

UNIVERSIDADE DO MINHO

Programação em Lógica Estendida e Conhecimento Imperfeito

MESTRADO INTEGRADO EM ENGENHARIA
INFORMÁTICA

SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO

2º SEMESTRE 2018/19

Grupo:

Etienne Costa A76089

Joana Cruz A76270

Rui Azevedo A80789

Maurício Salgado A71407

João Coutinho A86272

Docente:

César Analide

23 de Abril de 2019

Conteúdo

1	Resumo	2
2	Introdução	3
3	Preliminares	4
3.1	Programação em lógica e PROLOG	4
4	Descrição do Trabalho e Análise dos Resultados	5
5	Base de conhecimento	5
5.1	Entidades	5
5.1.1	Utente	5
5.1.2	Serviço	6
5.1.3	Consulta	6
5.1.4	Prestador	6
6	Representação de Conhecimento	7
6.1	Conhecimento positivo	7
6.2	Conhecimento negativo	7
7	Conhecimento Imperfeito	8
7.1	Incerto	8
7.2	Impreciso	8
7.3	Interdito	9
8	Integridade da Base de Conhecimento	10
8.1	Inserção de Conhecimento	11
8.2	Remoção de conhecimento	11
8.3	Conhecimento Incerto	12
8.4	Conhecimento Impreciso	13
8.5	Conhecimento Interdito	15
9	Sistema de Inferência	15
10	Conclusão	17
11	Referências Bibliográficas	18
12	Funções Auxiliares	19

1 Resumo

O presente trabalho tem como principal objetivo aprofundar os conhecimentos na linguagem de programação em lógica PROLOG.

Com base nisso foi desenvolvido o relatório explicando o desenvolvimento do segundo exercício prático no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Este trabalho desenvolvido consiste na implementação de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde tirando partido da programação em lógica estendida e à representação de conhecimento imperfeito, recorrendo à temática dos valores nulos.

2 Introdução

O relatório apresentado diz respeito ao segundo exercício proposto no âmbito da unidade curricular de Sistemas de Conhecimento de Representação e Raciocínio, utilizando a linguagem de programação em lógica PROLOG. O universo de discurso para qual estamos a desenvolver este sistema é o universo de prestação de cuidados de saúde, assim esta base de conhecimento consiste em utentes, serviços, consultas e prestadores de serviços.

3 Preliminares

Para o desenvolvimento deste projeto foi necessário alguns conhecimentos previamente adquiridos de programação em lógica, e a utilização da linguagem PROLOG. Este conhecimento foi absorvido durante as aulas de Sistemas de Representação de Conhecimento e Raciocínio, e também com alguma pesquisa nossa. De seguida, apresentamos alguns conceitos fundamentais para a compreensão e realização deste trabalho.

3.1 Programação em lógica e PROLOG

Uma das principais ideias da programação em lógica é de que um algoritmo é constituído por dois elementos disjuntos: a lógica e o controle. O componente lógico corresponde à definição do que deve ser solucionado, enquanto que o componente de controle estabelece como a solução pode ser obtida. O programador precisa somente descrever o componente lógico de um algoritmo, deixando o controle da execução para ser exercido pelo sistema de programação em lógica utilizado. Em outras palavras, a tarefa do programador passa a ser simplesmente a especificação do problema que deve ser solucionado, razão pela qual as linguagens lógicas podem ser vistas simultaneamente como linguagens para especificação formal e linguagens para a programação de computadores. O paradigma fundamental da programação em lógica é o da programação declarativa, em oposição à programação procedimental típica das linguagens convencionais. Um programa em lógica é então a representação de determinado problema ou situação expressa através de um conjunto finito de um tipo especial de sentenças lógicas denominadas cláusulas. Pode-se então expressar conhecimento (programas e/ou dados) em Prolog por meio de cláusulas de dois tipos: fatos e regras. Um fato denota uma verdade incondicional, enquanto que as regras definem as condições que devem ser satisfeitas para que uma certa declaração seja considerada verdadeira. Como fatos e regras podem ser utilizados conjuntamente, nenhum componente dedutivo adicional precisa ser utilizado. Além disso, como regras recursivas e não-determinismo são permitidos, os programadores podem obter descrições muito claras, concisas e não-redundantes da informação que desejam representar. Como não há distinção entre argumentos de entrada e de saída, qualquer combinação de argumentos pode ser empregada. Os termos "programação em lógica" e "programação Prolog" tendem a ser empregados indistintamente. Deve-se, entretanto, destacar que a linguagem Prolog é apenas uma particular abordagem da programação em lógica.

4 Descrição do Trabalho e Análise dos Resultados

De modo a complementar o sistema a ser desenvolvido foi implementado o conhecimento imperfeito e a lógica estendida, sendo assim foi introduzido um valor de verdade novo, isto é, o desconhecido. Para além disso, surge a possibilidade de representar conhecimento negativo, positivo e o imperfeito, sendo que este último pode ser incerto, impreciso ou mesmo interdito. Foram construídas certas regras da negação dos predicados de modo que seja possível obter resposta mesmo quando não exista facto negativos e para tal foi adotado o pressuposto do mundo fechado.

5 Base de conhecimento

Um programa em Prolog é um conjunto de axiomas e de regras de inferência definindo relações entre objectos que descrevem um dado problema. A este conjunto chama-se normalmente base de conhecimento.

A base de conhecimento do sistema desenvolvido é essencial à representação do conhecimento e raciocínio, tendo em conta o sistema foram desenvolvidas as seguintes entidades:

- utente: $\#IdUt, Nome, Idade, Sexo, Cidade \rightarrow \{V, F\}$
- serviço: $\#IdServ, Especialidade, Instituição, Cidade \rightarrow \{V, F\}$
- consulta: $\#IdConsult, \#IdUt, \#IdPrestador, \#IdServ, Descrição, Custo, Data \rightarrow \{V, F\}$
- prestador: $\#IdPrestador, \#IdServ, Nome, Idade, Sexo \rightarrow \{V, F\}$

5.1 Entidades

Nesta secção são apresentadas e caracterizadas as Entidades acima propostas.

5.1.1 Utente

Num mundo real o utente é caracterizado por diversos factores, que por sua vez no contexto do Prolog esses factores passam a ser denominados por átomos cujo o seu propósito é identificar os objectos. Sendo assim para os utentes optou-se por usar os seguintes átomos:

- IdUt: Identificador do utente, que por sua vez é um valor único.
- Nome: Nome do utente.
- Idade: Idade do utente.
- Sexo: Sexo do utente.
- Cidade: Cidade aonde mora o utente.

5.1.2 Serviço

Um serviço por sua vez é caracterizado pelos seguintes átomos:

- IdServ: Identificador do serviço, que por sua vez é uma valor único.
- Especialidade: Referente ao serviço prestado.
- Instituição: Espaço físico aonde é efectuado o serviço.
- Cidade: Espaço geográfico aonde se encontra a instituição.

5.1.3 Consulta

Uma consulta será sempre referente a um utente, a um prestador de serviço e a um serviço, sendo que a mesma é realizada numa data específica e tem sempre uma breve descrição e custo associado , portanto optou-se por representar o conhecimento do seguinte modo:

- IdConsult: Identificador da consulta, que por sua vez é uma valor único.
- IdUt: Identificador do utente.
- IdPrestador: Identificador do prestador de serviço.
- IdServ: Identificador do serviço prestado.
- Descrição: Breve descrição da consulta.
- Custo: Custo associado à consulta.
- Data: Data da realização da consulta.

5.1.4 Prestador

O prestador por sua vez é a entidade que presta os serviços existentes na base de conhecimento e para tal o mesmo é caracterizado pelos seguintes átomos:

- IdPrestador: Identificador do prestador, que por sua vez é uma valor único.
- IdServ: Identificador do serviço prestado.
- Nome: Nome do prestador de serviço.
- Idade: Idade do prestador.
- Sexo: Sexo do prestador.

6 Representação de Conhecimento

Sendo que é necessário apresentar uma distinção entre o conhecimento positivo e negativo, surgiu a necessidade de implementar o operador "-". Sendo que esse conhecimento negativo é representado por uma negação forte, isto é, informa que um dado termo é falso.

Sendo assim eis à seguinte listagem das negações fortes implementadas :

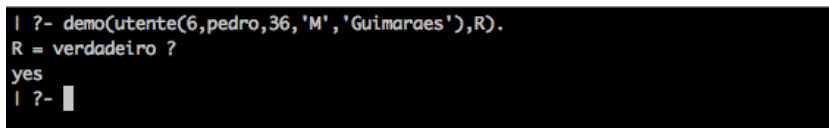
```
-utente (IdUt ,Nome, Idade , Sexo , Cidade):-  
    nao ( utente (IdUt ,Nome, Idade , Sexo , Cidade)) ,  
    nao (excecao ( utente (IdUt ,Nome, Idade , Sexo , Cidade))) .  
  
-servico (IdServ ,Especialidade , Instituicao , Cidade):-  
    nao (servico (IdServ ,Especialidade , Instituicao , Cidade)) ,  
    nao (excecao (servico (IdServ ,Especialidade , Instituicao , Cidade))) .  
  
-consulta (IdConsult ,IdUt ,IdPrestador ,IdServ ,Descricao ,Custo ,Data):-  
    nao ( consulta (IdConsult ,IdUt ,IdPrestador ,IdServ ,Descricao ,Custo ,Data)) ,  
    nao (excecao ( consulta (IdConsult ,IdUt ,IdPrestador ,IdServ ,Descricao ,Custo ,Data))) .  
  
-prestador (IdPrestador ,IdServ ,Nome, Idade , Sexo):-  
    nao ( prestador (IdPrestador ,IdServ ,Nome, Idade , Sexo)) ,  
    nao (excecao ( prestador (IdPrestador ,IdServ ,Nome, Idade , Sexo))) .
```

6.1 Conhecimento positivo

Quanto ao conhecimento positivo , este faz referência à informação existente na base de conhecimento. Podemos interpretar o seguinte exemplo:

- `utente(6,pedro,36,'M','Guimaraes')`.

É verdadeiro que o Pedro é um utente que deu entrada no centro de saúde com o identificador igual à 6, e o mesmo reside em Guimarães e possui o sexo masculino.



```
| ?- demo(utente(6,pedro,36,'M','Guimaraes'),R).  
R = verdadeiro ?  
yes  
| ?- █
```

Figura 1: Representação do conhecimento positivo.

6.2 Conhecimento negativo

Quanto ao conhecimento negativo, este faz referência à informação inexistente na base de conhecimento. Podemos interpretar o seguinte exemplo:

- `-utente(8,belo,45,'M','Dubai')`.

É falso que o Belo seja um utente que deu entrada no centro de saúde com o identificador igual à 8, que possua 45 anos e que reside na cidade do Dubai possuindo o sexo masculino.


```

| ?- demo(utente(8,beio,45,'M','Dubai'),R).
R = falso ?
yes
| ?- █

```

Figura 2: Representação do conhecimento negativo.

7 Conhecimento Imperfeito

Nesta secção pretende-se fazer uma breve explicação e representação do tipo de valores mais comuns que podem surgir numa situação de informação incompleta. Valores esses que são designados por valores nulo, sendo que os mesmos surgem como uma estratégia para a enumeração de caso, para os quais se pretende fazer a distinção entre situações em que as respostas a questões deverão ser concretizadas como verdadeiras, falsas ou desconhecidas.

Serão três os valores nulos aqui representados:

- Incerto: Desconhecido, de um conjunto indeterminado de hipóteses.
- Impreciso: Desconhecido, mas de um conjunto determinado de hipóteses.
- Interdito: Desconhecido e não permitido conhecer.

7.1 Incerto

Para descrever um utente que tenha dado entrada em algum centro de saúde apesar do seu nome ser desconhecido adotou-se a seguinte representação:

```

utente(11,name1,33,'F','Famalicao').
excecao(utente(IdUt,Nome,Idade,Sexo,Cidade)):-
utente(IdUt,name1,Idade,Sexo,Cidade).

```

Utilizou-se o átomo name1 para representar o valor incerto relativo ao utente tirando partido da exceção do mesmo para representar o valor incerto.

Como resposta ao exemplo apresentado obtemos a seguinte resposta:

```

| ?- demo(utente(11,cesar,33,'F','Famalicao'),R).
R = desconhecido ?
yes
| ?- █

```

Figura 3: Conhecimento Incerto de um Utente.

Dado que não existe conhecimento perfeito em relação ao utente acima mencionado, obtemos como esperado a resposta "desconhecido".

7.2 Impreciso

Como o próprio nome indica, este tipo de conhecimento posiciona-se em situações em que sabe-se que a informação é desconhecida mas pertence a uma gama de hipóteses. Esta imprecisão pode ser dividida em dois grupos:

- Conjunto de hipóteses distinguíveis.

- Intervalo de valores.

Para o conjunto de hipóteses distinguíveis temos casos em que o conhecimento representado é impreciso face aos valores que um determinado facto apresenta. Por exemplo temos a utente Marta de 55 anos cuja à sua morada é imprecisa pois não se sabe ao certo se a Marta reside em Guimarães ou Braga.

- `excecao(utente(14,marta,55,'F','Guimaraes'))`.
- `excecao(utente(14,marta,55,'F','Braga'))`.

```
| ?- demo(utente(14,marta,55,'F','Guimaraes'),R).
R = desconhecido ?
yes
| ?- demo(utente(14,marta,55,'F','Lisboa'),R).
R = falso ?
yes
| ?- █
```

Figura 4: Conhecimento Impreciso de um Utente.

Quanto ao intervalo de valores o exemplo mais prático seria a representação de um intervalo de idades, isto é, não ser preciso ao ponto de afirmar que o utente tem x anos de vida, mas sim que a sua idade esteja coomprendida num intervalo de valores. Para tal temos o exemplo da Teresa que é uma senhora que reside na cidade de Paris, identificada pelo Id 13 mas a sua idade é imprecisa visto que a idade da Teresa pertence ao intervalo [18,32].

- `excecao(utente(13,teresa,Idade,'F','Paris')):- Idade>=18, Idade=<=32`.

```
| ?- demo(utente(13,teresa,20,'F','Paris'),R).
R = desconhecido ?
yes
| ?- demo(utente(13,teresa,50,'F','Paris'),R).
R = falso ?
yes
| ?- █
```

Figura 5: Conhecimento Impreciso de um Utente.

7.3 Interdito

O conhecimento imperfeito do tipo interdito recai na situação em que o conhecimento relativo à uma entidade é desconhecido e permanecerá assim, não sendo possível saber o seu valor.

Como exemplo temos o utente bernardo que por questões de confidencialidade não se sabe à sua cidade, sendo que este conhecimento é identificado a custa do predicado nulo.

- `nulo(interdito1)`.
- `utente(15,bernardo,24,'M',interdito1)`.
- `excecao(utente(IdUt,Nome,Idade,Sexo,Cidade)):- utente(IdUt,Nome,Idade,Sexo,interdito1)`.

```
| ?- demo(utente(15,bernardo,24,'M','Luanda'),R).
R = desconhecido ?
yes
| ?- █
```

Figura 6: Conhecimento Interdito de um Utente.

8 Integridade da Base de Conhecimento

De forma a manter a integridade da base do conhecimento, e esta esteja de acordo com a realidade que pretendemos representar, é necessário implementar algum mecanismo que nos garanta isso. Assim, ao longo do trabalho fomos dando uso ao conceito de invariante. Estes permitem-nos controlar em específico a correta inserção e remoção na Base de Conhecimento. Os invariantes em PROLOG são representados da seguinte forma:

- +Termo :: Premissas.
- -Termo :: Premissas.

Em Prolog, já existem predicados que nos permitem inserir e remover factos da Base de Conhecimento. Estes são o `assert` e o `retract`, respetivamente. No entanto, a utilização destes predicados, exclusivamente, não garante consistência. Sendo assim, é necessário a criação de predicados auxiliares que nos garantam a integridade da Base de Conhecimento. Portanto para a realização da evolução deve ser garantido que não exista conhecimento positivo, negativo e imperfeito sendo que este último é definido a custa de exceções. Com base nisso os invariantes adotados tornaram-se muito semelhantes e obedecem o seguinte padrão:

```
+prestador(IdPrestador, IdServ, Nome, Idade, Sexo) :: (
    integer(IdPrestador),
    validaSexo(Sexo),
    Idade >= 0,
    findall(IdPrestador, prestador(IdPrestador, __, __, __), S1),
    findall(IdPrestador, -prestador(IdPrestador, __, __, __), S2),
    findall(IdPrestador, excecao(prestador(IdPrestador, __, __, __)), S3),
    length(S1, N1),
    length(S2, N2),
    length(S3, N3),
    N is N1+N2+N3,
    N <= 1).

-prestador(IdPrestador, IdServ, Nome, Idade, Sexo) :: (
    findall(IdConsult, prestador(IdPrestador, __, __, __), S1),
    length(S1, N),
    N == 0).

+(-prestador(IdPrestador, IdServ, Nome, Idade, Sexo)) :: (
    findall(IdPrestador, prestador(IdPrestador, __, __, __), S1),
    findall(IdPrestador, -prestador(IdPrestador, __, __, __), S2),
```

```

    findall(IdPrestador, excecacao(prestador(IdPrestador, __, __, __, __)), S3),
    length(S1, N1),
    length(S2, N2),
    length(S3, N3),
    N is N1+N2+N3,
    N=<1).

```

Sendo que para os restantes basta uma alteração para o respectivo predicado e os respectivos átomos.

8.1 Inserção de Conhecimento

Como já havia sido dito anteriormente o uso exclusivo da função assert não garante a integridade, para tal surgiu um conjunto de condições que são verificadas de modo a preservar a consistência da base de conhecimento, e só assim inserir conhecimento. Este processo foi denominado por evolução. É Implementado do seguinte modo:

```

evolucao(T) :-
    findall( I,+T::I,Li ),
    insercao( T ),
    teste( Li ).

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

```

Sendo que agora existe a necessidade de lidarmos com o conhecimento negativo surgiu a necessidade de criar uma função que lidasse com a evolução deste conhecimento, tendo sido ela implementada da seguinte maneira:

```

evolucaoNeg(T) :- findall( I,+(-T)::I,Li ),
                  teste( Li ),
                  assert(-T).

```

8.2 Remoção de conhecimento

Para a remoção do conhecimento o processo foi análogo ao de inserção, só que para este caso concreto o uso exclusivo do retract não garante a integridade, para tal surgiu um conjunto de condições de modo a preservar a consistência da base de conhecimento, e só assim remover conhecimento. Este processo foi denominado por involução. É implementado do seguinte modo:

```

involucao(T) :- T,
    findall( I,-T::I,Li ),
    remocao(T),
    teste( Li ).

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

```

Sendo que agora existe a necessidade de lidarmos com o conhecimento negativo surgiu a necessidade de criar uma função que lidasse com a evolução deste conhecimento, tendo sido ela implementada da seguinte maneira:

```

involucaoNeg(T) :- findall( I,+(-T)::I,Li ),
                   teste( Li ),
                   retract(-T).

```

8.3 Conhecimento Incerto

Para lidar com o conhecimento incerto optou-se por seguir dois processos distintos:

1º Inserir conhecimento incerto sem ser repetido, tendo sido tirado partido da seguinte função:

```
evolucaoNameIncerto(utente(IdUt, Nome, Idade, Sexo, Cidade)) :-
    evolucao(utente(IdUt, Nome, Idade, Sexo, Cidade)),
    assert(((excecao(utente(Id, N, I, S, C))) :- utente(Id, Nome, I, S, C))),
    assert(uncertaintyName(Nome)).
```

A imagem a seguir, retrata a tentativa de inserir um conhecimento incerto já existente e um conhecimento incerto novo, sendo o primeiro caso rejeitado face a existência do mesmo.

```
| ?- listing(utente).
utente(1, joana, 24, 'F', 'Braga').
utente(2, mauricia, 22, 'F', 'Porto').
utente(3, rui, 22, 'M', 'Coimbra').
utente(4, etienne, 25, 'M', 'Lisboa').
utente(5, rafael, 32, 'M', 'Aveiro').
utente(6, pedro, 36, 'M', 'Guimaraes').
utente(11, name1, 33, 'F', 'Famalicao').
utente(12, messi, 31, 'M', adress1).
utente(15, bernardo, 24, 'M', interdito1).
utente(16, martina, interdito2, 'F', 'Madrid').

yes
| ?- evolucaoNameIncerto(utente(11, name1, 33, 'F', 'Famalicao')).
no
| ?- evolucaoNameIncerto(utente(100, name1, 33, 'F', 'Famalicao')).
yes
| ?- listing(utente).
utente(1, joana, 24, 'F', 'Braga').
utente(2, mauricia, 22, 'F', 'Porto').
utente(3, rui, 22, 'M', 'Coimbra').
utente(4, etienne, 25, 'M', 'Lisboa').
utente(5, rafael, 32, 'M', 'Aveiro').
utente(6, pedro, 36, 'M', 'Guimaraes').
utente(12, messi, 31, 'M', adress1).
utente(15, bernardo, 24, 'M', interdito1).
utente(16, martina, interdito2, 'F', 'Madrid').
utente(11, name1, 33, 'F', 'Famalicao').
utente(100, name1, 33, 'F', 'Famalicao').
```

Figura 7: Evolução do conhecimento Incerto.

2º Modificação de conhecimento incerto para Conhecimento Perfeito:

Para este caso em concreto optou-se por corrigir o conhecimento incerto partindo do princípio que o novo valor a ser inserido fosse o correto, e para a realização desta etapa criou-se por exemplo o seguinte predicado:

```
evolucaoAdressPerfeito(utente(IdUt, Nome, Idade, Sexo, Cidade)) :-
    nao(existeAdressIncerto(utente(IdUt, Nome, Idade, Sexo, Cidade))),
    existeAdressIncerto2(utente(IdUt, Nome, Idade, Sexo, Cidade), L),
    involucaoAdressIncerto(utente(IdUt, Nome, Idade, Sexo, L)),
    evolucao(utente(IdUt, Nome, Idade, Sexo, Cidade)).
```

A imagem a seguir, retrata a evolução de um conhecimento incerto para conhecimento perfeito.

```

| ?- listing(utente).
utente(1, joana, 24, 'F', 'Braga').
utente(2, mauricia, 22, 'F', 'Porto').
utente(3, rui, 22, 'M', 'Coimbra').
utente(4, etienne, 25, 'M', 'Lisboa').
utente(5, rafael, 32, 'M', 'Aveiro').
utente(6, pedro, 36, 'M', 'Guimaraes').
utente(11, name1, 33, 'F', 'Famalicao').
utente(12, messi, 31, 'M', 'adress1').
utente(15, bernardo, 24, 'M', 'interdito1').
utente(16, martina, interdito2, 'F', 'Madrid').

yes
| ?- evolucaoAdressPerfeito(utente(4,etienne,25,'M','Lisboa')).
no
| ?- evolucaoAdressPerfeito(utente(12,messi,31,'M','Barcelona')).
yes
| ?- listing(utente).
utente(1, joana, 24, 'F', 'Braga').
utente(2, mauricia, 22, 'F', 'Porto').
utente(3, rui, 22, 'M', 'Coimbra').
utente(5, rafael, 32, 'M', 'Aveiro').
utente(6, pedro, 36, 'M', 'Guimaraes').
utente(11, name1, 33, 'F', 'Famalicao').
utente(15, bernardo, 24, 'M', 'interdito1').
utente(16, martina, interdito2, 'F', 'Madrid').
utente(4, etienne, 25, 'M', 'Lisboa').
utente(12, messi, 31, 'M', 'Barcelona').

yes
| ?- █

```

Figura 8: Evolução do conhecimento Incerto.

8.4 Conhecimento Impreciso

Relativamente ao conhecimento impreciso decidiu-se fazer modificações ao conhecimento incerto encontrado de modo a ter-se uma transição entre conhecimentos na nossa base de conhecimento, isto é, puder sair de um conhecimento do tipo incerto para algo do tipo impreciso, embora que qualquer interrogação feita sobre o mesmo o resultado será idêntico. Para implementar esta funcionalidade tirou-se partido de funções que posteriormente serão anexadas.

Optou-se por uma questão de simplificação que o conhecimento impreciso a ser evoluído seria extraído de um conjunto de valores, e ser evoluído recursivamente.

```

% (1) Consulta -----

evolucaoCostImpreciso(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),Lista,result):-
    existeCostImpreciso(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),
    nao(existeCostIncerto(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data))),
    existeCostIncerto2(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),L),
    involucaoCostIncerto(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,L,Data)),
    costRecursivo(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),Lista),
    assert(costImpreciso(IdConsult,IdUt,IdPrestador,IdServ,Custo)).

evolucaoCostImpreciso(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),Lista,result):-
    existeCostImpreciso(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),
    existeCostIncerto(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data)),
    costRecursivo(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),Lista),
    assert(costImpreciso(IdConsult,IdUt,IdPrestador,IdServ,Custo)).

involucaoCostImpreciso(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),Lista,result):-
    costAux(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),Lista),
    retract(costImpreciso(IdConsult,IdUt,IdPrestador,IdServ,Custo)).

```

Figura 9: Evolução do conhecimento Impreciso.

A imagem a seguir demonstra a utilização do código apresentado e os resultados esperados.

```

| ?- evolucaoCostImpreciso(consulta(26, 1, 12, 9, 'Consulta de Rotina', 23, (201
9,7,4)),[31,42,45],result).
yes
| ?- evolucaoCostImpreciso(consulta(26, 1, 12, 9, 'Consulta de Rotina', 23, (201
9,7,4)),[31,42,45],result).
no
| ?- evolucaoCostImpreciso(consulta(18, 1, 12, 9, 'Consulta de Rotina', cost1, (
2019,7,4)),[31,42,45],result).
yes
| ?- listing(consulta).
consulta(1, 1, 12, 9, 'Consulta de rotina', 15.0, (2018,3,11)).
consulta(2, 1, 9, 10, 'bla bla bla', 149.99, (2015,9,1)).
consulta(3, 1, 12, 9, 'bla bla bla', 12.5, (2015,9,1)).
consulta(4, 2, 8, 8, 'bla bla bla', 25.0, (2019,2,13)).
consulta(5, 2, 3, 2, 'bla bla bla', 60.0, (2017,3,28)).
consulta(6, 3, 1, 3, 'bla bla bla', 33.95, (2010,4,1)).
consulta(7, 3, 10, 11, 'bla bla bla', 25.0, (2010,4,2)).
consulta(8, 4, 11, 9, 'bla bla bla', 25.0, (2019,1,1)).
consulta(9, 4, 3, 2, 'bla bla bla', 90.0, (2011,6,12)).
consulta(10, 4, 2, 1, 'bla bla bla', 120.0, (2012,6,12)).
consulta(11, 5, 6, 7, 'bla bla bla', 32.33, (2015,9,1)).
consulta(12, 5, 9, 10, 'bla bla bla', 149.99, (2015,9,1)).
consulta(13, 6, 1, 3, 'bla bla bla', 25.0, (2018,12,1)).
consulta(14, 6, 1, 3, 'bla bla bla', 25.0, (2018,12,1)).
consulta(15, 6, 1, 3, 'bla bla bla', 25.0, (2018,12,1)).
consulta(16, 6, 1, 3, 'bla bla bla', 25.0, (2018,12,1)).
consulta(17, 6, 1, 3, 'bla bla bla', 25.0, (2018,12,1)).
consulta(19, 1, 12, 9, 'Consulta de rotina', interdito3, (2019,3,11)).

yes
| ?- demo(consulta(26, 1, 12, 9, 'Consulta de Rotina',45, (2019,7,4)),R).
R = desconhecido ?
yes
| ?-

```

Figura 10: Evolução do conhecimento Impreciso.

8.5 Conhecimento Interdito

Relativamente ao conhecimento interdito, este só deve ser inserido e nunca modificado sendo que para isso bastou inserir a respectiva exceção bem como o seu invariante.

```
evolucaoAdressInterdito(utente(IdUt, Nome, Idade, Sexo, Cidade)) :-  
    nao(excecao(utente(IdUt, Nome, Idade, Sexo, Cidade))),  
    evolucao(utente(IdUt, Nome, Idade, Sexo, Cidade)),  
    assert((excecao(utente(Id, N, I, S, C)):- utente(Id, N, I, S, Cidade))),  
    assert(nulo(Cidade)),  
    assert(+utente(ID, No, Ida, Se, Cid)::(findall((ID, No, Ida, Se, C), (utente(IdUt, Nome, Idade, Sexo, C), nao(nulo(C))))  
    length(L, 0))).
```

Figura 11: Evolução do conhecimento Interdito.

```
| ?- listing(utente).  
utente(1, joana, 24, 'F', 'Braga').  
utente(2, mauricia, 22, 'F', 'Porto').  
utente(3, rui, 22, 'M', 'Coimbra').  
utente(4, etienne, 25, 'M', 'Lisboa').  
utente(5, rafael, 32, 'M', 'Aveiro').  
utente(6, pedro, 36, 'M', 'Guimaraes').  
utente(11, name1, 33, 'F', 'Famalicao').  
utente(12, messi, 31, 'M', adress1).  
utente(15, bernardo, 24, 'M', interdito1).  
utente(16, martina, interdito2, 'F', 'Madrid').  
  
yes  
| ?- evolucaoAdressInterdito(utente(150, bernardo, 24, 'M', interdito1)).  
yes  
| ?- listing(utente).  
utente(1, joana, 24, 'F', 'Braga').  
utente(2, mauricia, 22, 'F', 'Porto').  
utente(3, rui, 22, 'M', 'Coimbra').  
utente(4, etienne, 25, 'M', 'Lisboa').  
utente(5, rafael, 32, 'M', 'Aveiro').  
utente(6, pedro, 36, 'M', 'Guimaraes').  
utente(11, name1, 33, 'F', 'Famalicao').  
utente(12, messi, 31, 'M', adress1).  
utente(15, bernardo, 24, 'M', interdito1).  
utente(16, martina, interdito2, 'F', 'Madrid').  
utente(150, bernardo, 24, 'M', interdito1).  
  
yes  
| ?- █
```

Figura 12: Evolução do conhecimento Interdito.

9 Sistema de Inferência

Foi desenvolvido um sistema que permite implementar mecanismos de inferência sobre três tipos de valores nulos, que concretizam situações de informação incompleta. Este sistema suporta três respostas possíveis:

- verdadeiro.

- falso.
- desconhecido.

Para tal foi desenvolvido o predicado `demo` que por sua vez recebe dois argumentos, sendo um deles a Questão a ser colocada e o segundo a Resposta da questão colocada.

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

Sendo que esse predicado apresenta limitações no que concerne a quantidade de questões decidiu-se abranger as conjunções e disjunções ao nosso demonstrador e por fim gerar uma espécie de map do demonstrador podendo assim operar sobre um conjunto de questões.

De seguida são representadas todas as combinações possíveis das conjunções e disjunções bem como as cláusulas que as representam:

Disjunção:

```
demo(Q1, ou, Q2, F) :- demo(Q1, F1) ,
                        demo(Q2, F2) ,
                        disjuncao(F1, F2, F) .
```

Questão1	Questão2	Resposta
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Verdadeiro	Desconhecido	Verdadeiro
Falso	Falso	Falso
Falso	Verdadeiro	Verdadeiro
Falso	Desconhecido	Desconhecido
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Verdadeiro
Desconhecido	Falso	Desconhecido

Conjunção:

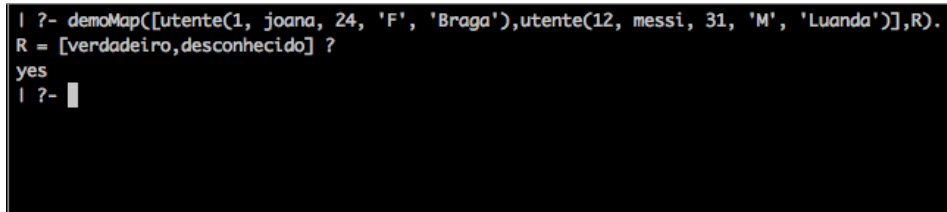
```
demo(Q1, e, Q2, F) :- demo(Q1, F1) ,
                       demo(Q2, F2) ,
                       conjuncao(F1, F2, F) .
```

Questão1	Questão2	Resposta
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Verdadeiro	Desconhecido	Desconhecido
Falso	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Desconhecido	Falso
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Desconhecido
Desconhecido	Falso	Falso

DemoMap:

O demoMap recebe dois argumentos, uma lista de questões e uma lista de respostas para as questões passadas como parâmetro, foi implementado de modo a ser possível serem respondidas várias questões em simultâneo.

```
demoMap([], []).  
demoMap([X|L], [R|S]) :- demo(X, R),  
                           demoMap(L, S).
```



```
| ?- demoMap([utente(1, joana, 24, 'F', 'Braga'),utente(12, messi, 31, 'M', 'Luanda')],R).  
R = [verdadeiro,desconhecido] ?  
yes  
| ?- █
```

Figura 13: Map Demo.

10 Conclusão

A realização deste trabalho prático permitiu consolidar o conhecimento adquirido ao longo das aulas, no que concerne à programação em lógica estendida e conhecimento imperfeito. Como tal, o suporte utilizado para caracterizar um universo de discurso na área da prestação de cuidados de saúde foi o PROLOG. Em forma de conclusão, o grupo considera que conseguiu corresponder as expectativas e consolidar os conceitos relativos ao conhecimento imperfeito e às variantes que este apresenta. Sendo que uma das principais dificuldades encontradas no desenvolvimento deste sistema passou pela forma como era feita a evolução de conhecimentos face aos diferentes tipo de conhecimento, sendo assim no que concerne a melhorias , passa por num futuro próximo inserir outros factos de modo a aproximar o trabalho de um contexto mais real, factos esses como receitas médicas e medicamentos.

11 Referências Bibliográficas

Referências

- [1] Cesar Analide, Jose Neves. *"Representação de Informação Incompleta"*.
- [2] Ivan Bratko. *"PROLOG: Programming for Artificial Intelligence"*.

12 Funções Auxiliares

```
%-----
demo( Questao , verdadeiro ) :- Questao .

demo( Questao , falso ) :- ~ Questao .

demo( Questao , desconhecido ) :- nao( Questao ) ,
                                nao( ~ Questao ) .

%-----
- - - - -

demo( Q1 , ou , Q2 , F ) :- demo( Q1 , F1 ) ,
                             demo( Q2 , F2 ) ,
                             disjuncao( F1 , F2 , F ) .

demo( Q1 , e , Q2 , F ) :- demo( Q1 , F1 ) ,
                           demo( Q2 , F2 ) ,
                           conjuncao( F1 , F2 , F ) .

%-----
- - - - -

disjuncao( verdadeiro , X , verdadeiro ) .
disjuncao( X , verdadeiro , verdadeiro ) .
disjuncao( desconhecido , Y , desconhecido ) :- Y \= verdadeiro .
disjuncao( Y , desconhecido , desconhecido ) :- Y \= verdadeiro .
disjuncao( falso , falso , falso ) .

%-----
- - - - -

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

%-----
- - - - -

demoMap( [] , [] ) .
demoMap( [X|L] , [R|S] ) :- demo( X , R ) ,
                             demoMap( L , S ) .
```

```

%
nao(T) :- T, !, fail.
nao(T).
%
teste([]).
teste([I|Is]):-I, teste(Is).

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

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

evolucao(T) :- findall(I,+T::I,Li),
               insercao(T),
               teste(Li).

involucao(T) :- T,
               findall(I,-T::I,Li),
               remocao(T),
               teste(Li).

evolucaoNeg(T) :- findall(I,+(-T)::I,Li),
                  teste(Li),
                  assert(-T).

involucaoNeg(T) :- findall(I,+(-T)::I,Li),
                   teste(Li),
                   retract(-T).

%-----PREDICADOS AUXILIARES-----


---


existeAdressIncerto(utente(IdUt,Nome,Idade,Sexo,Cidade)) :-
  (findall(City,(utente(IdUt,N,I,S,City),uncertaintyAdress(City)),L),
   length(L,0)).

existeAdressIncerto2(utente(IdUt,Nome,Idade,Sexo,Cidade),L) :-
  (findall((City),(utente(IdUt,N,I,S,City),uncertaintyAdress(City)),[L|Ls])).

%-----


---


existeNameIncerto(utente(IdUt,Nome,Idade,Sexo,Cidade)) :-
  (findall(Name,(utente(IdUt,Name,I,S,C),uncertaintyName(Name)),L),
   length(L,0)).

existeNameIncerto2(utente(IdUt,Nome,Idade,Sexo,Cidade),L) :-
  (findall((Name),(utente(IdUt,Name,I,S,C),uncertaintyName(Name)),[L|Ls])).

%-----


---



```

```

existeInstitutionIncerto(servico(IdServ,Especialidade,Instituicao,Cidade)) :-
(findall(Insti,(servico(IdServ,E,Insti,C),uncertaintyInstitution(Insti)),L),
length(L,0)).

existeInstitutionIncerto2(servico(IdServ,Especialidade,Instituicao,Cidade),L) :-
(findall((Insti),(servico(IdServ,E,Insti,C),uncertaintyInstitution(Insti)),[L|Ls])

%-----
existeCostIncerto(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),
(findall(Cost,(consulta(IdConsult,IdUt,IdPrestador,IdServ,D,Cost,Data), uncertaintyInstitution(D,Cost)),L),
length(L,0)).

existeCostIncerto2(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),
(findall((Cost),(consulta(IdConsult,IdUt,IdPrestador,IdServ,D,Cost,Data), uncertaintyInstitution(D,Cost)),L),
length(L,0)).

existeCostImpreciso(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data):-
(findall((Idc,Idu,Idp,Ids),(costImpreciso(IdConsult,IdUt,IdPrestador,IdServ,Custo,Data),Idc,Idu,Idp,Ids)),L),
length(L,0)).

%-----
costRecurso(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),[Head|Tail],
evolucao(excecao(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Head,Data),Tail))).

costRecurso(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),[Head|Tail],
evolucao(excecao(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Head,Data),Tail))).

costRecurso(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),Tail,
evolucao(excecao(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Head,Data),Tail))).

costAux(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),[Head|Tail],
evolucao(excecao(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Head,Data),Tail))).

costAux(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),[Head|Tail],
evolucao(excecao(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Head,Data),Tail))).

costAux(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Custo,Data),Tail,
evolucao(excecao(consulta(IdConsult,IdUt,IdPrestador,IdServ,Descricao,Head,Data),Tail))).

```
