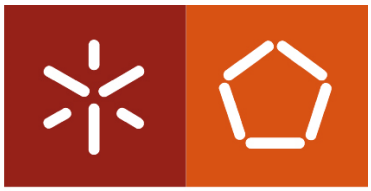

Escola de Engenharia
Universidade do Minho

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, 25 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 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 representação do conhecimento imperfeito. O objetivo final será a construção de uma aplicação capaz de armazenar conhecimento e sobre registo de eventos numa instituição de saúde e a interação com o sistema será desenvolvida em JAVA com recurso à biblioteca JASPER.

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, conhecimento imperfeito, 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.Será também feita uma interface em JAVA com recurso à biblioteca JASPER.

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.

Como tal seguindo o enunciado, em que o tema é caracterizar um universo de discurso na àrea de prestação de cuidados de saúde criaram-se predicados que permitem a inserção de conhecimento dos utentes, serviços e consultas:

- utente(IdUten,Nome,Idade,Morada) ->V,F,D

```
utente(1,gil,12,rua_Braga).
utente(2,carlos,20,rua_Guimaraes).
utente(7,filipe,13,rua_Guimaraes).
utente(18, celia,22, barcelos).
```

- servico(ID,Descricao,Instituicao,Cidade) ->V,F,D

```
servico(1,cardiologia,hospital_Braga,braga).
servico(2,urologia,hospital_Braga,braga).
servico(3,neurologia,hospital_Guimaraes,guimaraes).
```

- consulta(Data,IDUtente,IDServico,Custo) ->V,F,D

```
consulta(data(2010,1,10),1,1,200).
consulta(data(2011,2,11),2,2,20).
consulta(data(2012,3,12),3,3,210).
```

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).
servico(1,cardiologia,hospital_Braga,braga).
servico(2,urologia,hospital_Braga,braga).
servico(3,neurologia,hospital_Guimaraes,guimaraes).
consulta(data(2010,1,10),1,1,200).
consulta(data(2011,2,11),2,2,20).
consulta(data(2012,3,12),3,3,210).
```

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(13,joaquim,60,rua_Fafe).
```

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 é desconhecida:

```
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 caso 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

De modo a tornar a aplicação mais funcional e perspetivel adicionámos algumas funcionalidades que nos pareceram relevantes:

```
1
Existe 11 utentes na base de conhecimento
Prima ENTER para continuar.
```

Figura 3.1: Resposta da aplicação à questão quantos utentes estão inseridos no sistema

```
2
Existe 8 servicos na base de conhecimento
Prima ENTER para continuar.
```

Figura 3.2: Resposta da aplicação à questão qual o nr de serviços registados

```
3
Existe 9 consultas na base de conhecimento
Prima ENTER para continuar.
```

Figura 3.3: Resposta da aplicação à questão qual o nr de serviços registados

```
4
Escreva o nome do utente que deseja procurar
gil
{Idade=12, IdUtente=1, Morada=rua_Braga}
É a solução que pretendia?
n
{Idade=12, IdUtente=20, Morada=rua_Braga}
É a solução que pretendia?
n
Não existe mais utentes com o nome gil
Prima ENTER para continuar.
```

Figura 3.4: Encontrar um utente com um dado nome

```
5
Escreva o nome do servico que deseja procurar
cardiologia
{IdServico=1, Instituicao=hospital_Braga, Cidade=braga}
É a solução que pretendia?
n
{IdServico=10, Instituicao=hospital_Lisboa, Cidade=lisboa}
É a solução que pretendia?
y
Prima ENTER para continuar.
```

Figura 3.5: Informações sobre determinado serviço

```
6
Escreva a data da consulta que deseja procurar
Exemplo data(ano,mes,dia)
data(2010,1,10)
{IdServico=1, IdUtente=1, Custo=200}
É a solução que pretendia?
n
Não existe mais consultas com essa data data(2010,1,10)
Prima ENTER para continuar.
```

Figura 3.6: Procurar por data de consulta

```
7
Escreva a data da consulta que deseja procurar
Exemplo data(ano,mes,dia)
data(2010,1,10)
Escreva o id do utente que quer consultar
1
{IdServico=1, Custo=200}
É a solução que pretendia?
n
Não existe mais consultas na data data(2010,1,10) para o utente 1
Prima ENTER para continuar.
```

Figura 3.7: Procurar por data de consulta e id de utente

```
8
Escreva o id do utente que quer consultar
1
{Idade=12, Morada=rua_Braga, Nome=gil}
Prima ENTER para continuar.
```

Figura 3.8: Procurar por id de utente

```
9
Escreva o id do servico que quer consultar
1
{Descricao=cardiologia, Instituicao=hospital_Braga, Cidade=braga}
Prima ENTER para continuar.
```

Figura 3.9: Procurar por id de serviço

```
10
{IdServico=1, IdUtente=7, Custo=200}
É a solução que pretendia?
n
Não existe mais consultas em que a data seja desconhecida
Prima ENTER para continuar.
```

Figura 3.10: Procurar as consultas que não se sabe a data

```
11
{IdServico=2, IdUtente=1, Data=data(2014,6,15)}
É a solução que pretendia?
n
Não existe mais consultas sem que o custo seja desconhecido
Prima ENTER para continuar.
```

Figura 3.11: Procurar as consultas que não se sabe o custo

```
12
{Idade=10, IdUtente=8, Nome=johnny}
É a solução que pretendia?
n
{Idade=30, IdUtente=11, Nome=filipe}
É a solução que pretendia?
y
Prima ENTER para continuar.
```

Figura 3.12: Procurar os utentes com morada desconhecida

3.2.7 Interação com o Sistema em Java

Foi sugerido que desenvolvessemos uma aplicação Java para tornar o uso do programa mais intuitivo do que o Prolog. Para isso recorremos à biblioteca *Jasper* e criámos um programa com ligação ao *SICStus Prolog*.

Mostrámos de seguida o menu na figura abaixo com as diversas operações possiveis a realizar:

```
run:
**********************************************************************
**************************SRCR**********************************
**********************Exercício 2*****************************
**********************************************************************



1. Número de utentes resgistados.
2. Número de Servicos Registados.
3. Número de consultas.
4. Encontrar informações sobre um utente com um dado nome.
5. Informações sobre um determinado Serviço.
6. Pesquisar consultas por data.
7. Para uma data e um id do utente ver os servicos que ele frequentou e o custo.
8. Dado um id do utente saber quem é esse utente.
9. Dado um id do serviço saber qual é esse servico.
10. Ver as consultas que nao se sabe a data.
11. Ver as consultas que nao se sabe o custo.
12. Ver os utentes em que a morada seja desconhecida.
13. Adicionar utente
14. Adicionar servico
15. Adicionar consulta
16. Remover utente
17. Remover servico
18. Remover consulta
19. Demonstração do utente
20. Demonstração do servico
21. Demonstração da consulta
0. Sair.
```

Figura 3.13: Menu da aplicação

```
13
Introduza o id do utente
100
Introduza o nome do utente
gil
Introduza a idade do utente
3
Introduza a morada do utente
rua_Coimbra
Utente inserido com sucesso
Prima ENTER para continuar.
```

Figura 3.14: Adicionar Utente

```
13
Introduza o id do utente
1
Introduza o nome do utente
f
Introduza a idade do utente
4
Introduza a morada do utente
rt
Já existe o id 1
Prima ENTER para continuar.
```

Figura 3.15: Exemplo de falha na inserção de um utente já existente

```
14
Introduza o id do servico
20
Introduza o nome do servico
clinica_geral
Introduza a instituicao do servico
hospital_Guimaraes
Introduza a cidade da instituicao
guimaraes
Servico inserido com sucesso
Prima ENTER para continuar.
```

Figura 3.16: Adicionar Serviço

```
14
Introduza o id do servico
10
Introduza o nome do servico
clinica_geral
Introduza a instituicao do servico
hospital_Guimaraes
Introduza a cidade da instituicao
guimaraes
Já existe o id 10
Prima ENTER para continuar.
```

Figura 3.17: Exemplo de falha na inserção de um serviço já existente

```
15
Introduza a data da consulta
Exemplo data(ano,mes,dia)
data(2010,2,10)
Introduza o id do utente
1
Introduza servico ao qual o utente frequentou
1
Introduza o custo da consulta
2000
Consulta inserida com sucesso
Prima ENTER para continuar.
```

Figura 3.18: Adicionar Consulta

```
15
Introduza a data da consulta
Exemplo data(ano,mes,dia)
data(2010,1,10)
Introduza o id do utente
1
Introduza servico ao qual o utente frequentou
1
Introduza o custo da consulta
200
Já existe essa consulta
Prima ENTER para continuar.
```

Figura 3.19: Exemplo de falha na inserção de uma consulta já existente

```
16
Introduza o id do utente que quer remover
20
Utente removido com sucesso
Prima ENTER para continuar.
```

Figura 3.20: Remover Utente

```
16
Introduza o id do utente que quer remover
1
Não se pode remover o utente cujo o id é 1
Prima ENTER para continuar.
```

Figura 3.21: Exemplo de falha na remoção de um utente

```
16
Introduza o id do utente que quer remover
100
O id= 100 nao existe
Prima ENTER para continuar.
```

Figura 3.22: Exemplo de falha na remoção de um utente quando este não exite

```
17
Introduza o id do servico que quer remover
10
Servico removido com sucesso
Prima ENTER para continuar.
```

Figura 3.23: Remover Serviço

```
17
Introduza o id do servico que quer remover
1
Não se pode remover o servico cujo o id é 1
Prima ENTER para continuar.
```

Figura 3.24: Exemplo de falha na remoção de um serviço

```
17
Introduza o id do servico que quer remover
100
O id do servico = 100 nao existe
Prima ENTER para continuar.
```

Figura 3.25: Exemplo de falha na remoção de um serviço quando id não existe

```
18
Introduza a data da consulta
Exemplo data(ano,mes,dia)
data(2010,1,10)
Introduza o id do utente
1
Introduza servico ao qual o utente frequentou
1
Introduza o custo da consulta
200
ola
Consulta removida com sucesso
Prima ENTER para continuar.
```

Figura 3.26: Remover Consulta

```
18
Introduza a data da consulta
Exemplo data(ano,mes,dia)
data(2014,2,1)
Introduza o id do utente
100
Introduza servico ao qual o utente frequentou
1
Introduza o custo da consulta
20
Não existe esse utente
Prima ENTER para continuar.
```

Figura 3.27: Exemplo de falha na remoção de uma consulta quando utente não existe

```
18
Introduza a data da consulta
Exemplo data(ano,mes,dia)
data(2015,2,3)
Introduza o id do utente
3
Introduza servico ao qual o utente frequentou
100
Introduza o custo da consulta
1
Não existe esse servico
Prima ENTER para continuar.
```

Figura 3.28: Exemplo de falha na remoção de uma consulta quando serviço não existe

```
18
Introduza a data da consulta
Exemplo data(ano,mes,dia)
data(2015,2,24)
Introduza o id do utente
1
Introduza servico ao qual o utente frequentou
1
Introduza o custo da consulta
20
A consulta não existe
Prima ENTER para continuar.
```

Figura 3.29: Exemplo de falha na remoção de uma consulta quando consulta não existe

```
19
Introduza o id do utente
1
Introduza o nome do utente
gil
Introduza a idade do utente
12
Introduza a morada do utente
rua_Braga
{Questao=verdadeiro}
Prima ENTER para continuar.
```

Figura 3.30: Demo do utente

```
19
Introduza o id do utente
11
Introduza o nome do utente
filipe
Introduza a idade do utente
30
Introduza a morada do utente
fafe
{Questao=desconhecido}
Prima ENTER para continuar.
```

Figura 3.31: Exemplo conhecimento indeterminado

4. Conclusão

A utilização de Programação Lógica Estendida surgiu como o principal desafio neste exercício. Com este exercício conseguimos perceber melhor o conceito de conhecimento imperfeito bem como o significado de cada um dos valores nulos que tínhamos estudado, bem como a utilidade para situações da vida real.

A aplicação dos três tipos de conhecimento imperfeito não levantou problemas e, todos os resultados dos testes e exemplos práticos foram os esperados. Após identificadas as condições de inserção de registo, a construção dos invariantes foi também bastante simples e rápida.

Contudo tivemos algumas dificuldades na criação de exemplos relacionados com a realidade para a criação da base de conhecimento. 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. O facto de ser implementada uma interface em JAVA com recurso à biblioteca JASPER, fez com que a interação com o utilizador ficasse mais simplificada.

A. Código implementado

A.1 PROLOG

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

%--------------------Exercício 2 ------------ - - - - - - - - - - -  -  -
% 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/4.
:-dynamic servico/4.
:-dynamic consulta/4.

% Extensao do predicado utente(IdUten,Nome,Idade,Morada) ->{V,F,D}

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

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

-utente(13,joaquim,60,rua_Fafe).

% Nao sabemos a morada nem nunca vamos deixar saber que se saiba

utente(8,johnny,10,morada_desconhecido).

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

nulo(morada_desconhecido).


+utente(ID,NO,I,R) ::
(solucoes( (ID,NO,I,R),(utente(ID,johnny,I,R),nao(nulo(R))),S ),
comprimento( S,N ), N == 0
).

% Nao sabemos se mora em guimaraes ou em fafe

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


% Nao sabemos a idade ao certo do utente

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

utente(11,filipe,30,morada_desconhecido).
utente(12,celia,22,morada_desconhecido).


%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensao do predicado servico(ID,Descricao,Instituicao,Cidade) ->{V,F,D


servico(1,cardiologia,hospital_Braga,braga).
servico(2,urologia,hospital_Braga,braga).
servico(3,neurologia,hospital_Guimaraes,guimaraes).
servico(4,cirugia_Geral,hospital_Guimaraes,guimaraes).
servico(5,ortopedia,hospital_Coimbra,coimbra).
servico(6,psiquiatria,hospital_Fafe,fafe).
servico(7,ginecologia,hospital_Fafe,fafe).
servico(10,cardiologia,hospital_Lisboa,lisboa).


-servico(ID,D,I,C):- nao(servico(ID,D,I,C)),
nao(excecao(servico(ID,D,I,C))).
```

```
% Nao sabemos se endocrinologia é prestado no hospital_Fafe ou no de hosp

excecao(servico(8,endocrinologia,hospital_Fafe,fafe)).
excecao(servico(8,endocrinologia,hospital_Fafe_Novo ,fafe)).


%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensao do predicado consulta(Data,IDUtente,IDServico,Custo) ->{V,F,D}
% data (ano,mes,dia)


consulta(data(2010,1,10),1,1,200).
consulta(data(2011,2,11),2,2,20).
consulta(data(2012,3,12),3,3,210).
consulta(data(2013,4,13),4,4,220).
consulta(data(2014,5,14),5,5,24).
consulta(data(2015,6,15),6,6,26).


-consulta(D,IDU,IDS,C):- nao(consulta(D,IDU,IDS,C)),
nao(excecao(consulta(D,IDU,IDS,C))).


% Nunca podemos saber a data em que o utente realizou uma consulta de um

consulta(data_desconhecida,7,1,200).
nulo(data_desconhecida).

excecao(consulta(D,IDU,IDS,C)):-consulta(data_desconhecida,IDU,IDS,C).

+consulta(D,IDU,IDS,C) ::
(solucoes( (D,IDU,IDS,C),(consulta(D,7,1,C),nao(nulo(D))),S ),
comprimento( S,N ), N == 0
).


% Nao sabemos quanto ele pagou por um determinado servico

excecao(consulta(D,IDU,IDS,C)):-
consulta(D,IDU,IDS,custo_desconhecido).

consulta(data(2014,6,15),1,2,custo_desconhecido).

% Nao sabemos ao certo quanto ele pegou por um determinado servico
excecao(consulta(data(2015,6,15),1,3,C)):- (C>=10,C=<100).
```

```
% Nunca podemos vir a saber que tipo de servico o utente teve naquela dat

consulta(data(2015,6,15),1,servico_desconhecido,20).
nulo(servico_desconhecido).
excecao(consulta(D,ID,IDS,C)):- consulta(D,ID,servico_desconhecido,C).

+consulta(D,ID,IDS,C) ::
(solucoes((D,ID,IDS,C),(consulta(data(2015,6,15),1,IDS,C),
nao(nulo(IDS))),S),
comprimento(S,N),
N==0).

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

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)).


%---------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado que permite a remocao de conhecimento: Termo -> {

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)).


% Contar o numero de utentes

conta_Utente(Numero):-
(findall((ID,N,I,M),(utente(ID,N,I,M)),S),comprimento(S,Tamanho),
Numero is Tamanho).


% Contar o numero de Servicos

conta_Servico(Numero):-
(findall((ID,N,I,M),servico(ID,N,I,M),S),comprimento(S,Tamanho),
Numero is Tamanho).

% Contar o numero de Consultas

conta_Consulta(Numero):-
(findall((ID,N,I,M),consulta(ID,N,I,M),S),comprimento(S,Tamanho),
Numero is Tamanho).
```

```
%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% 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.
%--------------------------------- - - - - - - - - - -  -  -  -  -  -


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 ).

%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do meta-predicado demo: Questao,Resposta -> {V,F}

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

%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do meta-predicado nao: Questao -> {V,F}

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

```
nao( Questao ).

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

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

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


/* ==================Invariante=========================*/


% 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).

% Nao deixa inserir o mesmo id em relacao aos utentes

+utente(ID,_,_,_) ::
(solucoes(NO,utente(ID,NO,_,_),S),comprimento(S,N), N==1).


/*=======Servicos ==============*/

% 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 servicos com o mesmo ID

+servico(ID,_,_,_) ::
(solucoes(D,servico(ID,D,_,_),S),comprimento(S,N),N==1).

/*=======Consultas ==============*/

% 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).

% So podemos adicionar consultar se o id do utente existir

+consulta(_,ID,_,_) ::
```

```
(solucoes(NO,utente(ID,NO,_,_),S),comprimento(S,N),N==1).

% So podemos adicionar consultar se o id do servico existir


+consulta(_,_,IDS,_) ::
(solucoes(NO,servico(IDS,NO,_,_),S),comprimento(S,N),N==1).


/*===========Remover==================*/

% Nao deixar remover utentes que estejam nas consultas

-utente(ID,_,_,_) :: (nao(consulta(_,ID,S,_)),nao(utente(ID,_,_,_))).

/*=========Servicos =============*/
% Nao deixar remover um servico que esta nas consultas

-servico(ID,X,Y,Z) :: (nao(consulta(P,L,ID,K)),nao(servico(ID,X,Y,Z))).
```

A.2 Java

A.2.1 menu

```
public class Menu {

public Menu(){
int i=0;
System.out.println("*****************************************************
System.out.println("***************************SRCR*********************
System.out.println("************************Exercício 2*****************
System.out.println("*****************************************************
System.out.println("\n\n");
String args[]={ ". Número de utentes resgistados.",
". Número de Servicos Registados.",
". Número de consultas.",
". Encontrar informações sobre um utente com um dado nome.",
". Informações sobre um determinado Serviço.",
". Pesquisar consultas por data.",
". Para um data e um id ver os servicos que ele foi e o custo.",
". Dado um id do utente saber quem é esse utente.",
". Dado um id do serviço saber qual é esse servico.",
". Ver as consultas que nao se sabe a data.",
". Ver as consultas que nao se sabe o custo.",
". Ver os utentes em que a morada seja desconhecida.",
". Adicionar utente",
". Adicionar servico",
```

```
". Adicionar consulta",
". Remover utente",
". Remover servico",
". Remover consulta",
". Demonstração do utente",
". Demonstração do servico",
". Demonstração da consulta",
". Sair."
};

while(i<args.length-1) {

System.out.println(i+1+args[i]);

i++;
}

System.out.println(0+args[i]);
System.out.print("\n");

}
}
```