Universidade do Minho
Escola de Engenharia

Sistemas de Representação de Conhecimento e Raciocínio

TRABALHO PRÁTICO Nº 2

Mestrado Integrado em Engenharia Informática

Grupo 30

78416 Francisco José Moreira Oliveira

79617 Raul Vilas Boas

79175 Vitor Emanuel Carvalho Peixoto

Ano letivo 2017/2018

Braga, abril de 2018

RESUMO

Este trabalho prático foi realizado com o intuito de desenvolver e evoluir as competências adquiridas, na unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, usando programação lógica com o PROLOG.

Este segundo trabalho incide na temática do conhecimento imperfeito, relativamente a uma área de prestação de cuidados de saúde e tem como objetivo desenvolver um sistema para este caso de estudo.

Este relatório serve então para explicar o processo de desenvolvimento e raciocínio, bem como as escolhas tomadas no decorrer deste.

ÍNDICE

1. INTRODUÇÃO

Este segundo trabalho, consiste na realização de um conjunto de exercícios envolvendo um sistema desenvolvido à volta de uma área de prestação de cuidados de saúde e tem como objetivo desenvolver um sistema de representação de conhecimento e raciocínio para este caso de estudo. Logo, para isso, vai ser necessário caracterizar o conhecimento como apresentado no enunciado.

Vai ser necessário representar o conhecimento perfeito e imperfeito dos utentes, prestadores e cuidados de saúde, onde cada um possui a sua própria informação. Para representar esse conhecimento foram definidos os predicados para cada uma dessas entidades.

2. PRELIMINARES

Neste trabalho foi necessário armazenar a informação sobre os vários utentes, prestadores e cuidados de saúde, no entanto é preciso seguir algumas regras para representar o conhecimento perfeito, pois apenas existem dois valores de prova, o "verdadeiro" e o "falso".
Sendo assim tivemos de ter em atenção aos seguintes pressupostos:

- **Pressuposto dos Nomes Únicos:** quaisquer duas variáveis iguais são necessariamente duas entidades iguais, enquanto que se forem diferentes tem um valor atómico diferente;
- **Pressuposto do Mundo Fechado:** todo a informação que não existe na base de dados é considerada como falsa;
- **Pressuposto do Domínio Fechado:** não existe mais nenhum objeto para além dos quais já constam na base de dados.

No entanto que para o conhecimento imperfeito dois destes pressupostos são alterados, sendo eles o pressuposto do mundo fechado e o pressuposto do domínio fechado passando a serem abertos.

- **Pressuposto do Mundo Aberto:** pode haver outras conclusões verdadeiras para além daquelas que estão representadas na base de conhecimento;
- **Pressuposto do Domínio Aberto:** podem existir mais objetos sem ser aqueles que constam na base de conhecimento

Para além disto, também foi necessário arranjar uma forma de ser possível representar o desconhecido, para quando não for possível provar a veracidade nem a falsidade de uma questão.
Para isso criou-se um predicado demo que consegue que permite descobrir o resultado de um conhecimento, sendo ele verdadeiro, falso ou desconhecido.

Assim, se existir uma prova de veracidade da questão na base de conhecimento a resposta será verdadeira, enquanto que se houver uma negação forte a resposta será falsa. Caso contrário, quando não é possível provar a Questão ou a -Questao é considerado desconhecido

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

Figura 1 - Predicado demo

3. DESCRIÇÃO DO TRABALHO E ANÁLISE DE RESULTADOS

Neste trabalho existem 3 tipos de predicados que representam o conhecimento, sendo, eles os **utentes**, os **prestadores** e os **cuidados de saúde**. Os utentes possuem um identificador, um nome, uma idade e a morada. Os prestadores contêm um identificador, um nome, uma especialidade e uma instituição. E por fim, os cuidados de saúde possuem uma data, o identificador do utente, o identificar do prestador, uma descrição e um custo.

- utente: #IdUt, Nome, Idade, Morada, Genero ↝ { 𝕍, 𝔽 }
- prestador: #IdPrest, Nome, Especialidade, Instituição ↝ { 𝕍, 𝔽 }
- cuidado: Data, #IdUt, #IdPrest, Descrição, Custo ↝ { 𝕍, 𝔽 }

Para além disto, também foi adicionado nos utentes o tipo "género" que distingue os utentes do sexo masculino dos utentes do sexo feminino.

3.1 Representação do Conhecimento Positivo

Para representar o conhecimento positivo, visto que é baseado em factos, foi necessário povoar a base de conhecimento com exemplos de utentes, prestadores e cuidados de saúde. Para isso, utilizou-se o conhecimento que já tinha sido criado no primeiro exercício.

Primeiro começamos pelos utentes. Estes possuem identificador, nome, idade, morada e género:

```
%utente(Id,Nome,Idade,Morada,Genero)
utente(1,'Raul',20,'Campos','Masculino').
utente(2,'Francisco',20,'Joane','Masculino').
utente(3,'Vitor',20,'Vermoim','Masculino').
utente(4,'Carlos',7,'Campos','Masculino').
utente(5,'Bruno',20,'Campos','Masculino').
utente(6,'Ana',3,'Cerveira','Feminino').
utente(7,'Susana',20,'Cerveira','Feminino').
utente(8,'Cristina',40,'Cerveira','Feminino').
utente(9,'Fatima',77,'Braga','Feminino').
utente(10,'Filipe',33,'Braga','Masculino').
utente(11,'Carla',11,'Porto','Feminino').
utente(12,'Fabio',88,'Famalicao','Masculino').
```

Figura 2 - Base de conhecimento dos utentes

Depois seguem-se os prestadores, caraterizados pelo identificador, nome, especialidade e instituição:

```
%prestador(IDPrestador,Nome,Especialidade,Instituição)
prestador(1,'Tiago','Ortopedia','Hospital de Braga').
prestador(2,'Guilherme','Urologia','Hospital de Braga').
prestador(3,'Renato','Radiologia','Hospital de Santa Maria').
prestador(4,'Filipe','Psiquiatria','Hospital de Santa Maria').
prestador(5,'Tiago','Cirurgia','Hospital de Braga').
prestador(6,'Vitor','Pediatria','Hospital de Santo Antonio').
prestador(7,'Gil','Cirurgia','Hospital de Braga').
prestador(8,'Joao','Ortopedia','Hospital de Braga').
prestador(9,'Diana','Psiquiatria','Hospital de Braga').
```

Figura 3 - Base do conhecimento dos prestadores

Por fim, temos os cuidados de saúde que possuem uma data, a identificação do utente e prestador, uma descrição e um custo:

```
%cuidado(Data,IDU,IDP,Descrição,Custo)
cuidado('01-01-2018', 3, 1,'Dor de barriga', 15).
cuidado('04-04-2018', 7, 2,'Braco partido',20).
cuidado('23-01-2018', 4, 2,'Perna partida', 30).
cuidado('01-05-2018', 2, 3,'Reacao alergica', 5).
cuidado('03-03-2018', 12,8, 'Febre', 4).
cuidado('31-12-2018', 10,5, 'Febre', 5).
cuidado('30-03-2018', 7, 9,'Analises', 5).
cuidado('31-12-2018', 7, 6,'Urgencia', 200).
cuidado('31-12-2018', 1, 6,'Braco partido', 1).
cuidado('07-08-2018', 11, 9,'Reacao alergica', 5).
cuidado('01-04-2018', 1, 4,'Dor de cabeca', 30).
```

Figura 4 - Base do conhecimento dos cuidados de saúde

3.2 Representação do Conhecimento Negativo

O conhecimento negativo pode ser representado de duas formas, a **negação forte** e a **negação por falha**. A negação forte é representada pela conectiva '-', enquanto que a negação por falha é representada pelo termo 'não'.

A negação forte permite definir informação negativa relativamente aos utentes, prestadores e/ou cuidados de saúde. Assim, criou-se 3 exemplos deste tipo de conhecimento, um para os utentes, outra para os prestadores e, por fim um para os cuidados de saúde. A negação forte fornece-nos informação sobre o que é falso em relação ao conhecimento, por exemplo, para o caso do utente com identificador 21 ele não se chama 'Tobias', não tem 20 anos de idade, não vive na 'Carvalha' e não é do sexo masculino.

```
% Negação Forte

-utente(21,'Tobias',20,'Carvalha','Masculino').
-prestador(20,'Marta','Ortopedia','Hospital de Braga').
-cuidado('05-06-2018',3,1,'Febre',3).
```

Figura 5 – Exemplos de conhecimento com negação forte

Por outro lado, a negação por falha é utilizada para quando não se consegue provar que o conhecimento é verdadeiro. Em que, todo o conhecimento que não seja considerado verdadeiro ou não seja uma exceção é considerado falso.

```
% Negação por falha

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

-prestador(Id,N,E,I) :-
                    nao(prestador(Id,D,I,C)),
                    nao(excecao(prestador(Id,D,I,C))).

-cuidado(D,IDU,IDP,Des,C) :-
                    nao(cuidado(D,IDU,IDP,Des,C)),
                    nao(excecao(cuidado(D,IDU,IDP,Des,C))).
```

Figura 6 - Negação por falha

3.3 Representação do Conhecimento Imperfeito

Para representar o conhecimento imperfeito existem três possibilidades: o incerto, o impreciso e o interdito. Para assim, distinguir melhor as situações em que o conhecimento é apresentado como 'desconhecido'.

3.3.1 Conhecimento Imperfeito Incerto (Tipo I)

Para este tipo de conhecimento imperfeito está em causa a incerteza sobre certas informações sobre os utentes, prestadores e/ou cuidados de saúde, isto é, desconhece-se o valor desse conhecimento. Para isso recorreu-se a alguns exemplos que mostram este tipo de conhecimento.

Por exemplo, para o caso dos utentes, criou-se um utente em que se desconhece o seu nome, e por esse motivo, foi criado com 'inc1' no lugar do seu nome. Para além disto, é necessário criar uma exceção para que sempre que apareça um utente com este nome, se saber que estamos perante uma exceção.

```
utente(15,inc1,71,'Vermoim','Masculino').
excecao(utente(IDU,N,I,M,G)) :- utente(IDU,inc1,I,M,G).
```

Figura 7 – Desconhece-se o nome do utente com iD=15

Para além deste, criou-se também outros exemplos para os utentes, os prestadores e os cuidados de saúde.

```
prestador(12,'Carlos',inc5,'Hospital de Santa Maria').
excecao(prestador(IDP,N,E,I)) :- prestador(IDP,N,inc5,I).
```

Figura 9 – Desconhece-se a especialidade do prestador com ID=12

```
cuidado('09-04-2018',6,6,inc8,22).
excecao(cuidado(D,IDU,IDP,Desc,C)) :- cuidado(D,IDU,IDP,inc8,C).
```

Figura 8 – Desconhece-se a descrição do cuidado prestado

3.3.2 Conhecimento Imperfeito Impreciso (Tipo II)

Este tipo de conhecimento apresenta um valor desconhecido, mas, apenas sobre um conjunto limitado de valores, isto é, a informação apenas pode ter o valor de entre as várias opções limitadas, apenas não se sabe qual é o verdadeiro.

Por exemplo, para os utentes, criou-se uma utente chamada Maria com o identificador 16, no entanto não se sabe ao certo qual é a sua morada, pois existem duas opções, ou é Vila Nova de Cerveira ou é Vila Nova de Famalicão, por esse motivo, foi necessário representar as exceções destes casos.

```
excecao(utente(16,'Maria',20,'Vila Nova de Cerveira','Feminino')).
excecao(utente(16,'Maria',20,'Vila Nova de Famalicao','Feminino')).
```

Figura 10 – Não se sabe ao certo qual é a morada da Maria

Depois, também se criou mais dois novos casos para os prestadores e para os cuidados de saúde.

```
excecao(prestador(13,'Catarina','Psiquiatria','Hospital de Braga')).
excecao(prestador(13,'Catarina','Cirurgia','Hospital de Braga')).
excecao(prestador(13,'Catarina','Ortopedia','Hospital de Braga')).
```

Figura 12 – Não se tem a certeza de qual é a especialidade do prestador com ID=13

```
excecao(cuidado('01-03-2018',5,5,'Urgencia',6)).
excecao(cuidado('02-03-2018',5,5,'Urgencia',6)).
```

Figura 11 – Não se sabe ao certo em que data foi realizado o cuidado

3.3.3 Conhecimento Imperfeito Interdito (Tipo III)

Por último, temos o conhecimento imperfeito interdito, este tipo de conhecimento para além de identificar os valores desconhecidos não permite que haja evolução desse conhecimento, isto é, não é permitido especificar ou conhecer o seu valor.

Para mostrar este tipo de conhecimento criou-se um utente chamado Salvador, com ID=17, do qual não se sabe a sua idade, visto que está representada com 'idadeY'. Como neste tipo de conhecimento é impossível alguma vez descobrir a idade deste utente, temos de criar um invariante que não permita a evolução deste conhecimento e para isso, criou-se um predicado nulo para identificar o 'idadeY'. Assim, ao procurar pelo utente em específico, se existir o predicado 'nulo(idadeY)' nunca vai ser possível evoluir as informações relativas à idade deste utente.

```
utente(17,'Salvador',idadeY,'Barcelos','Masculino').
excecao(utente(IDU,N,I,M,G)) :- utente(IDU,N,idadeY,M,G).
nulo(idadeY).

+utente(IDU,N,Idade,M,G) ::(
    solucoes(I,(utente(17,'Salvador',I,'Barcelos','Masculino'),nao(nulo(I))),L),
    comprimento(L,R),
    R==0).
```

Figura 13 – Desconhece-se a idade do utente com ID=17 e nunca se pode conhecer

Para além deste, também se criou mais um exemplo deste conhecimento para um prestador em que se desconhecia o seu nome.

```
prestador(15,nomeY,'Urologia','Hospital de Santa Maria').
excecao(prestador(IDP,N,E,I)) :- prestador(IDP,nomeY,E,I).
numo(nomeY).

+prestador(IDP,N,E,I) ::(
    solucoes(N, (prestador(15,N,'Urologia','Hospital de Santa Maria'), nao(nulo(N))),L),
    comprimento(L,R),
    R == 0).
```

Figura 14 – Desconhece.se o nome do prestador com ID=15 e nunca se pode conhecer

3.4 Invariantes

Para impedir a evolução de certos casos foi necessário a criação de certos invariantes que impedem estas situações. Uma delas é a impossibilidade de haver dois utentes com o mesmo identificador e, por isso criou-se um invariante que dado um utente com um certo ID vai procurar na base de conhecimento se há algum utente com esse mesmo ID, caso haja, coloca esse ID numa lista. Depois, é só analisar o tamanho da lista e verificar se a condição é respeitada.

Visto que, o conhecimento do utente em causa é adicionado antes dos invariantes serem verificados, o tamanho mínimo da lista será 1 que representará o utente que queremos adicionar. No entanto, se o tamanho for igual a 2, isto significa que já existia na base de conhecimento um utente com esse identificador e por esse motivo, não podemos adicionar o utente à base, sendo por isso removido.

```
% Não pode haver mais do que uma ocorrência de um utente
+utente(ID,No,I,M,G) :: (solucoes(ID,utente(ID,X,Y,Z,W),S ),
                         comprimento(S,N),
                         N == 1).
```

Figura 15 – Invariante que não permite inserção de conhecimento de utentes repetidos

Para além deste, também se criou outro invariante que não permite a inserção de um utente com uma idade inválida (que não pertença aos números naturais com o zero inclusive) e outro um que não permite que um utente seja de outro sexo para além de masculino ou feminino.

```
% O utente a ser inserido não pode ter uma idade inválida
+utente(ID,N,I,M,G) :: naturais(I).


% O utente a ser inserido apenas pode ter dois géneros, masculino ou femininos
+utente(ID,N,I,M,G) :: genero(G).
```

Figura 16 - Invariante para a idade e para o género

Para o prestador, tal como para o utente, criou-se um invariante que não permite a inserção de um prestador com um identificador igual a um prestador já existente.

```
% Não pode haver mais do que uma ocorrência de um prestador
+prestador(ID,No,E,I) :: (solucoes(ID,prestador(ID,X,Y,Z),S),
                          comprimento(S,N),
                          N == 1).
```

Figura 17 - Invariante dos prestadores

Por último, para os cuidados criou-se três invariantes: O primeiro não permite a inserção de cuidados, caso o identificador do utente a ser inserido não exista; O segundo não permite a inserção caso o identificador do prestador não exista na base de conhecimento; O terceiro invariante não permite que o custo do cuidado a ser inserido seja inválido, isto é, menor que zero.

```
% O custo do cuidado inserido não pode ser negativo
+cuidado(D,U,S,C,A) :: C >= 0.


% Invariante que não permite a remoção de utentes caso exista cuidados de saúde com eles
-utente(ID,No,I,M,G) :: (solucoes(ID,cuidado(_,ID,_,_,_),L),
                         comprimento(L,X),
                         X == 0).


% Invariante que não permite a remoção de prestadores caso exista cuidados de saúde com eles
-prestador(ID,No,E,I) :: (solucoes(ID,cuidado(_,_,ID,_,_),L),
                          comprimento(L,X),
                          X == 0).
```

Figura 18 - Invariantes dos cuidados

```
% Invariante que não permite a remoção de utentes caso exista cuidados de saúde com eles
-utente(ID,No,I,M,G) :: (solucoes(ID,cuidado(_,ID,_,_,_),L),
                         comprimento(L,X),
                         X == 0).

% Invariante que não permite a remoção de prestadores caso exista cuidados de saúde com eles
-prestador(ID,No,E,I) :: (solucoes(ID,cuidado(_,_,ID,_,_),L),
                          comprimento(L,X),
                          X == 0).
```

Figura 19 - Invariantes de remoção

Para além deste, também se criou-se invariantes que restringem a remoção de conhecimento. Um dos casos em que se verificou que uma remoção seria inválida, era quando se tentaria remover um utente ou um prestador, quando existem cuidados de saúde com eles. Pois isso depois implicaria que haveria cuidados com utentes ou prestadores que não existiam na base de conhecimento e caso se quisesse saber a informações deles, seria impossível. Logo, criou-se dois invariantes que não permitem que se remova tanto um utente como um prestador, caso existam cuidados de saúdes com eles. Para isso, por exemplo, ao ser fornecido o utente para remover vamos procurar nos cuidados se há algum com o identificador do utente e caso haja é colocado numa lista. Depois, após calcular o comprimento da lista, retira-se as conclusões, isto é, caso o tamanho da lista seja maior ou igual a 1, quer dizer que existem cuidados com esse utente, caso o tamanho da lista seja 0, quer dizer que não existem cuidados com esse utente, e por isso, podemos removê-lo da base de conhecimento.

Depois, também houve a necessidade de criar mais invariantes que impedem certos acontecimento. Tais como, a impossibilidade de inserir excecoes iguais, inserir conhecimento negativo repetido, inserir conhecimento negativo que contradiz o positivo ou vice versa.

```
% Não permite que haja excecoes iguais
+excecao(T) :: ( solucoes(T,excecao(T),S ),
                 comprimento( S,L ),
                 L == 1 ).

% Não permite adicionar conhecimento negativo repetido
+(-Termo) :: (solucoes(Termo, -Termo, S),
              comprimento(S,N),
               N == 1).

% Permite adicionar conhecimento positivo que contradiz o conhecimento negativo
+Termo :: nao(-Termo).

% Não deixa adicionar conhecimento negativo que contradiz o conhecimento positivo
+(-Termo) :: nao(Termo).
```

Figura 20 - Exemplos de invariantes,

3.5 Evolução do Conhecimento

Neste capitulo vamos retratar como resolvemos a problemática da evolução do conhecimento imperfeito para conhecimento perfeito para os casos dos valores nulos incertos e imprecisos, visto que o interdito jamais pode ser evoluído.

3.5.1 Conhecimento Imperfeito Incerto

Para passar de conhecimento incerto para conhecimento perfeito é necessário remover todas as exceções que este conhecimento possui e atualizar o conhecimento do utente, pois este tem um dos campos com a informação desconhecida.

Usando como exemplo o seguinte caso em que desconhecemos o nome do utente com o identificador 15.

```
utente(15,inc1,71,'Vermoim','Masculino').
excecao(utente(IDU,N,I,M,G)) :- utente(IDU,inc1,I,M,G).
```

Figura 21 - Exemplo de conhecimento incerto.

Logo, para tornar este conhecimento perfeito temos de remover as suas exceções, remover o utente com a variável 'inc1' no nome e adicionar novamente o utente com o nome correto. Para isso usou-se a função 'evolucaoIncertoN' que faz o pretendido. Para além deste, também se criou um predicado

para cada tipo de informação dos utentes, prestadores e cuidados de saúde que seguem uma estrutura semelhante a este.

```
evolucaoIncertoN(utente(IDU,N,I,M,G)):-
    demo(utente(IDU,N,I,M,G),desconhecido),
    solucoes( (excecao(utente(IDU,N,I,M,G)) :- utente(IDU,X,I,M,G)), (utente(IDU,X,I,M,G),nao(nulo(X))), L),
    retractL(L),
    remove(utente(IDU,X,I,M,G)),
    evolucao(utente(IDU,N,I,M,G)).
```

Figura 22 - Predicado 'evolucaoIncertoN'.

Apresentação do output:

```
| ?- listing(excecao).
excecao(utente(A,_,B,C,D)) :-
        utente(A, inc1, B, C, D).
excecao(utente(16,'Maria',20,'Vila Nova de Cerveira','Feminino')).
excecao(utente(16,'Maria',20,'Vila Nova de Famalicao','Feminino')).
excecao(prestador(13,'Catarina','Psiquiatria','Hospital de Braga')).
excecao(prestador(13,'Catarina','Cirurgia','Hospital de Braga')).
excecao(prestador(13,'Catarina','Ortopedia','Hospital de Braga')).
excecao(cuidado('01-03-2018',5,5,'Urgencia',6)).
excecao(cuidado('02-03-2018',5,5,'Urgencia',6)).
excecao(utente(A,B,_,C,D)) :-
        utente(A, B, idadeY, C, D).
```

Figura 25 - Lista das exceções antes de aplicar o predicado.

```
| ?- evolucaoIncertoN(utente(15,'Carlos',71,'Vermoim','Masculino')).
yes
```

Figura 24 - Aplicação do predicado 'evolucaoIncertoN'.

```
| ?- listing(utente).
utente(1, 'Raul', 20, 'Campos', 'Masculino').
utente(2, 'Francisco', 20, 'Joane', 'Masculino').
utente(3, 'Vitor', 20, 'Vermoim', 'Masculino').
utente(4, 'Carlos', 7, 'Campos', 'Masculino').
utente(5, 'Bruno', 20, 'Campos', 'Masculino').
utente(6, 'Ana', 3, 'Cerveira', 'Feminino').
utente(7, 'Susana', 20, 'Cerveira', 'Feminino').
utente(8, 'Cristina', 40, 'Cerveira', 'Feminino').
utente(9, 'Fatima', 77, 'Braga', 'Feminino').
utente(10, 'Filipe', 33, 'Braga', 'Masculino').
utente(11, 'Carla', 11, 'Porto', 'Feminino').
utente(12, 'Fabio', 88, 'Famalicao', 'Masculino').
utente(15, 'Carlos', 71, 'Vermoim', 'Masculino').

yes
| ?- listing(excecao).
excecao(utente(16,'Maria',20,'Vila Nova de Cerveira','Feminino')).
excecao(utente(16,'Maria',20,'Vila Nova de Famalicao','Feminino')).
excecao(prestador(13,'Catarina','Psiquiatria','Hospital de Braga')).
excecao(prestador(13,'Catarina','Cirurgia','Hospital de Braga')).
excecao(prestador(13,'Catarina','Ortopedia','Hospital de Braga')).
excecao(cuidado('01-03-2018',5,5,'Urgencia',6)).
excecao(cuidado('02-03-2018',5,5,'Urgencia',6)).
excecao(utente(A,B,_,C,D)) :-
        utente(A, B, idadeY, C, D).
```

Figura 23 - Alterações no conhecimento após o predicado

3.5.2 Conhecimento Imperfeito Impreciso

Neste caso, ao contrário do anterior, apenas é necessário remover as exceções e adicionar o caso do conhecimento correto. Por exemplo, neste caso, como não se sabe ao certo qual é a morada do prestador com id 16, temos de remover estas exceções e adicionar o caso correto que apenas pode ser ou 'Vila Nova de Cerveira' ou 'Vila Nova de Famalicão', visto que apenas uma dessas é a informação correta.

```
excecao(utente(16,'Maria',20,'Vila Nova de Cerveira','Feminino')).
excecao(utente(16,'Maria',20,'Vila Nova de Famalicao','Feminino')).
```

Figura 26 - Exemplo de conhecimento impreciso

Para isso utilizou-se o predicado 'evolucaoImpreciso' que faz o pretendido para o caso dos prestadores, mas também para o conhecimento impreciso dos utentes e dos cuidados de saúde.

```
evolucaoImpreciso(utente(IDU,Nome,Idade,Morada,Genero)):-
        demo(utente(IDU,Nome,Idade,Morada,Genero),desconhecido),
        solucoes(excecao(utente(IDU,N,I,M,G)), excecao(utente(IDU,N,I,M,G)),L),
        retractL(L),
        evolucao(utente(IDU,Nome,Idade,Morada,Genero)).
```

Figura 27 - Predicado 'evolucaoImpreciso'.

Para mostrar melhor o que este predicado faz vamos mostrar as alterações que acontecem no conhecimento. Inicialmente, temos as seguintes exceções onde estão incluídas as exceções relativas ao exemplo apresentado anteriormente da utente com identificador 16.

```
| ?- listing(excecao).
excecao(utente(A,_,B,C,D)) :-
        utente(A, inc1, B, C, D).
excecao(utente(16,'Maria',20,'Vila Nova de Cerveira','Feminino')).
excecao(utente(16,'Maria',20,'Vila Nova de Famalicao','Feminino')).
excecao(prestador(13,'Catarina','Psiquiatria','Hospital de Braga')).
excecao(prestador(13,'Catarina','Cirurgia','Hospital de Braga')).
excecao(prestador(13,'Catarina','Ortopedia','Hospital de Braga')).
excecao(cuidado('01-03-2018',5,5,'Urgencia',6)).
excecao(cuidado('02-03-2018',5,5,'Urgencia',6)).
excecao(utente(A,B,_,C,D)) :-
        utente(A, B, idadeY, C, D).
```

Figura 28 - Lista das exceções

Depois utilizando o predicado criado e analisado novamente a listagens das exceções e também dos utentes conclui-se que as exceções relativas a esse conhecimento impreciso foram removidas e houve a adição desse utente à base de conhecimento como conhecimento perfeito.

```
| ?- evolucaoImpreciso(utente(16,'Maria',20,'Vila Nova de Cerveira','Feminino')).
yes
```

Figura 29 - Output do predicado 'evolucaoImpreciso'.

```
| ?- listing(excecao).
excecao(utente(A,_,B,C,D)) :-
        utente(A, inc1, B, C, D).
excecao(prestador(13,'Catarina','Psiquiatria','Hospital de Braga')).
excecao(prestador(13,'Catarina','Cirurgia','Hospital de Braga')).
excecao(prestador(13,'Catarina','Ortopedia','Hospital de Braga')).
excecao(cuidado('01-03-2018',5,5,'Urgencia',6)).
excecao(cuidado('02-03-2018',5,5,'Urgencia',6)).
excecao(utente(A,B,_,C,D)) :-
        utente(A, B, idadeY, C, D).
```

Figura 30 - Lista de exceções.

```
| ?- listing(utente).
utente(1, 'Raul', 20, 'Campos', 'Masculino').
utente(2, 'Francisco', 20, 'Joane', 'Masculino').
utente(3, 'Vitor', 20, 'Vermoim', 'Masculino').
utente(4, 'Carlos', 7, 'Campos', 'Masculino').
utente(5, 'Bruno', 20, 'Campos', 'Masculino').
utente(6, 'Ana', 3, 'Cerveira', 'Feminino').
utente(7, 'Susana', 20, 'Cerveira', 'Feminino').
utente(8, 'Cristina', 40, 'Cerveira', 'Feminino').
utente(9, 'Fatima', 77, 'Braga', 'Feminino').
utente(10, 'Filipe', 33, 'Braga', 'Masculino').
utente(11, 'Carla', 11, 'Porto', 'Feminino').
utente(12, 'Fabio', 88, 'Famalicao', 'Masculino').
utente(16, 'Maria', 20, 'Vila Nova de Cerveira', 'Feminino').
```

Figura 31 - Lista de utentes.

3.6 Sistemas de Inferência

Para analisar o conhecimento criou-se um predicado 'demo' que permite determinar qual o valor do conhecimento, sendo que este pode ser verdadeiro, desconhecido ou falso, como já foi explicado anteriormente.

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

Figura 32 - Predicado 'demo'

Utilizado o predicado 'demo' com um conhecimento positivo definido na base de conhecimento retorna verdadeiro

```
| ?- demo(utente(1,'Raul',20,'Campos','Masculino'),R).
R = verdadeiro ?
```

Figura 33 - Conhecimento positivo

No caso de ser utilizado um caso de conhecimento imperfeito incerto o predicado demo vai retornar desconhecido, pois neste caso não se sabe ao certo o nome do utente 15, sendo que o nome 'Carlos' é uma possibilidade, mas pode não ser o correto.

```
| ?- demo(utente(15,'Carlos',71,'Vermoim','Masculino'),R).
R = desconhecido ?
```

Figura 34 - Conhecimento imperfeito

Por último, se for fornecido um caso de um conhecimento que não existe obtemos como resultado falso, visto que ele não está inserido na base de conhecimento.

```
| ?- demo(utente(99,'Raul',20,'Campos','Masculino'),R).
R = falso ?
```

Figura 35 - Conhecimento falso

Para além deste, também se criou um predicado que permite analisar conjunções e disjunções de uma lista de conhecimento. Para isso, primeiro definiu-se todos os casos entre os diversos tipos de valoração nas conjunções, como por exemplo, numa conjunção um conhecimento verdadeiro e um conhecimento falso retorna falso.

```
conjuncao(verdadeiro,verdadeiro,verdadeiro).
conjuncao(verdadeiro,desconhecido,desconhecido).
conjuncao(desconhecido,verdadeiro,desconhecido).
conjuncao(desconhecido,desconhecido,desconhecido).
conjuncao(falso,_,falso).
conjuncao(_,falso,falso).
```

Figura 36 - Tipos de conjunções.

Depois de definidas criou-se dois predicados, 'demoC' e 'demoD' que recebem uma lista de conhecimento e retornam a valoração dessa lista, tanto para os exemplos de conjunções como para os de disjunções.

```
disjuncao(verdadeiro,_,verdadeiro).
disjuncao(_,verdadeiro,verdadeiro).
disjuncao(falso,falso,falso).
disjuncao(falso,desconhecido,desconhecido).
disjuncao(desconhecido,falso,desconhecido).
disjuncao(desconhecido,desconhecido,desconhecido).
```

Figura 37 - Tipos de disjunções.

```
% Extenão do predicado demoC: Lista, R -> {V,F,D}
demoC([],R).
demoC([Q],R) :- demo(Q,R).
demoC([Q1,e,Q2|T],R) :-
    demo(Q1,R1),
    demoC([Q2|T],R2),
    conjuncao(R1,R2,R).

% Extenão do predicado demoD: Lista, R -> {V,F,D}
demoD([],R).
demoD([Q],R) :- demo(Q,R).
demoD([Q1,ou,Q2|T],R) :-
    demo(Q1,R1),
    demoD([Q2|T],R2),
    disjuncao(R1,R2,R).
```

Figura 39 - Predicado 'demoC' e 'demoD'.

```
| ?- demoC([utente(99,'Raul',20,'Campos','Masculino'),e,utente(16,'Maria',20,'Vila Nova
de Cerveira','Feminino'),e,utente(1,'Raul',20,'Campos','Masculino')],R).
R = falso ?
yes
| ?- demoD([utente(99,'Raul',20,'Campos','Masculino'),ou,utente(16,'Maria',20,'Vila Nova
 de Cerveira','Feminino'),ou,utente(1,'Raul',20,'Campos','Masculino')],R).
R = verdadeiro ?
yes
```

Figura 38 - Output dos predicados.

4. FUNÇÕES AUXILIARES

Tal como usado no trabalho prático 1, voltou-se a usar o predicado 'evolucao' que tinha sido criado, assim como o 'involucao' para pudermos inserir ou remover conhecimento tendo em atenção aos invariantes que criamos.

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

insere(P) :- assert(P).
insere(P) :- retract(P),!,fail.

evolucao(Termo) :- solucoes(Inv,+Termo::Inv,S),
                   insere(Termo),
                   teste(S).
```

Figura 40 – Predicado 'evolucao' e suas funções auxiliares.

Também foi necessário criar uma função que remove o conhecimento de uma determinada lista. Para além destes, também foram usados vários predicados, tal como, o não, comprimento, soluções entre outros.

```
remove(P) :- retract(P).
remove(P) :- assert(P),!,fail.

involucao( Termo ) :-  Termo,
                       solucoes(Inv,-Termo::Inv,S),
                       remove(Termo),
                       teste(S).
```

Figura 43 - Predicado 'involucao' e as suas funções auxiliares.

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

Figura 42 - Predicado 'retractL'

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

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

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

genero(X) :- X == 'Masculino'.
genero(X) :- X == 'Feminino'.

naturais(0).
naturais(X) :- N is X-1, N >= 0, naturais(N).
```

Figura 41 - Funções Auxiliares.

5. CONCLUSÕES E SUGESTÕES

Após concluir todas as funcionalidades propostas do enunciado, podemos passar a uma análise final do trabalho realizado.

Assim sendo, podemos observar positivamente a realização deste trabalho, permitindo ainda solidificar conhecimentos aprendidos nas aulas e também adquirir novos conceitos acerca da programação lógica com o PROLOG sobre os diversos tipos de conhecimento.

Num projeto deste tipo, há sempre melhorias que podem ser implementadas, para permitir uma melhor manipulação do conhecimento armazenado e acrescentar mais funcionalidades. Esse é sem dúvida um trabalho a desenvolver futuramente, que seria capaz de melhorar este sistema.