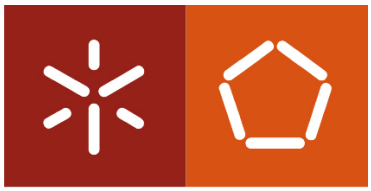

Escola de Engenharia
Universidade do Minho

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, 23 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 nos últimos dois capítulos será apresentada a bibliografia consultada, no quinto capítulo e os anexos no sexto.

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 factores, tal como, número de utente, número de contribuinte, número de cartão de cidadão entre muitos outros mas para a resolução dos critérios mínimos apenas são requeridos os mencionados anteriormente, pois as questões não pretendem incidir nesses campos.

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 determinados 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, mas um profissional é algo mais que isto.

Uma instituição é caracterizada apenas pelo seu nome por forma a simplificar este sistema

visto que os requisitos mínimos não pretendem questionar nada de especial que implique que a instituição tenha de ter algo mais no seu predicado além do seu 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)
- 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 ).
```

Como uma instituição tem a si associado respetivo serviço usámos o predicado **instserv** com a assinatura **instserv (intituicao,servico).**, para relacionar a intituição com o respetivo serviço, como se pode verificar no exemplo abaixo:

```
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 **utentinst(utente,instituicao).**

```
utentinst(1, hospital_barcelos ).
utentinst(3, hospital_porto ).
utentinst(2, hospital_trofa ).
```

3.1.5 Relacionamento entre Entidades

Relacionamos instituição de saúde com os seviços prestados pelo profissional com o utente. Introduzimos na base de conhecimento o predicado **ins_serv_uten_profi(intituicao, servico,Codigo utente, Codigo profissional).**

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

3.2 Funcionalidades Básicas

Após a apresentação da base de conhecimento do exercício torna-se possivel desenvolver alhuns teoremas e axiomas 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 que estejam ligados à instituição pretendida através do axioma *servicoInst(Inst,Serv)*, este que nos devolve uma lista de todos os serviços. 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 do programa para este axioma.

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

Figura 3.1: Exemplo de output do programa

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 :

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):-
```

```
% 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
```

Figura 3.2: Exemplo da execução do axioma utentesInst

```
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 3.3: Exemplo da execução do axioma servUtente

3.2.4 Identificar os utentes de um determinados 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, a intituição e 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 3.4: Exemplo da execução do axioma utenServInst

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

```
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]).
```

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

Figura 3.5: Exemplo da execução do axioma instServicos

3.2.6 Identificar os serviços que não se podem encontrar numa 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])).
```

3.2.7 Determinar as instituições onde um profissional presta serviço

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

```
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 3.6: Exemplo da execução do axioma servicosForaInst

```
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).
```

```
yes
| ?- profiServico((4,celia),X).
X = [hospital_porto] ?
yes
| ?- profiServico((1,marcus),X).
X = [hospital_braga,hospital_trofa] ?
yes
```

Figura 3.7: Exemplo da execução do axioma profiServico

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

```
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).
```

```
yes
| ?- instSerProf((1,jose),X,inst).
X = [hospital_braga,hospital_trofa,hospital_barcelos] ?
yes
| ?- instSerProf((1,jose),X,serv).
X = [cardiologia,endocrinologia] ?
yes
| ?- instSerProf((1,jose),X,prof).
X = [(1,marcus),(2,maria)] ?
yes
```

Figura 3.8: Exemplo da execução do axioma instSerProf

3.2.9 Registar utentes, profissionais, serviços ou instituições

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

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

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

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

```
yes
| ?- adicionarUtentes(10,maria_joão).
yes
| ?- adicionarUtentes(10,maria_joão).
no
| ?- adicionarServico(cirgia_plastica).
yes
| ?- adicionarProfissional(10, celia_figueiredo).
yes
| ?- adicionarInstituicao(hospital_joca).
yes
```

Figura 3.9: Exemplo da execução do axioma adicionar utente, profissionais, serviço ou instituições

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

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

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

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

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

3.2.11 Funcionalidades Extra

Figura 3.10: Exemplo da execução do axioma profiServico

4. Conclusão