumpridos. Construímos uma base de conhecimento de acordo com a especificação e implementámos as funcionalidades necessárias. Para além disto, decidimos alargar não só a base de conhecimento como as funcionalidades, acrescentando características que achamos que seriam interessantes existir neste sistema.

Analisando os resultados obtidos, todas as funcionalidades criadas funcionam de acordo com as expectativas e estão de acordo com a base de conhecimento criada, isto é, as funcionalidades desenvolvidas são bastante úteis caso este sistema se tratasse de um sistema real.

Alguns dos problemas que tivemos durante o desenvolvimento prendem-se com a representação de datas em PROLOG e algumas funcionalidades mais complexas.

15. Bibliografia

[Analide, 2011] ANALIDE, César, NOVAIS, Paulo, NEVES, José,
“Sugestões para a Elaboração de Relatórios”, Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.

“SICStus Prolog User’s Manual”,
SICS Swedish ICT AB PO Box 1263 SE-164 29 Kista, Sweden, dezembro 2016

16. Anexo

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

% TRABALHO DE GRUPO - 1 EXERCICIO

% GRUPO 2

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


%-------------------------------------------------------------------------
-------------------------
%--------------------Declaracões Iniciais--------------------------
-------------------------
%-------------------------------------------------------------------------
-------------------------

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

:- op( 900,xfy,'::' ).
:- dynamic utente/4.
:- dynamic medico/4.
:- dynamic especialidade/2.
:- dynamic cuidado/4.
:- dynamic atomedico/6.

%-------------------------------------------------------------------------
-------------------------
%------Extensão do predicado que permite encontrar todas as
soluções-------------------------
%-------------------------------------------------------------------------
-------------------------

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

```
%-------------------------------------------------------------------------
-------------------------
%------Extensao do predicado removerElemento: [X | L], Y, [A | B] ->
{V,F}-------------------
%-------------------------------------------------------------------------
-------------------------

removerElemento( [],_,[] ).

removerElemento( [X|L],X,NL ) :-
    removerElemento( L,X,NL ).

removerElemento( [X|L],Y,[X|NL] ) :-
    X \== Y, removerElemento( L,Y,NL ).


%-------------------------------------------------------------------------
-------------------------
%------Extensao do predicado removerRepetidos: [X | L], [A | B] ->
{V,F}---------------------
%-------------------------------------------------------------------------
-------------------------

removerRepetidos( [],[] ).

removerRepetidos( [X|L],[X|NL] ) :-
    removerElemento( L,X,TL ), removerRepetidos( TL,NL ).


%-------------------------------------------------------------------------
-------------------------
%------Extensão do predicado que permite a remocao do conhecimento:
T -> {V,F}---------------
%-------------------------------------------------------------------------
-------------------------

remocao( Termo ) :- solucoes(Inv, -Termo::Inv, Linv),
                    retract(Termo),
                    testa(Linv).
remocao( Termo ) :- assert(Termo).

%-------------------------------------------------------------------------
-------------------------
%------Extensão do predicado que calcula os elementos de uma lista:
T -> {V,F}---------------
%-------------------------------------------------------------------------
-------------------------
```

```
comprimento(S,N) :- length(S,N).


%-------------------------------------------------------------------
-------------------------
%------Extensão do predicado contem: H,[H|T] -> {V, F}--------------
-------------------------
%-------------------------------------------------------------------
-------------------------

contem(H, [H|T]).
contem(X, [H|T]) :-
    contem(X, T).


%-------------------------------------------------------------------
-------------------------
%------Extensão do predicado contem: [H|T],L -> {V, F}--------------
-------------------------
%-------------------------------------------------------------------
-------------------------

contemTodos([], _).
contemTodos([H|T], L) :-
    contem(H, L), contemTodos(T, L).


%-------------------------------------------------------------------
-------------------------
%------Extensão do predicado que permite a evolucao do conhecimento:
T -> {V,F}--------------
%-------------------------------------------------------------------
-------------------------

insere( Termo ) :- solucoes( Inv, +Termo::Inv, Linv),
                   assert(Termo),
                   testa(Linv).
insere( Termo ) :- retract(Termo).

testa([]).
testa([H|T]):- H, testa(T).


%-------------------------------------------------------------------
-------------------------
%------Extensão do predicado que permite juntar conhecimento:
L1,L2,L  -> {V,F}--------------
```

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

acrescenta(L1,L2,L) :- append(L1,L2,L).



%-------------------------------------------------------------------
-------------------------
%------Extensão do predicado que permite somar os elementos de uma
lista: [H|T],R -> {V, F}--
%-------------------------------------------------------------------
-------------------------

somatorio([],0).
somatorio([H|T],Sum):-
  somatorio(T,Rest),
  Sum is H+Rest.


%-------------------------------------------------------------------
-------------------------
%------Utente: #IdUt, Nome, Idade, Morada -> {V,F}------------------
-------------------------
%-------------------------------------------------------------------
-------------------------

utente(u1,'Bruno Pereira',21,'Braga').
utente(u2,'Maria Brito',21,'Braga').
utente(u3,'Gustavo Andrez',36,'Guimaraes').
utente(u4,'Rogerio Moreira',20,'Braga').
utente(u5,'Samuel Ferreira',20,'Sao Joao da Madeira').
utente(u6,'Asdrubal Amora',80,'Guimaraes').


%-------------------------------------------------------------------
-------------------------
%------Médico : #IdMed, Nome, Idade, #IdEsp -> {V,F}----------------
-------------------------
%-------------------------------------------------------------------
-------------------------

medico(m1, 'Fernando Fernandes', 34, e1).
medico(m2, 'Paulo Pinho', 48, e2).
medico(m3, 'Augusto Agostinho', 52, e3).
medico(m4, 'Bernardo Barros', 44, e4).
medico(m5, 'Carlos Costa', 45, e5).
medico(m6, 'Dario Dias', 61, e6).
```

```
medico(m7, 'Nadia Nascimento', 29, e4).
medico(m8, 'Antonio Egas Moniz',163,e7).


%-------------------------------------------------------------------
-------------------------
%------Especialidade : #IdEspecialidade, Designação----------------
-------------------------
%-------------------------------------------------------------------
-------------------------
especialidade(e1, 'geral').
especialidade(e2, 'pediatria').
especialidade(e3, 'urologia').
especialidade(e4, 'ginecologia').
especialidade(e5, 'psiquiatria').
especialidade(e6, 'radiologia').
especialidade(e7, 'cirurgia').


%-------------------------------------------------------------------
-------------------------
%------Cuidado prestado: #IdServ, Descrição, Instituição, Cidade ->
{V,F}--------------------
%-------------------------------------------------------------------
-------------------------

cuidado(s1,e5,'hospital sao joao','porto').
cuidado(s2,e4,'hospital sao marcos','braga').
cuidado(s3,e6,'hospital da luz','lisboa').
cuidado(s4,e2,'hospital sao joao','porto').
cuidado(s5,e1,'centro de saude do caranda','braga').
cuidado(s6,e3,'hospital sao joao','porto').
cuidado(s7,e2,'maternidade alfredo da costa','lisboa').
cuidado(s8,e7,'hospital da luz','lisboa').


%-------------------------------------------------------------------
-------------------------------------------
%------Ato medico: Hora (hh:mm) ,Data(dd-mm-yyyy), #IdUt, #IdServ,
#IdMed, Custo(€€.€€) -> {V,F}----------
%-------------------------------------------------------------------
-------------------------------------------

atomedico('12:20',10-03-2017, u1, s5, m1, 12.20).
atomedico('21:00',2-03-2017, u2, s4, m2, 16.50).
atomedico('09:40',14-02-2017, u3, s6, m3, 21.00).
atomedico('22:00',21-01-2017, u4, s1, m5, 18.20).
atomedico('17:20',12-03-2017, u5, s2, m4,  14.80).
```

```
atomedico('08:20',20-02-2017, u5, s3, m6,  54.32).
atomedico('10:20',28-03-2017, u5, s2, m4,  54.32).
atomedico('11:20',28-02-2017, u2, s6, m3,  12.20).
atomedico('14:20',15-03-2017, u3, s4, m2,  12.20).
atomedico('13:20',15-03-2017, u5, s1, m5,  12.20).
atomedico('09:45',15-03-2017, u1, s5, m1,  12.20).
atomedico('00:00',25-12-2016, u2, s7, m8,  891.24).


%-------------------------------------------------------------------
-------------------------
%------Registo de Utente: #IdUt, Nome, Idade, Morada) -> {V,F}------
-------------------------
%-------------------------------------------------------------------
-------------------------

registoUtente(Id,Nome,Idade,Morada) :-
            insere(utente(Id,Nome,Idade,Morada)).


%-------------------------------------------------------------------
-------------------------
%------Registo de Médico: #IdMed, Nome, Idade, IdEsp) -> {V,F}------
-------------------------
%-------------------------------------------------------------------
-------------------------

registoMedico(IdMed,Nome,Idade,IdEsp) :-
            insere(medico(IdMed,Nome,Idade,IdEsp)).


%-------------------------------------------------------------------
-------------------------
%------Registo de Especialidade: #IdEspecialidade, Designação ->
{V,F}-----------------------
%-------------------------------------------------------------------
-------------------------

registoEspecialidade(IdEsp,Desig) :-
            insere(especialidade(IdEsp,Desig)).


%-------------------------------------------------------------------
-------------------------
%------Registo de Cuidado: #IdCuidado,Descricao,Instituicao,Cidade -
> {V,F}-----------------
%-------------------------------------------------------------------
-------------------------
```

```
registoCuidado(Id,Descricao,Instituicao,Cidade) :-
            insere(cuidado(Id,Descricao,Instituicao,Cidade)).


%-------------------------------------------------------------------
-----------------------------------
%------Registo de atos medicos: Hora, Data, Id-Utilizador, Id-
Servico, Id-Medico, Custo -> {V,F}--------
%-------------------------------------------------------------------
-----------------------------------

registoAtoMedico(Hora,Data,IdU,IdS,IdMed,Custo) :-
            insere(atomedico(Hora,Data,IdU,IdS,IdMed,Custo)).


%-------------------------------------------------------------------
-------------------------
%------Predicado que remove um utente-----------------------------
-------------------------
%-------------------------------------------------------------------
-------------------------

removerUtente(Id,Nome,Idade,Morada) :-
        remocao(utente(Id,Nome,Idade,Morada )).


%-------------------------------------------------------------------
-------------------------
%-------Predicado que remove um medico-----------------------------
-------------------------
%-------------------------------------------------------------------
-------------------------

removerMedico(IdMed, Nome, Idade, IdEsp) :-
        remocao(medico(IdMed, Nome, Idade, IdEsp)).


%-------------------------------------------------------------------
-------------------------
%------Predicado que remove uma especialidade----------------------
-------------------------
%-------------------------------------------------------------------
-------------------------

removerEspecialidade(IdEsp, Desig) :-
        remocao(especialidade(IdEsp, Desig)).
```

```
%-------------------------------------------------------------------
-------------------------
%------Predicado que remove um cuidado-----------------------------
-------------------------
%-------------------------------------------------------------------
-------------------------

removerCuidado(Id,Descricao,Instituicao,Cidade) :-
        retract(cuidado(Id,Descricao,Instituicao,Cidade)),
        solucoes(Id, atomedico(_, _, _, Id, _, _), L),
        removerTodosAtosMedicosPorIdS(Id,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que remove todos os atos médicos cujo cuidado já
foi removido---------------
%-------------------------------------------------------------------
-------------------------

removerTodosAtosMedicosPorIdS(Id, []).
removerTodosAtosMedicosPorIdS(Id, [H|T]) :-
        retract(atomedico(_,_,_,Id,_,_)),
        removerTodosAtosMedicosPorIdS(Id,T).


%-------------------------------------------------------------------
-----------------------------------------
%------Predicado que conta o numero de atos medicos efetuados por um
determinado medico: IdM , L-> {V,F}------
%-------------------------------------------------------------------
-----------------------------------------

contaAtosMedicos(IdM, L):-
  solucoes(IdM, atomedico(_,_,_,_,IdM,_),L1),
  length(L1,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que remove um ato medico--------------------------
-------------------------
%-------------------------------------------------------------------
-------------------------

removerAtoMedico(Hora,Data,IdU,IdS,IdMed,Custo) :-
       remocao(atomedico(Hora,Data,IdU,IdS,IdMed,Custo)).
```

```
%-------------------------------------------------------------------
------------------------
%------Predicado que identifica um servico pelo seu id: {S,A},Y ->
{V,F}---------------------
%-------------------------------------------------------------------
------------------------

mostrarServico({S,A},Y) :-
  solucoes({T,A},especialidade(S, T),Y).


%-------------------------------------------------------------------
------------------------
%------Predicado que identifica um servico pelo seu id numa lista:
L, Y -> {V,F}-------------
%-------------------------------------------------------------------
------------------------

mostrarServicoLista([],[]).
mostrarServicoLista([H|T],Y):-
    mostrarServico(H,X),
    mostrarServicoLista(T,Z),
    acrescenta(X,Z,Y).


%-------------------------------------------------------------------
------------------------
%-------Predicado que identifica os utentes pela sua idade: C,L ->
{V,F}---------------------
%-------------------------------------------------------------------
------------------------

listarUtentesIdade(C,L) :-
    solucoes({I1,I2,I3},utente(I1,I2,C,I3),L).


%-------------------------------------------------------------------
------------------------
%------Predicado que identifica os utentes pela sua cidade: C,L ->
{V,F}---------------------
%-------------------------------------------------------------------
------------------------

listarUtentesCidade(C,L) :-
    solucoes({I1,I2,I3},utente(I1,I2,I3,C),L).
```

```
%-------------------------------------------------------------------------
-------------------------
%------Predicado que identifica os utentes pelo seu nome: C,L ->
{V,F}----------------------
%-------------------------------------------------------------------------
-------------------------

listarUtentesNome(C,L) :-
    solucoes({I1,C,I2,I3},utente(I1,C,I2,I3),L).


%-------------------------------------------------------------------------
-------------------------
%------Predicado que identifica os utentes dado todos os seus dados:
C,L -> {V,F}------------
%-------------------------------------------------------------------------
-------------------------

listarUtentesDados(C,L) :-
    solucoes({I1,I2,I3},utente(C,I1,I2,I3),L).


%-------------------------------------------------------------------------
-----------------------------
%------Predicado que identifica as instituicoes prestadoras de
cuidados de saude: C,L -> {V,F}---
%-------------------------------------------------------------------------
-----------------------------

listarInstituicoes(L) :-
    solucoes(C,cuidado(I1,I2,C,I3),L1),
    removerRepetidos(L1,L).


%-------------------------------------------------------------------------
---------------------------------
%------Predicado que identifica os cuidados prestados por uma
determinada instituicao: C,L -> {V,F}---
%-------------------------------------------------------------------------
---------------------------------

listarCuidadosInstituicao(C,L) :-
  solucoes({I2,I3,I1},cuidado(I1,I2,C,I3),L1),
  removerRepetidos(L1,L2),
  mostrarServicoLista(L2,L).
```

```
%-------------------------------------------------------------------
-------------------------
%------Predicado que identifica os cuidados prestados numa
determinada cidade: C,L -> {V,F}--
%-------------------------------------------------------------------
-------------------------

listarCuidadosCidade(C,L) :-
   solucoes({I2,I3},cuidado(I1,I2,I3,C),L1),
   removerRepetidos(L1,L2),
   mostrarServicoLista(L2,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que identifica os utentes por cuidado: C,L ->
{V,F}-------------------------
%-------------------------------------------------------------------
-------------------------

listarUtentesCuidado(C,L) :-
    solucoes(Idu, atomedico(_,_,Idu,C,_,_),AUX),
    listarUtentesPorIds(AUX,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que identifica os utentes de uma instituicao:
Inst,L -> {V,F}---------------
%-------------------------------------------------------------------
-------------------------

listarUtentesInstituicao(Inst,S) :-
    solucoes(I1,cuidado(I1,I2,Inst,I3),S1),
    listarUtentesPorCuidados(S1,L),
    listarUtentesPorIds(L,S2),
    removerRepetidos(S2,S).


%-------------------------------------------------------------------
-------------------------
%------Predicado que identifica os utentes através do seu id: T, L -
> {V,F} -----------------
%-------------------------------------------------------------------
-------------------------

listarUtentesPorIds([],[]).
listarUtentesPorIds([H|T],L) :-
```

```
    utente(H,I1,I2,I3),
    listarUtentesPorIds(T,X),
    acrescenta([{H,I1,I2,I3}],X,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que identifica os atos médios por Id de Serviços:
T, L -> {V,F}-------------
%-------------------------------------------------------------------
-------------------------

listarAtosMedicosPorIdServico([],[]).
listarAtosMedicosPorIdServico([H|T],L) :-

solucoes({Hora,Data,Idu,IdMed,Custo},atomedico(Hora,Data,Idu,H,IdMed
,Custo),AUX),
    listarAtosMedicosPorIdServico(T,X),
    acrescenta(AUX,X,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que lista os Ids dos medicos por Ids de serviço: T,
L -> {V,F}-------------
%-------------------------------------------------------------------
-------------------------
listarIdMedicosPorIdServico([],[]).
listarIdMedicosPorIdServico([H|T],L) :-
    solucoes(IdMed,atomedico(_,_,_,H,IdMed,_),AUX),
    listarIdMedicosPorIdServico(T,X),
    acrescenta(AUX,X,Z),
    removerRepetidos(Z,L).

%-------------------------------------------------------------------
-------------------------
%------Predicado que identifica os atos médios por cuidado: T, L ->
{V,F}--------------------
%-------------------------------------------------------------------
-------------------------

listarUtentesPorCuidados([],[]).
listarUtentesPorCuidados([H|T],L) :-
    solucoes(Idu,atomedico(_,_,Idu,H,_,_),AUX),
    listarUtentesPorCuidados(T,X),
    acrescenta(AUX,X,L).
```

```
%-------------------------------------------------------------------
-------------------------
%------Predicado que determina as instituições de um médico: IdMed,
L -> {V,F}---------------
%-------------------------------------------------------------------
-------------------------

instituicoesPorMedico(IdMed, L) :-
    solucoes(IdS, atomedico(_,_,_,IdS,IdMed,_),R),
    listarInstituicoesVariosServicos(R,Z),
    removerRepetidos(Z,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que lista  todas instituições de um conjunto de
Serviços: T, P -> {V,F}-----
%-------------------------------------------------------------------
-------------------------

listarInstituicoesVariosServicos([], []).
listarInstituicoesVariosServicos([H|T], P) :-
    listarInstituicoesServico(H,R),
    listarInstituicoesVariosServicos(T,L),
    acrescenta(R,L,P).


%-------------------------------------------------------------------
-------------------------
%------Predicado que identifica as instituições por um determinado
serviço: C, L -> {V,F}----
%-------------------------------------------------------------------
-------------------------

listarInstituicoesServico(C,L) :-
    solucoes(Inst,cuidado(C,_,Inst,_),L1),
    removerRepetidos(L1,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que lista toda a informação de uma lista de
serviços: T, L -> {V,F}---------
%-------------------------------------------------------------------
-------------------------

listarInfoServicos([],[]).
listarInfoServicos([H|T],L) :-
```

```
    solucoes({Desc,Inst,Cidade},cuidado(H,Desc,Inst,Cidade),AUX),
    listarInfoServicos(T,X),
    acrescenta(AUX,X,L1),
    mostrarServicoLista(L1,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que lista os atos médicos para um utente: Idu, L ->
{V,F}-------------------
%-------------------------------------------------------------------
-------------------------

atosMedicosPorUtente(Idu,L) :-

solucoes({Hora,Data,Ids,IdMed,Custo},atomedico(Hora,Data,Idu,Ids,IdM
ed,Custo),L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que lista os atos médicos para um serviço: Ids, L -
> {V,F}------------------
%-------------------------------------------------------------------
-------------------------

atosMedicosPorServico(Ids,L) :-

solucoes({Hora,Data,Idu,IdMed,Custo},atomedico(Hora,Data,Idu,Ids,IdM
ed,Custo),L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que lista os atos médicos para uma instituicao:
Inst, L -> {V,F}-----------
%-------------------------------------------------------------------
-------------------------

atosMedicosPorInstituicao(Inst,L) :-
    solucoes(Ids,cuidado(Ids,Desc,Inst,Cidade),L1),
    listarAtosMedicosPorIdServico(L1,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que lista os ids dos médicos de uma instituicao:
Inst, L -> {V,F}-----------
```

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

idMedicosPorInstituicao(Inst,L) :-
    solucoes(Ids,cuidado(Ids,Desc,Inst,Cidade),L1),
    listarIdMedicosPorIdServico(L1,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que lista os serviços associados a um utente: Idu,
L -> {V,F}---------------
%-------------------------------------------------------------------
-------------------------

servicosPorUtente(Idu,L) :-
    solucoes(Ids,atomedico(Hora,Data,Idu,Ids,IdMed,Custo),L1),
    listarInfoServicos(L1,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que calcula o custo total de atos medicos de um
utente: Idu, R -> {V,F}----
%-------------------------------------------------------------------
-------------------------

calculoCustoTotalAtosUtente(Idu,R) :-
    solucoes(C, atomedico(_,_,Idu,_,_,C),X),
    somatorio(X,R).


%-------------------------------------------------------------------
-------------------------
%------Predicado que calcula o custo total dos atos medicos: R ->
{V,F}----------------------
%-------------------------------------------------------------------
-------------------------

calculoCustoTotalAtos(R) :-
    solucoes(P,atomedico(_,_,_,_,_,P),X),
    somatorio(X,R).


%-------------------------------------------------------------------
-------------------------
%------Predicado que calcula o custo total dos atos medicos numa
data: C, R -> {V,F}---------
```

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

calculoCustoTotalData(C,R) :-
    solucoes(P,atomedico(_,C,_,_,_,P),X),
    somatorio(X,R).

%-------------------------------------------------------------------
-------------------------
%------Predicado que calcula o custo total dos atos medicos de um
cuidado: C, R -> {V,F}---------
%-------------------------------------------------------------------
-------------------------

calculoCustoTotalServico(C,R) :-
    solucoes(P,atomedico(_,_,_,C,_,P),X),
    somatorio(X,R).

%-------------------------------------------------------------------
-------------------------
%-------Predicado que calcula o custo total dos atos medicos numa
instituição: C, R -> {V,F}---------
%-------------------------------------------------------------------
-------------------------

calculoCustoTotalInstituicao(IdI, L):-
  solucoes(IdS,cuidado(IdS,_,IdI,_),P),
  listaCustosServicos(P,L1),
  somatorio(L1,L).

listaCustosServicos([],[]).
listaCustosServicos([H|T],R):-
  listaCustosServicos(T,R1),
  solucoes(C, atomedico(_,_,_,H,_,C),X),
  somatorio(X,Y),
  acrescenta([Y],R1,R).


%-------------------------------------------------------------------
-------------------------
%------Predicado que mostra os utentes associados a um médico: C, R
-> {V,F}-----------------
%-------------------------------------------------------------------
-------------------------

utentesPorMedico(C,R) :-
   solucoes(U,atomedico(_,_,U,_,C,_),R1),
   removerRepetidos(R1,R2),
```

```
   listarUtentesPorIds(R2,R).


%-------------------------------------------------------------------
-------------------------
%------Predicado que lista todos os médicos por Id: T, L -> {V,F}---
-------------------------
%-------------------------------------------------------------------
-------------------------

listarMedicosPorIds([],[]).
listarMedicosPorIds([H|T],L) :-
    medico(H,I1,I2,I3),
    listarMedicosPorIds(T,X),
    acrescenta([{H,I1,I2,I3}],X,L).


%-------------------------------------------------------------------
-------------------------
%------Predicado que determina os médicos de um utente: IdU, R ->
{V,F}----------------------
%-------------------------------------------------------------------
-------------------------

medicoPorUtente(IdU,R) :-
    solucoes(IdM,atomedico(_,_,IdU,_,IdM,_),R1),
    removerRepetidos(R1,R2),
    listarMedicosPorIds(R2,R).


% -----------------------------------------------------------------
---------------------------------------
%---------------------------------- INVARIANTES --------------------
---------------------------------------
% -----------------------------------------------------------------
---------------------------------------

% --------------- Invariantes de Inserção -------------------------
-------------

%-------------------------------------------------------------------
-------------
%------Invariante que não permite a inserção de utentes com o mesmo
ID-----------
%-------------------------------------------------------------------
-------------
```

```
+utente(Idu,Nome,Idade,Morada) ::
(solucoes(Idu,utente(Idu,_,_,_),S),
                                  comprimento(S,N),
                                  N == 1).


%-----------------------------------------------------------------
-------------
%------Invariante que não  permite a inserção de médicos com o mesmo
ID----------
%-----------------------------------------------------------------
-------------

+medico(IdMed,Nome,Idade,IdEsp) ::
(solucoes(IdMed,medico(IdMed,_,_,_),S),
                                   comprimento(S,N),
                                   N == 1).

%-----------------------------------------------------------------
-------------
%------Invariante que não  permite a inserção de médicos com
especialidade que não exista----------
%-----------------------------------------------------------------
-------------

+medico(IdMed,Nome,Idade,IdEsp) :: (especialidade(IdEsp,_)).


%-----------------------------------------------------------------
-------------
%------Invariante que não permite a inserção de especialidades com o
mesmo ID----
%-----------------------------------------------------------------
-------------

+especialidade(IdEsp, Desig) ::
(solucoes(IdEsp,especialidade(IdEsp,_),S),
                                   comprimento(S,N),
                                   N == 1).


%-----------------------------------------------------------------
-----------------------------------------------------------------
------
%------Invariante que não permite a inserção de cuidados com o mesmo
ID, ou, em alternativa, com a descrição, instituição e cidades
iguais----
```

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

+cuidado(Ids,Desc,Inst,Cidade) ::
(solucoes(Ids,cuidado(Ids,_,_,_),S),
                                    comprimento(S,N),
                                    N == 1,

solucoes(Ids,cuidado(_,Desc,Inst,Cidade),S),
                                    comprimento(S,N),
                                    N == 1).


%------------------------------------------------------------------
---------------------------------------
%------Invariante que não permite a inserção de atos médicos em que
o custo seja inferior ou igual a zero----
%------------------------------------------------------------------
---------------------------------------

+atomedico(Hora,Data,IdU,IdS,IdMed,Custo) :: Custo > 0.


%------------------------------------------------------------------
-------------------------------------------------------------
%------Invariante que não permite a inserção de atos médicos em que
a especialidade do cuidado seja diferente da do ato médico----
%------------------------------------------------------------------
-------------------------------------------------------------

+atomedico(Hora,Data,IdU,IdS,IdMed,Custo) ::
(medico(IdMed,_,_,IdEsp),

cuidado(IdS,IdEsp,_,_)).

%------------------------------------------------------------------
-----------------------------------------------
%------Invariante que não permite a inserção de um ato médico em que
o utente, o serviço ou o médico não existam----
%------------------------------------------------------------------
-----------------------------------------------

+atomedico(Hora,Data,IdU,IdS,IdMed,Custo) :: (solucoes(IdU,
utente(IdU,_,_,_),S),
                                                comprimento(S,N),
                                                N == 1,
```

```
                                                    solucoes(IdMed,
medico(IdMed,_,_,_),L),
                                                    comprimento(L,M),
                                                    M == 1,
                                                    solucoes(IdS,
cuidado(IdS,_,_,_),R),
                                                    comprimento(R,Z),
                                                    Z == 1).


%-------------------------------------------------------------------
-------------------------------------------------
%------Invariante que não permite a inserção de um ato médico em que
o médico já tenha consulta à hora estipulada-----
%-------------------------------------------------------------------
-------------------------------------------------

+atomedico(Hora,Data,IdU,IdS,IdMed,Custo) :: (solucoes(IdMed,
atomedico(Hora,Data,_,_,IdMed,_),S),
                                              comprimento(S,N),
                                              N==1).


%-------------------------------------------------------------------
-----------------------------------------------
%------Invariante que não permite a inserção de um ato médico à
mesma hora, na mesma data com o mesmo utente------
%-------------------------------------------------------------------
-----------------------------------------------

+atomedico(Hora,Data,IdU,IdS,IdMed,Custo) :: (solucoes(IdU,
atomedico(Hora,Data,IdU,_,_,_),S),
                                               comprimento(S,N),
                                               N == 1).

% --------------- Invariantes de Remoção----------------------------
-----------

%-------------------------------------------------------------------
--------------------
%------Invariante que não permite a remoção de um utente com atos
médicos marcados------
%-------------------------------------------------------------------
--------------------

-utente(Id, Nome, Idade, Morada) :: (solucoes(Id,
atomedico(_,_,Id,_,_,_), S),
                                    comprimento(S,N),
```

```
                                    N==0).

%--------------------------------------------------------------------
--------------------
%------Invariante que não permite a remoção de um médico com atos
médicos marcados------
%--------------------------------------------------------------------
--------------------

-medico(Id,Nome,Idade,IdEsp) ::
(solucoes(Id,atomedico(_,_,_,_,Id,_),S),
                                    comprimento(S,N),
                                    N==0).


%--------------------------------------------------------------------
-------------------------
%------Invariante que não permite a remoção de uma especialidade se
um médico a possuir------
%--------------------------------------------------------------------
-------------------------

-especialidade(IdEsp, Desc) ::
(solucoes(IdEsp,medico(_,_,_,IdEsp),S),
                                  comprimento(S,N),
                                  N==0).


%--------------------------------------------------------------------
--------------------------
%------Invariante que não permite a remoção de uma especialidade se
um cuidado a possuir-------
%--------------------------------------------------------------------
--------------------------

-especialidade(IdEsp, Desc) ::
(solucoes(IdEsp,cuidado(_,IdEsp,_,_),S),
                                  comprimento(S,N),
                                  N==0).
```