UNIVERSIDADE DO MINHO

Exercício nº 2

Programação em lógica estendida e Conhecimento Imperfeito

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Ano Letivo 2016/2017

2º Semestre

A73674 – Alexandre Lopes Mandim da Silva

A74219 – Hugo Alves Carvalho

A74260 – Luís Miguel da Cunha Lima

Braga,

9 de Abril de 2017

Resumo

O trabalho apresentado neste relatório foi desenvolvido no âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio com o objetivo de desenvolver competências na utilização da linguagem de programação em lógica PROLOG.

O exercício consiste num sistema de representação de conhecimento imperfeito com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde pela realização de serviços de atos médicos, recorrendo á utilização de valores nulos e da criação de mecanismos de raciocínio adequados.

Ao longo deste relatório iremos explicar todo o processo de construção de soluções indicadas aos desafios propostos no enunciado do segundo exercício prático.

Índice

1. Introdução

Este relatório foi realizado no âmbito do trabalho nº2, proposto na unidade curricular de SRCR (Sistemas de Representação de Conhecimento e Raciocínio) do curso MIEI (Mestrado Integrado em Engenharia Informática) da Universidade do Minho.

O principal objetivo deste exercício é, utilizando o PROLOG, aprofundar e refinar os nossos conhecimentos da programação lógica recorrendo á utilização de valores nulos e da criação de mecanismos de raciocínio adequados. O presente relatório relata a descrição destes mesmos exercícios, a maneira de como foram abordados e finalmente, como o grupo os resolveu.

O tema dos exercícios propostos é a prestação de cuidados de saúde pela realização de serviços de atos médicos. Posto isto, é necessário representar o conhecimento sobre utentes (identificação, nome, idade e uma morada), cuidados prestados (identificação do serviço, uma descrição, instituição e cidade) e, por fim, representar conhecimento que relaciona os utentes com os cuidados prestados denominado por atos médicos, em que cada ato é caracterizado por data, utente, serviço e o custo do ato.

Com o conhecimento que queremos retratar bem definido, como para o primeiro exercício, foram criados predicados para representar esse mesmo conhecimento:

- utente: IdU,Nome,Idade,Morada -> {V,F,D}
- cuidadoPrestado: IdC,Descricao,Instituicao,Cidade -> {V,F,D}
- atoMedico: Data,IdU,IdC,Custo -> {V,F,D}

Por fim, serão implementadas funcionalidades que complementem o problema proposto pelo enunciado do segundo exercício. Esse conjunto de funcionalidades e respetiva solução das mesmas serão ilustradas no capitulo 3.

2. Preliminares

De forma a conseguirmos realizar o trabalho proposto foi necessário, através das aulas desta Unidade Curricular, compreender os conceitos teóricos e métodos de aplicação dos mesmos, onde fazemos aqui uma pequena introdução teórica á linguagem lógica.

A programação em lógica segue os seguintes pressupostos:

- **Pressuposto dos Nomes Únicos (PNU)** – qualquer constante com o nome idêntico representa o mesmo objeto, como constantes diferentes representam obrigatoriamente objetos distintos.
- **Pressuposto do Mundo Fechado (PMF)** – que aceita que somente é verdadeiro aquilo sobre o que se possui conhecimento, isto é, caso não exista na base de conhecimento alguma informação sobre os peixes poderem andar em terra, neste caso é considerado que os peixes não andam na terra. Tal facto desobedece a lógica do mundo real pois, caso não haja conhecimento não indica que tal seja falso será apenas desconhecido!
- **Pressuposto do Domínio Fechado (PDF)** – defende que somente existem os objetos que existem na base de conhecimento, ou seja, por exemplo um registo de jogadores de futebol, caso o Hugo não esteja registado então o Hugo não é verdadeiro, não existe. Tal como o antecedente este pressuposto não pode ser ponderado para situações gerais.

Procurando retificar as falhas referidas em PMF e PDF a programação em lógica apoia-se então no pressuposto de mundo e domínio abertos conservando, porém, o pressuposto dos nomes únicos. Esta extensão introduz ainda um novo tipo de negação, a negação forte, esta negação só validará a pergunta em verdadeiro ou falso se e só se existirem provas para isso.

Neste tipo de programação existe três valores de verdade: falso, verdadeiro ou desconhecido. O valor desconhecido representa informação incompleta podendo ser:

- **Incerto** – desconhece se o valor do conhecimento.
- **Impreciso** – o valor sabe-se estar entre uma gama de valores possíveis.
- **Interdito** – não é permitido o acesso ao valor.

Todos estes conceitos irão ser aplicados neste exercício prático.

3. Descrição do Trabalho

Como já explicado anteriormente, existem três predicados que representam o conhecimento relativo aos utentes, cuidados prestados e atos médicos que faz a associação entre estes últimos dois:

utente: IdU,Nome,Idade,Morada -> {V,F,D}

Este é um predicado que permite guardar os dados relativos a um utente numa instituição. Como apresentado em cima, consideramos o "IdU" (código de diferenciação entre os vários utentes), "Nome" (nome do utente), "Idade" (idade do utente) e "Morada" (morada do utente) como variáveis para representar este predicado.

cuidadoPrestado: IdC,Descricao,Instituicao,Cidade -> {V,F,D}

Este predicado permite guardar os dados referentes a um cuidado prestado numa instituição de saúde. As variáveis que representam este predicado, como já apresentadas em cima, são um "IdC" (código do cuidado), "Descriçao" (nome do cuidado), "Instituiçao" (nome da instituição onde é prestado este cuidado) e "cidade" (nome da cidade da instituição prestadora deste cuidado).

É de referir que estes dois predicados são referenciados pelos seus códigos.

atoMedico: Data,IdU,IdC,Custo -> {V,F,D}

Este predicado permite guardar os dados relativos a um ato médico numa instituição, considerando as variáveis "Data" (data da realização do ato médico), "IdU" (código do utente que realizou o ato médico), "IdC" (código do cuidado prestado) e ainda custo (custo do ato médico).

Posto isto, é necessário dar solução a um conjunto de funcionalidades propostas no enunciado. As respetivas funcionalidades são descritas em seguida.

3.1. Representar conhecimento positivo e negativo

```
atoMedico(01-01-2017, 1, 6, 15).
atoMedico(13-01-2017, 3, 4, 30).
atoMedico(14-01-2017, 2, 7, 8).
atoMedico(20-01-2017, 4, 2, 20).
```

Figura 1 – Exemplo de representação de conhecimento positivo

Tendo como base o conhecimento positivo já inserido pela realização do primeiro exercício (exemplificado em cima), explicamos agora o predicado que permite representar a negação do conhecimento através da inserção do caractere '- 'no principio de cada predicado, permitindo assim representar o conhecimento negativo.

```
-utente(11,mario,30,braga).
```

Figura 2- Exemplo de conhecimento negativo

```
%-----------------------------------------------------
% Representaçao de conhecimento negativo, positivo

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

-cuidadoPrestado(Id,D,I,C) :-
                    nao(cuidadoPrestado(Id,D,I,C)),
                    nao(excecao(cuidadoPrestado(Id,D,I,C))).

-atoMedico(D,IdU,Id,C) :-
                    nao(atoMedico(D,IdU,Id,C)),
                    nao(excecao(atoMedico(D,IdU,Id,C))).
```

Figura 2 - Representação de conhecimento negativo

Para inserir este conhecimento foi indispensável desenvolver predicados que possibilitem deduzir falsidades de acordo com o conhecimento negativo apresentado. Assim, consideramos que tudo aquilo que não é considerado verdadeiro nem uma exceção à verdade deve ser considerado falso.

Para o auxilio na representação de conhecimento negativo, introduzimos dois predicados: "não" e "exceção". O predicado "não" caracteriza-se por ser uma negação por falha na prova. Caso não se consiga provar o termo, ou seja, se este não estiver presente na base de conhecimento, então como consequência o resultado da questão é verdadeiro, podendo concluir que este termo não existe. O predicado "exceção" aparece para ser possível representar o conhecimento imperfeito que será apresentado nos próximos tópicos.

3.2. Representar casos de conhecimento imperfeito

3.2.1. Conhecimento imperfeito incerto

Para representar conhecimento imperfeito incerto, o grupo recorreu a alguns exemplos que correspondem a este tipo de conhecimento.

Como tal, podemos considerar o exemplo em que se sabe que existe um utente com id nº13, 55 anos de idade, com morada em Vizela, mas não se sabe o seu nome. Neste caso, é necessário usar "xpto2" no lugar do nome do utente, e considerar que sempre que aparece utente com este nome, estamos perante uma exceção.

```
utente(13,xpto2,55,vizela).
excecao( utente(Id,N,I,M) ) :-
    utente(Id,xpto2,I,M).
```

Figura 3 – Exemplo de conhecimento imperfeito incerto: não se sabe o nome do utente de id 13, com 55 anos e residente em Vizela

Outro exemplo que representamos é a possibilidade de ser conhecido um utente com id nº11, chamado Mário e com 30 anos de idade, mas em que não se sabe a sua morada. Tal como foi explicado no exemplo anterior, neste caso utilizamos "xpto1" no lugar da morada.

No entanto, consideremos que apesar de ser desconhecida a morada do utente, sabe-se também que este não reside em Braga. Neste caso, foi necessário representar conhecimento negativo relativo à cidade de Braga, ficando assim esta como falsa e todas as restantes como desconhecidas.

```
utente(11,mario,30,xpto1).
excecao( utente(Id,N,I,M) ) :-
    utente(Id,N,I,xpto1).

% representação de conhecimento negativo

-utente(11,mario,30,braga).
```

Figura 4 – Exemplo de conhecimento imperfeito incerto: não se sabe a morada do utente id11, com nome Mario e 30 anos de idade. No entanto sabe-se que o utente não reside em Braga

3.2.2. Conhecimento imperfeito impreciso

Tal como no tipo anterior, podemos representar conhecimento imperfeito impreciso quando não temos a certeza sobre um determinado termo.

Com este objetivo, podemos considerar o caso em que se sabe que existe um utente com id nº12, chamado Felisberto e residente em Aveiro, mas não se sabe se a idade deste é de 20 ou 21 anos. Como sabemos que o valor da idade toma um destes dois valores, estamos perante conhecimento impreciso, ou seja, é necessário representar exceções.

```
excecao( utente(12,felisberto,20,aveiro) ).
excecao( utente(12,felisberto,21,aveiro) ).
```

Figura 5 - Exemplo de conhecimento imperfeito impreciso: não se sabe se o utente Felisberto (id12 e residente em Aveiro) tem 20 ou 21 anos

3.2.3. Conhecimento imperfeito interdito

No caso do conhecimento imperfeito interdito temos que considerar situações em que nunca se poderá saber o valor de um determinado termo.

Para exemplificar, consideramos o caso em que existe um atoMedico associado ao cliente 1 e ao serviço 4 no dia 29-03-2017 e em que nunca se poderá saber o valor que o cliente pagou.

Neste caso, para além se representarmos a exceção, é necessário representar “xpto4” como nulo e impedir que seja adicionado conhecimento sobre este ato médico com um custo não nulo. Para isso, é desenvolvido um invariante que restringe esta situação.

```
atoMedico(29-03-2017, 1, 4, xpto4).
excecao( atoMedico(D,U,S,C) ) :-
            atoMedico(D,U,S,xpto4).
nulo(xpto4).
+atoMedico( D,U,S,C ) :: (solucoes((D,U,S), (atoMedico(29-03-2017, 1, 4, Cs),nao(nulo(Cs))),Servs ),
                          comprimento( Servs,N ),
                          N == 0).
```

Figura 6 - Exemplo de conhecimento imperfeito interdito: nunca será possível saber qual o preço que o cliente 1 pagou pelo serviço 4 no dia 29-03-2017

3.3. Invariantes de inserção e remoção de conhecimento do sistema

Para além do invariante referido no tópico 3.2.3, é necessário ter em consideração outras situações essências para o correto funcionamento do sistema.

Deste modo, foi necessário manipular alguns invariantes com restrições relativas ao processo de inserir ou remover conhecimento.

Por exemplo, não deve ser permitido inserir um utente ou um cuidado prestado quando o esse id que estamos a inserir já está associado a algum utente/cuidadoPrestado existente na base de conhecimento.

```
% Invariante que não permite a inserção de conhecimento de um utente com um id já existente

+utente(Id, N, I, M) :: (solucoes((Id, N), utente(Id, X, Y, Z), S),
                         comprimento(S, L),
                         L == 1).
```

Figura 7 - Invariante que não permite a inserção de um utente com um id já existente na base de conhecimento

```
% Invariante que não permite a inserção de conhecimento de um cuidadoPrestado com um id já existente

+cuidadoPrestado(Id,D,I,C) :: (solucoes((Id, D), cuidadoPrestado(Id, X, Y, Z), S),
                               comprimento(S, L),
                               L == 1).
```

Figura 8 - Invariante que não permite a inserção de um utente com o id já existente na base de conhecimento

Estes invariantes procuram as soluções que obedecem a um determinado termo, e coloca-as numa lista. Assim, ao procurar todas a soluções para um determinado id utente/cuidado prestado, só deve ser possível inserir conhecimento se o tamanho da lista com esse elemento for 1.

Por outro lado, apenas é possível inserir conhecimento sobre atos médicos quando estes correspondem a um utente e a um cuidado médico existentes na base de conhecimento.

```
% Invariante que não permite a inserção de conhecimento de um atoMedico quando o id do utente/cuidadoPrestado
% não existem na base de conhecimento

+atoMedico(D,U,S,C) :: (solucoes((S, Xs), cuidadoPrestado(S, Xs, Ys, Zs), Servs),
                        comprimento(Servs, Ls),
                        Ls == 1,
                        solucoes((U, Xu), utente(U, Xu, Yu, Zu), Uts),
                        comprimento(Uts, Lu),
                        Lu == 1).
```

Figura 9 - Invariante que não permite a inserção de conhecimento de um atoMedico quando o id utente/cuidadoPrestado não existem na base de conhecimento.

A remoção de conhecimento deve ter em consideração alguns cuidados, como por exemplo não permitir a remoção de um utente ou serviço que estejam associados a atos médicos, uma vez que resulta numa perca de conhecimento. Para além disso, só deve ser permitido remover conhecimento se este de facto existir.

Com o objetivo de implementar um correto predicado de remoção de conhecimento, foi necessário implementar invariantes, que são utilizados no predicado “findall”.

```
%-----------------------------------------------------------------------------------------------------
% Invariante que não permite a remocao de conhecimento de um utente não presente na base de conhecimento
% e com id associado a atoMedico

-utente(Id,N,I,M) :: (solucoes((Id), utente(Id,N,I,M), Uts),
                      comprimento(Uts, Lu),
                      Lu == 1,
                      solucoes((Id), atoMedico(X, Id, Y, Z), R),
                      comprimento(R, L),
                      L == 0).
```

Figura 10 - Invariante que não permite a remoção de conhecimento de um utente não presente na base de conhecimento ou com o id associado atos médicos

```
%-----------------------------------------------------------------------------------------------------
% Invariante que não permite a remocao de conhecimento de um cuidadoPrestado não presente na base de
% conhecimento e com o id associado a atoMedico

-cuidadoPrestado(Id, D, I, C) :: (solucoes((Id), cuidadoPrestado(Id,D,I,C), Servs),
                                  comprimento(Servs, Lc),
                                  Lc == 1,
                                  solucoes((Id), atoMedico(X, Y, Id, Z), R),
                                  comprimento(R, L),
                                  L == 0).
```

Figura 11 - Invariante que não permite a remoção de conhecimento de um cuidado prestado não presente na base de conhecimento ou com o id associado atos médicos

Por outro lado, é também necessário implementar um invariante que não permita que seja removido conhecimento sobre um ato médico que não exista, pela mesma situação anteriormente referida.

```
%-----------------------------------------------------------------------------------------------------
% Invariante que não permite a remocao de conhecimento de um atoMedico não presente na base de conhecimento

-atoMedico(D,U,S,C) :: (solucoes((D,U,S,C), atoMedico(D,U,S,C), A),
                        comprimento(A, L),
                        L == 1).
```

Figura 12 - Invariante que não permite a remoção de conhecimento de um ato médico não existente na base de conhecimento

3.4. Evolução de conhecimento

De forma a que seja possível evoluir o conhecimento desconhecido ou falso, foi necessário desenvolver um predicado "evolução" que permite a atualização do conhecimento existe.

Tanto para os utentes como também para os cuidados prestados e atos médicos, o raciocínio foi o mesmo. Em primeiro lugar, verificamos se o conhecimento que estamos a tentar atualizar é falso ou desconhecido, de acordo com o sistema de inferência "demo" (explicado no tópico 3.5).

No caso deste ser desconhecido, utilizados o "findall" para procurar todos os utentes com esse Id que queremos inserir, e através do predicado "remocaoL" retiramos o conhecimento desconhecido anteriormente existente. Este predicado é explicado no tópico dos predicados auxiliares.

Por outro lado, se o conhecimento foi falso, basta recorrer ao predicado "registar" - também descrito nos predicados auxiliares – para inserir o conhecimento.

```
evolucao(utente(Id,Nome,Idade,Morada)):-
        demo(utente(Id,Nome,Idade,Morada),desconhecido),
        findall(utente(Id,N,I,M), utente(Id,N,I,M),L),
        remocaoL(utente(Id,Nome,Idade,Morada),L).

evolucao(utente(Id,Nome,Idade,Morada)):-
        demo(utente(Id,Nome,Idade,Morada),falso),
        registar(utente(Id,Nome,Idade,Morada)).
```

Figura 13 – Evolução de conhecimento desconhecido/falso sobre utente

```
evolucao(cuidadoPrestado( Id,Desc,Inst,Cidade ) ):-
        demo(cuidadoPrestado(Id,Desc,Inst,Cidade) ,desconhecido),
        findall(cuidadoPrestado(Id,E,I,C), cuidadoPrestado(Id,E,I,C),L),
        remocaoL(cuidadoPrestado( Id,Desc,Inst,Cidade ),L).

evolucao(cuidadoPrestado( Id,Desc,Inst,Cidade ) ):-
        demo(cuidadoPrestado(Id,Desc,Inst,Cidade),falso),
        evolucao(cuidadoPrestado( X,Desc,Inst,Cidade)).
```

Figura 14 - Evolução de conhecimento desconhecido/falso sobre cuidadoPrestado

```
evolucao(atoMedico(Data,IdU, IdS,Custo)):-
        demo(atoMedico(Data,IdU, IdS,Custo),desconhecido),
        findall(atoMedico(Data,IdU, IdS,Custo), atoMedico(Data,IdU, IdS,Custo),L),
        remocaoL(atoMedico(Data,IdU, IdS,Custo),L).

evolucao(atoMedico(Data,IdU, IdS,Custo)):-
        demo(atoMedico(Data,IdU, IdS,Custo),falso),
        evolucao(atoMedico(Data,IdU, IdS,Custo)).
```

Figura 15– Evolução de conhecimento desconhecido/falso sobre atoMedico

3.5. Sistema de inferência

O sistema de inferência capaz de implementar os mecanismos de raciocínio adequados a estes sistemas tem três tipos de resposta possíveis: verdadeiro, falso e desconhecido.

Este meta-predicado foi desenvolvido tendo em conta três possibilidades:

- se existir conhecimento positivo de uma determinada Questão, esta é verdadeira;
- se existir conhecimento negativo de uma determinada Questão, esta é falsa;
- caso não haja conhecimento positivo nem negativo de uma determinada Questão, esta é então desconhecida;

```
%------------------------------ - - - - - - - - - - -  -  -  -  -   -
% Extensao do meta-predicado demo: Questao,Resposta -> {verdadeiro, falso, desconhecido}

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

Figura 16 - Sistema de inferência capaz de implementar mecanismos de raciocínio

3.6. Predicados auxiliares

3.6.1. Soluções

O predicado "soluções" encontra todas as possibilidades de prova de um teorema passado, sendo verdadeira sempre que o predicado "findall" também for.

```
% -----------------------------------------------
% Extensão do predicado solucoes: X,Y,Z -> {V,F}

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

Figura 17 - Predicado solucoes

3.6.2.Testar

Este predicado recebe uma lista e verifica se esta é válida.

```
% -----------------------------------------------
% Extensão do predicado testar: Lista -> {V,F}

testar([]).
testar([I|L]) :-
    I,
    testar(L).
```

Figura 18 - Predicado testar

3.6.3. Predicado não

Este predicado calcula o valor de verdade contrário à resposta a uma determinada questão.

```
% Extensao do predicado nao: Q -> {V,F}

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

Figura 19 - Predicado não

3.6.4. Comprimento

Este predicado verifica se a lista X tem o tamanho Z.

```
% ------------------------------------------------------------
% Extensão do predicado comprimento: Lista, Tamanho -> {V,F}

comprimento(X, Z) :-
    length(X, Z).
```

Figura 20 - Predicado comprimento

3.6.5. Registar

Os predicados "registar" e "inserção" tratam da inserção na base de conhecimento, tendo em consideração os invariantes de inserção de conhecimento explicados anteriormente neste relatório.

```
% Extensão do predicado que permite a registar conhecimento

registar(Termo) :-
    findall(Invariante, +Termo::Invariante, Lista),
    insercao(Termo),
    testar(Lista).

insercao(T) :-
    assert(T).
insercao(T) :-
    retract(T), !, fail.
```

Figura 21 - Predicados registar e insercao

3.6.6. Remover

Os predicados "remover" e "remoção" tratam da inserção na base de conhecimento, tendo em consideração os invariantes de remoção de conhecimento explicados anteriormente neste relatório.

```
% Extensão do predicado que permite a remover conhecimento

remover(Termo) :-
    findall(Invariante, -Termo::Invariante, Lista),
    testar(Lista),
    remocao(Termo).

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

Figura 22 - Predicados remover e remocao

3.6.7. RemocaoL

Este predicado permite remover o Termo da lista L, utilizando o predicado "registar" para adicionar conhecimento.

```
% Extensão do predicado que permite a remover conhecimento de uma lista

remocaoL( Termo,L ) :-
    retractL( L ),
    registar(Termo).
remocaoL( Termo,L) :-
    assertL( L ),!,fail.

retractL([]).
retractL([X|L]):-
        retract(X),
        retractL(L).

assertL([]).
assertL([X|L]):-
        assert(X),
        assertL(L).
```

Figura 23 - Predciados remocaoL, retractL e assertL

4. Conclusões e Sugestões

Este trabalho prático foi realizado com a intenção de solidificar os conhecimentos da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio. Concluído este trabalho, conseguimos perceber melhor o conceito de conhecimento imperfeito, o funcionamento de cada um dos valores nulos que lecionamos, e ainda a utilidade da utilização destes conhecimentos para situações práticas.

Foram aplicados os conhecimentos da linguagem de PROLOG que adquirimos ao longo das aulas, sendo estes desenvolvidos à medida que o trabalho ia sendo realizado, uma vez que foi necessário resolver um conjunto de novas situações e criar mecanismos de raciocínio para a resolução de problemas.

De uma forma geral, os resultados produzidos foram bastante satisfatórios e enriquecedores para todos os elementos do grupo, pois conseguimos praticar e desenvolver definições e construção de predicados recorrendo à programação em lógica e entender o porquê da sua existência e necessidade.

5. Bibliografia

[Analide, 2011] ANALIDE, César, NOVAIS, Paulo, NEVES, José,

"Sugestões para a Redacção de Relatórios Técnicos",

Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011

[Analide, 1996] ANALIDE, César, NEVES, José,

"Representação de Informação Incompleta",

Departamento de Informática, Universidade do Minho, Portugal, 2011.