UNIVERSIDADE DO MINHO
ESCOLA DE ENGENHARIA

Sistemas de Representação de Conhecimento e Raciocínio

2º EXERCÍCIO - GRUPO 17

Adriana Henriques Esteves Teixeira Meireles A82582
Ana Marta Santos Ribeiro A82474
Carla Isabel Novais da Cruz A80564
Jéssica Andreia Fernandes Lemos A82061

MIEI - 3º Ano - 2º Semestre

Braga, 22 de abril de 2019

1 Resumo

O presente relatório tem como objetivo a documentação do segundo exercício prático da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio. Seguindo a estrutura do primeiro exercício prático, primeiramente é feita uma breve introdução, posteriormente é detalhado todo o raciocínio para o desenvolvimento das várias funcionalidades propostas no enunciado do problema bem como apresentados alguns resultados. Por fim, é realizada uma breve crítica ao trabalho realizado.

Índice

2 Introdução

Este relatório surge na realização do segundo exercicío prático proposto na unidade curricular de *Sistemas de Representação de Conhecimento e Raciocínio*. Tal como na fase anterior este deverá ser elaborado utilizando a linguagem de programação PROLOG. Contudo, o objetivo incide na representação de conhecimento imperfeito recorrendo à utilização de valores nulos e da elaboração de mecanismos de raciocínios adequeados.

Assim, como no trabalho prático anterior, pretende-se que seja desenvolvido um sistema de representação de conhecimento e raciocínio capaz de caracterizar um universo na área de prestação de cuidados de saúde. Deste modo, o conhecimento base definido é utente, cuidado e prestador. O objetivo é permitir adicionar e remover este conhecimento, tendo em atenção as restrições associadas que constituem o conhecimento imperfeito.

3 Preliminares

O sistema desenvolvido no segundo exercício deste projeto prático pretende caracterizar o universo de discurso na área de prestação de cuidados médicos. A realização do exercício anterior permitiu que a manipulação da linguagem fosse relativamente mais fácil. Através das aulas frequentadas, foi possível implementar os novos objetivos propostos da forma mais imediata. Foram realizadas pesquisas no âmbito do trabalho a realizar e após a leitura detalhada do enunciado foi realizada a distribuição do trabalho. Isto possibilitou uma maior organização do grupo e que o trabalho fosse concluído na data limite proposta.

4 Descrição do trabalho e análise de resultados

4.1 Conhecimento Positivo e Negativo

Todo o conhecimento positivo foi representado no exercicio 1, onde foi explicado o significado dos factos que compoem a base de conhecimento. Posto isto, a representaçao do conhecimento negativo sera feita nesta fase.

A representaçao de conhecimento negativo e realizada com base na negaçao forte,indicando que um facto e considerado falso caso nao exista na base de conhecimento e nao possua exceçao a si associada.

De seguida, encontram-se alguns exemplos de inserção destes tipos de conhecimento:

```
-utente( 72, 'Anacleto Sousa', 83,'Setubal' ).
negativo(72,utente).

-prestador(30,'Pedro Falc o','Oftalmologia','Hospital Sao Joao').
negativo(30,prestador).

-cuidado('24-05-2017', 18:45, 2, 8,'Depressao', 60).
negativoCuidado('24-05-2017', 18:45, 8, cuidado).
```

4.2 Representação de conhecimento imperfeito

Nesta fase do projeto, foi necessário adicionar conhecimento imperfeito, que diz respeito às situações em que não é possível obter resposta às questões, ou seja, o resultado é desconhecido. Existe três tipos deste conhecimento, nomeadamente o incerto, o impreciso e o interdito, que serão explicados detalhadamente de seguida.

4.2.1 Conhecimento Incerto

Este tipo de conhecimento está associado às situações em que um determinado valor é desconhecido, ou seja, não se conhece o seu valor, no entanto existe a possibilidade de o vir a descobrir. Para tal, foi necessário criar exceções para cada uma das situações em que determinado valor é desconhecido, como pode ser observado de seguida a título de exemplo:

```
excecao(utente(IdUt,Nome,Idade,_)):-
    utente(IdUt,Nome,Idade,desconhecido).

excecao(prestador(IdPrest,Nome,Especialidade,_)) :-
    prestador(IdPrest,Nome,Especialidade,desconhecido).

excecao(cuidado(Data,Hora,IdUt,IdPrest,Descricao,_)) :-
    cuidado(Data,Hora,IdUt,IdPrest,Descricao,desconhecido).
```

De seguida, serão apresentados alguns exemplos de inserção deste tipo de conhecimento:

```
utente(13, 'Andreia Silva', 17, desconhecido).
incerto(13,utente).

prestador(12, 'Fernando Silva', 'Cardiologia',desconhecido).
incerto(12, prestador).

cuidado('16-03-2017', 14:00, 6, 3, 'Cataratas', desconhecido).
incertoCuidado('16-03-2017', 14:00, 3, cuidado).
```

4.2.2 Conhecimento Impreciso

O conhecimento impreciso verifica-se quando determinado facto é desconhecido, no entanto a dúvida incide sobre um determinado conjunto de valores bem determinado. Assim, de seguida são apresentados alguns exemplos deste tipo de conhecimento:

```
% Duvida sobre a Morada e a idade do utente cujo identificador    9.
excecao(utente(9, 'Benedita Andrade', 60, 'Porto')).
excecao(utente(9, 'Benedita Andrade', 61, 'Porto')).
excecao(utente(9, 'Benedita Andrade', 60, 'Braganca')).
excecao(utente(9, 'Benedita Andrade', 61, 'Braganca')).
impreciso(9, utente).

% Duvida sobre a especialidade associada ao prestador cujo
    identificador    6.
excecao(prestador(6,'Daniel Ferreira','Pediatria','Hospital de
    Braga')).
excecao(prestador(6,'Daniel Ferreira','Cirurgia','Hospital de
    Braga')).
```

```
impreciso(6, prestador).

% Duvida sobre o custo do cuidado, mas sabe-se que    entre 50 e 60
    euros.
excecao(cuidado('30-04-2018', 9:30, 4, 3,'Sopro no coracao', 50)).
excecao(cuidado('30-04-2018', 9:30, 4, 3,'Sopro no coracao', 60)).
imprecisoCuidado('30-04-2018', 9:30, 4, cuidado).
```

4.2.3 Conhecimento Interdito

Este tipo de conhecimento ocorre quando determinado facto não se conhece e nunca se saberá o seu valor. Assim, foi necessário utilizar o predicado nulo para que este conhecimento desconhecido não possa ser evoluído. Deste modo, de seguida serão apresentados alguns exemplos:

```
excecao(utente(IdUt,Nome,_,Morada)):-
    utente(IdUt,Nome,semIdade,Morada).

excecao(prestador(IdPrest,Nome,Especialidade,_)) :-
    prestador(IdPrest,Nome,Especialidade,semInstituicao).

excecao(cuidado(Data,Hora,IdUt,IdPrest,_,Custo)) :-
    cuidado(Data,Hora,IdUt,IdPrest,semDescricao,Custo).

nulo(semIdade).
nulo(semInstituicao).
nulo(semDescricao).
```

Em baixo serão apresentados alguns exemplos de inserção deste tipo de conhecimento, nomeadamente um utente ao qual não se conhece a idade, um prestador com instituição desconhecida e um cuidado sem uma descrição associada.

```
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
interdito(16,utente)).

prestador(13, 'Antonieta Goncalves', 'Psiquiatria', semInstituicao).
interdito(13, prestador).

cuidado('17-03-2017', 16:00, 4, 5, semDescricao, 30).
incertoCuidado('17-03-2017', 16:00, 5, cuidado).
```

4.3 Manipular invariantes que designem restrições à inserção e à remoção de conhecimento do sistema

É necessário implementar diversos invariantes para que a nossa base de conhecimento funcione corretamente, sendo estes responsáveis por restringir a inserção e remoção de conhecimento da mesma. Com a utilização de alguns predicados como auxilio, é possível com que base de conhecimento não evolua de forma errada.

Neste exercício é agora possível a inserção de conhecimento positivo e negativo bem como incerto, impreciso e interdito. Para que a nossa base de conhecimento seja coerente, foram criados invariantes de modo a não ser possível inserir conhecimento repetido:

```
+utente(IdUt,Nome,Idade,Morada)::
        (listaSolucoes(IdUt,utente(IdUt,Nome,Idade,Morada),L),
        comprimento(L,R),
        R==1).

+prestador(IdPrest,Nome,Especialidade,Instituicao)::
        (listaSolucoes(IdPrest,
        prestador(IdPrest,Nome,Especialidade,Instituicao),L),
        comprimento(L,R),
        R==1).

+cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo)::
        (listaSolucoes((Data,Hora,IdPrest),
        cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo),L),
        comprimento(L,R),
        R==1).
```

Assim, podemos ver no exemplo que se verifica a impossibilidade de inserção de conhecimento já existente.

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
```

```
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?- evolucao(utente(60, 'Joao', 18, 'Braga'), positivo).
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
utente(60, 'Joao', 18, 'Braga').
| ?- evolucao(utente(1, 'Joana Antunes', 18, 'Braga'), positivo).
no
```

Como verificamos, não inseriu o utente 'Joana Antunes' e inseriu o utente 'Joao'.

Ao ser realizada a inserção de conhecimento negativo, temos de verificar se não é o conhecimento contraditório de algum conhecimento positivo existente, isto é um conhecimento positivo exatamente com a mesma informação, este não deve ser inserido. Deste modo, utilizando como exemplo a inserção de utentes, realizamos os seguintes invariantes:

```
+(-utente(IdUt,Nome,Idade,Morada))::
        (listaSolucoes(IdUt,
```

```
        utente(IdUt,Nome,Idade,Morada),L),
        comprimento(L,R),
        R==0).

+(-prestador(IdPrest,Nome,Especialidade,Instituicao))::
        (listaSolucoes(IdPrest,
        prestador(IdPrest,Nome,Especialidade,Instituicao),L),
        comprimento(L,R),
        R==0).

+(-cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo))::
        (listaSolucoes((Data,Hora,IdPrest),
        cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo),L),
        comprimento(L,R),
        R==0).
```

Assim, provavamos o invariante utilizado:

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
utente(60, 'Joao', 18, 'Braga').
yes
| ?- evolucao(-utente(60, 'Joao', 18, 'Braga'), negativo).
no
```

Também foram adicionados invariante que controlam a inserção de conhecimento relativo a fatores que se encontram no campo desconhecidos. Desta forma, é possível manter a coerência. Apresentamos, por exemplo, os invariantes criados para impedir a

inserção de conhecimento repetido na base de conhecimento, no caso de adicionar-mos conhecimento interdito, em que abrange todas as situações possíveis:

```
intr(utente(IdUt,_,_,_))::
        (listaSolucoes(IdUt,utente(IdUt,_,_,_),L),
        comprimento(L,R),
        R==1).

intr(prestador(IdPrest,_,_,_))::
        (listaSolucoes(IdPrest,prestador(IdPrest,_,_,_),L),
        comprimento(L,R),
        R==1).

intr(cuidado(Data,Hora,_,IdPrest,_,_))::
        (listaSolucoes((Data,Hora,IdPrest),
        cuidado(Data,Hora,_,IdPrest,_,_),L),
        comprimento(L,R),
        R==1).
```

Assim ao tentar adicionar acontecimento interdito que já exista, um utente sem idade, temos:

```
| ?- listing(interdito).
interdito(16, utente).
interdito(18, utente).
interdito(13, prestador).
yes
| ?- evolucao(utente(16, 'Carlos Costa', semIdade, 'Lisboa'),
    interdito, semIdade).
no
```

Podemos também testar a inserção de um conhecimento interdito sem mora é possível que não exista.

```
| ?- evolucao(utente(20, 'Joao', 18, semMorada),  interdito, morada).
yes
| ?- listing(interdito).
interdito(16, utente).
interdito(18, utente).
```

```
interdito(13, prestador).
interdito(20, utente).
yes
```

Caso, tentemos adicionar novamente este conhecimento não é possível pois está repetido.

```
| ?- evolucao(utente(20, 'Joao', 18, semMorada),  interdito, morada).
no
| ?- listing(interdito).
interdito(16, utente).
interdito(18, utente).
interdito(13, prestador).
interdito(20, utente).
yes
```

Achamos ainda de importância elevada o impedimento de inserção de conhecimento positivo e negativo em simultâneo. Apresentamos assim a implementação dos invariantes que impeçam este acontecimento:

```
+utente(IdUt,Nome,Idade,Morada)::
        (listaSolucoes(IdUt,-utente(IdUt,Nome,Idade,Morada),L),
        comprimento(L,R),
        R==0).

+prestador(IdPrest,Nome,Especialidade,Instituicao)::
        (listaSolucoes(IdPrest,
        -prestador(IdPrest,Nome,Especialidade,Instituicao),L),
        comprimento(L,R),
        R==0).

+cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo)::
        (listaSolucoes((Data,Hora,IdPrest),
        -cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo),L),
        comprimento(L,R),
        R==0).
```

Por fim, tornou-se imperativo a implementação de invariantes de remoção de modo

a que verificar se esse conhecimento existee por fim removê-lo. Apreserntamos assim o exmplo para os utentes:

```
-utente(IdUt,_,_,_):: (listaSolucoes(IdUt,utente(IdUt,_,_,_),L),
                                    comprimento(L,R),
                                    R == 1).
```

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
utente(60, 'Joao', 18, 'Braga').

yes
| ?- involucao(utente(60,'Joao',18,'Braga'),positivo).
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).

yes
```

```
-inc(utente(IdUt,_,_,_))::
    (listaSolucoes(IdUt,incerto(IdUt,utente),L),
                                    comprimento(L,R),
                                    R == 1).

-inc(prestador(IdPrest,_,_,_,_))::
    (listaSolucoes(IdPrest,incerto(IdPrest,prestador),L),
                                comprimento(L,R),
                                R == 1).

-inc(cuidado(Data,Hora,_,IdPrest,_,_))::
    (listaSolucoes((Data,Hora,IdPrest),
              incertoCuidado(Data,Hora,IdPrest,cuidado),
              L),
              comprimento(L,R),
              R == 1).
```

Tivemos também em atenção que apenas será possível remover informação de um utente caso este não tenha consultas, e remover um prestador no caso de não ter nenhuma consulta também, tal como podemos verificar de seguida:

```
-utente(IdUt,_,_,_,_)::
        (listaSolucoes(IdUt,cuidado(_,_,IdUt,_,_,_,_),L),
         comprimento(L,R),
        R == 0).

-prestador(IdPrest,_,_,_,_)::
        (listaSolucoes(IdPrest,cuidado(_,_,_,_,IdPrest,_,_),L),
        comprimento(L,R),
        R == 0).
```

4.4 Lidar com a problemática da evolução do conhecimento, criando os procedimentos adequados

4.4.1 Evolução

De modo a adicionar conhecimento é necessário averiguar se a inserção é válida. Para isso, aplicou-se o predicado listaSolucoes, que é responsável por listar todos os invariantes associados. Depois, é adicionado o termo pretendido e são testados os invariantes através do testaInv, que se encontra representado em baixo:

```
% testaInv: L -> {V,F}
testaInv([]).
testaInv([I|Ls]) :- I, testaInv(Ls).
```

⇒ **Conhecimento Positivo**

Os predicados que se seguem pretendem adicionar utentes, prestadores ou cuidados na base de conhecimento positiva.

```
evolucao( utente( IdUt,Nome,Idade,Morada ), Tipo ) :-
    Tipo == positivo,
    listaSolucoes(I, +utente(IdUt,Nome,Idade,Morada)::I, S),
    insere(utente(IdUt,Nome,Idade,Morada)),
    insere(positivo(IdUt,utente)),
    testaInv(S),
    removeUtente(IdUt),
    removeIncerto(IdUt,utente),
    removeImpreciso(IdUt,utente).
```

De seguida , encontra-se um exemplo que insere um utente com id que ainda não existe.

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
```

```
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).

 ?- evolucao(utente(70,'Adriana',23,'Braga'),positivo).
yes

| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
utente(70, 'Adriana', 23, 'Braga').

| ?- listing(positivo).
positivo(1, utente).
positivo(2, utente).
positivo(3, utente).
positivo(4, utente).
positivo(5, utente).
positivo(6, utente).
positivo(7, utente).
positivo(1, prestador).
positivo(2, prestador).
positivo(3, prestador).
positivo(4, prestador).
positivo(5, prestador).
positivo(70, utente).
```

Segue-se um exemplo que impede que seja inserido um utente com um id já exis-

tente.

```
| ?- evolucao(utente(2,'Joana',14,'Porto'),positivo).
no
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
utente(70, 'Adriana', 23, 'Braga').
```

O mesmo foi feito para o caso do prestador e do cuidado.

```
evolucao( prestador( IdPrest,Nome,Especialidade,Instituicao ), Tipo )
    :-
    Tipo == positivo,
    listaSolucoes(I,
        +prestador(IdPrest,Nome,Especialidade,Instituicao)::I, S),
    insere(prestador(IdPrest,Nome,Especialidade,Instituicao)),
    insere(positivo(IdPrest,prestador)),
    testaInv(S),
    removePrestador(IdPrest),
    removeIncerto(IdPrest,prestador),
    removeImpreciso(IdPrest,prestador).

evolucao( cuidado( Data,Hora,IdUt,IdPrest,Descricao,Custo ), Tipo ) :-
    Tipo == positivo,listaSolucoes(I,
        +cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo)::I, S),
    insere(cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo)),
    insere(positivoCuidado(Data,Hora,IdPrest,cuidado)),
    testaInv(S),
    removeCuidado(Data,Hora,IdPrest),
    removeIncerto(Data,Hora,IdPrest,cuidado),
    removeImpreciso(Data,Hora,IdPrest,cuidado).
```

⇒ Conhecimento Negativo

Os predicados que se seguem pretendem adicionar utentes, prestadores ou cuidados na base de conhecimento negativa.

```
evolucao( utente(IdUt, Nome, Idade ,Morada) , Tipo ) :-
    Tipo == negativo,
    listaSolucoes( I, +(-utente(IdUt, Nome, Idade ,Morada))::I, S ),
    insere(-utente(IdUt, Nome, Idade ,Morada) ),
    insere(negativo(IdUt,utente)),
    testaInv(S).
```

Segue-se um exemplo da inserção de um utente com um id inexistente na base de conhecimento negativa.

```
| ?- listing(negativo).
negativo(17, utente).
negativo(34, utente).
negativo(72, utente).
negativo(30, prestador).
negativo(45, prestador).
negativo(98, prestador).

| ?- evolucao(utente(40,'Sara',25,'Braga'),negativo).
yes

| ?- listing(negativo).
negativo(17, utente).
negativo(34, utente).
negativo(72, utente).
negativo(30, prestador).
negativo(45, prestador).
negativo(98, prestador).
negativo(40, utente).
```

O mesmo foi feito para o caso do prestador e do cuidado.

```
evolucao(prestador(IdPrest, Nome, Especialidade ,Instituicao) , Tipo
    ) :-
```

```
    Tipo == negativo,
    listaSolucoes( I, +(-prestador(IdPrest, Nome, Especialidade
        ,Instituicao)) ::I, S ),
    insere(-prestador(IdPrest, Nome, Especialidade ,Instituicao) ),
    insere(negativo(IdPrest,prestador)),
    testaInv(S).

evolucao(cuidado( Data,Hora,IdUt,IdPrest,Descricao,Custo ) , Tipo )
    :-
    Tipo == negativo,
    listaSolucoes( I, +(-cuidado(
        Data,Hora,IdUt,IdPrest,Descricao,Custo )) ::I, S ),
    insere(-cuidado( Data,Hora,IdUt,IdPrest,Descricao,Custo ) ),
    insere(negativoCuidado( Data,Hora,IdPrest,cuidado )),
    testaInv(S).
```

⇒ Conhecimento imperfeito incerto

De modo a permitir a evolução dos diferentes tipos de conhecimento, realizamos algumas alteração no predicado evolução.

Relativamento ao conhecimento imperfeito incerto adicionamos alguns termos ao predicado, como podemos observar no exemplo seguinte, que permite adicionar um utente cuja morada é desconhecida.

```
% Permite evolucao de utentes com morada desconhecida
evolucao( utente( IdUt,Nome,Idade,Morada ), Tipo, Desconhecido ) :-
    Tipo == incerto,
    Desconhecido == morada,
    listaSolucoes(I,
    incerto(utente(IdUt,Nome,Idade,desconhecido))::I,
    S),
    insere(utente(IdUt,Nome,Idade,desconhecido)),
    testaInv(S),
    insere(incerto(IdUt,utente)).
```

Como podemos verificar no exemplo seguinte, o sistema é capaz de inserir a utente Ana Silva cuja morada não é conhecida.

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?- evolucao(utente(19,'Ana Silva',34,desconhecido),incerto,morada).
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
utente(19, 'Ana Silva', 34, desconhecido).
yes
| ?-
```

De forma semelhante foi implementado para os restantes casos tais como:

- Utente/prestador com nome desconhecido
- Utente com morada desconhecida
- Prestador com instituição desconhecida

- Cuidado com custo desconhecido
- Cuidado com descrição desconhecida

⇒ **Conhecimento imperfeito interdito**

Para o caso do conhecimento interdito apresentamos também um exemplo do utente, em que não se conhece nem será possível conhecer a morada.

```
% Permite evolucao de utentes com morada interdita
evolucao( utente( IdUt,Nome,Idade,Morada ), Tipo, Desconhecido ) :-
    Tipo == interdito,
    Desconhecido == morada,
    listaSolucoes(I,
    interdito(utente(IdUt,Nome,Idade,semMorada))::I,
    S),
    insere(utente(IdUt,Nome,Idade,semMorada)),
    insere(interdito(IdUt,utente)),
    testaInv(S).
```

O sistema permite a evolução do conhecimento do utente Carlos Novais cuja morada é desconhecida e não será possível conhecer.

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?- evolucao(utente(20,'Carlos
    Novais',54,semMorada),interdito,morada).
```

```
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
utente(20, 'Carlos Novais', 54, semMorada).
yes
| ?-
```

O mesmo foi realizador de forma idêntica para:

- Utente sem idade
- Prestador sem instituição
- Cuidado sem custo
- Cuidado sem descrição

⇒ **Conhecimento imperfeito impreciso**

No caso da evolução do conhecimento impreciso optamos por representar todo as possibilidades do conhecimento numa lista. Para este foi necessário recorrer a um predicado que permite a remoção de conhecimento incerto, *removeIncerto* e ao predicado *removeUtente*, que apresentamos de seguida.

```
% remove conhecimento incerto de um utente
removeIncerto(IdUt,utente) :-
        incerto(IdUt,utente),
        remove(incerto(IdUt,utente)).
removeIncerto(IdUt,utente).
```

```
6
7  % remove conhecimento incerto de um cuidado
8  removeIncerto(Data,Hora,IdPrest,cuidado) :-
9          incertoCuidado(Data,Hora,IdPrest,cuidado),
10         remove(incertoCuidado(Data,Hora,IdPrest,cuidado)).
11 removeIncerto(Data,Hora,IdPrest,cuidado).
12
13 % remove conhecimento incerto de um prestador
14 removeIncerto(IdPrest,prestador) :-
15         incerto(IdPrest,prestador),
16         remove(incerto(IdPrest,prestador)).
17 removeIncerto(IdPrest,prestador).
```

Desenvolvemos este predicado de modo a permitir a remoção de utentes. O mesmo foi feito para permitir a remoção de prestadores e cuidados de forma semelhante, para auxiliarem a evolução do conhecimento impreciso destes.

```
1  % remove utente do conhecimento incerto
2  removeUtente(IdUt) :-
3        incerto(IdUt,utente),
4        remove(utente(IdUt,_,_,_)).
5  removeUtente(IdUt).
```

Apresentamos de seguida o exemplo do predicado que permitirá a evolução deste tipo de conhecimento para os utentes.

```
1  evolucao( [utente( IdUt,Nome,Idade,Morada ) | R], Tipo ) :-
2      Tipo = impreciso,
3      listaSolucoes( I, imp(utente(IdUt,Nome,Idade,Morada))::I, S ),
4      insere(excecao(utente(IdUt,Nome,Idade,Morada))),
5      insereImpreciso(R,IdUt,utente),
6      testaInv(S),
7      removeUtente(IdUt),
8      removeIncerto(IdUt,utente),
9      evolucao( R, impreciso ).
10 evolucao( [], tipo ).
```

Sendo assim, o sistema irá permitir a evolução da utente Julia Sousa, cuja idade

não se tem certeza ser 48.

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?- listing(impreciso).
impreciso(8, utente).
impreciso(10, utente).
impreciso(11, utente).
impreciso(9, utente).
impreciso(6, prestador).
impreciso(7, prestador).
impreciso(8, prestador).
impreciso(9, prestador).
yes
| ?- evolucao([utente(20,'Julia Sousa',48,'Braga')],impreciso).
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
```

```
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?- listing(impreciso).
impreciso(8, utente).
impreciso(10, utente).
impreciso(11, utente).
impreciso(9, utente).
impreciso(6, prestador).
impreciso(7, prestador).
impreciso(8, prestador).
impreciso(9, prestador).
impreciso(20, utente).
yes
| ?-
```

Elaboramos o mesmo predicado para o caso dos prestadores e dos cuidados, permitindo assim a evolução de conhecimento imperfeito destes.

4.4.2 Involução

Com os novos tipos de conhecimento foi também necessário alterar o predicado involução. De forma a eliminar conhecimento é necessário verificar se a remoção é válida. Para tal, recorre-se ao predicado listaSolucoes, que lista todos os invariantes associados. De seguida, através do testaInv averiguamos se estes são respeitados. Na eventualidadede não serem a remoção falha.

⇒ Conhecimento Positivo

Os predicados que se seguem pretendem eliminar utentes, prestadores ou cuidados na base de conhecimento positiva.

```
involucao(utente(IdUt, Nome,Idade,Morada), Tipo) :-
    Tipo == positivo,
    listaSolucoes(I, -utente(IdUt, Nome,Idade,Morada)::I, S),
    testaInv(S),
    remove(utente(IdUt, Nome,Idade,Morada)),
    remove(positivo(IdUt, utente)).
```

De seguida, é removido o utente adicionado anteriormente.

```
 | ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
utente(70, 'Adriana', 23, 'Braga').
 | ?- involucao(utente(70,'Adriana',23,'Braga'),positivo).
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
```

O mesmo foi feito para prestador e cuidado, de forma idêntica.

```
involucao( prestador(IdPrest, Nome,Especialidade,Instituicao), Tipo)
    :-
    Tipo == positivo,
    listaSolucoes(I,
        -prestador(IdPrest,Nome,Especialidade,Instituicao)::I, S),
    testaInv(S),
    remove(prestador(IdPrest, Nome, Especialidade,Instituicao)),
    remove(positivo(IdPrest,prestador)).

```

```
8  involucao(cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo), Tipo) :-
9      Tipo == positivo,
10     listaSolucoes(I,
           -cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo)::I, S),
11     testaInv(S),
12     remove(cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo)),
13     remove(positivoCuidado(Data,Hora,IdPrest,cuidado)).
```

⇒ Conhecimento Negativo

Os predicados que se seguem pretendem eliminar utentes, prestadores ou cuidados na base de conhecimento negativa.

```
1  involucao(utente(IdUt, Nome,Idade,Morada),Tipo ) :-
2      Tipo == negativo,
3      listaSolucoes( I, -(-utente(IdUt, Nome,Idade,Morada) )::I, S ),
4      testaInv(S),
5      remove(-utente(IdUt, Nome,Idade,Morada) ),
6      remove(negativo(IdUt, utente)).
```

De seguida, mostra-se o exemplo da remoção do utente inserido anteriormente.

```
1  | ?- listing(negativo).
2  negativo(17, utente).
3  negativo(34, utente).
4  negativo(72, utente).
5  negativo(30, prestador).
6  negativo(45, prestador).
7  negativo(98, prestador).
8  negativo(40, utente).
9  | ?- involucao(utente(40,'Sara',25,'Braga'),negativo).
10 yes
11 | ?- listing(negativo).
12 negativo(17, utente).
13 negativo(34, utente).
14 negativo(72, utente).
15 negativo(30, prestador).
16 negativo(45, prestador).
17 negativo(98, prestador).
18 yes
```

O mesmo foi feito para prestador e cuidado.

```
involucao( prestador(IdPrest,Nome,Especialidade,Instituicao),Tipo ) :-
    Tipo == negativo,
    listaSolucoes( I,
        -(-prestador(IdPrest,Nome,Especialidade,Instituicao) )::I, S ),
    testaInv(S),
    remove(-prestador(IdPrest,Nome,Especialidade,Instituicao) ),
    remove(negativo(IdPrest,prestador)).

involucao(cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo),Tipo ) :-
    Tipo == negativo,
    listaSolucoes( I,
        -(-cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo) )::I, S ),
    testaInv(S),
    remove(-cuidado(Data,Hora,IdUt,IdPrest,Descricao,Custo) ),
    remove(negativoCuidado(Data,Hora,IdPrest,cuidado)).
```

⇒ **Conhecimento imperfeito incerto**

Para o caso do conhecimento incerto temos os mesmo casos apresentados na secção da evolução. Da mesma forma, permitimos a involução de:

- Utente com morada desconhecida
- Utente com idade desconhecida
- Utente com nome desconhecido
- Prestador com instituição desconhecida
- Prestador com nome desconhecido
- Cuidado com custo desconhecido
- Cuidado com descrição desconhecida

Tendo em conta que estes são bastante semelhantes, apresentamos de seguida como exemplo o predicado que permite remover conhecimento incerto de utentes cujo nome é desconhecido.

```
involucao( utente(IdUt,Nome,Idade,Morada), Tipo, Desconhecido) :-
    Tipo == incerto,
    Desconhecido == nome,
    listaSolucoes(I, -inc(utente(IdUt,desconhecido,Idade,Morada))::I,
       S),
    testaInv(S),
    remove(utente(IdUt,desconhecido,Idade,Morada)),
    remove(incerto(IdUt,utente)).
```

Assim, o sistema consegue remover da base de conhecimento o utente com identificador 14 cujo nome não é conhecido.

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?- involucao(utente(14,desconhecido,65,'Algarve'),incerto,nome).
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
```

```
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?-
```

⇒ **Conhecimento imperfeito interdito**

O mesmo se verifica para o caso do conhecimento interdito, em que foram implementados predicados para os casos:

- Utente sem morada
- Utente sem idade
- Prestador sem instituição
- Cuidado sem custo
- Cuidado sem descrição

A implementação para todos é semelhante, pelo que iremos apresentar um exemplo. Este permite a remoção de utentes com morada desconhecida e não será possível conhecer.

```
involucao( utente(IdUt,Nome,Idade,Morada), Tipo, Desconhecido) :-
    Tipo == interdito,
    Desconhecido == morada,
    listaSolucoes(I, -intr(utente(IdUt,Nome,Idade,semMorada))::I, S),
    testaInv(S),
    remove(utente(IdUt,Nome,Idade,semMorada)),
    remove(interdito(IdUt,utente)).
```

Como esperado, torna-se possível remover conhecimento da utente Carlota Carvalho cuja morada não é conhecida nem será.

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
```

```
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?- involucao(utente(18,'Carlota
    Carvalho',25,semMorada),interdito,morada).
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
yes
| ?-
```

⇒ Conhecimento imperfeito impreciso

Para o caso do conhecimento ser impreciso será possível remover várias das possibilidades na lista do conhecimento. Assim, implementamos o predicado involução para os diferentes conhecimentos, utente, prestador e cuidado. Como exemplificação apresentamos o caso da involução para o utente.

```
involucao( [utente(IdUt,Nome,Idade,Morada) | R], Tipo ) :-
    Tipo == impreciso,
    listaSolucoes( Inv, -imp(utente(IdUt,Nome,Idade,Morada))::Inv, S),
    testaInv(S),
```

```
    remove(excecao(utente(IdUt,Nome,Idade,Morada))),
    remove(impreciso(IdUt,utente)),
    involucao( R, impreciso ).
involucao( [], Tipo ).
```

Como podemos constatar, assim, o sistema é capaz de remover conhecimento da utenta Juliana Fernandes cujo conhecimento é impreciso.

```
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?- listing(impreciso).
impreciso(8, utente).
impreciso(10, utente).
impreciso(11, utente).
impreciso(9, utente).
impreciso(6, prestador).
impreciso(7, prestador).
impreciso(8, prestador).
impreciso(9, prestador).
yes
| ?- involucao([utente(8,'Julia Fernandes',38,'Braga')],impreciso).
yes
| ?- listing(impreciso).
impreciso(10, utente).
impreciso(11, utente).
impreciso(9, utente).
impreciso(6, prestador).
impreciso(7, prestador).
```

```
impreciso(8, prestador).
impreciso(9, prestador).
yes
| ?- listing(utente).
utente(1, 'Joana Antunes', 18, 'Braga').
utente(2, 'Manuel Sousa', 37, 'Guimaraes').
utente(3, 'Ricardo Pimenta', 25, 'Braga').
utente(4, 'Carla Ribeiro', 67, 'Coimbra').
utente(5, 'Paula Costa', 21, 'Porto').
utente(6, 'Anabela Silva', 52, 'Braga').
utente(7, 'Rodrigo Bento', 9, 'Porto').
utente(12, 'Ricardo Lemos', desconhecido, 'Guarda').
utente(13, 'Andreia Silva', 17, desconhecido).
utente(14, desconhecido, 65, 'Algarve').
utente(16, 'Carlos Costa', semIdade, 'Lisboa').
utente(18, 'Carlota Carvalho', 25, semMorada).
yes
| ?-
```

4.5 Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas

Com o intuito de obter respostas relativamente a determinadas questões, tornou-se essencial desenvolver sistemas de inferências capazes de implementar os mecanismos de raciocício inerentes a estes.

Assim, começamos por implementar o caso simples em que o sistema apenas recebe uma questão. Na eventualidade de a questão estar na base de conhecimento a resposta será verdadeiro e se não estiver será falso. Caso não haja nenhuma informação acerca da mesma na base de conhecimento esta será desconhecido.

```
% Predicado demo:
% Questao, Resposta -> {V,F,D}
demo( Questao,verdadeiro ) :-
    Questao.
demo( Questao,falso ) :-
    -Questao.
demo( Questao,desconhecido ) :-
    nao( Questao ),
```

```
    nao( -Questao ).
```

A título de exemplo, recorremos ao prestador cujo identificador é 12 e verificamos que a informação é deconhecida. Tal deve-se ao facto, de este ser um conhecimento incerto, dado que não existe informação acerca da sua morada,tal como se pode observar de seguida:

```
| ?- listing(prestador).
prestador(1, 'Maria Lemos', 'Psiquiatria', 'Hospital de Braga').
prestador(2, 'Carla Sousa', 'Cardiologia', 'Hospital da Luz').
prestador(3, 'Catarina Machado', 'Oftalmologia', 'Hospital de Braga').
prestador(4, 'Antonio Costa', 'Clinica Geral', 'Santa Maria').
prestador(5, 'Ana Sousa', 'Obstetricia', 'CUF Coimbra').
prestador(10, desconhecido, 'Psiquiatria', 'CUF Porto').
prestador(11, 'Catarina Pereira', desconhecido, 'Hospital da Luz').
prestador(12, 'Fernando Silva', 'Cardiologia', desconhecido).
prestador(13, 'Antonieta Goncalves', 'Psiquiatria', semInstituicao)
```

Assim o resultado da *demo* é desconhecido, tal como expectável, pois não se sabe se este prestador realmente na trabalho no Hospital da Luz.

```
| ?- demo(prestador(12,'Fernando Silva','Cardiologia', 'Hospital da
    Luz'),R).
R = desconhecido ?
yes
| ?-
```

De seguida, elaboramos um sistema que recebe uma lista de questões e devolve uma lista com as respostas às mesmas,tal como se pode observar de seguida:

```
% meta-predicado demoLista:
% Lista de questoes, Lista de respostas -> {V,F, D}
demoLista([],[]).
demoLista([Q|T],[R|S]) :- demo(Q,R), demoLista(T,S).
```

Em baixo este será ilustrado com um exemplo, em que a primeira questão é verdadeira e a segunda é desconhecida.

```
| ?- demoLista([utente(1,'Joana
    Antunes',18,'Braga'),prestador(12,'Fernando Silva','Cardiologia',
    'Hospital da Luz')],R).
```

```
R = [verdadeiro,desconhecido] ?
yes
| ?-
```

Realizamos ainda um sistema de inferência, capaz de dar resposta à composição de um conjunto de questões, tal como se pode observar de seguida:

```
demoComposicao(Q && C, R) :- demo(Q,RQ), demoComposicao(C,RC),
    conjuncao(RQ,RC,R).
demoComposicao(Q $$ C, R) :- demo(Q,RQ), demoComposicao(C,RC),
    disjuncao(RQ,RC,R).
demoComposicao(Q, R) :- demo(Q,R).
```

Para definir este predicado, foi necessário definir os meta-predicado *conjuncao* e *disjuncao*. É importante referir que consideramos que && representa uma conjunção e $$ uma disjunção.

O resultado de uma conjunção só deverá ser verdadeiro, caso ambas as questões sejam verdadeiras e deverá ser falso na eventualidade de existir uma questão falsa. Nas restantes situações o resultado deverá ser desconhecido. Tal comportamento, pode ser verificado na tabela seguinte:

Questão	Questão	Resultado
Verdadeiro	Veradadeiro	Verdadeiro
Verdadeiro	Desconhecido	Desconhecido
Verdadeiro	Falso	Falso
Falso	Falso	Falso
Falso	Veradadeiro	Falso
Falso	Desconhecido	Falso
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Veradadeiro	Desconhecido
Desconhecido	Falso	Falso

Tabela 1. Resultado de uma conjunção

Na disjunção o resultado deverá ser verdadeiro sempre que uma das questões seja verdadeira e será falso caso ambas sejam falsas. Nas outras situações o resultado deverá ser desconhecido, como se pode observar de seguida:

Deste modo, de seguida será apresentado o resultado de uma conjunção de questões. Sendo que a primeira questão é verdadeira e a segunda será desconhecido, dado que a

Questão	Questão	Resultado
Verdadeiro	Veradadeiro	Verdadeiro
Verdadeiro	Desconhecido	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Falso	Falso
Falso	Veradadeiro	Verdadeiro
Falso	Desconhecido	Desconhecido
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Veradadeiro	Verdadeiro
Desconhecido	Falso	Desconhecido

Tabela 2. Resultado de uma disjunção

instituição do prestador com identificador 12 é desconhecida. Assim, o resultado será desconhecido, tal como esperado.

```
1 | ?- demoComposicao(utente(1,'Joana Antunes',18,'Braga') &&
    prestador(12,'Fernando Silva','Cardiologia', 'Hospital da Luz'),R).
2 R = desconhecido ?
3 yes
4 | ?-.
```

5 Conclusão

A elaboração deste projeto permitiu ao grupo consolidar os conhecimentos relativos à representação de conhecimento perfeito, positivo e negativo, bem como conhecimento imperfeito, com a utilização da ferramenta PROLOG. Neste tivemos o cuidado de permitir a evolução e involução destes tipos de conhecimento através da elaboração de invariantes que permitissem o progresso correto da base de conhecimento. Assim, o nosso projeto é capaz de representar o conhecimento pedido bem como a evolução e involução do mesmo.

Em suma, consideramos que os objetivos estabelecidos foram cumpridos. Tendo em conta que todas as funcionalidade propostas no enunciado foram implementadas bem como alguns extras, cosideramos que este exercício foi resolvido com sucesso.