UNIVERSIDADE DO MINHO

Programação em Lógica e Invariantes

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2º Semestre – 2017/2018)

Grupo 23

A61855 Ana Paula Carvalho
A73831 João Pires Barreira
A61799 Rafael Braga Costa

Braga,
Março 2018

Resumo

Neste segundo exercício prático foi desenvolvido um aperfeiçoamento do sistema de representação de conhecimento e raciocínio desenvolvido na entrega anterior, referente a um universo de discurso na área da prestação de cuidados de saúde, introduzindo-lhe a representação de conhecimento imperfeito.

Conteúdo

1 Preliminares

A realização do primeiro trabalho prático tornou-se, como seria esperado, numa parte vital no prosseguimento para este segundo exercício.

Da mesma forma, foram utilizados todos os conhecimento assimilados até ao instante de submissão da primeira entrega e adicionadas as novas competências, agora referentes à matéria de Conhecimento Imperfeito. Foram necessárias novos conceitos como os dois tipos de negação, conjuntos de respostas e o valor do tipo desconhecido.

Ocasionalmente, o grupo recorreu a algumas pesquisas na *web* sobre programação em lógica e exemplos da sua utilização, obtendo uma perceção mais clara das suas potencialidades.

Para além do referido, uma distribuição uniforme das tarefas pelos elementos do grupo contribuiu para o sucesso deste exercício

2 Introdução

Após o desenvolvimento da caracterização de um universo de discurso na área de cuidados de saúde no primeiro exercício prático, foi-nos colocado o desafio de complementar o nosso sistema de representação de conhecimento e raciocínio incluindo a representação de conhecimento imperfeito. Assim sendo, foi introduzido um novo valor de verdade – o desconhecido – abrangendo situações em que a informação existente nos predicados não esteja completa. Este novo panorama confere flexibilidade ao sistema e institui a implementação de esquemas de raciocínio não-monótono.

Ao longo deste relatório explicitaremos a nossa abordagem ao problema apresentado e todas as as alterações a que a resolução anterior foi submetida.

3 Descrição do Trabalho e Análise dos Resultados

3.1 Negação forte e demonstradores

Por oposição ao conhecimento positivo (por exemplo: o utente 1 está registado na base de conhecimento), considerou-se a criação de um novo tipo de conhecimento, o conhecimento negat ivo. Os nomes dos predicados que representam este tipo de conhecimento foram precedidos pela etiqueta "-". O seguinte exemplo:

```
-utente(1, 'Joana', 24, 'Braga').
```

informa que não existe um utente com identificador igual a 1, com o nome de Joana, que tenha 24 anos e que more em Braga. De notar que este tipo de conhecimento difere da utilização do predicado não, ou seja, difere do seguinte caso:

```
nao(utente(1, 'Joana', 24, 'Braga')).
```

Este último exemplo representa uma negação por falha da prova, ou seja, apenas informa se existe ou não conhecimento de um determinado termo. Já o tipo de conhecimento negativo representa uma negação forte, ou seja, informa que um dado termo é falso.

De modo a se poder demonstrar a veracidade de um determinado termo criou-se uma espécie de demonstrador que através de um determinado termo devolve o valor da sua veracidade. No entanto, considere-se que se tenta demonstrar um dado termo que não existe na base de conhecimento. Não se pode considerar esse termo como sendo verdadeiro, já que esse termo não se encontra na base de

conhecimento. Assim sendo, de acordo com a linguagem *PROLOG* e de acordo com o pressuposto do mundo fechado, esse termo seria falso. No entanto, tal como se explicou acima apenas se considera um determinado termo como sendo falso quando se informa do mesmo à base de conhecimento através de um conhecimento negativo. Esta nova característica obrigou à criação de um novo tipo de conhecimento, o tipo desconhecido. Este novo tipo é caracterizado por conhecimento ao qual não existe nenhuma informação e obrigou, deste modo, a abandonar o pressuposto do mundo fechado.

O demonstrador base que permite então avaliar um qualquer termo segundo as restrições impostas é o seguinte:

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

Este demonstrador, apresenta no entanto algumas restrições, como por exemplo, não permitir avaliar um conjunto de questões. Deste modo, decidiu-se criar dois demonstradores complementares. Um destes demonstradores é o seguinte:

```
demo(Q1, eq, Q2, F) :- demo(Q1, F1),
                       demo(Q2, F2),
                       equivalencia(F1, F2, F).
demo(Q1, impl, Q2, F) :- demo(Q1, F1),
                         demo(Q2, F2),
                         implicacao(F1, F2, F).
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).
```

Este demonstrador recebe como argumentos duas questões (*Q1* e *Q2*), uma *flag F* que representa o resultado e um tipo. Este tipo pode representar uma equivalência, implicação, disjunção ou conjunção. Para cada um destes tipos é usado o demonstrador base para determinar a veracidade das duas questões (*Q1* e *Q2*) e no final realizada uma equivalência, implicação, disjunção ou conjunção dos valores obtidos. A seguintes tabelas ilustram os diferentes resultados para cada um destes tipos:

Questão1	Questão2	Resultado
Falso	Falso	Verdadeiro
Falso	Desconhecido	Desconhecido
Falso	Verdadeiro	Falso
Desconhecido	Falso	Desconhecido
Desconhecido	Desconhecido	Verdadeiro
Desconhecido	Verdadeiro	Desconhecido
Verdadeiro	Falso	Falso
Verdadeiro	Desconhecido	Desconhecido
Verdadeiro	Verdadeiro	Verdadeiro

Tabela 1: Resultados de uma equivalência

Questão1	Questão2	Resultado
Falso	Falso	Verdadeiro
Falso	Desconhecido	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Desconhecido	Falso	Desconhecido
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Desconhecido
Verdadeiro	Falso	Falso
Verdadeiro	Desconhecido	Desconhecido
Verdadeiro	Verdadeiro	Verdadeiro

Tabela 2: Resultados de uma implicação

Questão1	Questão2	Resultado
Falso	Falso	Falso
Falso	Desconhecido	Desconhecido
Falso	Verdadeiro	Verdadeiro
Desconhecido	Falso	Desconhecido
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Verdadeiro	Desconhecido	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro

Tabela 3: Resultados de uma disjunção

Questão1	Questão2	Resultado
Falso	Falso	Falso
Falso	Desconhecido	Falso
Falso	Verdadeiro	Falso
Desconhecido	Falso	Falso
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Desconhecido
Verdadeiro	Falso	Falso
Verdadeiro	Desconhecido	Desconhecido
Verdadeiro	Verdadeiro	Verdadeiro

Tabela 4: Resultados de uma conjunção

Além do demonstrador anterior, que permite a demonstração de duas questões com recurso a operadores lógicos, considerou-se a criação de um demonstrador que recebe como argumento um conjunto qualquer de questões e produz como resultado o conjunto de resultados obtidos às mesmas questões. O demonstrador é o seguinte:

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

3.2 Representação de conhecimento imperfeito

3.2.1 Tipo incerto

Este tipo de conhecimento imperfeito diz respeito a situações em que não se conhece o valor de um determinado campo. Ou seja, casos em que parte da informação é desconhecida e está dentro de um conjunto indeterminado de hipóteses.

Por exemplo, se existir um utente cujo nome é desconhecido, este conhecimento é facilmente descrito da seguinte forma, através do uso de uma variável que demarque o campo que não se conhece e da inserção de uma nova exceção associada.

```
utente(11, inc01, 49, 'Rua do Forno').

excecao(utente(Id, Nome, Idade, Morada)) :-
        utente(Id, inc01, Idade, Morada).
```

No exemplo anterior, existe utente cujo ID é "11", Idade é "49"e Morada é "Rua do Forno"mas cujo nome se desconhece. Então, utilizou-se uma variável *inc01* para sinalizar o conhecimento imperfeito do tipo incerto associado ao campo "Nome". Depois, bastou indicar que o utente cujo campo "Nome"fosse *inc01* correspondesse a uma exceção do próprio predicado utente.

Podemos, então, testar esta nova funcionalidade relativa ao suporte de conhecimento imperfeito do tipo incerto por parte do nosso sistema. Para isso, basta-nos utilizar o predicado *demo*, descrito anteriormente.

```
| ?- demo(utente(11, 'Filomena', 49, 'Rua do Forno'), F).
F = desconhecido ?
yes
| ?-
```

Figura 1: Demonstração de conhecimento imperfeito do tipo incerto

Como se pode verificar pela imagem acima, a existência de um utente cujos campos ID, Nome, Idade e Morada sejam, respetivamente, "11", "Filomena", "49"e "Rua do Forno"é indicada com o valor desconhecido por, na verdade, não haver informação (i.e. conhecimento perfeito) que nos permita indicar o nome do utente referido.

3.2.2 Tipo impreciso

Este tipo de conhecimento imperfeito diz respeito a situações em que existe informação desconhecida mas sabe-se que está dentro de um conjunto determinado de valores.

Para este tipo, distinguimos duas variantes: uma variante correspondente a um valor do tipo impreciso cujo conjunto de hipóteses é totalmente discriminado; e outra variante também correspondente

a um valor do tipo impreciso mas cujo conjunto de hipóteses é dado por uma gama de valores.

De modo a podermos diferenciar ambas as situações, o primeiro caso foi marcado com o predicado *impreciso*, enquanto que o segundo foi denotado com o predicado *imprecisoIntervalo*.

Conjunto de hipóteses discriminado Esta variante corresponde, por exemplo, à existência de um utente em que não haja certeza acerca da sua morada, sabendo-se no entanto que é uma de duas possíveis. Esta situação está descrita no exemplo seguinte:

```
excecao(utente(24, 'Zueiro', 46, 'Rua de tras')).
excecao(utente(24, 'Zueiro', 46, 'Curral de Moinas')).
```

Não se sabe se o utente "Zueiro"mora na "Rua de tras"ou em "Curral de Moinas". Assim sendo, este conhecimento imperfeito do tipo impreciso é indicado com o auxílio de uma execeção para cada umas das possibilidades.

Através do predicado *demo*, descrito anteriormente, poder-se-á verificar que se questionarmos a existência deste utente com uma das duas moradas, iremos obter a resposta "desconhecido", pelo facto de não se ter a certeza em qual das duas moradas este utente habita. No entanto, se perguntarmos se o utente habita em "Rua de cima", a resposta já será negativa.

```
| ?- demo(utente(24, 'Zueiro', 46, 'Rua de tras'), F).
F = desconhecido ?
yes
| ?- demo(utente(24, 'Zueiro', 46, 'Rua de Cima'), F).
F = falso ?
yes
```

Figura 2: Demonstração de conhecimento imperfeito do tipo incerto (hipóteses discriminadas)

Gama de valores Uma situação em que, por exemplo, não se tenha a certeza acerca da idade de um utente mas em que se saiba que este valor está situado entre um intervalo de idades, corresponde a este tipo de conhecimento imperfeito impreciso. Esse conhecimento é denotado como uma exceção em que um dos campos é menor que um determinado valor e maior do que outro determinado valor.

```
excecao(utente(13, 'Zulmira', Idade, 'Rua da Mina')) :-
        Idade >= 25,
        Idade =< 30.
```

Por exemplo, no caso apresentado acima, não se conhece o valor da Idade da utente cujo ID é "13", Nome é "Zulmira"e Morada é "Rua da Mina". Com a introdução de uma nova exceção, pôde-se indicar o conhecimento perfeito relativo aos campos ID, Nome e Morada, e, como se sabe que a utente tem entre 25 e 30 anos de idade, pôde-se também limitar o seu campo Idade adequadamente.

Desta forma, através do predicado *demo*, pode-se verificar que ao questionarmos acerca da existência de uma utente com os campos referidos anteriormente mas com, por exemplo, 27 anos, a resposta será "desconhecido"por apenas se saber que a idade da utente se encontra entre os 25 e 30 anos (desconhecendo-se o seu valor exato). No entanto, se questionarmos a existência de uma mesma utente cuja idade seja 50 anos, a resposta será negativa, pelo facto de este valor não se encontrar dentro do intervalo referido.

```
| ?- demo(utente(13, 'Zulmira', 27, 'Rua da Mina'), F).
F = desconhecido ?
yes
| ?- demo(utente(13, 'Zulmira', 50, 'Rua da Mina'), F).
F = falso ?
yes
```

Figura 3: Demonstração de conhecimento imperfeito do tipo incerto (gama de valores)

3.2.3 Tipo interdito

Este tipo de conhecimento imperfeito diz respeito a situações em que existe informação desconhecida mas em que se sabe que é impossível vir a conhecer tal informação.

Consequentemente, o conhecimento deste tipo nunca poderá ser evoluído. Assim, foi necessário a criação de um predicado *nulo*, que servirá para identificar informação deste tipo. Depois, será ainda necessário alterar os invariantes de modo a que não se possa alterar conhecimento do tipo interdito.

De modo a inserir conhecimento imperfeito do tipo interdito, será então necessário marcar o campo correspondente com uma variável específica, inserir uma nova exceção correspondente a essa variável e declará-la como do tipo nulo:

```
utente(50, 'Costa', 67, int01).

excecao(utente(Id, Nome, Idade, Morada)) :-
        utente(Id, Nome, Idade, int01).

    nulo(int01).
```

Podemos confirmar esta nova funcionalidade através do predicado demo:

```
| ?- demo(utente(50, 'Costa', 67, 'Rua do Girassol'), F).
F = desconhecido ?
yes
```

Figura 4: Demonstração de conhecimento imperfeito do tipo interdito

```
| ?- evolucao(utente(50, 'Costa', 67, 'Rua do Girofle')).
no
```

Figura 5: Demonstração de conhecimento imperfeito do tipo interdito (evolução não é permitida)

Na **Figura 4** o resultado é "desconhecido"por ser impossível saber se existe verdadeiramente um utente chamado "Costa"de 67 anos cujo "ID"é 50 e que mora na "Rua do Girassol", pelo facto de que a sua morada é desconhecida e sempre o será.

Na **Figura 5** é possível ainda verificar que não será possível inserir uma nova morada para este mesmo utente, por ter sido informado anteriormente que o campo "Morada"permanecerá permamentemente desconhecido.

3.3 Adição e remoção de conhecimento

O sistema de conhecimento construído fornece métodos que permitem um aumento e uma diminuição de conhecimento. Esses métodos são denominados por evolução e involução. De notar que é permitido, evoluir, ou involuir o sistema através de conhecimento positivo ou negativo. Além disso, o sistema deve responder de um modo consistente quando se pretende um aumento de conhecimento já existindo previamente conhecimento imperfeito. Optou-se pela construção de um sistema que fosse capaz de lidar com estes casos sem a percepção do utilizador. Isto é, um utilizador apenas necessita de invocar um destes predicados (evolução ou involução) e o próprio sistema decide como lidar com os casos de inserção ou remoção de conhecimento positivo/negativo, bem como o tratamento de conhecimento imperfeito já existente.

As duas seguintes secções explicam com detalhe o funcionamento destes dois predicados. Para cada caso é ilustrado um exemplo com recurso ao termo *utente*. Os outros termos que representam a base de conhecimento desenvolvida (*prestador*, *cuidado*, *instituicao* e *consulta*) não são exemplificados já que o processo que envolve a inserção ou remoção dos mesmos é muito semelhante.

3.3.1 Evolução do sistema

A inserção de conhecimento do sistema é caracterizada pelo predicado *mevolução*. Tal como já foi mencionado o sistema encontra-se preparado para receber conhecimento positivo ou negativo.

No que toca ao conhecimento positivo, considere-se o caso de uma inserção de um conhecimento qualquer sobre ao qual não existe conhecimento imperfeito. O predicado responsável pelo tratamento deste caso é o seguinte:

```
evolucao(Termo) :- nao(imperfeito(Termo)),
                   solucoes(Inv, +Termo::Inv, S),
                   teste(S),
                   assert(Termo).
```

Em que caso não exista conhecimento imperfeito relativo a um termo que se pertende inserir e, se respeitar os eventuais invariantes relativos a esse termo, é então realizado o seu respetivo *assert*. Os predicados *imperfeito*, *solucoes* e *teste* são fornecidos em anexo.

A título de exemplo considere-se a inserção de um novo utente dado por:

```
utente(24, 'Joana', 30, 'Braga').
```

Considere-se que este novo utente de identificador 24 não existe na base de conhecimento e que não há qualquer representação de conhecimento imperfeito sobre um utente com o identificador 24. Assim, para a inserção deste novo utente o seguinte invariante deve ser respeitado:

```
+utente(Id, Nome, Idade, Morada) :: (
    integer(Id),
    integer(Idade),
    Idade >= 0,
    Idade =< 125,
    solucoes(Id, utente(Id, _, _, _), S1),
    solucoes(Id, -utente(Id, Nome, Idade, Morada), S2),
    solucoes(M, (utente(Id, Nome, Idade, M), nulo(M)), S3),
    comprimento(S1, 0),
    comprimento(S2, N2),
```

```
    comprimento(S3, 0),
    N2 >= 0,
    N2 =< 1
).
```

Este invariante indica que um utente deve possuir um identificador inteiro, uma idade compreendida entre os 0 e os 125 anos e que não deve haver informação positiva acerca de um utente com o mesmo identificador. Além disso, não permite a existência de informação que contradiza outra informação já existente. Ou seja, neste caso, não deverá ser possível a inserção deste utente se já existir o seguinte conhecimento negativo:

```
-utente(24, 'Joana', 30, 'Braga').
```

Ainda relativamente a este caso, não deve ser possível a inserção de conhecimento relativa a valores interditos. No caso de um utente não deverá ser possível a modificação de um utente que tenha a informação acerca da sua morada como interdita.

Considere-se agora um aumento de conhecimento relativo a um utente, mas que já existe conhecimento imperfeito sobre o mesmo. O predicado responsável por este aumento de informação é o seguinte:

```
evolucao(Termo) :- imperfeito(Termo),
                   testeImperfeito(Termo),
                   removeIncerto(Termo),
                   solucoes(Inv, +Termo::Inv, S),
                   teste(S),
                   assert(Termo),
                   removeImperfeito(Termo).
```

Em que existindo conhecimento imperfeito acerca de um dado termo é efetuado um teste acerca de esse termo contradizer esse conhecimento imperfeito e, caso isto não aconteça, procede-se ao seu *assert* caso este termo respeite os eventuais invariantes.

O predicado *testeImperfeito* é válido caso o resultado da demonstração desse mesmo termo seja de um valor desconhecido. Isto é, existe conhecimento imperfeito sobre esse mesmo termo e esse conhecimento é válido, isto é, não possui o valor falso. O predico *removeIncerto* remove todo o possível conhecimento imperfeito do tipo incerto que possa existir sobre um termo. É importante realizar esta operação antes do teste ao invariante, já que para muitos termos deve haver apenas conhecimento de um identificador único. Como exemplo considere-se a existência do seguinte conhecimento imperfeito do tipo incerto:

```
utente(11, inc01, 49, 'Rua do Forno').

excecao(utente(Id, Nome, Idade, Morada)) :-
    utente(Id, inc01, Idade, Morada).

incerto(utente(11)).
```

Em que o valor *inc01* representa um valor incerto relativamente ao nome de um utente. Considere-se a inserção de um utente com o identificador 11, de nome João, de 49 anos, mas que more na Rua de Baixo. O resultado é o seguinte:

```
| ?- evolucao(utente(11, 'Joao', 49, 'Rua de Baixo')
no
```

Figura 6: Resultado de uma inserção que contradiz um conhecimento incerto

O resultado é o esperado já que o conhecimento que se pretende inserir contradiz o conhecimento imperfeito já presente, uma vez que o utente 11 não mora na Rua de Baixo. Considere-se então a inserção do mesmo utente mas com a morada correta:

```
| ?- evolucao(utente(11, 'Joao', 49, 'Rua do Forno')).
yes
| ?- listing(utente).
utente(0, 'Jose', 45, 'Rua do Queijo').
utente(1, 'Maria', 41, 'Rua de Cima').
utente(2, 'Gertrudes', 26, 'Rua Carlos Antonio').
utente(3, 'Paula', 73, 'Rua da Mina').
utente(4, 'Sebastiao', 83, 'Rua da Poeira').
utente(5, 'Zeca', 9, 'Rua do Poeira').
utente(6, 'Jorge', 44, 'Rua da Poeira').
utente(7, 'Rafaela', 23, 'Rua da Poeira').
utente(8, 'Anabela', 42, 'Rua de Baixo').
utente(9, 'Antonio', 57, 'Rua do Forno').
utente(10, 'Zueiro', 33, 'Rua do Mar').
utente(50, 'Costa', 67, int01).
utente(11, 'Joao', 49, 'Rua do Forno').

yes
| ?- listing(incerto).
incerto(prestador(11)).
incerto(instituicao(5)).
incerto(consulta(5,7,data_hora(27,1,2018,16,40))).

yes
```

Figura 7: Resultado de uma insersão de conhecimento positivo que remove conhecimento incerto

Como se pode verificar, o utente é inserido na base de conhecimento é já não há conhecimento incerto acerca do mesmo (o predicado *incerto(utente(11))* foi removido).

Considere-se agora o seguinte caso:

```
excecao(utente(24, 'Zueiro', 46, 'Rua de tras')).
excecao(utente(24, 'Zueiro', 46, 'Curral de Moinas')).

impreciso(utente(24)).
```

Em que há conhecimento imperfeito relativamente ao utente 24 do tipo impreciso (mora na Rua de tras ou em Curral de Moinas). Assim, a inserção de conhecimento perfeito relativamente ao utente 24 deve conter os mesmos valores de identificador, nome e idade e indicar qual das duas moradas está correta. A figura seguinte exemplifica uma tentativa de inserção de conhecimento positivo que contradiz o conhecimento imperfeito já que a morada não é válida.

```
| ?- evolucao(utente(24, 'Zueiro', 46, 'Braga')).
no
| ?-
```

Figura 8: Resultado de uma insersão de conhecimento positivo que contradiz um conhecimento impreciso

De seguida, apresenta-se uma inserção de conhecimento positivo para o caso anterior que é válida.

```
| ?- evolucao(utente(24, 'Zueiro', 46, 'Curral de Moinas')).
yes
| ?- listing(utente).
utente(0, 'Jose', 45, 'Rua do Queijo').
utente(1, 'Maria', 41, 'Rua de Cima').
utente(2, 'Gertrudes', 26, 'Rua Carlos Antonio').
utente(3, 'Paula', 73, 'Rua da Mina').
utente(4, 'Sebastiao', 83, 'Rua da Poeira').
utente(5, 'Zeca', 9, 'Rua do Poeira').
utente(6, 'Jorge', 44, 'Rua da Poeira').
utente(7, 'Rafaela', 23, 'Rua da Poeira').
utente(8, 'Anabela', 42, 'Rua de Baixo').
utente(9, 'Antonio', 57, 'Rua do Forno').
utente(10, 'Zueiro', 33, 'Rua do Mar').
utente(11, inc01, 49, 'Rua do Forno').
utente(50, 'Costa', 67, int01).
utente(24, 'Zueiro', 46, 'Curral de Moinas').

yes
| ?- listing(excecao).
excecao(utente(13,'Zulmira',A,'Rua da Mina')) :-
        A>=25,
        A=<30.
excecao(utente(13,'Zulma',A,'Rua da Mina')) :-
        A>=25,
        A=<30.
excecao(utente(A,_,B,C)) :-
        utente(A, inc01, B, C).
excecao(utente(A,B,C,_)) :-
        utente(A, B, C, int01).
excecao(prestador(A,B,C,_)) :-
        prestador(A, B, C, inc01).
excecao(prestador(12,'Luis','Oncologia',3)).
excecao(prestador(12,'Luis','Dermatologia',3)).
excecao(prestador(12,'Luis','Cardiologia',3)).
excecao(prestador(13,'Catia','Radiografia',A)) :-
        pertence(A, [1,2,3,4]).
excecao(cuidado(data(27,3,2018),10,0,'Consulta Cardiologia',A)) :-
        A>=10,
        A=<50.
excecao(cuidado(A,B,C,_,D)) :-
        cuidado(A, B, C, int01, D).
excecao(instituicao(A,_,B)) :-
        instituicao(A, inc01, B).
excecao(instituicao(6,'Curral de Moinas Hospital','Curral de Moinas')).
excecao(instituicao(6,'Curral de Moinas Hospital','Moinas do Curral')).
excecao(consulta(A,B,C,_)) :-
        consulta(A, B, C, inc01).
excecao(consulta(6,8,data_hora(26,3,2018,13,0),data_hora(26,3,2018,14,50))).
excecao(consulta(6,8,data_hora(26,3,2018,13,0),data_hora(26,3,2018,14,30))).

yes
| ?- listing(impreciso).
impreciso(utente(A,_,_,_)) :-
        impreciso(utente(A)).
impreciso(prestador(A,_,_,_)) :-
        impreciso(prestador(A)).
impreciso(instituicao(A,_,_)) :-
        impreciso(instituicao(A)).
impreciso(cuidado(A,B,C,_,_)) :-
        impreciso(cuidado(A,B,C)).
impreciso(consulta(A,B,C,_)) :-
        impreciso(consulta(A,B,C)).
impreciso(prestador(12)).
impreciso(instituicao(6)).
impreciso(consulta(6,8,data_hora(26,3,2018,13,0))).

yes
```

Figura 9: Resultado de uma insersão de conhecimento positivo que remove conhecimento impreciso

Como se pode verificar, o utente 24 foi inserido na base de conhecimento, as duas exceções que ilustravam as suas duas possíveis moradas foram removidas e, por sua vez, este conhecimento deixou de

ser impreciso (o predicado (*impreciso(utente(24))* foi removido).

Ainda relativamente à inserção de conhecimento positivo falta considerar casos de existência de conhecimento imperfeito do tipo impreciso que representem a possibilidade de uma gama de valores. Considere-se o seguinte exemplo:

```
excecao(utente(13, 'Zulmira', Idade, 'Rua da Mina')) :-
Idade >= 25,
Idade =< 30.

excecao(utente(13, 'Zulma', Idade, 'Rua da Mina')) :-
Idade >= 25,
Idade =< 30.

imprecisoIntervalo(utente(13)).
```

Neste caso, sabe-se que o utente com o identificador número 13 chama-se Zulmira ou Zulma, mora na Rua da Mina e que tem uma idade compreendida entre os 25 e os 30 anos. Assim, a inserção de um utente com o identificador número 13, que não se chame Zulmira nem Zulma ou que a sua idade não esteja compreendida entre os 25 e os 30 anos, não é possível. A figura seguinte apresenta uma possível inserção de conhecimento perfeito positivo relativamente a este caso:

```
| ?- evolucao(utente(13, 'Zulma', 28, 'Rua da Mina')).
yes
| ?- listing(utente).
utente(0, 'Jose', 45, 'Rua do Queijo').
utente(1, 'Maria', 41, 'Rua de Cima').
utente(2, 'Gertrudes', 26, 'Rua Carlos Antonio').
utente(3, 'Paula', 73, 'Rua da Mina').
utente(4, 'Sebastiao', 83, 'Rua da Poeira').
utente(5, 'Zeca', 9, 'Rua do Poeira').
utente(6, 'Jorge', 44, 'Rua da Poeira').
utente(7, 'Rafaela', 23, 'Rua da Poeira').
utente(8, 'Anabela', 42, 'Rua de Baixo').
utente(9, 'Antonio', 57, 'Rua do Forno').
utente(10, 'Zueiro', 33, 'Rua do Mar').
utente(11, inc01, 49, 'Rua do Forno').
utente(50, 'Costa', 67, int01).
utente(24, 'Zueiro', 46, 'Curral de Moinas').
utente(13, 'Zulma', 28, 'Rua da Mina').

yes
| ?- listing(imprecisoIntervalo).
imprecisoIntervalo(prestador(13)).
imprecisoIntervalo(cuidado(data(27,3,2018),10,0)).

yes
```

Figura 10: Resultado de uma insersão de conhecimento positivo válida para uma gama de valores

Relativamente à inserção de conhecimento negativo, considere-se a inserção de conhecimento negativo sobre ao qual não existe conhecimento imperfeito. O predicado responsável pelo tratamento deste caso é o seguinte:

```
evolucao(-Termo) :- nao(impreciso(Termo)),
                    solucoes(Inv, +(-Termo)::Inv, S),
                    teste(S),
                    assert(-Termo).
```

Em que caso não exista conhecimento do tipo impreciso sobre um determinado termo e, caso o termo a inserir respeite os demais invariantes associados ao mesmo, então esse termo pode ser inserido na base de conhecimento. Os casos que respeitam conhecimento do tipo impreciso serão falados mais à frente nesta mesma secção.

Considere-se, novamente, o predicado *utente*. Suponha-se que se pretende informar acrescentar à base de conhecimento que o utente com o identificador número 60, de nome Ana, com uma idade de 40 anos e que mora em Guimarães não existe. A inserção desse mesmo termo é feita do seguinte modo:

```
evolucao(-utente(60, 'Ana', 40, 'Guimaraes')).
```

Tal como foi mencionado, para que o processo de evolução seja válido o termo a inserir deve respeitar os invariantes relacionados com o mesmo. No caso de um utente, quando se pretende acrescentar conhecimento negativo relativamente ao mesmo, deve ser respeitado o seguinte invariante:

```
+(-utente(Id, Nome, Idade, Morada)) :: (
    solucoes(Id, utente(Id, Nome, Idade, Morada), S1),
    solucoes(Id, -utente(Id, Nome, Idade, Morada), S2),
    solucoes(M, (utente(Id, N, I, M), nulo(M)), S3),
    comprimento(S1, 0),
    comprimento(S2, N2),
    comprimento(S3, 0),
    N2 >= 0,
    N2 =< 1
).
```

Ou seja, o conhecimento negativo não deve ser igual a outro conhecimento negativo já existente na base de conhecimento, não deve contradizer conhecimento positivo presente na base de conhecimento (neste caso não deveria existir na base de conhecimento o termo *utente(60, 'Ana', 40, 'Guimaraes')*) e, não deve acrescentar informação a conhecimento do tipo interdito. No caso de um utente foi criada a opção de não se poder saber qual a morada do mesmo. Caso isto aconteça, considerou-se que não se deve também acrescentar à base locais onde um utente não mora. Novamente, este invariante é muito semelhante para os outros predicados que representam a base de conhecimento desenvolvida.

Considere-se agora a inserção de conhecimento negativo quando já existe informação do tipo incerta relativamente a um determinado termo. Considere-se o seguinte caso:

```
utente(11, inc01, 49, 'Rua do Forno').

excecao(utente(Id, Nome, Idade, Morada)) :-
        utente(Id, inc01, Idade, Morada).

incerto(utente(11)).
```

Em que se desconhece ao certo o nome do utente cujo número de identificação é igual a 11. Considere-se então que se informa à base de conhecimento que esse mesmo utente não se chama Manuel. O resultado é o seguinte:

```
| ?- evolucao(-utente(11, 'Manuel', 49, 'Rua do Forno')).
yes
| ?- listing(-).
-utente(A,B,C,D) :-
        nao(utente(A,B,C,D)),
        nao(excecao(utente(A,B,C,D))).
-prestador(A,B,C,D) :-
        nao(prestador(A,B,C,D)),
        nao(excecao(prestador(A,B,C,D))).
-cuidado(A,B,C,D,E) :-
        nao(cuidado(A,B,C,D,E)),
        nao(excecao(cuidado(A,B,C,D,E))).
-instituicao(A,B,C) :-
        nao(instituicao(A,B,C)),
        nao(excecao(instituicao(A,B,C))).
-consulta(A,B,C,D) :-
        nao(consulta(A,B,C,D)),
        nao(excecao(consulta(A,B,C,D))).
-utente(11,'Manuel',49,'Rua do Forno').

yes
| ?- listing(incerto).
incerto(utente(11)).
incerto(prestador(11)).
incerto(instituicao(5)).
incerto(consulta(5,7,data_hora(27,1,2018,16,40))).

yes
```

Figura 11: Resultado de uma insersão de conhecimento negativo quando há presença de conhecimento incerto

Como se pode verificar, a informação é adicionada à base de conhecimento com sucesso, já que esta respeita o invariante relacionado com o termo *utente*. De notar que o tipo de conhecimento associado ao utente com identificador número 11 continua a ser do tipo incerto. Isto faz sentido já que continua a não se poder determinar o nome do utente. Apenas se sabe que, de uma infinidade de soluções, o utente em questão não se chama Manuel.

Relativamente a um exemplo de conhecimento impreciso temos:

```
excecao(utente(24, 'Zueiro', 46, 'Rua de tras')).
excecao(utente(24, 'Zueiro', 46, 'Curral de Moinas')).

impreciso(utente(24)).
```

Em que não se sabe ao certo se o utente 24 mora na Rua de tras ou em Curral de Moinas. Quando se trata do processo de evolução de conhecimento negativo na presença de conhecimento do tipo incerto, o predicado de evolução passa a ser descrito da seguinte forma:

```
evolucao(-Termo) :- impreciso(Termo),
                    removeImpreciso(Termo, L),
                    comprimento(L, N),
                    N > 1,
                    solucoes(Inv, +(-Termo)::Inv, S),
                    teste(S),
                    assert(-Termo).


evolucao(-Termo) :- impreciso(Termo),
                    removeImpreciso(Termo, [T|[]]),
                    removeImperfeito(Termo),
                    assert(T).
```

Como se pode verificar foram criados dois predicados de evolução distintos para este caso de presença de conhecimento impreciso.

Analisando o primeiro predicado de evolução considere-se o predicado *removeImpreciso*. Este predicado recebe como argumento um termo e produz uma lista. Essa lista corresponde a todas as exceções que caracterizam o tipo impreciso desse mesmo termo excepto aquela que se pretende remover. Considere-se a seguinte execução:

```
| ?- removeImpreciso(utente(24, 'Zueiro', 46, 'Rua de tras'), L).
L = [utente(24,'Zueiro',46,'Curral de Moinas')] ?
```

Figura 12: Remoção de uma excecão relativa a conhecimento impreciso

Ou seja, ao se indicar que o utente 24 não mora na Rua de tras, então (para este caso) a única excecão presente será a que este utente mora em Curral de Moinas. Continuando a análise do mesmo predicado de evolução, calcula-se o comprimento da lista de resultados e testa-se se o mesmo é superior a 1. Neste caso este predicado iria falhar visto que esta lista apenas possui comprimento igual a 1. Este predicado serve para inserir conhecimento negativo quando existem várias excecões relativas a conhecimento impreciso. Caso o conhecimento negativo a inserir coincida com alguma das demais exceções, esta é então removida e o conhecimento negativo inserido na base de conhecimento.

O outro predicado de evolução trata do caso exemplificado, em que existe duas exceções (Rua de tras ou Curral de Moinas) e uma delas é dada como falsa através de uma negação forte. Ora através disto, o sistema deduz então que a restante exceção que não foi eliminada consiste em conhecimento positivo do tipo perfeito. O resultado desta operação é o seguinte:

```
| ?- evolucao(-utente(24, 'Zueiro', 46, 'Rua de tras')).
yes
| ?- listing(utente).
utente(0, 'Jose', 45, 'Rua do Queijo').
utente(1, 'Maria', 41, 'Rua de Cima').
utente(2, 'Gertrudes', 26, 'Rua Carlos Antonio').
utente(3, 'Paula', 73, 'Rua da Mina').
utente(4, 'Sebastiao', 83, 'Rua da Poeira').
utente(5, 'Zeca', 9, 'Rua do Poeira').
utente(6, 'Jorge', 44, 'Rua da Poeira').
utente(7, 'Rafaela', 23, 'Rua da Poeira').
utente(8, 'Anabela', 42, 'Rua de Baixo').
utente(9, 'Antonio', 57, 'Rua do Forno').
utente(10, 'Zueiro', 33, 'Rua do Mar').
utente(11, inc01, 49, 'Rua do Forno').
utente(50, 'Costa', 67, int01).
utente(24, 'Zueiro', 46, 'Curral de Moinas').

yes
| ?- listing(excecao).
excecao(utente(13,'Zulmira',A,'Rua da Mina')) :-
        A>=25,
        A=<30.
excecao(utente(13,'Zulma',A,'Rua da Mina')) :-
        A>=25,
        A=<30.
excecao(utente(A,_,B,C)) :-
        utente(A, inc01, B, C).
excecao(utente(A,B,C,_)) :-
        utente(A, B, C, int01).
excecao(prestador(A,B,C,_)) :-
        prestador(A, B, C, inc01).
excecao(prestador(12,'Luis','Oncologia',3)).
excecao(prestador(12,'Luis','Dermatologia',3)).
excecao(prestador(12,'Luis','Cardiologia',3)).
excecao(prestador(13,'Catia','Radiografia',A)) :-
        pertence(A, [1,2,3,4]).
excecao(cuidado(data(27,3,2018),10,0,'Consulta Cardiologia',A)) :-
        A>=10,
        A=<50.
excecao(cuidado(A,B,C,_,D)) :-
        cuidado(A, B, C, int01, D).
excecao(instituicao(A,_,B)) :-
        instituicao(A, inc01, B).
excecao(instituicao(6,'Curral de Moinas Hospital','Curral de Moinas')).
excecao(instituicao(6,'Curral de Moinas Hospital','Moinas do Curral')).
excecao(consulta(A,B,C,_)) :-
        consulta(A, B, C, inc01).
excecao(consulta(6,8,data_hora(26,3,2018,13,0),data_hora(26,3,2018,14,50))).
excecao(consulta(6,8,data_hora(26,3,2018,13,0),data_hora(26,3,2018,14,30))).

yes
| ?- listing(impreciso).
impreciso(utente(A,_,_,_)) :-
        impreciso(utente(A)).
impreciso(prestador(A,_,_,_)) :-
        impreciso(prestador(A)).
impreciso(instituicao(A,_,_)) :-
        impreciso(instituicao(A)).
impreciso(cuidado(A,B,C,_,_)) :-
        impreciso(cuidado(A,B,C)).
impreciso(consulta(A,B,C,_)) :-
        impreciso(consulta(A,B,C)).
impreciso(prestador(12)).
impreciso(instituicao(6)).
impreciso(consulta(6,8,data_hora(26,3,2018,13,0))).

yes
```

Figura 13: Dedução de conhecimento positivo perfeito através de uma negação forte

Como se pode verificar, o utente 24 foi inserido, deixou de ser do tipo impreciso e, as duas exceções relativas à sua morada foram removidas.

Resta, finalmente, falar do caso da inserção de conhecimento negativo quando existe conhecimento imperfeito que suporta uma gama de valores. Considere-se o seguinte caso:

```
excecao(utente(13, 'Zulmira', Idade, 'Rua da Mina')) :-
    Idade >= 25,
    Idade =< 30.
```

A inserção de conhecimento negativo sobre este mesmo utente, mas que não esteja dentro da gama de valores de idade é suportada e é claramente verdadeira. Isto porque sabe-se que este utente tem uma idade compreendida entre os 25 e os 30 anos e, ao indicar que a idade do mesmo não está fora desse intervalo (24 por exemplo), é claramente verdadeiro.

Considere-se então a inserção do seguinte conhecimento negativo:

```
evolucao(-utente(13, 'Zulma', 27, 'Rua da Mina')).
```

Esta inserção é possível já que a idade deste mesmo utente pode ser vista como o conjunto $[25, 26, 27, 28, 29, 30] \setminus \{30\}$.

3.3.2 Involução do sistema

A remoção de conhecimento do sistema é levada a cabo pelo predicado *involução*.

No que diz respeito à remoção de conhecimento positivo, o predicado encontra-se praticamente igual ao do exercício 1 e apresenta-se abaixo:

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

Mantém-se a utilização do símbolo de "-" para designar o invariante de remoção de um termo, sendo feito ainda um teste por forma a evitar remoções de conhecimento que não exista.

Relativamente à remoção de conhecimento negativo, esta surge de forma análoga a anterior, sendo que apenas deixamos de tratar um "Termo", passando a tratar um "-Termo" que corresponde à negação forte do mesmo. Assim, apresenta-se abaixo o predicado resultante:

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

Passando agora a um exemplo concreto de remoção de conhecimento, atentemos na definição no invariante de remoção associado ao termo utente:

```
-utente(IdUt, _, _, _) :: (
        solucoes(IdUt, cuidado(_, IdUt, _, _, _), S1),
        solucoes(IdUt, consulta(IdUt, _, _, _), S2),
        comprimento(S1, 0),
        comprimento(S2, 0)
).
```

Este invariante serve para controlar a remoção de utentes da base de conhecimento. Mais concretamente, não permite que se efetue a remoção de um utente se existirem cuidados e/ou consultas associadas ao mesmo.

Considere-se o seguinte caso em que existe um utente chamado José (cujo identificador é 0):

```
utente(0, 'Jose', 45, 'Rua do Queijo').
```

Considere-se também que a base de conhecimento possui também informação relativa a uma consulta de Neurologia efetuada pelo utente José no dia 13 de Novembro de 2017:

```
cuidado(data(13, 11, 2017), 0, 1, 'Consulta Neurologia', 39.99).
```

Como seria de esperar, não se irá conseguir remover o utente refernte ao José, pelo facto de ter uma consulta associada a si na base de conhecimento. Esta situação encontra-se patente na figura abaixo:

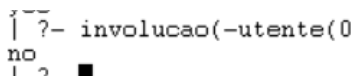

Figura 14: Falha ao tentar remover utente José por haver consulta associada

4 Conclusões e Sugestões

Dado por finalizado o trabalho prático, o grupo considera que conseguiu corresponder às expectativas na implementação um sistema de representação de conhecimento e raciocínio consistente e capaz de induzir sobre os vários tipos de conhecimento.

Esta tarefa concedeu-nos uma melhor competência na Programação em Lógica, consolidando os conhecimentos adquiridos nas últimas aulas da Unidade Curricular referentes ao tema da Representação de Conhecimento Imperfeito.

Adaptámos, de um modo criterioso, mecanismos já implementados na exercício anterior de modo a que fossem compatíveis com o novo panorama de informação incompleta e soubemos, satisfatoriamente, lidar com as problemáticas que foram surgindo ao longo do tempo.

Como sugestão para trabalho futuro – e em virtude da ampla diversidade de serviços relacionados com a área médica –, poderiam ser implementadas novas funcionalidades como, por exemplo, a inserção de receitas médicas, gerenciamento de pessoal de enfermagem, etc.

5 Anexos

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


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


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


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).



\\ Verifica se o conhecimento relativo a um utente e imperfeito
imperfeito(utente(Id, _, _, _)) :- incerto(utente(Id)).
imperfeito(utente(Id, _, _, _)) :- impreciso(utente(Id)).
imperfeito(utente(Id, _, _, _)) :- imprecisoIntervalo(utente(Id)).


\\ Verifica se o conhecimento relativo a um prestador e imperfeito
imperfeito(prestador(Id, _, _, _)) :- incerto(prestador(Id)).
imperfeito(prestador(Id, _, _, _)) :- impreciso(prestador(Id)).
imperfeito(prestador(Id, _, _, _)) :-
    imprecisoIntervalo(prestador(Id)).
```

```
\\ Verifica se o conhecimento relativo a uma instituicao e imperfeito
imperfeito(instituicao(Id, _, _)) :- incerto(instituicao(Id)).
imperfeito(instituicao(Id, _, _)) :- impreciso(instituicao(Id)).
imperfeito(instituicao(Id, _, _)) :-
    imprecisoIntervalo(instituicao(Id)).


\\ Verifica se o conhecimento relativo a um cuidado e imperfeito
imperfeito(cuidado(D, U, P, _, _)) :- incerto(cuidado(D, U, P)).
imperfeito(cuidado(D, U, P, _, _)) :- impreciso(cuidado(D, U, P)).
imperfeito(cuidado(D, U, P, _, _)) :-
    imprecisoIntervalo(cuidado(D, U, P)).


\\ Verifica se o conhecimento relativo a uma consulta e imperfeito
imperfeito(consulta(U, P, HI, _)) :- incerto(consulta(U, P, HI)).
imperfeito(consulta(U, P, HI, _)) :- impreciso(consulta(U, P, HI)).
imperfeito(consulta(U, P, HI, _)) :-
    imprecisoIntervalo(consulta(U, P, HI)).



teste([]).
teste([H | T]) :- H, teste(T).


solucoes(F, Q, S) :- findall(F, Q, S).


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