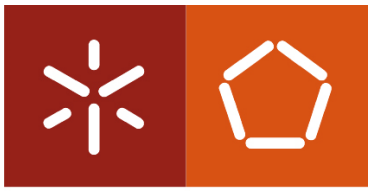

Escola de Engenharia
Universidade do Minho

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

Exercício 2

Extensão à Programação em Lógica e Conhecimento Imperfeito

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, 22 de Abril de 2016

Conteúdo

Lista de Figuras

Resumo

O presente relatório documenta o segundo trabalho prático da Unidade Curricular de Sistemas de Representação Conhecimento e Racocínio. Nesta fase o objetivo será construir um mecanismo de representação de conhecimento com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde. O objetivo é demonstrar as funcionalidades subjacentes à programação em lógica estendida e à representação de conhecimento imperfeito, recorrendo à temática dos valores nulos. Em termos gerais, foi usada a linguagem PROLOG, esta que utiliza um conjunto de fatos, predicados e regras de derivação de lógica e haverá ainda uma interação com o sistema criado para a implementação do caso prático deverá ser desenvolvida em JAVA, com recurso à biblioteca JASPER. 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.

A extensão à programação em lógica surge da necessidade de abandonar conceitos restritos obrigatoriamente associados à programação em lógica abordada no primeiro trabalho prático. A extensão à programação e lógica permite então abandonar o conceito de mundo fechado, que consiste em apenas assumir verdadeiro aquilo que se conhece (aquilo que está presente na base de conhecimento) e também abandonar o conceito de domínio fechado, que consiste em assumir que não existem mais objetos que não aqueles mencionados na base de conhecimento. Assim sendo existem, ao contrário da programação em lógica três valores de verdade, são eles os valores falso, verdadeiro e desconhecido. Para além destas mudanças, a extensão à programação em lógica introduz o conceito de negação forte, sabendo e reconhecendo a existência da negação fraca ou negação por falha na prova (predicado não), a negação forte permite representar conhecimento negativo e é mais completa em relação à anterior pois só declara o seu valor de verdade quando de facto existe prova para tal, já a negação forte, perante falta de conhecimento considera, pelo pressuposto de mundo fechado que se este não existe então o seu valor é falso o que não é de todo correto para situações reais onde o pressuposto mundo fechado não é aplicável. É ainda de notar que, apesar destas alterações na passagem de programação em lógica para a sua extensão os restantes conceitos mantêm-se como é o caso do pressuposto dos nomes únicos.

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 cinco 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 elaboração deste exercício e algumas das ferramentas fundamentais para a realização 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.

2.2 Base de dados vs Representação de Conhecimento

Os sistemas de Bases de Dados (BD's) e os sistemas de representação de conhecimento lidam com aspectos concretos do mundo real e podem-se comparar nos termos em que dividem a utilização da informação [Analide,2002].

Posto isto, apresentamos os pressupostos no qual se baseiam as linguagens de manipulação, tanto das bases de dados como dos sistemas de representação de conhecimento.

2.2.1 Base de Dados

Os pressupostos em que se baseiam as linguagens de manipulação de base de dados seguem o principio de que apenas existe e é válida a informação contida nesta. Assim sendo, apresentámos então, os seguintes pressupostos:

- **Pressuposto do Mundo Fechado** - toda a informação não mencionada na base de dados é considerada falsa;

- **Pressuposto dos Nomes Únicos** – duas constantes diferentes (que definam valores atómicos ou objetos) designam, necessariamente duas entidades diferentes do universo de discurso;

- **Pressuposto do Domínio Fechado** – não existem mais objetos no universo para além daqueles designados por constantes na base de dados.

2.2.2 Sistema de Representação de Conhecimento

Porém, nem sempre se pretende assumir que a informação representada é a única que se pode considerar válida e que as entidades representadas sejam as únicas existentes. Posto isto, apresentámos, então, os seguintes pressupostos:

- **Pressuposto do Mundo Aberto** - podem existir outros factos ou conclusões verdadeiros para além daqueles representados na base de conhecimento;

- **Pressuposto dos Nomes Únicos** – duas constantes diferentes (que definam valores atómicos ou objetos) designam, necessariamente duas entidades diferentes do universo de discurso;

- **Pressuposto do Domínio Aberto** – podem existem mais objetos do universo de discurso para além daqueles designados pelas constantes da base de conhecimento.

2.3 Representação da Informação Incompleta

Muitas vezes presenciámos situações onde a informação existente é insuficiente e/ou incompleta. Perante estas situações, é importante saber representá-las para além do verdadeiro ou falso; temos de ser capazes de demonstrar que a questão ou situação em causa é incompleta ou desconhecida e que o seu valor não pode ser definido no imediato, ao contrário do que acontece, na maior parte das situações, nas mais variadas linguagens de programação.

E aqui que a programação em lógica estendida se apresenta como capaz de resolver estas situações. Através da programação em lógica estendida seremos capazes de representar tais situações desconhecidas. A forma de representar o conhecimento é apresentada a seguir:

- **Verdadeiro**: quando for possivel provar uma questão(Q) na base de conhecimento;

- **Falso**: quando for possível provar a falsidade de uma questão(-Q) na base de conhecimento;

- **Desconhecido**: quando não for possível provar a questãoQ nem a questão-Q

De facto, para cumprir com os pressupostos definidos anteriormente, teremos de definir outro tipo de informação, além da informação positiva e a informação negativa. Esta pode ser representada da seguinte forma:

- **Negação por Falha**: quando não existe nenhuma prova. É representada pelo termo *não*, anteriormente utilizado no *Exercício 1*;

- **Negação Forte ou clássica**: representada pela conectiva (**-**) e que afirma que determinado predicado é falso.

Por fim, atendendo aos requisitos do trabalho, será ainda necessário obdecer às seguintes caracteristicas:

- Representar casos de conhecimento imperfeito, pela utilização de valores nulos de todos os tipos estudados;
- Manipular invariantes que designem restrições de inserção e remoção de conhecimento do sistema;
- Lidar com a problemática da evolução do conhecimento, criando os procedimentos adequados;
- Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas.

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.

3.2 Funcionalidades Básicas

Nesta secção serão apresentados e explicados os métodos usados para a resolução das funcionalidades básicas propostas.

3.2.1 Representar Conhecimento Positivo e Negativo

A representação do conhecimento positivo é feito nos factos, por exemplo:

```
utente(1,gil,12,rua_Braga).
utente(20,gil,12,rua_Braga).
utente(2,carlos,20,rua_Guimaraes).
utente(3,sandro,30,rua_Lisboa).
utente(4,ana,10,rua_Varzim).
utente(5,filipa,15,rua_Coimbra).
utente(6,antonio,13,rua_Pacos).
utente(7,filipe,13,rua_Guimaraes).
utente(8, celia,22, barcelos).
```

O conhecimento negativo é representado de duas formas, ou utilizando a negação por falha ou a negação forte:

Negação por falha:

```
-utente(ID,N,I,M):- nao(utente(ID,N,I,M)),
nao(excecao(utente(ID,N,I,M))).
```

Negação forte:

```
-utente(10,quim,60,mouquim).
```

3.2.2 Representar casos de Conhecimento Imperfeito, pela utilização de valores nulos de todos os tipos estudados

Os valores nulos surgem como uma estratégia para a enumeração de casos, para os quais se pretende fazer a distinção entre situações em que as respostas a questões deverão ser concretizadas como conhecidas (V ou F) ou desconhecidas. Assim existem três tipos de valores nulos:

- Valor nulo do tipo desconhecido;
- Valor nulo do tipo desconhecido de um conjunto dado de valores;
- Valor nulo do tipo não permitido.

Para a representação do utente cuja morada é desconhecida na base de conhecimento, utilizámos os valores nulos do tipo desconhecido:

Para o caso em que o utente pode ter a morada de guimarães ou fafe usámos valores nulos do tipo desconhecido de um conjunto de valores, assim como a idade variar entre 30 e 40 anos:

```
excecao(utente(9,carlos,60,guimaraes)).
excecao(utente(9,carlos,60,fafe)).

excecao(utente(10,lourenco,I,fafe)):- (I>=30,I=<40).
```

Para o tipo não permitido, usámos o exemplo em que a morada de um utente não pode ser conhecida:

```
utente(8,johnny,10,morada_desconhecido).

excecao(utente(ID,NO,I,R)):-
utente(ID,NO,I,morada_desconhecido).

nulo(morada_desconhecido).
```

3.2.3 Manipular Invariantes que designem restrições à inserção e à remoção de conhecimento no sistema

Quando existe a necessidade de remover ou inserir novos factos na base de conhecimento temos de ter em atenção se nada dependes deles, ou se eles já não se encontram presentes, ou se já existem.

Inserção de Conhecimento

As extensões de predicados que permitem a inserção de conhecimento são as seguintes:

```
inserirUtente(ID,NO,I,M):-evolucao(utente(ID,NO,I,M)).
inserirServico(ID,D,I,C):-evolucao(servico(ID,D,I,C)).
inserirConsulta(D,IDU,IDS,C):-evolucao(consulta(D,IDU,IDS,C)).
```

No caso particular de adicionar uma nova consulta esta só pode ser inserida se o utente e o serviço estiverem registados:

```
% Só se pode adicionar uma consulta se o id do utente existir
+consulta(_,ID,_,_) ::
(solucoes(NO,utente(ID,NO,_,_),S),comprimento(S,N),N==1).

% Só se pode adicionar consultar se o id do servico existir
+consulta(_,_,IDS,_) ::
(solucoes(NO,servico(IDS,NO,_,_),S),comprimento(S,N),N==1).
```

No caso de se pretender inserir conhecimento que já está na base de conhecimento, também não é permitido com o uso dos invariantes:

```
% Nao deixa inserir o mesmo conhecimento em relacao as consultas
+consulta(D,IDU,IDS,C) ::
(solucoes((D,IDU,IDS,C),consulta(D,IDU,IDS,C),S),comprimento(S,N),
N==1).

% Nao deixa inserir o mesmo conhecimento em relacao aos servicos
+servico(ID,D,I,C) ::
(solucoes((ID,D,I,C),servico(ID,D,I,C),S),comprimento(S,N),N==1).

% Nao deixa inserir o mesmo conhecimento em relacao aos utentes
+utente(ID,NO,I,M) ::
(solucoes((ID,NO,I,M),utente(ID,NO,I,M),S),comprimento(S,N), N==1).
```

Assim como a repetição do identificador do utente, serviço

```
% Nao deixa inserir o mesmo id em relacao aos utentes
+utente(ID,_,_,_) ::
(solucoes(NO,utente(ID,NO,_,_),S),comprimento(S,N), N==1).

% Nao deixa inserir servicos com o mesmo ID
+servico(ID,_,_,_) ::
(solucoes(D,servico(ID,D,_,_),S),comprimento(S,N),N==1).
```

Remoção de Conhecimento

Para a remoção de conhecimento utilizamos os seguintes predicados:

```
removerUtente(ID,NO,I,M):-remover(utente(ID,NO,I,M)).
removerServico(ID,NO,I,C):-remover(servico(ID,NO,I,C)).
removerConsulta(D,IDU,IDS,C):-remover(consulta(D,IDU,IDS,C)).
```

Para o caso da remoção de conhecimento é necessário não permitir que seja removido um utente que está a ter uma consulta, assim como um serviço que está a ser utilizado, para tal foram desenvolvidos os seguintes invariantes:

```
% Nao deixar remover utentes que estejam nas consultas
-utente(ID,_,_,_) ::
(nao(consulta(_,ID,S,_)),nao(utente(ID,_,_,_))).

% Nao deixar remover um servico que está nas consultas
-servico(ID,X,Y,Z) ::
(nao(consulta(P,L,ID,K)),nao(servico(ID,X,Y,Z))).
```

3.2.4 Lidar com a problemática da evolução do conhecimento, criando procedimentos adequados

O tratamento da problemática da evolução de conhecimento prende-se com o facto de inserir ou remover conhecimento na base de conhecimento, deixar essa base coesa e fidedigna. Para que isso aconteça, não podemos apagar informação que seja dependência de outra, ou inserir informação repetida; assim aquando da alteração da base de conhecimento, teremos que testar se não irá corromper a base do conhecimento. De forma a garantir esta coerência, usámos os invariantes para a inserção e remoção já falados acima, nas funções *solucoes* e *remover*.

- Função de evolução do conhecimento:

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

insercao( Termo ) :-
assert( Termo ).

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

teste( [] ).
teste( [R|LR] ) :-
R,
teste( LR ).
```

- Função de remoção de conhecimento

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

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

3.2.5 Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas

O pretendido neste ponto é que se implemente um mecanismo interpretador de questões, que mediante uma questão obtenhamos uma de três respostas possíveis: *Verdadeiro*, *Falso* ou *Desconhecido*.

O conceito utilizado foi o do **Princípio do Mundo Fechado**, ou seja, toda a informação que não exista mencionada na base de conhecimento é considerada falsa.

O interpretador dada uma questão, caso se possa concluir a veracidade da questão na base de dados a resposta será *'verdadeiro'*, caso esteja presente como uma negação forte na base de dados a resposta será *'falso'*; e para o cao em que não está presente na base de dados como verdadeira, nem negada fortemente terá como resposta *'desconhecido'*. Desta forma conseguimos um programa em lógica estendido.

Mostramos de seguida o predicado demo:

```
demo( Questao,verdadeiro ) :-
Questao.
demo( Questao, falso ) :-
-Questao.
demo( Questao,desconhecido ) :-
nao( Questao ),
nao( -Questao ).
```

3.2.6 Funcionalidades extra

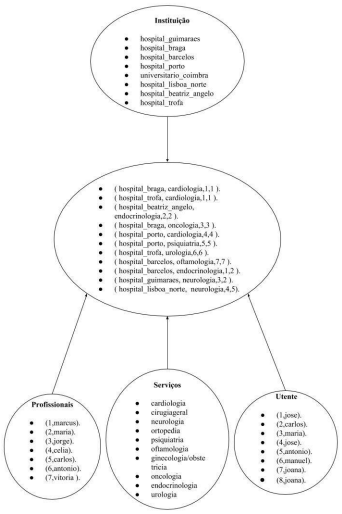

Figura 3.1: Esquema da base de conhecimento

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 ).
```