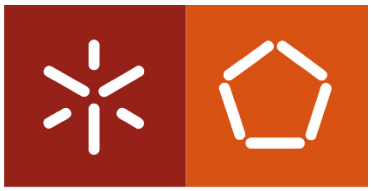

Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado Integrado em Engenharia Informática
Sistemas de Representação Conhecimento e Racocínio

Exercício 1

Registo de eventos numa instituição de saúde

Grupo 19

Célia Natália Lemos Figueiredo
Aluna a67367

Gil Gonçalves
Aluno a67738

José Carlos Pedrosa Lima de Faria
Aluno a67638

Judson Quissanga Coge Paiva
Aluno E6846

Braga, 26 de Março de 2016

Conteúdo

Lista de Figuras

Resumo

O presente relatório documenta o primeiro trabalho prático da Unidade Curricular de Sistemas de Representação Conhecimento e Racocínio. Nesta primeira fase o objetivo foi construir um mecanismo de representação de conhecimento para o registo de eventos numa instituição de saúde. Em termos gerais, foi usada a linguagem PROLOG, esta que utiliza um conjunto de fatos, predicados e regras de derivação de lógica. Uma execução de um programa é, na verdade, uma prova de um teorema, iniciada por uma consulta. Neste relatório pretende-se apresentar a forma como a aplicação foi construída bem como explicar algumas decisões tomadas.

1. Introdução

Este primeiro capítulo tem como objetivo apresentar uma breve introdução ao exercício a realizar. Sendo assim, é necessário perceber os motivos que levaram à resolução do exercício assim como os objetivos pretendidos. A Programação em Lógica é um tipo específico de programação cujo objetivo é a implementação de um programa cujo conteúdo se prende em factos (registos que se sabem verdadeiros), predicados (associados aos factos) e regras. A este programa podem ser estruturadas questões sobre o seu conteúdo e obter-se-ão respostas válidas e corroboradas pela lógica em si. A programação em lógica baseia-se em dois princípios básicos para a "descoberta" das respostas (soluções) a essas questões: lógica, usada para representar os conhecimentos e informação, e Inferência, regras aplicadas à Lógica para manipular o conhecimento.

1.1 Motivação e Objetivos

O PROLOG é uma linguagem declarativa, pois fornece uma descrição do problema que se pretende computar utilizando uma coleção de factos e regras lógicas que indicam como deve ser resolvido o problema proposto. Sendo também uma linguagem que é especialmente associada com a inteligência artificial e linguística computacional este é um dos grandes motivos que nos levou a querer aprender este tipo de linguagem, esta mais direcionada ao conhecimento do que aos algoritmos.

Após os conhecimentos adquiridos na linguagem de programação lógica PROLOG, este exercício surge com o objetivo de consolidar conhecimentos e obter experiência e prática face a problemas de programação em lógica. O objetivo final será a construção de um programa capaz de armazenar conhecimento sobre registo de eventos numa instituição de saúde e através deste solucionar questões deste tema.

1.2 Estrutura do documento

O presente relatório encontra-se organizado em seis capítulos. Sendo que neste primeiro introduzimos a linguagem e o tema a tratar menciona-se também a motivação e os objetivos que nos levaram à realização deste exercício. No segundo capítulo será feito um estudo prévio da linguagem de modo a que o leitor possa entender o exercício. No terceiro capítulo explicaremos o que foi desenvolvido para a implementação do exercício. No quarto capítulo apresentaremos as conclusões e interpretação dos resultados obtidos. Por fim será apresentados um anexo com o código desenvolvido.

2. Preliminares

Neste capítulo vao ser apresentados alguns conceitos fundamentais para a elaboraçãoo deste exercício e algumas das ferramentas fundamentais para a elaboração do mesmo.

2.1 Estudos anteriores

Para a realização deste trabalho foram necessários alguns conhecimentos anteriores sobre programação em lógica e, posteriormente, o uso da linguagem de programação PROLOG. Este conhecimento foi adquirido ao longo das aulas teóricas (programação em lógica) e aulas teórico-práticas (PROLOG) de Sistemas de Representação de Conhecimento e Raciocínio. Sobre estes conhecimentos, devemos destacar todos os conceitos que foram aprendidos tais como o que são predicados, o que são cláusulas, o que é a base de conhecimento, entre outros conceitos de programação em lógica que serão explicados ao longo deste documento. Após termos alguns conhecimentos de programação em lógica falta colocá-los em prática, e, é aqui, que entram os conhecimentos de PROLOG e da ferramenta *SICSTus* usada para compilar e interpretar o código desenvolvido nesta linguagem.

Para o desenvolvimento desses predicados foi necessário fazer uma análise de conhecimentos de cada um dos membros sobre o tema e acompanhado de uma pequena pesquisa sobre as características destes.

O estudo inicial passou por caracterizar um utente, este que é definido pelo nome, serviço em que está inscrito ou consultado, profissional atribuído e a instituição onde o profissional exerce e onde o utente é atendido, que devem coincidir. Na realidade, o utente é composto por muitos outros elementos, tal como, número de utente, número de contribuinte, número de cartão de cidadão entre muitos outros, porém para simplificar e como não é necessário para este caso em estudo, vamos apenas utilizar um código e o nome do utente.

Um serviço é caracterizado pelo nome, isto é o nome do serviço terá de ser elucidativo, por exemplo, “pediatria” e pela instituição a que corresponde ao nome do local onde se presta esse serviço aos utentes, e como tal será algo como “hospital....” ou “centro de saúde...” ou algo semelhante que reflita que esse local é um local em que se prestam serviços na área da saúde.

Um profissional é caracterizado pelo seu nome, serviço em que está inserido, sendo que pode estar em mais que um serviço e a instituição onde labora. E estes são apenas os requisitos mínimos que permitem ao sistema determinar as respostas a todas as questões.O profissional também terá um códiogo e um nome para a sua identificação.

Uma instituição é caracterizada apenas pelo seu nome por forma a simplificar este sistema visto que os requisitos não pretendem questionar algo implique que a instituição tenha mais

atributos no seu predicado além do nome.

2.2 Programação em Lógica e PROLOG

De modo a que a leitura deste documento seja perceptível em termos de conceitos e símbolos é necessário fazer referências breves a noções básicas de PROLOG, a linguagem em que é desenvolvido este trabalho. Tal como foi mencionado anteriormente uma linguagem de programação lógica utiliza a lógica para representar conhecimento e inferências para manipular informação. Um programa neste tipo de programação possui então os seguintes parâmetros:

- Factos - constatações sobre algo que se conhece e se sabe verdadeiro, por exemplo cor(azul), mae(sofia, joão)
- Predicados – implementam relações, por exemplo o predicado filho(filho, pai) implementa a relação de descendência direta (ser filho de)
- Regras – utilizadas para definir novos predicados.

Estes são alguns exemplos dos conhecimentos base para perceber a programação em lógica. Após se ter estes conhecimentos, é necessário traduzir estes e aplicá-los na linguagem PROLOG. Deixamos, então, alguns exemplos importantes para a usar:

- . utilizado para terminar uma declaração;
- :- significa “se”;
- , possui o significado “e”;
- ; significa “ou”;
- // representa a unificação;

É ainda necessário referir que as variáveis representam-se por maiúsculas e constantes, predicados e factos com minúsculas. Com estas noções como base passa-se agora ao desenvolvimento das tarefas do exercício.

3. Descrição do Trabalho e Análise de Resultados

Nesta parte do documento serão explicitadas todas as etapas de resolução dos desafios fornecidos bem como todas as decisões efetuadas no processo.

3.1 Base de Conhecimento

A base de Conhecimento define bases de dados ou conhecimento acumulados sobre determinado assunto. Para a elaboração deste exercício tornou-se importante definir uma base de conhecimento que possa responder aos pedidos do enunciado.

3.1.1 Instituições

De acordo com o enunciado existem as instituições de saúde estas que têm a elas associados serviços e estes têm profissionais e respetivos utentes.

Para a implementação do exercício foi necessário criar uma base de conhecimento das instituições existentes Como tal usamos o predicado instituição para descrever esta relação: **instituição(nome).**, como no exemplo:

```
instituicao( hospital_guimaraes ).
instituicao( hospital_braga ).
instituicao( hospital_barcelos ).
```

Inicialmente tinhamos pensado relacionar a instituição com o serviço, pois como uma instituição tem a si associado respetivo/s serviço/s, usámos o predicado **instserv** com a assinatura **instserv (intituicao,servico).**, porém repensámos e decidimos que poderiamos interligar todas entidades num só predicado. Este facto pode ser verificado no capítulo 3.1.5, entretanto fica um exerto da base de conhecimento de como se tinha pensado:

```
instserv( hospital_braga, cardiologia ).
instserv( hospital_beatriz_angelo, endocrinologia ).
```

3.1.2 Serviços

Um requisito a incluir na base de conhecimento é a existencia de serviços, para representar esta situação temos o predicado servico com a assinatura **servico(nome).**

```
servico( cardiologia ).
servico( cirugiageral ).
servico(  neurologia ).
```

3.1.3 Profissionais

Mais um requisito do sistema a incluir na base de conhecimento é a existencia de profissionais com a seguinte assinatura **profissional(codigo, nome).**

```
profissional(1,marcus).
profissional(2,maria).
profissional(3,jorge).
```

3.1.4 Utentes

Por fim, incluimos os utentes na base de conhecimento estes com a assinatura **utente(codigo,nome).**

```
utente(1,jose).
utente(2,carlos).
utente(3,maria).
```

Como os utentes pertencem a uma certa instituição de saúde foi necessário introduzir esse conhecimento na base sendo a sua assinatura

3.1.5 Relacionamento entre Entidades

Como a cada isntituição estão relacionados serviços e esses serviços necessitam de profissionais de saúde e para terminar o ciclo são precisos os utentes. Resolvemos criar um facto na base de conhecimento que relacionasse as quatro entidades. Para tal introduzimos na base de conhecimento o predicado **ins_serv_uten_profi(intituicao, servico,Codigo utente, Codigo profissional).**, segue de seguida um exerto e uma imagem ilustrativa da base de conhecimento implementada.

```
ins_serv_uten_profi( hospital_braga, cardiologia,1,1 ).
ins_serv_uten_pofi( hospital_trofa, cardiologia,1,1 ).
```

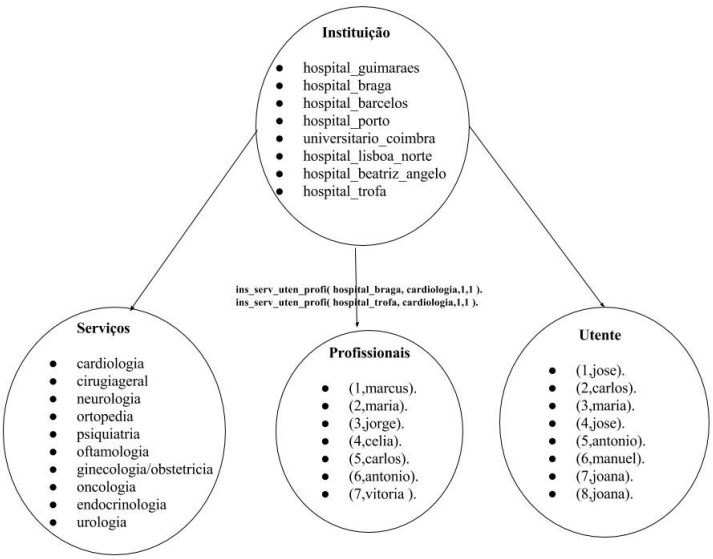

Figura 3.1: Esquema da base de conhecimento

3.2 Funcionalidades Básicas

Após a apresentação da base de conhecimento do exercício torna-se possivel desenvolver alguns teoremas que consigam responder aos desafios apresentados. Desta forma, apresentaremos e explicaremos a forma como resolvemos cada um dos desafios e quais os resultados obtidos pelas soluções por nós apresentadas.

3.2.1 Identificar os serviços existentes numa instituição

Pretende-se que seja possivel apresentar todos os serviços existentes numa determinada instituição de saúde. Para tal basta procurar todos os serviços (*findall*) que estejam ligados à instituição pretendida através do axioma *servicoInst(Inst,Serv)*, este que nos devolve uma lista com todos os serviços existentes numa instituição. Segue de seguida o axioma implementado:

```
servicoInst(Inst,Serv):-
findall(K,ins_serv_uten_profi(Inst,K,_,_),Serv).

servicoInst(Inst,[Serv|K]):-
ins_serv_uten_profi(Inst,Serv,_,_),
servicoInst(Inst,K).

servicoInst(Inst,[Serv]):-
ins_serv_uten_profi(Inst,Serv,_,_).
```

Mostramos de seguida o output obtido do programa para este axioma:

```
| ?- servicoInst(hospital_barcelos,X).
X = [oftamologia,endocrinologia] ?
yes
% 1
| ?- servicoInst(hospital_braga,V).
V = [cardiologia,oncologia] ?
yes
% 1
```

Figura 3.2: Exemplo de output do axioma servicoInst

3.2.2 Identificar os utentes de uma instituição

Para identificar todos os utentes de uma instituição, o método será semelhante ao anterior, pois é necessário selecionar os utentes que pertencem a determinada instituição. Usamos assim o axioma *utentesInst(Inst,Uten)*, que permite encontrar todos os utentes de uma dada instituição, mostramos de seguida o axioma implementado para este desafio:

```
utentesInst(Inst,Uten):-
findall((K,J),(ins_serv_uten_profi(Inst,_,K,_),utente(K,J)),Uten).

utentesInst(Inst,[(Cod,Uten)|K]):-
ins_serv_uten_profi(Inst,_,Cod,_),
utente(Cod,Uten),
utentesInst(Inst,K).
```

```
utentesInst(Inst,[(Cod,Uten)]):-
ins_serv_uten_profi(Inst,_,Cod,_),
utente(Cod,Uten).
```

O output produzido pela execução do axioma é:

```
% 1
| ?- utentesInst(hospital_barcelos, Y).
Y = [(7,joana),(1,jose)] ?
yes
% 1
| ?- utentesInst(hospital_guimaraes, Y).
Y = [(3,maria)] ?
yes
% 1
| ?- utentesInst(hospital_braga, X).
X = [(1,jose),(3,maria)] ?
yes
% 1
```

Figura 3.3: Exemplo da execução do axioma utentesInst

3.2.3 Identificar os utentes de um determinado serviço

Mais uma vez a forma de identificar os utentes de determinado serviço é semelhante às anteriores em que usámos a funcionalidade *findall* que nos devolve uma lista com os utentes, neste caso o código de utente e o respetivo nome.

```
servUtente(Serv,Ute):-
findall((K,J),(ins_serv_uten_profi(_,Serv,K,_),utente(K,J)),Ute).

servUtente(Serv,[(Cod,Uten)|K]):- ins_serv_uten_profi(_,Serv,Cod,_),
utente(Cod,Uten),
servUtente(Serv,K).

servUtente(Serv,[(Cod,Uten)]):- ins_serv_uten_profi(_,Serv,Cod,_),
utente(Cod,Uten).
```

Mostrámos de seguida o output da execução do axioma:

```
% 1
| ?- servUtente(cardiologia, U).
U = [(1,jose),(1,jose),(4,jose)] ?
yes
% 1
| ?- servUtente(neurologia, U).
U = [(3,maria),(4,jose)] ?
yes
% 1
```

Figura 3.4: Exemplo da execução do axioma servUtente

3.2.4 Identificar os utentes de um determinado serviço numa instituição

Para a identificação dos utentes que estão num determinado serviço numa certa instituição de saúde, implementamos o axioma *utenServInst(Serv,Inst,Uten)*, este que procura o serviço e a intituição e nos devolve uma lista com os utentes.

```
utenServInst(Serv,Inst,Uten):-
findall((K,J),(ins_serv_uten_profi(Inst,Serv,K,_),utente(K,J)),Uten).

utenServInst(Serv,Inst,[(Cod,Uten)|K]):-
utente(Cod,Uten),
ins_serv_uten_profi(Inst,Serv,Cod,_),
utenServInst(Serv,Inst,K).

utenServInst(Serv,Inst,[(Cod,Uten)]):-
ins_serv_uten_profi(Inst,Serv,Cod,_),
utente(Cod,Uten).
```

```
% 1
| ?- utenServInst(cardiologia, hospital_braga, U).
U = [(1,jose)] ?
yes
% 1
| ?- utenServInst(neurologia, hospital_guimaraes, U).
U = [(3,maria)] ?
yes
% 1
```

Figura 3.5: Exemplo da execução do axioma utenServInst

3.2.5 Identificar as instituições onde seja prestado um dado serviço ou conjunto de serviços

A implementação deste axioma é semelhante aos anteriores, pois é necessário relacionar o serviço ou vários serviços à respetiva instituição. Para tal implementamos o axioma **instServico: ([Serviço],[Instituição])**, que recorre ao **servicoInst** para garantir que o serviço está na instituição e também elimina os nomes dos serviços que apareçam repetidos.

```
instServicos([Serv|K],I):-
findall(H,servicoInst(H,[Serv|K]),L),
eliminarRepetidos(L,I).

inst_Servico(S,[I|K]):- servicoInst(I,[S]),
inst_Servico(S,K).

inst_Servico(S,[I]):- servicoInst(I,[S]).

instServicos([S|T],[I|K]):-
inst_Servico(S,[I|K]),
instServicos(T,[I|K]).

instServicos([S],[I|K]):-inst_Servico(S,[I|K]).

instServicos([S],[I]):-inst_Servico(S,[I]).
```

De seguida mostramos o output obtido:

```
| ?- instServicos([cardiologia], X).
X = [hospital_braga,hospital_trofa,hospital_porto] ?
yes
| ?- instServicos([oftamologia, endocrinologia], X).
X = [hospital_barcelos] ?
yes
| ?-
```

Figura 3.6: Exemplo da execução do axioma instServicos

3.2.6 Identificar os serviços que não se podem encontrar numa instituição

Neste desafio é necessário saber quais os serviços exitentes, depois saber quais os serviços que fazem parte de uma instituição. No final subtraimos o total de serviços aos existentes em cada instituição e devolve assim os serviços que não existem em determinada instituição. Foi implementado o axioma **difList** para subtrair todos o serviços existentes aos existentes em cada instituição.

```
todosServicos(L):-
findall(S,servico(S),L).

servicosForaInst(Ins,Serv,todos):-todosServicos(P),
servicoInst(Ins,K),
difList(P,K,Serv).

servicosForaInst(Ins,[Serv|K]):-nao(servicoInst(Ins,[Serv|K])).
```

Mostramos de seguida um exemplo da execução do programa:

```
yes
| ?- servicosForaInst(hospital_guimaraes,X,todos).
X = [cardiologia,cirugiageral,ortopedia,psiquiatria,oftamologia,ginecologia/obstetricia,oncologia,endocrinologia,urologia] ?
yes
| ?- servicosForaInst(hospital_braga,X,todos).
X = [cirugiageral,neurologia,ortopedia,psiquiatria,oftamologia,ginecologia/obstetricia,endocrinologia,urologia] ?
yes
| ?- servicosForaInst(hospital_porto,X,todos).
X = [cirugiageral,neurologia,ortopedia,oftamologia,ginecologia/obstetricia,oncologia,endocrinologia,urologia] ?
yes
| ?-
```

Figura 3.7: Exemplo da execução do axioma servicosForaInst

3.2.7 Determinar as instituições onde um profissional presta serviço

Para este desafio foi necessário descobrir quais as instituições onde determinado profissional presta serviço. Para tal usamos o método de procurar todas as intituições que satisfazem o objetivo de ser um profissional que presta serviço numa determinada instituição.

```
profiServico((Cod,Prof),Inst):-
findall(K,(ins_serv_uten_profi(K,_,_,Cod),profissional(Cod,Prof)),Inst).

profiServico((Cod,Prof),[Inst|K]):-ins_serv_uten_profi(Inst,_,_,Cod),
profissional(Cod,Prof),
profiServico((Cod,Prof),K).

profiServico((Cod,Prof),[Inst]):-ins_serv_uten_profi(Inst,_,_,Cod),
profissional(Cod,Prof).
```

Mostramos de seguida um exemplo de execução do axioma:

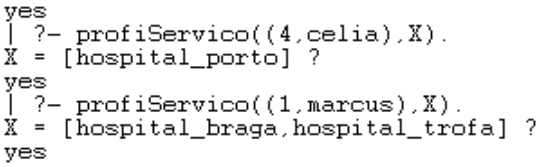

Figura 3.8: Exemplo da execução do axioma profiServico

3.2.8 Determinar todas as instituições(ou serviços ou profissionais) a quem o utente já recorreu

Este desafio foi realizado por partes, pois é necessário saber quais as intituições todas a que um utente já recorreu, começamos por criar o axioma **instSerProf((Cod,Uten),Inst,ints)**, em que quando se dá um utente devolve uma lista de instituições onde esse utente já frequentou.

De seguida implementamos o axioma **instSerProf((CodU,Uten),Serv,serv)**, que permite saber quais os serviços que um utente já frequentou. Por fim construímos o axioma **instSerProf((CodU,Uten),Prof,prof)**, que nos devolve uma lista com os profissionais a quem determinado utente recorreu. Todos estes axiomas estão a eliminar informação repetida.

```
instSerProf((Cod,Uten),Inst,ints):-
 findall(K,(ins_serv_uten_profi(K,_,Cod,_),utente(Cod,Uten)),L),
eliminarRepetidos(L,Inst).

instSerProf((CodU,Uten),Serv,serv):-
findall(K,(ins_serv_uten_profi(_,K,CodU,_),utente(CodU,Uten)),L),
eliminarRepetidos(L,Serv).

instSerProf((CodU,Uten),Prof,prof):-
findall((K,Nome),(ins_serv_uten_profi(_,_,CodU,K),utente(CodU,Uten),
profissional(K,Nome)),L),
eliminarRepetidos(L,Prof).
```

Exemplificamos na imagem abaixo a execução dos axiomas acima descritos:

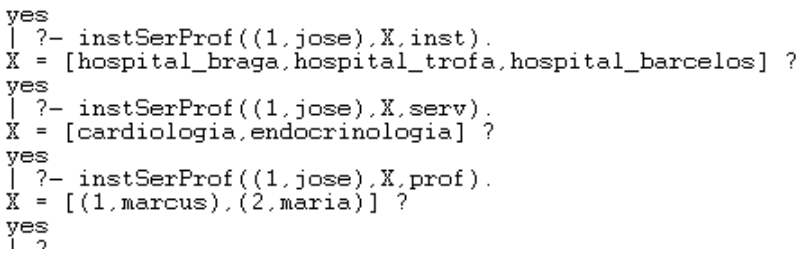

Figura 3.9: Exemplo da execução do axioma instSerProf

3.2.9 Registar utentes, profissionais, serviços ou instituições

Para possibilitar a inserção de conhecimento na base implementamos o teorema **inserirConhecimento**, este teorema que necessita de verificar os invariantes definidos por nós, pois não podiamos deixar que informação repetida fosse adicionada à base de conhecimento. Fica de seguida um exerto de como implementamos as inserções na base de conhecimento:

```
adicionarUtentes(Codigo,Utente):-
inserirConhecimento(utente(Codigo,Utente)).

adicionarServico(Servico):-
inserirConhecimento(servico(Servico)).

adicionarProfissional(Codigo,Profissional):-
 inserirConhecimento(profissional(Codigo,Profissional)).

adicionarInstituicao(Nome):-
inserirConhecimento(instituicao(Nome)).
```

Criamos vários invariantes para garantir que não é inserido conhecimento repetido no sistema.

O primeiro invariante criado não permite que seja introduzido de novo um utente já existente, assim como um profissional:

```
+utente(Codigo,Uten) :: (solucoes( (Codigo,Uten),
 utente(Codigo,Uten), S),comprimento(S,N),N==1).
```

O segundo invariante não permite introduzir um código de utente já existente, assim como também implementamos para o profissional o mesmo invariante:

```
+utente(Codigo,_) :: (solucoes( Uten, utente(Codigo,Uten), S),
comprimento(S,N),N==1).
```

No terceiro invariante não permitimos que se introduzisse o mesmo nome a uma intituição, partindo do facto que não existem instituições com o mesmo nome.

```
+instituicao(Nome) :: (solucoes( Nome, instituicao(Nome), S),
comprimento(S,N),N==1).
```

No quarto inavariante não permitimos que se introdizem o mesmo nome para um serviço.

```
+servico(Nome) :: (solucoes( Nome, servico(Nome), S),
comprimento(S,N),N==1).
```

Mostramos de seguida a execução dos teoremas e a respetiva listagem com os dados inseridos na base de conhecimento.

3.2.10 Remover utente (profissionais, serviços, instituições) dos registos

Para possibilitar a remoção de conhecimento foi necessário criar o predicado **remover**, este predicado também necessita de verificar invariantes criados por nós. Mostramos de seguida o exercerto dos predicados de remoção :

```
removerUtentes(Codigo,Utente):- remover(utente(Codigo,Utente)).

removerServico(Servico):- remover(servico(Servico)).

removerProfissional(Codigo,Profissional):- remover(profissional(Codigo

removerInstituicao(Nome):- remover(instituicao(Nome)).
```

```
yes
| ?- adicionarUtentes(10, joao_leite).
yes
| ?- adicionarServico(medicina_dentaria).
yes
| ?- listing(servico).
servico(cardiologia).
servico(cirugiageral).
servico(neurologia).
servico(ortopedia).
servico(psiquiatria).
servico(oftamologia).
servico(ginecologia/obstetricia).
servico(oncologia).
servico(endocrinologia).
servico(urologia).
servico(medicina_dentaria).

yes
| ?- listing(utente).
utente(1, jose).
utente(2, carlos).
utente(3, maria).
utente(4, jose).
utente(5, antonio).
utente(6, manuel).
utente(7, joana).
utente(8, joana).
utente(9, natalia_lemos).
utente(11, antonio_campos).
utente(10, joao_leite).
```

Figura 3.10: Exemplo da execução do axioma adicionar utente ou serviço com a respetiva listagem de dados

O primeiro invariante criado não permite que seja removido um utente enquanto estiver a ser consultado:

```
-utente(Codigo,Nome) :: (nao(ins_serv_uten_profi(_,_,Codigo,_)),nao(ut
```

O segundo invariante não permite remover um profissional enquanto este estiver num serviço ou instituição.

```
-profissional(Codigo,Nome) :: (nao(ins_serv_uten_profi(_,_,_,Codigo)),
```

O terceiro invariante não permite que seja removido um servico com profissionais a trabalhar nele ou utentes a usá-lo, ou estar na instituicao

```
-servico(Nome) :: (nao(ins_serv_uten_profi(_,Nome,_,_))).
```

O quarto invariante não permite remover uma instituicao com profissionais, utentes, ou servicos

```
-instituicao(Nome) :: (nao(ins_serv_uten_profi(Nome,_,_,_))).
```

Mostramos de seguida um exemplo de remoção de conhecimento:

```
yes
| ?- removerUtentes(11,antonio_campos).
yes
| ?- removerServico(medicina_dentaria).
yes
| ?- listing(utente).
utente(1, jose).
utente(2, carlos).
utente(3, maria).
utente(4, jose).
utente(5, antonio).
utente(6, manuel).
utente(7, joana).
utente(8, joana).
utente(9, natalia_lemos).
utente(10, joao_leite).

yes
| ?- listing(servico).
servico(cardiologia).
servico(cirugiageral).
servico(neurologia).
servico(ortopedia).
servico(psiquiatria).
servico(oftamologia).
servico(ginecologia/obstetricia).
servico(oncologia).
servico(endocrinologia).
servico(urologia).
```

Figura 3.11: Exemplo da execução do axioma profiServico

3.2.11 Funcionalidades Extra

Após a conclusão das funcionalidades básicas exigidas e de forma a enriquecer o nosso exercício decidimos implementar funcionalidades extra. Assim sendo, passaremos a explicar estas novas funcionalidades, que se baseiam na contagem de entidades no sistema.

Contagem de utentes, serviços, profissionais e instituições

Para a inserção desta funcionalidade foi necessária a criação de um predicado que permitisse calcular o comprimento de uma lista, para tal implementou-se o predicado **comprimento**:

```
comprimento([], 0).
comprimento([_|T], R) :-
comprimento(T, X),
R is 1+X.
```

Para a contagem das caracteristicas pretendidas em cada caso o método utilizado foi semelhante, aplicámos o predicado **comprimento** aos elementos pretendidos, como segue no exemplo de contar o número de serviços existentes em determinada instituição de saúde:

```
numero_Servico(Inst,N):- servicoInst(Inst,K),comprimento(K,Contador),
N is Contador.
```

Um outro exemplo é o da contagem dos profissionais existentes em determinada instituição de saúde em que se utilizou o **findall**. Este mecanismo serve para encontrar todos os profissionais em determinada instituição, com o objetivo de obter o comprimento da lista (contador). O resultado será unificado, **N** passará a assumir esse resultado, como se pode verificar na implementação deste predicado abaixo:

```
numero_Profissionais(Inst,N):-
 findall(Prof,ins_serv_uten_profi(Inst,_,_,Prof),L),
comprimento(L,Contador),
N is Contador.
```

Mostramos de seguida todos os exemplos de execução da nossas funcionalidades extras:

```
| ?- numero_Servico(hospital_braga,N).
N = 2 ?
yes
| ?- numero_Profissionais(hospital_braga,N).
N = 2 ?
yes
| ?- numero_Utentes(hospital_braga,N).
N = 2 ?
yes
| ?- numero_Utentes_Int_Serv(hospital_barcelos,oftamologia,N).
N = 1 ?
yes
| ?- numero_Ins(N).
N = 8 ?
yes
| ?- numero_Servico(N).
N = 10 ?
yes
| ?- numero_Utentes(N).
N = 8 ?
yes
| ?- numero_Profissionais(N).
N = 7 ?
yes
```

Figura 3.12: Exemplo da execução dos extras do exercício

4. Conclusão

Uma vez finalizado o projeto, podemos concluir que os objetivos propostos foram alcançados com sucesso. Construímos uma base de conhecimento de acordo com o que era pretendido no enunciado e implementámos as funcionalidades necessárias. Para além disso, decidimos ainda implementar algumas caracteristicas ou funcionalidades que achamos que seriam interessantes incluir neste sistema. Analisando os resultados obtidos, temos que todas as funcionalidades funcionam de acordo com as nossas expetativas e estão de acordo com a nossa base de conhecimento. Um dos problemas com a realização deste exercício está relacionada com a área creativa do grupo, pois tivemos de inventar/imaginar novas funcionalidades úteis para integrar neste exercício, algo que teve de ser bastante discutido no grupo. Em suma, estamos muito satisfeitos com o trabalho que desenvolvemos.

A. Código implementado

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

%---------------------------------- - - - - - - - - - -  -  -  -  -
% Base de Conhecimento do registo de eventos numa instituição de saúde

%---------------------------------- - - - - - - - - - - - - - - - -
% SICStus PROLOG: Declaracoes iniciais

:- op( 900,xfy,'::' ).
:- set_prolog_flag( discontiguous_warnings,off ).
:- set_prolog_flag( single_var_warnings,off ).
:- set_prolog_flag( unknown,fail ).

/* permitir adicionar a base de conhecimento        */

:-dynamic utente/2.
:-dynamic instituicao/1.
:-dynamic profissional/2.
:-dynamic servico/1.

%---------------------------------- Base de Conhecimento - - - - - - -
% Extensao do predicado instituição(nome).

instituicao( hospital_guimaraes ).
instituicao( hospital_braga ).
instituicao( hospital_barcelos ).
instituicao( hospital_porto ).
instituicao( universitario_coimbra ).
instituicao( hospital_lisboa_norte ).
instituicao( hospital_beatriz_angelo ).
instituicao( hospital_trofa ).

%---------------------------------- - - - - - - - - - -  -  -  -  -
% Extensao do predicado servico(nome).

servico( cardiologia ).
servico( cirugiageral ).
servico(  neurologia ).
```

```
servico(  ortopedia ).
servico(  psiquiatria ).
servico(  oftamologia ).
servico(  ginecologia/obstetricia ).
servico(  oncologia ).
servico(  endocrinologia ).
servico(  urologia ).

%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensao do predicado utente(codigo,nome).

utente(1,jose).
utente(2,carlos).
utente(3,maria).
utente(4,jose).
utente(5,antonio).
utente(6,manuel).
utente(7,joana).
utente(8,joana).

%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensao do predicado profissional(codigo,nome).

profissional(1,marcus).
profissional(2,maria).
profissional(3,jorge).
profissional(4,celia).
profissional(5,carlos).
profissional(6,antonio).
profissional(7,vitoria ).

%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensao do predicado ins_serv_uten_profi(intituicao,servico,Codigo

ins_serv_uten_profi( hospital_braga, cardiologia,1,1 ).
ins_serv_uten_profi( hospital_trofa, cardiologia,1,1 ).
ins_serv_uten_profi( hospital_beatriz_angelo, endocrinologia,2,2 ).
ins_serv_uten_profi( hospital_braga, oncologia,3,3 ).
ins_serv_uten_profi( hospital_porto, cardiologia,4,4 ).
ins_serv_uten_profi( hospital_porto, psiquiatria,5,5 ).
ins_serv_uten_profi( hospital_trofa, urologia,6,6 ).
ins_serv_uten_profi( hospital_barcelos, oftamologia,7,7 ).
ins_serv_uten_profi( hospital_barcelos, endocrinologia,1,2 ).
ins_serv_uten_profi( hospital_guimaraes, neurologia,3,2 ).
ins_serv_uten_profi( hospital_lisboa_norte,  neurologia,4,5).


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

```
% Extensao do predicado concat:(Lista,Lista,Lista)->{V,F}

concat([],L,L).
concat([X|L1],L2,[X|L3]):- concat(L1,L2,L3).

%------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensao do predicado nao
nao( Questao) :-
Questao, !,fail.
nao(Questao).

%------------------------- - - - - - - - - - -  -  -  -  -   -
difList([],_,[]).

difList([H1|T1],L2,[H1|L3]):-
nao(member(H1,L2)), difList(T1,L2,L3).

difList([_|T1],L2,L3):-
difList(T1,L2,L3).


%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensão do predicado eliminarRepetidos: Lista,Resultados -> {V, F}

eliminarRepetidos( [],[] ) .
eliminarRepetidos( [H|T],[H|R] ) :- eliminaElemento( H,T,T2 ),
eliminarRepetidos( T2,R ).


%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensão do predicado eliminaElemento: Elemento,Lista,Resultados ->

eliminaElemento( _,[],[] ) .
eliminaElemento( E,[E|T],T1 ) :- eliminaElemento( E,T,T1 ).
eliminaElemento( E,[H|T],[H|T1] ) :- E\==H,
eliminaElemento( E,T,T1 ).

%------------------------- - - - - - - - - - -  -  -  -  -   -
% 1-Identificar os serviços existentes de uma instituicao
% Extensao do predicado  servicoInst  : Instituicao,[servico]->{V,F}

servicoInst(Inst,Serv):-
findall(K,ins_serv_uten_profi(Inst,K,_,_),Serv).

servicoInst(Inst,[Serv|K]):- ins_serv_uten_profi(Inst,Serv,_,_),
servicoInst(Inst,K).

servicoInst(Inst,[Serv]):- ins_serv_uten_profi(Inst,Serv,_,_).
```

```
%------------------------ - - - - - - - - - - - - - - - - -
% 2-Identificar os utentes de uma instituicao
% Extensao do predicado  utentesInst  : Instituicao,[utentes]->{V,F}


utentesInst(Inst,Uten):-
findall((K,J),(ins_serv_uten_profi(Inst,_,K,_),utente(K,J)),Uten).

utentesInst(Inst,[(Cod,Uten)|K]):- ins_serv_uten_profi(Inst,_,Cod,_),
utente(Cod,Uten),
utentesInst(Inst,K).

utentesInst(Inst,[(Cod,Uten)]):-ins_serv_uten_profi(Inst,_,Cod,_),
utente(Cod,Uten).


%------------------------ - - - - - - - - - - - - - - - - -
% 3-Identificar os utentes de um determinado servico
% Extensao do predicado  servUtente  : servico,[utentes]->{V,F}

servUtente(Serv,Ute):-
findall((K,J),(ins_serv_uten_profi(_,Serv,K,_),utente(K,J)),Ute).

servUtente(Serv,[(Cod,Uten)|K]):- ins_serv_uten_profi(_,Serv,Cod,_),
utente(Cod,Uten),
servUtente(Serv,K).

servUtente(Serv,[(Cod,Uten)]):- ins_serv_uten_profi(_,Serv,Cod,_),
utente(Cod,Uten).


%------------------------ - - - - - - - - - - - - - - - - -
% 4- Identificar os utentes de um determinado servico numa instituicao
% Extensao do predicado utenservinst:(servico, instituicao,[utentes])-

utenServInst(Serv,Inst,Uten):-
findall((K,J),(ins_serv_uten_profi(Inst,Serv,K,_),utente(K,J)),Uten).

utenServInst(Serv,Inst,[(Cod,Uten)|K]):- utente(Cod,Uten),
ins_serv_uten_profi(Inst,Serv,Cod,_),
utenServInst(Serv,Inst,K).

utenServInst(Serv,Inst,[(Cod,Uten)]):-
 ins_serv_uten_profi(Inst,Serv,Cod,_),
utente(Cod,Uten).
```

```
%------------------------ - - - - - - - - - - - - - - - - -
% 5-Identificar as instituições onde seja prestado um dado serviço ou
% Extensao do predicado instServico: ([Serviço],[Instituição])->{V,F}

instServicos([Serv|K],I):-
findall(H,servicoInst(H,[Serv|K]),L),
eliminarRepetidos(L,I).

inst_Servico(S,[I|K]):- servicoInst(I,[S]),
inst_Servico(S,K).

inst_Servico(S,[I]):- servicoInst(I,[S]).

instServicos([S|T],[I|K]):-inst_Servico(S,[I|K]),instServicos(T,[I|K]).
instServicos([S],[I|K]):-inst_Servico(S,[I|K]).
instServicos([S],[I]):-inst_Servico(S,[I]).


%------------------------ - - - - - - - - - - - - - - - - -
% 6-Identificar os serviços que não se podem encontrar numa instituiçã

% Extensao do predicado servicosForaInst : Instituicao,Serviço->{V,F}

todosServicos(L):-
findall(S,servico(S),L).

servicosForaInst(Ins,Serv,todos):-todosServicos(P),
servicoInst(Ins,K),
difList(P,K,Serv).

servicosForaInst(Ins,[Serv|K]):-nao(servicoInst(Ins,[Serv|K])).

%------------------------ - - - - - - - - - - - - - - - - -
% 7-Determinar as instituições onde um profissional presta servico;
% Extensao do predicado profiServico : profissional,[instituições]->{V

profiServico((Cod,Prof),Inst):-
findall(K,(ins_serv_uten_profi(K,_,_,Cod),profissional(Cod,Prof)),
Inst).

profiServico((Cod,Prof),[Inst|K]):-ins_serv_uten_profi(Inst,_,_,Cod),
profissional(Cod,Prof),
profiServico((Cod,Prof),K).

profiServico((Cod,Prof),[Inst]):-ins_serv_uten_profi(Inst,_,_,Cod),
profissional(Cod,Prof).

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

```
% 8-Determinar todas as instituições(ou serviços, ou profissionais) a
% Extensao do predicado instSerProf: utente,instituicao ou servico ou

% Instituicao
instSerProf((Cod,Uten),Inst,inst):-
 findall(K,(ins_serv_uten_profi(K,_,Cod,_),utente(Cod,Uten)),L),
eliminarRepetidos(L,Inst).

instSerProf((Cod,Uten),[Inst|K]):-ins_serv_uten_profi(Inst,_,Cod,_),
utente(Cod,Uten),
instSerProf((Cod,Uten),K).

instSerProf((Cod,Uten),[Inst]):-ins_serv_uten_profi(Inst,_,Cod,_),
utente(Cod,Uten).


%Servico

instSerProf((CodU,Uten),Serv,serv):-
 findall(K,(ins_serv_uten_profi(_,K,CodU,_),utente(CodU,Uten)),L),
eliminarRepetidos(L,Serv).

instSerProf((CodU,Uten),[Serv|K]):-ins_serv_uten_profi(_,Serv,CodU,_),
utente(CodU,Uten),
instSerProf((CodU,Uten),K).

instSerProf((CodU,Uten),[Serv]):-ins_serv_uten_profi(_,Serv,CodU,_),
utente(CodU,Uten).


%Profissional

instSerProf((CodU,Uten),Prof,prof):-
 findall((K,Nome),(ins_serv_uten_profi(_,_,CodU,K),utente(CodU,Uten),
 profissional(K,Nome)),L),
eliminarRepetidos(L,Prof).

instSerProf((CodU,Uten),[(CodP,Prof)|K]):-
ins_serv_uten_profi(_,_,CodU,CodP),
profissional(CodP,Prof),
utente(CodU,Uten),
instSerProf((CodU,Uten),K).

instSerProf((CodU,Uten),[(CodP,Prof)]):-
ins_serv_uten_profi(_,_,CodU,CodP),
profissional(CodP,Prof),
utente(CodU,Uten).
```

```
%------------------------ - - - - - - - - - - - - - - - - -
% 9-Registar utentes, profissionais, servicos ou instituicoes;

% Extensao do predicado regista: utente,instituicao,servico ou profiss

adicionarUtentes(Codigo,Utente):-
 inserirConhecimento(utente(Codigo,Utente)).

adicionarServico(Servico):-
 inserirConhecimento(servico(Servico)).

adicionarProfissional(Codigo,Profissional):-
 inserirConhecimento(profissional(Codigo,Profissional)).

adicionarInstituicao(Nome):- inserirConhecimento(instituicao(Nome)).


%------------------------ - - - - - - - - - - - - - - - - -
% 10-Remover utentes, profissionais, servicos ou instituicoes;

% Extensao do predicado regista: utente,instituicao,servico ou profiss
removerUtentes(Codigo,Utente):- remover(utente(Codigo,Utente)).

removerServico(Servico):- remover(servico(Servico)).

removerProfissional(Codigo,Profissional):-
 remover(profissional(Codigo,Profissional)).

removerInstituicao(Nome):- remover(instituicao(Nome)).


/* ################## MECANISMOS ####################*/

%------------------------------- - - - - - - - - - - - - - - - -
% Extensão do predicado solucoes: X,Teorema,Solucoes -> {V, F}

solucoes(X, Teorema, _) :- Teorema,
assert(temp(X)),
fail.
solucoes(_, _, Solucoes) :- assert(temp(fim)),
construir(Solucoes).

construir(Solucoes) :- retract(temp(X)), !,
(X==fim, !, Solucoes=[];
Solucoes=[X | Resto],
construir(Resto)).

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

```
% Extensão do predicado comprimento: L, R -> {V, F}

comprimento([], 0).
comprimento([_|T], R) :-
comprimento(T, X),
R is 1+X.

%--------------------------------- - - - - - - - - - -  -  -  -  -
% Extensão do predicado que permite a inserção de conhecimento: Termo

inserirConhecimento(Termo) :-
solucoes( Invariante, +Termo::Invariante, Lista),
insercao(Termo),
teste( Lista ).

insercao(Termo) :-
assert(Termo).
insercao(Termo)        :-
retract(Termo), !, fail.

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

%--------------------------------- - - - - - - - - - -  -  -  -  -
% Extensão do predicado que permite a remoção de conhecimento: Termo -

remover(Termo):-
solucoes(Inv,-Termo::Inv,LInv),
remocao(Termo),
teste(LInv).

remocao(Termo):-
retract(Termo).
remocao(Termo):-
assert(Termo),!,fail.


/* #######Invariantes############*/

% Nao deixa introduzir o mesmo conhecimentos
+utente(Codigo,Uten) :: (solucoes( (Codigo,Uten), utente(Codigo,Uten),
 S),comprimento(S,N),N==1).

% Nao deixa inserir utentes com o mesmo codigo
+utente(Codigo,_) :: (solucoes( Uten, utente(Codigo,Uten),
 S),comprimento(S,N),N==1 ).
```

```
% Nao deixa introduzir nomes iguas para as instituicoes
+instituicao(Nome) :: (solucoes( Nome, instituicao(Nome),
 S),comprimento(S,N),N==1 ).

% Nao deixa introduzir nomes iguais para os servico
+servico(Nome) :: (solucoes( Nome, servico(Nome), S),comprimento(S,N),
N==1).

% Nao deixa introduzir o mesmo conhecimentos
+profissional(Codigo,Prof) :: (solucoes( (Codigo,Prof),
 profissional(Codigo,Prof), S),comprimento(S,N),N==1 ).

% Nao deixa inserir profissionais com o mesmo codigo
+profissional(Codigo,_) :: (solucoes( Prof, profissional(Codigo,Prof),
 S),comprimento(S,N),N==1 ).


/* ########## Remover #############*/

% Nao deixar remover um paciente enquanto estiver a ser consultado por
-utente(Codigo,Nome) ::
 (nao(ins_serv_uten_profi(_,_,Codigo,_)),nao(utente(Codigo,Nome))).


% Nao deixar remover um profissional enquanto estiver a ser consultado
-profissional(Codigo,Nome) :: (nao(ins_serv_uten_profi(_,_,_,Codigo)),
nao(profissional(Codigo,Nome))).

% Nao deixar remover um servico com profissionais a trabalhar nele ou
-servico(Nome) :: (nao(ins_serv_uten_profi(_,Nome,_,_))).


% Nao deixar remover uma instituicao com profissionais, utentes, ou se
-instituicao(Nome) :: (nao(ins_serv_uten_profi(Nome,_,_,_))).
```