Ângelo Dias Teixeira – a73312

Bruno Manuel Borlido Arieira – a70565

João Miguel Freitas Palmeira – a73864

Pedro Manuel Barbosa Almeida – a74301

## Resumo

Neste documento encontra-se apresentado o método de resolução do primeiro exercício do trabalho prático de Sistemas de Representação de Conhecimento e Raciocínio.

Apresentam-se as decisões que nos conduziram à resolução do problema, mostrando código e explicando o mesmo.

#### Índice

[1. Introdução 3](#_Toc508508296)

[1.1 Estrutura do Relatório 3](#_Toc508508297)

[2. Predicados 4](#_Toc508508298)

[3. Invariantes 10](#_Toc508508299)

[4. Outros Predicados 12](#_Toc508508300)

[5. Conclusões e Trabalho Futuro 14](#_Toc508508301)

# Introdução

­No presente ano letivo foi proposto aos alunos de Sistemas de Representação de Conhecimento e Raciocínio que desenvolvessem um projeto de programação em lógica que emulasse o funcionamento de uma instituição de saúde. O tema escolhido possibilita a construção de bases de conhecimento de complexidade média e assim permite a demonstração de conhecimentos de programação em lógica e de invariantes que garantam a consistência das bases de conhecimento.

# 1.1 Estrutura do Relatório

O relatório está dividido em três partes que contém os passos de resolução do exercício proposto. A primeira parte é relativa a predicados do problema, a segunda parte é sobre invariantes e a terceira parte contém predicados auxiliares que foram usados para concretizar as funcionalidades pedidas.

# Predicados

Os predicados auxiliares utilizados na resolução podem ser consultados na secção 4 do relatório.

###### Registar utentes, prestadores e cuidados de saúde:

###### Extensão do predicado que permite a evolução do conhecimento: P->{V,F}

evolucao( Termo ) :-

findall( Invariante,+Termo::Invariante,Lista ),

insercao( Termo ),

teste( Lista ).

insercao( Termo ) :-

assert( Termo ).

insercao( Termo ) :-

retract( Termo ),!,fail.

###### Remover utentes, prestadores e cuidados de saúde:

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

remover(Termo) :-

findall( Invariante, -Termo::Invariante, Lista),

teste(Lista),

retract(Termo).

###### Identificar os utentes por critérios de seleção:

###### Extensão do predicado listarUtenteN: Nome,S -> {V,F}

listarUtenteN(Nome,S):-

findall(X,utente(X,Nome,Z,W),S).

###### Extensão do predicado listarUtenteI: Idade,S -> {V,F}

listarUtenteI(Idade,S):- findall(X,utente(X,Y,Idade,W),S).

###### Extensão do predicado listarUtenteM: Morada,S -> {V,F}

listarUtenteM(Morada,S):- findall(X,utente(X,Y,Z,Morada),S).

###### Extensão do predicado listarUtenteNI: Nome,Idade,S -> {V,F}

listarUtenteNI(Nome,Idade,S):-

findall(X,utente(X,Nome,Idade,W),S).

###### Extensão do predicado listarUtenteNM: Nome,Morada,S -> {V,F}

listarUtenteNM(Nome,Morada,S):-

findall(X,utente(X,Nome,Z,Morada),S).

###### Extensão do predicado listarUtenteIM: Idade,Morada,S -> {V,F}

listarUtenteIM(Idade,Morada,S):-

findall(X,utente(X,Y,Idade,Morada),S).

###### Extensão do predicado listarUtenteNIM: Nome,Idade,Morada,S -> {V,F}

listarUtenteNIM(Nome,Idade,MoradaS):-

findall(X,utente(X,Nome,Idade,Morada),S).

###### Extensão do predicado listarUtenteID: IdUtente,S -> {V,F}

listarUtenteID(IdUtente,S):-

findall((IdUtente,Nome,Idade,Morada),utente(IdUtente,Nome,Idade,Morada),S).

###### Identificar as instituições prestadoras de cuidados de saúde:

###### Extensão do predicado listarInstituicoes: S -> {V,F}

listarInstituicoes( S ) :-

findall( W,prestador(X,Y,Z,W),S ).

###### Identificar cuidados de saúde prestados por instituição/cidade/datas:

###### Extensão do predicado prestadorInstituicao: Instituicao,S -> {V,F}

prestadorInstituicao(Instituicao,S) :-

findall( X,prestador(X,Y,Z,Instituicao),S ).

###### Extensão do predicado cuidadosInstituicao: Instituicao,S -> {V,F}

cuidadosInstituicao(I,S) :-

prestadorInstituicao(I,X),

icuidados(X,S).

icuidados([],[]).

icuidados([H|T],S) :-

findall((X,Y,H,W,Z),cuidado(X,Y,H,W,Z),NL),

concatenar(N,NL,S),

icuidados(T,N).

###### Extensão do predicado cuidadosData: Data,S -> {V,F}

cuidadosData(Data,S) :-

findall((Data,B,C,D,E),cuidado(Data,B,C,D,E),S).

###### Identificar os utentes de um prestador/especialidade/instituição:

###### Extensão do predicado utentesDeUmPrestador: IdUtente,S -> {V,F}

###### utentesDeUmPrestador(I,S) :-

###### findall(X,cuidado(A,X,I,D,E),NL).

###### removerRepetidos(NL,S).

**Extensão do predicado prestadorEspecialidade: Especialidade,S -> {V,F}**

prestadorEspecialidade(I,S) :-

findall(X,prestador(X,B,I,D),S).

**Extensão do predicado utentesEspecialidade: Especialidade,S -> {V,F}**

utentesEspecialidade(I,S) :-

prestadorEspecialidade(I,X),

utentesPrestador(X,S).

**Extensão do predicado utentesPrestador: ListaPrestadores,S -> {V,F}**

utentesPrestador([],[]).

utentesPrestador([H|T],S) :-

findall(X,cuidado(A,X,H,D,E),NL),

concatenar(N,NL,S),

utentesPrestador(T,N).

**Extensão do predicado prestadorInstituicao: Instituicao,S -> {V,F}**

prestadorInstituicao(I,S) :-

findall(X,prestador(X,Y,Z,I),S).

**Extensão do predicado utentesInstituicao: Instituicao,S -> {V,F}**

utentesInstituicao(I,S) :-

prestadorInstituicao(I,X),

utentesPrestador(X,S).

**Identificar cuidados de saúde realizados por utente/instituição/prestador:**

**Extensão do predicado cuidadoUtente: IdUtente,S -> {V,F}**

cuidadoUtente(I,S) :-

findall((A,I,C,D,E),cuidado(A,I,C,D,E),S).

**Extensão do predicado cuidadoPrestador: IdPrestador,S -> {V,F}**

cuidadoPrestador(I,S) :-

findall((A,B,I,D,E),cuidado(A,B,I,D,E),S).

**Determinar todas as instituições/prestadores a que um utente já recorreu:**

**Extensão do predicado instituicoesUtente: IdUt,S -> {V,F} 8**

instituicoesUtente(I,S) :-

prestadorUtente(I,X),

iu(X,S).

iu([],[]).

iu([H|T],S) :-

findall(X,prestador(H,B,C,X),NL),

concatenar(N,NL,S),

iu(T,N).

**Extensão do predicado prestadorUtente: IdUt,S -> {V,F}**

prestadorUtente(I,S) :-

findall(X,cuidado(A,I,X,D,E),NL),

removerRepetidos(NL,S).

**Calcular o custo total dos cuidados de saúde por utente/especialidade/prestador/datas:**

**Extensão do predicado custoUtente: IdUt,S -> {V,F}**

custoUtente(I,S) :-

findall(E,cuidado(A,I,C,D,E),NL),

somatorio(NL,S).

**Extensão do predicado prestadorEspecialidade: Especialidade,S -> {V,F}**

prestadorEspecialidade(I,S) :-

findall(X,prestador(X,B,I,D),S).

**Extensão do predicado custoEspecialidade: Especialidade,S -> {V,F}**

custoEspecialidade(I,S) :-

prestadorEspecialidade(I,X),

cEaux(X,Y),

somatorio(Y,S).

cEaux([],[]).

cEaux([H|T],S) :-

findall(X,cuidado(A,B,H,D,X),NL),

concatenar(N,NL,S),

cEaux(T,N).

**Extensao do predicado custoPrestador: Prestador,S -> {V,F}**

custoPrestador(I,S) :-

findall(X,cuidado(A,B,I,D,X),NL),

somatorio(NL,S).

**Extensao do predicado custoData: Data,S -> {V,F}**

custoData(I,S) :-

findall(X,cuidado(I,B,C,D,X),NL),

somatorio(NL,S).

# Invariantes

Para que seja possível adicionar ou remover na nossa base de conhecimento definimos os seguintes predicados:

:- dynamic utente/4.

:- dynamic prestador/4.

:- dynamic cuidado/5.

###### Para regular a inserção de conhecimentos, decidimos utilizar os seguintes invariantes.

###### Invariante Estrutural: não permitir a inserção de número de utente repetido

+utente( X,Y,Z,W ) :: (findall( X,(utente( X,B,C,D )),S ),

comprimento( S,N ),

N == 1

).

###### Invariante Estrutural: não permitir a inserção de número de prestador repetido

+prestador( X,Y,Z,W ) :: (findall(X, (prestador( X,B,C,D )),S ),

comprimento( S,N ),

N == 1

).

###### Invariante Estrutural: não permitir a inserção de cuidado repetido

+cuidado( X,Y,Z,W,W1 ) :: (findall( (X,Y,Z,W,W1),(cuidado( X,Y,Z,W,W1 )),S ),

comprimento( S,N ),

N == 1

).

###### Invariante Estrutural: não permitir a inserção de custo negativo

+cuidado(X,Y,Z,W,W1) :: (findall(W1,cuidado(A,B,C,D,W1),S),

naoNegativo(S)

).

###### Invariante Estrutural: não permitir a inserção de cuidado com utente inexistente

+cuidado( X,Y,Z,W,W1 ) :: (findall( A,(utente( A,B,C,D )),S ),

contem( Y,S)

).

###### Invariante Estrutural: não permitir a inserção de cuidado com prestador inexistente

+cuidado( X,Y,Z,W,W1 ) :: (findall( A,(prestador( A,B,C,D )),S ),

contem( Z,S)

).

# Outros Predicados

##### Outros Predicados:

###### Extensão do predicado teste: [R|LR] -> {V,F}

teste([]).

teste([R|LR]) :-

R,

teste(LR).

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

comprimento([],0).

comprimento([H|T],R) :- comprimento(T,N),

R is N+1.

###### Extensão do predicado naoNegativo: L -> {V,F}

naoNegativo([]). naoNegativo([H|T]) :- H>=0,

naoNegativo(T).

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

contem(H, [H|T]).

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

contem(X, T).

###### Extensão do predicado concatenar: L1,L2,L3 -> {V,F}

*concatenar( [],L,L ).*

*concatenar( [H|T],L2,[H|L] ) :- concatenar(T,L2,L).*

###### Extensão do predicado removerRepetidos: L1,L2 -> {V,F}

removerRepetidos( [],[] ).

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

removerElemento( L,X,TL ),

removerRepetidos( TL,NL ).

###### Extensão do predicado removerElemento: L1, Y, L2 -> {V,F}

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

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

X \== Y,

removerElemento( L,Y,NL ).

###### Extensão do predicado somatorio: L,R -> {V,F}

somatorio( [],0 ).

somatorio( [H|T],R ) :- somatorio(T,N), R is H+N.

# Conclusões e Trabalho Futuro

O principal objetivo deste trabalho era a utilização da linguagem de programação em lógica Prolog e o desenvolvimento de um sistema de representação de conhecimento fiável usando para isso mecanismos e metodologias de forma a permitir retirar informações semanticamente corretas e realizar com elas raciocínios de forma segura.

Assim, podemos dizer que estamos satisfeitos com o trabalho realizado, uma vez que foram cumpridos todos os requisitos e tivemos a oportunidade de aumentar o nosso conhecimento neste tipo de programação o que facilita a realização de futuros projetos.