Universidade do Minho
Escola de Engenharia
Mestrado Integrado em Engenharia Informática

Unidade Curricula de Sistemas de Representação de Conhecimento e Raciocínio

Ano Lectivo de 2018/2019

Programação em Lógica e Invariantes

Henrique Faria, João Marques, José Pereira, Ricardo Petronilho

Março, 2019

SRCR

Data de Recepção	
Responsável	
Avaliação	
Observações	

Programação em Lógica e Invariantes

Henrique Faria, João Marques,

José Pereira, Ricardo Petronilho

Março, 2019

Resumo

O presente trabalho tem como principal objetivo aprimorar a utilização da linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento e construção de raciocínio para a resolução de problemas. Deste modo, pretendemos descrever os esforços efetuados no sentido a atingir o objetivo proposto.

Neste relatório será apresentado o projeto, seguindo-se a apresentação das soluções elaboradas para cada exercício.

Por último é apresentada uma reflexão crítica sobre o trabalho que visa explanar as dificuldades encontradas, bem como os resultados obtidos.

Índice

Índice de Figuras

1. Introdução

No âmbito da Unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio do Mestrado integrado em Engenharia Informática da Universidade do Minho, foi-nos proposta a elaboração de um exercício para o desenvolvimento de uma base de conhecimento para utentes, serviços e consultas.

Para realização da mesma utilizou-se a linguagem de programação **PROLOG**, utilizada na disciplina, que tem como base a lógica matemática. Nesse sentido, permite expressar programas na forma lógica e usar processos de inferência para produzir resultados.

Na verdade, esta é uma linguagem de programação declarativa que difere da semântica imperativa, funcional ou orientada a objetos que foram abordadas durante o curso.

2. Preliminares

Para uma correta resolução do enunciado proposto são necessários os seguintes requerimentos:

- Saber PROLOG;
- Compreender os conceitos de predicado e axioma;
- Compreender corretamente o enunciado;

Para a obtenção destes conhecimentos as aulas teóricas e as práticas foram cruciais.

3. Descrição do trabalho e Análise de resultados

Nesta secção será explicado e exemplificado todo o processo de resolução do trabalho prático realizado. Desde a inserção de conhecimento a remoção da mesma e a predicados realizados sobre a base de conhecimento.

3.1. Inserção de Conhecimento

Para realizar predicados sobre a base de conhecimento foi necessário inserir inicialmente algum conhecimento à base, tendo sido inseridos os seguintes predicados.

- **Utente**

```
%Extenção do predicado utente: IdUt,Nome,Idade,Cidade -> {V,F}
utente(1,joao,22,lisboa).
utente(2,carlos,15,beja).
utente(3,alfredo,45,braga).
utente(4,dario,72,lisboa).
utente(5,quim,25,braga).
utente(6,henrique,20,braga).
utente(7,ricardo,20,braga).
utente(8,bruno,44,aveiro).
utente(9,nuno,33,porto).
utente(10,pedro,70,coimbra).
```

Figure 1: Inserção de conhecimento sobre Utentes

- **Serviço**

```
%Extenção do predicado servico: IdServ,Descrição,Instituição,Cidade -> {V,F}
servico(1,urgencias,hospital_braga,braga).
servico(2,autopsias,hospital_braga,braga).
servico(3,geral,hospital_braga,braga).
servico(4,analises,hospital_trofa_1,braga).
servico(5,radio_grafia,hospital_lisboa,lisboa).
servico(6,dentista,hospital_coimbra,coimbra).
servico(7,ecografia,hospital_aveiro,aveiro).
servico(8,fisioterapia,hospital_trofa_2,porto).
servico(9,urgencias,hospital_aveiro,aveiro).
servico(10,urgencias, hospital_porto,porto).
servico(11,radio_grafia,hospital_braga, braga).
servico(12,tumografia,hospital_braga, braga).
servico(13,dermatologia,hospital_coimbra,coimbra).
servico(14,rinoplastia, hospital_lisboa, lisboa).
servico(15,radio_grafia,hospital_braga,braga).
```

Figure 2: Inserção de conhecimento sobre Serviços

- **Consulta**

```
%Extenção do predicado consulta: Id,Data,IdUt,IdServ,IdMedico, Custo -> {V,F}
consulta(1,data(11,2,2019),1,1,1,15).
consulta(2,data(12,3,2018),2,2,4,20).
consulta(3,data(13,1,2019),3,3,1,25).
consulta(4,data(25,8,2017),4,1,4,15).
consulta(5,data(2,10,2007),5,15,1,30).
consulta(6,data(17,5,2018),9,8,2,30).
```

Figure 3: Inserção de conhecimento sobre Consultas

3.2. Invariantes Referenciais e Estruturais

- **Invariante Estrutural para inserção de Utentes/Serviços**

Apenas é possível adicionar um Utente ou Serviço à base de conhecimento se não existir nenhum Utente ou Serviço, respetivamente, com o mesmo identificador. Os invariantes criados para tal são os seguintes.

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

Figure 4: Invariante Estrutural para inserção de Utente

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

Figure 5: Invariante Estrutural para inserção de Serviços

- **Invariante Estrutural e Referencial para inserção de Consultas**

Para a inserção de uma Consulta são necessários dois invariantes, um invariante estrutural para certificar que não existem dois identificadores iguais e um invariante referencial para garantir que o identificador do Utente, Serviço e Médico existem.

```
+consulta(ID,_,_,_,_,_) :: (solucoes(ID,(consulta(ID,_,_,_,_,_)), S),
                            comprimento(S,N),
                            N == 1
                            ).
```

Figure 6: Invariante Estrutural para inserção de Consultas

```
+consulta(ID,_,IDU,IDS,IM,_) :: (solucoes(ID, (utente(IDU,_,_,_), servico(IDS,_,I,_), medico(IM,_,_,_,I)), S),
                                 comprimento(S,N),
                                 N == 1
                                 ).
```

Figure 7: Invariante Referencial para inserção de Consultas

- **Invariante Referencial para remoção de Utentes/Serviços**

Na remoção de um Utente ou Serviço é necessário verificar se os seus identificadores não estão a ser utilizados em Consultas, caso existam Consultas com o identificador de Utente ou Serviço, estes não podem ser removidos.

```
-utente(Id,_,_,_) :: (solucoes( Id,(consulta(_,_,Id,_,_,_)),S ),
                      comprimento( S,N ),
                      N == 0
                      ).
```

Figure 8: Invariante Referencial para remoção de Utente

```
-servico(Id,_,_,_) :: (solucoes( Id,(consulta(_,_,_,Id,_,_)),S ),
                       comprimento( S,N ),
                       N == 0
                       ).
```

Figure 9: Invariante Referencial para remoção de Serviço

- **Invariante Referencial para remoção de Consultas**

Não são permitidas remoções de consultas da base de conhecimento por serem o predicado principal de todo o exercício, por isso o invariante referencial criado para a remoção de consultas é o seguinte:

```
-consulta(_,_,_,_,_,_) :: (no).
```

Figure 10: Invariante Referencial para remoção de Consulta

3.3. Resolução dos Tópicos apresentados

- **Registar Utentes, Serviços e Consultas**

No registo de utentes, serviços e consultas é necessário verificar sempre os invariantes estruturais e referenciais, para se garantir integridade da base de conhecimento.

Deste modo, foi desenvolvido um predicado denominado **evolucao**, que recebe um termo, que poderá ser, utente (ID,Nome,Idade,Cidade), servico (ID,Descricao,Instituicao,Cidade) ou consulta (ID,Data,IDUtente,IDServiço,IDMedico,Custo), sendo que o objetivo deste predicado é garantir a inserção de novos termos respeitando os diversos invariantes. Para isso é necessário definir o predicado solucoes, insercao e teste.

```
%Extensão do predicado de evolucao: Termo -> {V,F}
evolucao(Termo) :- solucoes(I, +Termo::I,Lista),
                   insercao(Termo),
                   teste(Lista).
```

Figure 11: Predicado evolucao

O predicado solucoes, faz o findall dos invariantes referentes ao termo recebido como argumento e vai colocar os mesmos em S, ou em Lista, no caso da definição do predicado evolucao.

```
%Extensão do predicado de solucoes: Tipo,Condicao,Lista -> {V,F}
%utiliza o findall para encontrar todo o conhecimento que corresponda à condição
solucoes(T,C,S) :- findall(T,C,S).
```

Figure 12: Predicado solucoes

A insercao tem dois possíveis casos, o caso em que se verificam os invariantes, e adiciona-se o termo com o assert. Depois temos o caso em que não se verificam os invariantes e então é necessário remover o termo, logo faz-se retract do mesmo, tal como se pode ver na seguinte figura.

```
%Extensão do predicado de insercao: Termo -> {V,F}
%insere um elemento na base de conhecimento
insercao(T) :- assert(T).
insercao(T) :- retract(T), !, fail.
```

Figure 13: Predicado insercao

Por fim, é necessário o predicado teste, que irá testar os invariantes referentes a este termo, como se pode verificar na figura abaixo, em caso de falha dos invariantes, é necessário remover o Termo adicionado.

```
%Extensão do predicado de teste: Lista -> {V,F}
%testa todos os elementos da Lista
teste([]).
teste([I|L]) :- I, teste(L).
```

Figure 14: Predicado teste

- **Remover Utentes, Serviços e Consultas**

Tal como no registo de novos **termos** no momento da remoção dos mesmos é necessário cumprir **invariantes** de forma a manter a integridade na base de conhecimento.

Para tal é necessário cumprir alguns invariantes mostrados acima pelas Figuras (8, 9 e 10).

Tal como explicado no tópico anterior os invariantes usufruem do predicado **solucoes** uma vez que torna o código mais legível do que o uso do predicado **findAll**. De seguida é verificado que realmente não existem consultas ou serviços uma vez que o comprimento das soluções tem de ser 0.

Após a elaboração dos dois invariantes foi criado o predicado **remocao** tal como a seguinte figura demonstra:

```
%Extensão do predicado de remocao: Termo -> {V,F}
%remove um elemento da base de conhecimento
remocao(T) :- retract(T).
remocao(T) :- assert(T), !, fail.
```

Figure 15: Predicado remocao

Mais á frente é explicado o comportamento e necessidade deste predicado.

Por fim foi especificado o predicado **involucao** que permite a remoção efetivamente de um termo da base de conhecimento. A seguinte figura demonstra o mesmo:

```
%Extensão do predicado de involucao: Termo -> {V,F}
involucao(Termo) :- solucoes(I, -Termo::I,Lista),
                    remocao(Termo),
                    teste(Lista).
```

Figure 16: Predicado involucao

O predicado **remocao** tem um papel fundamental na correção da involução uma vez que caso o termo em questão, que já foi removido a meio da execução da involução, **não cumpra** a lista de invariantes, verificados pelo predicado **teste**, o remocao **regista novamente** o termo já que o mesmo não pode ser removido.

- **Identificar as Instituições prestadoras de Serviços**

Para identificar as instituições prestadoras de serviços é necessário examinar os predicados servicos(IdServ,Descrição,Instituição,Cidade).

Para tal, foi criado o predicado:

```
lerServicos(D) :- findall((C),servico(_,_,C,_),E), setof((C),member((C),E),D).
```

Figure 17: Predicado lerServicos

Neste predicado são primeiro recolhidas todas as ocorrências do nome de qualquer instituição através do predicado findall, sendo depois estas ordenadas por nome tendo as repetições do nome de cada instituição sendo reduzidas a apenas a uma fazendo uso do predicado setof.

- **Identificar Utentes/Serviços/Consultas por critério de seleção**

Para a realização desta parte do trabalho quisemos possibilitar a consulta de um cliente a partir do seu nome, para isso bastou usar o predicado “solucoes” e garantir que os clientes por ele encontrados tinham o mesmo nome que o dado para isso à variável que representa o nome dado foi colocada no campo do nome para que este fosse encontrado.

Soluções análogas foram usadas nos predicados clienteLocal, servicoNome, servicoId, servicoInstituicao, servicoLocal, consultaID, consultaUtente e consultaInstituicao.

```
%Extensão do predicado de clienteNome: Nome,Lista -> {V,F}
clienteNome(Nome,D) :- solucoes((Nome,Idade,Cidade),utente(_,Nome,Idade,Cidade),D).

%Extensão do predicado de clienteLocal: Cidade,Lista -> {V,F}
clienteLocal(Cidade,D) :- solucoes((Nome,Idade,Cidade), utente(_,Nome,Idade,Cidade), D).

%Extensão do predicado de servicoNome: Nome,Lista -> {V,F}
servicoNome(Nome,D) :- solucoes((Id,Nome), servico(Id,Nome,_,_),D).

%Extensão do predicado de servicoId: Id,Lista -> {V,F}
servicoId(Id,D) :- solucoes((Id,Nome), servico(Id,Nome,_,_),D).

%Extensão do predicado de servicoInstituicao: Instituicao,Lista -> {V,F}
servicoInstituicao(Inst,D) :- solucoes((Id,Nome),servico(Id,Nome,Inst,_),D).

%Extensão do predicado de servicoLocal: Cidade,Lista -> {V,F}
servicoLocal(Local,D) :- solucoes((Id,Nome), servico(Id,Nome,_,Local),D).

%Extensão do predicado de consultaCustosSuperiores: CustoMax,Lista -> {V,F}
consultaID(Id,D) :- solucoes((Id,Data,IdUtente,IdServ,IdMed,Custo), consulta(Id,Data,IdUtente,IdServ,IdMed,Custo),D).

%Extensão do predicado de consultaUtente: Utente,Lista -> {V,F}
consultaUtente(IdUtente,D) :- solucoes((Id,Data,IdUtente,IdServ,IdMed,Custo), consulta(Id,Data,IdUtente,IdServ,IdMed,Custo),D).

%Extensão do predicado de consultaInstituicao: Instituicao,Lista -> {V,F}
consultaInstituicao(IdInstituicao,D) :- solucoes((Id,Data,IdUt,IdInstituicao,IdMed,Custo), consulta(Id,Data,IdUt,IdInstituicao,IdMed,Custo),D).
```

Figure 18: Predicados cliente_, servico_ e consulta_

Para possibilitar a consulta por uma data especifica, adicionou-se ao segundo campo do predicado soluço uma restrição na qual é garantido que o dia, o mês e o ano da consulta presente na solução são iguais aos pedidos pelo inquisidor.

```
%Extensão do predicado de consultaData: Data,Lista -> {V,F}
(data(D,M,A),E) :- solucoes((Id,data(D2,M2,A2),IdUt,IdServ,IdMed,Custo), (consulta(Id,data(D2,M2,A2),IdUt,IdServ,IdMed,Custo),D2 == D,M2 == M, A2 == A),E).
```

Figure 19: Predicado consultaData

Decidimos ser também de extrema relevância a possibilidade de consultar quais as consultas ocorridas durante um intervalo de tempo para isso dá-se duas datas e no campo central de soluções a restrição é imposta fazendo uso do predicado entre que verifica se a data da consulta a ser consultada se encontra entre as duas datas pedidas através da redução de cada data a dias e comparação das mesmas tendo, para ser aceite, a data da consulta de ser superior á data de inicio e inferior á de fim do intervalo dado.

```
%Extensão do predicado de consultaDatas: Data,Data,Lista -> {V,F}
consultaDatas(data(D1,M1,A1),data(D2,M2,A2),E) :- solucoes((Id,data(D3,M3,A3),IdUt,IdServ,IdMed,Custo), (consulta(Id,data(D3,M3,A3),IdUt,IdServ,IdMed,Custo),entre(A1*365+M1*30+D1,A2*365+M2*30+D2,A3*365+M3*30+D3)),E).
```

Figure 20: Predicado consultaDatas

Decidimos também implementar a visualização de consultas por preço nas quais são filtradas para o primeiro predicado apresentado as consultas com custos inferiores

a um teto dado pelo inquisidor e no segundo predicado superiores a um custo mínimo decidido também pelo interrogador da nossa base de conhecimentos.

```
%Extensão do predicado de consultaCustosInferiores: CustoMax,Lista -> {V,F}
consultaCustosInferiores(CustoMax,D) :- solucoes((Id,Data,IdUt,IdServ,IdMed,Custo),(consulta(Id,Data,IdUt,IdServ,IdMed,Custo),Custo =< CustoMax),D).

%Extensão do predicado de consultaCustosSuperiores: CustoMax,Lista -> {V,F}
consultaCustosSuperiores(CustoMax,D) :- solucoes((Id,Data,IdUt,IdServ,IdMed,Custo),(consulta(Id,Data,IdUt,IdServ,IdMed,Custo),Custo >= CustoMax),D).
```

Figure 21: Predicados consultaCustosInferiores e consultaCustosSuperiores

- **Identificar Serviços prestados por Instituição/Cidade/Datas/Custo**

Para a identificação dos Serviços prestados Instituição, Cidade, Datas e Custo foram criados 4 predicados, servPrestPorInst, servPrestPorCidade, servPrestPorDatas e servPrestPorCusto.

Tanto servPrestPorInst como servPrestPorCidade apenas procuram diretamente na base de conhecimento por um serviço que contenha a Instituição ou Cidade em causa. Já o servPrestPorDatas e servPrestPorCusto necessitam de procurar as consultas associadas aos respetivos serviços uma vez que as variáveis datas a custo se encontram na consulta.

- **Identificar Utentes de um Serviço/Instituição**

Para a identificar Utentes através de Serviços ou Instituições foram criados dois predicados, o **identificarUtenteServico** e o **identificarUtenteInstituicao**, que identificam Utentes através de Servicos e Instituicoes, respetivamente.

O predicado **identificarUtenteServico** recebe o Identificador de um Servico e filtra todas as consultas que contenham esse Identificador, guardando o Identificador de Utente que aparece nessa mesma consulta para poder filtrar os Utentes para retirar o Nome do Utente.

```
%Extensão do predicado de identificarUtentesServico: Servico,Lista -> {V,F}
identificarUtentesServico(Serv,D) :- solucoes((Uten,Serv,Data),(consulta(_,Data,A,Serv,_,_),utente(A,Uten,_,_)),D).
```

Figure 22: Predicado identificarUtenteServico

O predicado **identificarUtenteInstituicao** é idêntico ao anterior apenas com a diferença que em vez de ser passado o Identificador do Servico é o Nome da Instituicao, é feito um filtro para retirar das Consultas todos os Identificadores de Servico que contenham o Nome da Instituicao passado ao predicado e nessas mesmas Consultas retirar o Identificador de Utente para retirar os respetivos Nomes.

```
%Extensão do predicado de identificarUtentesInstituicao: Cidade,Lista -> {V,F}
identificarUtentesInstituicao(Inst,D) :- solucoes((Uten,Inst,Data),(consulta(_,Data,A,G,_,_),servico(G,_,Inst,_),utente(A,Uten,_,_)),D).
```

Figure 23: Predicado identificarUtenteInstituicao

- **Identificar Serviços realizados por Utente/Instituição/Cidade**

Para a identificação dos Serviços realizados por Utente, Instituição ou Cidade foram criados três predicados, **identificarServicosUtente**, **identificarServicosInstituicao** e **identificarServicosCidade**.

O predicado **identificarServicosUtente** filtra todas as consultas através do Identificador do Utente guardando todos os Identificadores de Utentes dessas consultas que contêm o Identificador do Servico, guardando o Identificador do Utente, o Nome do Servico e a Data da sua realização.

```
%Extensão do predicado de identificarServicosUtente: Utente,Lista -> {V,F}
identificarServicosUtente(Uten,D) :- solucoes((Serv,Uten,Data),(consulta(_,Data,Uten,A,_,_),servico(A,Serv,_,_)),D).
```

Figure 24: Predicado identificarServicosUtente

Para o predicado **identificarServicosInstituicao** a diferença em relação ao anterior é que em vez do Identificador do Utente é passado ao predicado o Nome da Instituicao. O filtro para obter os Servicos consiste em ver todas as Consultas realizadas e através do Identificador do Servico guardado em Consulta filtramos esses Servicos pelo Nome de Instituicao passado ao predicado e guardamos os Nomes desses Servicos que passem no filtro.

```
%Extensão do predicado de identificarServicosInstituicao: Instituicao,Lista -> {V,F}
identificarServicosInstituicao(Inst,D) :- solucoes((Serv,Inst,Data),(consulta(_,Data,_,A,_,_),servico(A,Serv,Inst,_)),D).
```

Figure 25: Predicado identificarServicosInstituicao

O predicado **identificarServicosCidade** é igual ao **identifiarServicosInstituicao** mudando apenas o parâmetro passado ao predicado, em vez do Nome da Instituicao é o Nome da Cidade. O filtro é realizado do mesmo modo.

```
%Extensão do predicado de identificarServicosCidade: Cidade,Lista -> {V,F}
identificarServicosCidade(Loca,D) :- solucoes((Serv,Loca,Data),(consulta(_,Data,_,A,_,_),servico(A,Serv,_,Loca)),D).
```

Figure 26: Predicado identificarServicosCidade

- **Calcular o custo total dos cuidados de saúde por utente, serviço, instituição ou data.**

A elaboração dos predicados para calcular o custo total de consultas é feito através de filtros como utente, servico, instituição, data.

O predicado **custoCuidadosPorUtente**, responsável por calcular os custos por utente, recebe o Id do mesmo, e filtra as consultas que contenham o Id de utente igual ao recebido como argumento, e guarda as mesmas em S.

De seguida, invoca-se o predicado **sum**, que calcula a soma de todos os elementos de S, ou seja, os custos, tal como se pode ver na seguinte figura.

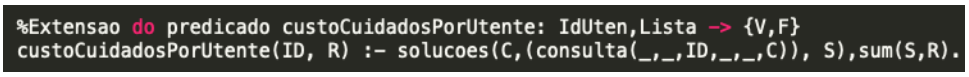

Figure 27: Predicado custoCuidadoPorUtente

Do mesmo modo, o predicado **custoCuidadosPorServico**, calcula os custos por serviço, recebe o **Id** do **serviço**, e filtra as consultas que contenham o mesmo Id de **serviço**, por fim, tal como no predicado anterior, soma-se todos os elementos que estão em S.

Figure 28: Predicado custoCuidadosPorServico

O predicado custoCuidadosPorInstituicao, calcula os custos por instituição, ou seja, recebe como argumento o Id da mesma, no entanto, a informação da instituição está presente no predicado serviço, logo, filtra-se os serviços com esta instituição e as consultas com o Id dos respetivos serviços, e no final soma-se os valores colocados em S, como se pode observar na figura seguinte.

Figure 29: Predicado custoCuidadosPorInstituicao

Por fim, e não menos importante, o predicado custoCuidadosPorData, que calcula os custos em consultas numa determinada data. Desta forma, recebe-se como argumento uma data e verifica-se em todas as consultas, quais foram realizadas na mesma. Por fim, tal como nos predicados anteriores, faz-se a soma dos elementos de S, para determinação dos custos, tal como se pode verificar na figura abaixo.

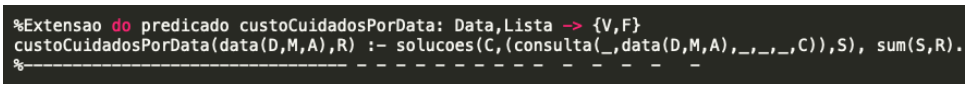

Figure 30: Predicado custoCuidadosPorData

3.4. Funcionalidades Extra

Para além dos predicados pedidos para a realização do trabalho prático o grupo dedicou-se a extender a sua base de conhecimento adicionando o predicado **Medico**. Do mesmo modo que foi realizado com os predicados **Utente** e **Servico** foram criados invariantes estruturais e referenciais para a inserção e remoção de Medicos da base de conhecimento, seguinte estes o mesmo processo dos outros predicados.

```
%Extenção do predicado medico: Id,Nome, Idade, Especialidade, Instituição -> {V,F}
medico(1,jose,30,pediatria,hospital_braga).
medico(2,joao,35,radiologia,hospital_porto).
medico(3,pedro,50,ginecologista,hospital_coimbra).
medico(4,joaquim, 44, cirugia, hospital_braga).
```

Figure 31: Inserção de conhecimento sobre Medicos

```
+medico(ID,_,_,_,_) :: (solucoes(ID,medico(ID,_,_,_,_),S),
                        comprimento(S,N),
                        N == 1
                        ).
```

Figure 32: Invariante estrutural para a inserção de Medicos

```
-medico(Id,_,_,_) :: (solucoes( Id,(consulta(_,_,_,_,Id,_)),S ),
                      comprimento( S,N ),
                      N == 0
                      ).
```

Figure 33: Invariante referencial para a remoção de Medicos

Conclusões e Sugestões

No final do trabalho realizado o grupo faz uma avaliação positiva do mesmo. Apesar de todos os as questões colocadas terem sido resolvidas e o grupo ter criado mais algumas questões extra poderiam ter sido geradas mais de maior ou menor complexidade.

A base de conhecimento inicial desenvolvida é estática podendo no futuro mudar-se a forma de inserção e remoção de conhecimento, em vez de existir apenas enquanto o programa funciona, esta poderá ser escrita para um ficheiro armazenando assim todo o conhecimento inicial não alterado e todo o novo conhecimento inserido.