Universidade do Minho

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho 1 - 2ª Fase

Grupo 34

Celso Rodrigues	Daniela Fernandes	Eduardo Araújo	José Gomes
A83655	A73768	A86012	A82418

3 de maio de 2021

Resumo

Este documento diz respeito à segunda fase do Trabalho Prático em Grupo da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio da Universidade do Minho.

Esta fase tem como objetivo utilização da extensão à programação em lógica, recorrendo ao PROLOG (linguagem de programação lógica), para representar conhecimento imperfeito, recorrendo à utilização de valores nulos.

Ao logo deste relatório vamos apresentar as formas de conhecimeno que adotamos, as características e funcionalidades do sistemas, assim como vamos esclarecer as estratégias que utilizamos para a realização do projeto.

De uma perspectiva geral, consideramos que a realização deste projeto foi relativamente bem sucedida, visto que pensamos ter cumprido com todos os requisitos mínimos propostos.

Conteúdo

1 Introdução

Para a segunda fase do trabalho de grupo, foi-nos proposto a demonstração de funcionalidades subjacentes à programação em lógica estendida e à representação do conhecimento imperfeito, num sistema com a capacidade para caracterizar um universo, na área da vacinação do covid-19.

Utilizando o sistema de representação do conhecimento que desenvolvemos na fase anterior, vamos ter que acrescentar os seguintes requisitos básicos:

- Representação do conhecimento positivo e negativo;
- Representar casos de conhecimento imperfeito, utilizando valores nulos de todos os tipos estudados;
- Manipular variantes que designem restrições à inserção e remoção de conhecimento no sistema;
- Lidar com a problemática da evolução do conhecimento, criando procedimentos adequados;
- Desenvolver um sistema de inferência capaz de implementar mecanismos de raciocínio inerentes a estes sistemas.

Com este projeto vamos utilizar a extensão à programação em lógica, usando a linguagem de programação PROLOG, no âmbito da representação de conhecimento imperfeito, recorrendo à utilização de valores nulos.

2 Preliminares

Na programação em lógica existem três pressupostos, que são:

- **Pressuposto dos Nomes Únicos -** duas constantes diferentes designam 2 entidades diferentes.
- **Pressuposto do Mundo Fechado -** toda a informação que não é mencionada é falsa.
- **Pressuposto do Domínio Fechado -** não há mais objetos no universo para além dos que designamos como constantes.

Na fase anterior, tivemos em conta todos os três pressupostos que enunciamos em cima. No entanto, como vamos abordar a extensão à programação em lógica, já não vamos ter em consideração o Pressuposto do Mundo Fechado. Vamos deixar de ter apenas o conhecimento *verdadeiro* e *falso*, vai ser considerado também o *desconhecido*.

Além disto, vamos aplicar dois novos conceitos, a *negação forte* e a *negação por falha*. A negação por falha não é suficiente para adquirir o conhecimento.

Relativamente ao conhecimento imperfeito, vamos poder representá-lo de três maneiras diferentes:

- **Incerto -** conhecimento acerca do qual não temos qualquer informação.
- **Impreciso -** conhecimento acerco do qual sabemos um conjunto ou intervalo de valores.
- **Interdito -** conhecimento que não pode ser representado nem conhecido.

3 Descrição do Trabalho e Análise dos Resultados

Nesta secção vamos apresentar a elaboração do trabalho que realizamos para resolver o problema que nos foi proposto. Além disso, vamos apresentar o conhecimento que representamos, os meta-predicados que desenvolvemos e umas demonstrações para comprovar a veracidade dos resultados obtidos.

3.1 Descrição do Sistema

Tal como feito na primeira fase do projeto, consideramos a representação do conhecimento da seguinte forma:

- *utente: idUtente, nº segurança social, nome, data_nasc, email, telefone, morada, profissão, [doencas_cron], centro_saude -> {V,F}*
- *staff: idStaff, idCentro, Nome, email -> {V,F}*
- *centro_saude: idCentro, nome, morada, telefone, email -> {V,F}*
- *vacinacao_covid: staff, utente, data, vacina, toma -> {V,F}*

3.2 Conhecimento positivo e negativo

Na primeira fase do projeto, apresentamos todo o conhecimento positivo, explicamos tudo o que faz parte da base do conhecimento que criamos. Nesta fase, vamos representar todo o conhecimento negativo.

Para o conhecimento negativo, recorremos à *negação forte*, indicamos que um facto é falso caso não exista na base de conhecimento que criamos e não possua uma exceção a si associada. Deste modo, concluimos que a informação é considerada *falsa*, caso não seja *verdadeira* ou *desconhecida*.

Para verificarmos a existência de uma determinada questão na base de conhecimento, criamos o meta-predicado **nao**, focada na negação por falha na prova. Na hipótese de uma questão não se encontrar na base de conhecimento o predicado vai indicar *verdadeiro*, utilizando o Pressuposto do mundo fechado.

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

Para representarmos o conhecimento negativo de um determinado predicado, vamos utilizar o meta-predicado **nao** relacionado com a negação por falha na prova e o **excecao** que vai indicar a presença do conhecimento imperfeito.

Neste exemplo, apresentamos o raciocínio para o predicado *utente* que vai servir que exemplos para os restantes predicados.

```
-utente(ID,S,No,D,E,T,M,P,L,C) :- nao(utente(ID,S,No,D,E,T,M,P,L,C)),
                                  nao(excecao(utente(ID,S,No,D,E,T,M,P,L,C))).
```

3.3 Representação de casos de conhecimento imperfeito

Na realização desta fase vamos representar o conhecimento imperfeito utilizando três tipos de valores nulos, que aprendemos nas aulas práticas e teóricas da unidade curricular:

- Incerto
- Interdito
- Impreciso

Como dissemos anteriormente o conhecimento imperfeito, trata-se de conhecimento *desconhecido* não de conhecimento *verdadeiro* ou *falso*. Para representarmos este tipo de conhecimento existente na base de conhecimento foi criado o meta-predicado *exceção*.

Deste modo, o conhecimento imperfeito pode ter associado a si uma *excecao* ou um *valor nulo*.

Para sabermos qual o tipo de conhecimento representado por uma questão, vamos recorrer ao meta-predicado **demo** que vai indicar se corresponde a um conhecimento *verdadeiro*, *falso* ou *desconhecido*.

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

Vamos apresentar nas proximações subsecções os diferentes casos que definimos de conhecimento imperfeito e os invariantes que definimos para a sua manipulação.

3.3.1 Conhecimento incerto

Neste conhecimento, não temos qualquer informação sobre o conhecimento desconhecido. Deste modo, vamos caracterizar o parâmetro desconhecido por um *valor nulo*, que vamos representar por *xpto*.

Vamos também associar uma **excecao** a esse valor, para que o conhecimento ser representado como *desconhecido*.

Definimos o conhecimento incerto para todos os predicados, utente, centro_saude, staff e vacinacao_covid, que temos definidos. Neste capítulo vamos mostrar alguns exemplos do que definimos para cada um dos predicados.

Começamos por mostrar como definimos o conhecimento incerto para **utente**:

- o atributo do **id**, substituimos por *x001* : o facto e a cláusula, indicam que para o utente *josué* não se conhece informação sobre o seu identificador de utente. Para verificarmos que o conhecimento é desconhecido, utilizamos o meta-predicado **demo**, que nos vai indicar *desconhecido* para qualquer ID que queiramos questionar sobre.

```
utente(x001, 152635, josue, date(1976,04,19), 'josueeee@ahgs.pt', 927643769, porto, bombeiro, [], centro_saude_porto).
excecao(utente(ID,S,No,D,E,T,M,P,L,C)) :- utente(x001,S,No,D,E,T,M,P,L,C).

cImperfeito(utente(ID,S,No,D,E,T,M,P,L,C),R) :- utente(x001,S,No,D,E,T,M,P,L,C),
                                                R = (utente(x001,S,No,D,E,T,M,P,L,C)).

cImperfeito(utente(ID,S,No,D,E,T,M,P,L,C),R) :- utente(x001,S,No,D,E,T,M,P,L,C) ,
                                                R = (excecao(utente(x001,S,No,D,E,T,M,P,L,C)) :- utente(x001,S,No,D,E,T,M,P,L,C)).
```

```
?- demo(utente(10,152635, josue, date(1976,04,19), 'josueeee@ahgs.pt',
 927643769, porto, bombeiro, [], centro_saude_porto),R).
R = desconhecido.

?- demo(utente(5,152635, josue, date(1976,04,19), 'josueeee@ahgs.pt',
927643769, porto, bombeiro, [], centro_saude_porto),R).
R = desconhecido.
```

Figura 1: Representação do conhecimento incerto acerca do ID do utente *Josué*

- o atributo do **nome**, substituímos por *x003* : o facto e a cláusula, indicam que para o utente cujo o ID seja *9*, não temos qualquer informação sobre o seu nome. Com o auxílio do meta-predicado **demo**, vai-nos indicar *desconhecido* para qualquer nome que tentemos associar.

```
utente(9, 152635, x003, date(1976,04,19), 'ariana@gmail.pt', 982327543, famalicao, secretaria, [], centro_saude_famalicao).
excecao(utente(ID,S,No,D,E,T,M,P,L,C)) :- utente(ID,S,x003,D,E,T,M,P,L,C).

cImperfeito(utente(ID,S,No,D,E,T,M,P,L,C),R) :- utente(ID,S,x003,D,E,T,M,P,L,C),
                                                R = (utente(ID,S,x003,D,E,T,M,P,L,C)).

cImperfeito(utente(ID,S,No,D,E,T,M,P,L,C),R) :- utente(ID,S,x003,D,E,T,M,P,L,C) ,
                                                R = (excecao(utente(ID,S,x003,D,E,T,M,P,L,C)) :- utente(ID,S,x003,D,E,T,M,P,L,C)).
```

```
[?- demo(utente(9, 152635, ariana, date(1976,04,19), 'ariana@gmail.pt',]
 982327543, famalicao, secretaria, [], centro_saude_famalicao),R).
R = desconhecido.

[?- demo(utente(9, 152635, joao, date(1976,04,19), 'ariana@gmail.pt', 9]
82327543, famalicao, secretaria, [], centro_saude_famalicao),R).
R = desconhecido.
```

Figura 2: Representação do conhecimento incerto acerca do *nome*do utente com ID=9.

- À semelhança deste fizemos o conhecimento incerto para todos os atributos de *utente*

Seguimos para o conhecimento incerto para o **centro_saude**:

- o atributo **id**, substituímos por *x011*: o facto e a cláusula, indicam que para o centro de saúde *centro_saude_guarda* não se conhece informação sobre o seu identificador de centro de saúde. Para verificarmos que o conhecimento é desconhecido, utilizamos o meta-predicado **demo**, que nos vai indicar *desconhecido* para qualquer ID que queiramos questionar sobre.

```
centro_saude(x011, centro_saude_guarda, guarda, 253424267, 'guarda@csg.pt').
excecao(centro_saude(ID,No,M,T,E)) :- centro_saude(x011,No,M,T,E).

cImperfeito(centro_saude(ID,No,M,T,E)) :- centro_saude(x011,No,M,T,E),
                                          R = (centro_saude(x011,No,M,T,E)).

cImperfeito(centro_saude(ID,No,M,T,E)) :- centro_saude(x011,No,M,T,E),
                                          R = (excecao(centro_saude(x011,No,M,T,E)) :- centro_saude(x011,No,M,T,E)).
```

```
[?- demo(centro_saude(5, centro_saude_guarda, guarda, 253424267, 'guard]
a@csg.pt'),R).
R = desconhecido.

[?- demo(centro_saude(20, centro_saude_guarda, guarda, 253424267, 'guar]
da@csg.pt'),R).
R = desconhecido.
```

Figura 3: Representação do conhecimento incerto acerca do ID do $centro_saude_guarda$

- o atributo **email**, substituímos por *x015* : o facto e a cláusula, indicam que para o centro de saúde cujo o ID seja *11*, não temos qualquer informação sobre o seu email. Com o auxílio do meta-predicado **demo**, vai-nos indicar *desconhecido* para qualquer email que tentemos associar.

```
centro_saude(11, centro_saude_viana, viana , 253427723, x015).
excecao(centro_saude(ID,No,M,T,E)) :- centro_saude(ID,No,M,T,x015).

cImperfeito(centro_saude(ID,No,M,T,E)) :- centro_saude(ID,No,M,T,x015),
                                          R = (centro_saude(ID,No,M,T,x015)).

cImperfeito(centro_saude(ID,No,M,T,E)) :- centro_saude(ID,No,M,T,x015),
                                          R = (excecao(centro_saude(ID,No,M,T,x015)) :- centro_saude(ID,No,M,T,x015)).
```

```
[?- demo(centro_saude(11, centro_saude_viana, viana , 253427723, 'email
@mn.p'),R).
R = desconhecido.
```

Figura 4: Representação do conhecimento incerto acerca do email do centro de saude com ID= 11

- À semelhança deste fizemos o conhecimento impreciso para todos os atributos de *centro_saude*

Agora o conhecimento incerto de **staff**:

- o atributo **idCentro**, substituímos por *x017* : o facto e a cláusula, indicam que para o staff cujo o ID seja *7*, não temos qualquer informação sobre o centro de saúde onde trabalha. Com o auxílio do meta-predicado **demo**, vai-nos indicar *desconhecido* para qualquer id de centro de saúde que tentemos associar.

```
staff(7,x017, bernardo, 'bernardo_guarda@csg.pt').
excecao(staff(ID,I,No,E)) :- staff(ID,x017,No,E).

cImperfeito(staff(ID,I,No,E)) :- staff(ID,x017,No,E),
                                 R = (staff(ID,x017,No,E)).

cImperfeito(staff(ID,I,No,E)) :- staff(ID,x017,No,E),
                                 R = (excecao(staff(ID,x017,No,E)) :- staff(ID,x017,No,E)).
```

```
[?- demo(staff(7,5, bernardo, 'bernardo_guarda@csg.pt'),R).
R = desconhecido.
```

Figura 5: Representação do conhecimento incerto acerca do id do centro de saúde que o staff com ID= 11 trabalha.

- À semelhança deste fizemos o conhecimento incerto para todos os atributos de *staff*

Por fim, para **vacinacao_covid**:

- o atributo **vacina**, substituímos por *x023* : o facto e a cláusula, indicam que para a vacina dada pelo staff com id = 1 ao utente com id = 5, não temos qualquer informação sobre a vacina que lhe é administrada. Com o auxílio do meta-predicado **demo**, vai-nos indicar *desconhecido* para qualquer vacina que tentemos associar.

```
vacinacao_covid(1, 5, '10-01-2021', x023, 2).
excecao(vacinacao_covid(Ids,Idu,D,V,T)) :- vacinacao_covid(Ids,Idu,D,x023,T).

cImperfeito(vacinacao_covid(Ids,Idu,D,V,T)) :- vacinacao_covid(Ids,Idu,D,x023,T),
                                               R = (vacinacao_covid(Ids,Idu,D,x023,T)).

cImperfeito(vacinacao_covid(Ids,Idu,D,V,T)) :- vacinacao_covid(Ids,Idu,D,x023,T),
                                               R = (excecao(vacinacao_covid(Ids,Idu,D,x023,T)) :- vacinacao_covid(Ids,Idu,D,x023,T)).
```

```
[?- demo(vacinacao_covid(1, 5, '10-01-2021', 'Pfizer', 2),R).          ]
R = desconhecido.

[?- demo(vacinacao_covid(1, 5, '10-01-2021', 'Moderna', 2),R).         ]
R = desconhecido.
```

Figura 6: Representação do conhecimento incerto acerca do nome da vacina que é adminitrada pelo o staff com id =1 ao utente com id = 5.

- À semelhança deste fizemos o conhecimento incerto para todos os atributos de *vacinacao_covid*

3.3.2 Conhecimento impreciso

Este conhecimento é desconhecido dentro de um conjunto de valores. Temos que representar nas *exceções* os casos em que se encontram nesse conjunto de valores, qualquer caso que não esteja dentro desse conjunto, é considerado falso se não estiver presente na base de conhecimento.

O conjunto de valores pode ser restrito a valores pontuais, como um intervalo de valores. Caso sejam casos pontuais, podemos representa-los recorrendo a um *facto*, os restantes podemos representá-los por uma *cláusula*.

Nesta capítulo vamos demonstrar alguns dos casos de conhecimento impreciso que definidos para os predicados **utente**, **centro_saude**, **staff** e **vacinacao_covid**.

Vamos começar pelo o **utente**:

- no **id** temos a dúvida qual será o id correspondente ao utente *mafalda*, sabemos que pode ser o *8* ou o *9*, que são valores pontuais de imprecisão, caso tentemos outro número, como o *10*, resultado será *falso*. Quando experimentamos o *8 ou 9* e com o auxílio do meta-predicado **demo**, vai-nos indicar *desconhecido* para qualquer um destes valores.

```
excecao(utente(8, 365337, mafalda, date(1998,05,29), 'mafalda@outlook.pt', 917236372, cascais, marketing,[], centro_saude_cascais)).
excecao(utente(9, 365337, mafalda, date(1998,05,29), 'mafalda@outlook.pt', 917236372, cascais, marketing,[], centro_saude_cascais)).

cImperfeito(utente(8, 365337, mafalda, date(1998,05,29), 'mafalda@outlook.pt', 917236372, cascais, marketing,[], centro_saude_cascais),R) :-
                    R = excecao(utente(8, 365337, mafalda, date(1998,05,29), 'mafalda@outlook.pt', 917236372, cascais, marketing,[], centro_saude_cascais)).

cImperfeito(utente(8, 365337, mafalda, date(1998,05,29), 'mafalda@outlook.pt', 917236372, cascais, marketing,[], centro_saude_cascais),R) :-
                    R = excecao(utente(8, 365337, mafalda, date(1998,05,29), 'mafalda@outlook.pt', 917236372, cascais, marketing,[], centro_saude_cascais)).
```

```
[?- demo(utente(8, 365337, mafalda, date(1998,05,29), 'mafalda@outlook.]
pt', 917236372, cascais, marketing,[], centro_saude_cascais),R).
R = desconhecido.

[?- demo(utente(9, 365337, mafalda, date(1998,05,29), 'mafalda@outlook.]
pt', 917236372, cascais, marketing,[], centro_saude_cascais),R).
R = desconhecido.

[?- demo(utente(10, 365337, mafalda, date(1998,05,29), 'mafalda@outlook]
.pt', 917236372, cascais, marketing,[], centro_saude_cascais),R).
[R = falso .                                                          ]
```

Figura 7: Representação do conhecimento impreciso relativamente ao id do utente *mafalda*.

- À semelhança deste fizemos o conhecimento impreciso para alguns dos atributos de *utente*.

De seguida passamos para o conhecimento imperfeito relativo ao **centro_saude**:

- Neste caso a dúvida surge relativamente ao **id** do centro de saúde da guarda que se encontra entre os valores de *9* e *15*, [9,15]. Com o auxílio do meta-predicado **demo**, se pretendermos saber se o **id** é igual a 9 ou 12 indica-nos que a informação é *desconhecida*, já quando pretendemos saber se é 20 a informação é *falsa*, pois não se encontra dentro dos valores estipulados.

```
excecao(centro_saude(I, centro_saude_guarda, guarda, 253764648, 'guarda@csg.pt')):- I >= 9, I=<15.
cImperfeito(centro_saude(I, centro_saude_guarda, guarda, 253764648, 'guarda@csg.pt'), R) :-
    R = (excecao(centro_saude(I, centro_saude_guarda, guarda, 253764648, 'guarda@csg.pt')) :-
    I >= 9,
    I =< 15).
```

```
[?- demo(centro_saude(9, centro_saude_guarda, guarda, 253764648, 'guard]
a@csg.pt'), R).
R = desconhecido.

[?- demo(centro_saude(12, centro_saude_guarda, guarda, 253764648, 'guar]
da@csg.pt'), R).
R = desconhecido.

[?- demo(centro_saude(20, centro_saude_guarda, guarda, 253764648, 'guar]
da@csg.pt'), R).
[R = falso .                                                          ]
```

Figura 8: Representação do conhecimento impreciso relativamente ao id do centro de saude da *guarda*.

- À semelhança deste fizemos o conhecimento impreciso para todos os atributos de *centro_saude*.

Para o predicado **satff** o conhecimento imperfeito:

- A dúvida é relativa a qual dos centros de saúde o staff *carmo* trabalha, o id do centro de saúde encontra-se entre *1* e *8*, isto é está dentro do seguinte intervalo [1,8]. Caso pretendamos saber se trabalho no centro de saúde com id=6, o meta-precidado **demo** indica-nos que a informação é *desconhecida*, já quando tentamos com o i = 11 (que não pertence ao intervalo) a informação é *falsa*.

```
excecao(staff(6,I, carmo, 'carmo_vilaverde@csvv.pt')):- I >= 1, I=<8.
cImperfeito(staff(6,I, carmo, 'carmo_vilaverde@csvv.pt'), R) :-
    R = (excecao(staff(6,I, carmo, 'carmo_vilaverde@csvv.pt')) :-
    I >= 1, I=<8).
```

```
?- demo(staff(6,6, carmo, 'carmo_vilaverde@csvv.pt'),R).
R = desconhecido.

?- demo(staff(6,8, carmo, 'carmo_vilaverde@csvv.pt'),R).
R = desconhecido.

?- demo(staff(6,11, carmo, 'carmo_vilaverde@csvv.pt'),R).
R = falso .
```

Figura 9: Representação do conhecimento impreciso relativamente ao id do centro de saude que o staff com o id = 6 trabalha.

- À semelhança deste fizemos o conhecimento impreciso para todos os atributos de *staff*

Por fim, o conhecimento imperfeito relativo à **vacinacao_covid**:

- no nome da **vacina** temos a dúvida qual será a vacina administrada ao utente cujo *id = 2* , sabemos que pode ser a *Moderna* ou a *Pfizer*, que são valores pontuais de imprecisão, caso tentemos outra vacina, como o *Astrazeneca*, resultado será *falso*. Com o auxílio do meta-predicado **demo**, vai-nos indicar *desconhecido* para quando queremos saber se a informação relativa à vacina da Pfizer e da Moderna, já quando tentamos saber a informação relativamente à Astrazeneca dá-nos *falso*.

```
excecao(vacinacao_covid(7, 2, '02-05-2021', 'Moderna',1)).
excecao(vacinacao_covid(7, 2, '02-05-2021', 'Pfizer',1)).

cImperfeito(vacinacao_covid(7, 2, '02-05-2021', 'Moderna',1),R) :-
                        R = excecao(vacinacao_covid(7, 2, '02-05-2021', 'Moderna',1)).

cImperfeito(vacinacao_covid(7, 2, '02-05-2021', 'Pfizer',1),R) :-
                        R = excecao(vacinacao_covid(7, 2, '02-05-2021', 'Pfizer',1)).
```

```
[?- demo(vacinacao_covid(7, 2, '02-05-2021', 'Moderna',1),R).          ]
 R = desconhecido.

[?- demo(vacinacao_covid(7, 2, '02-05-2021', 'Pfizer',1),R).           ]
 R = desconhecido.

[?- demo(vacinacao_covid(7, 2, '02-05-2021', 'Astrazeneca',1),R).      ]
[R = falso .                                                           ]
```

Figura 10: Representação do conhecimento impreciso relativamente ao id do centro de saude que o staff com o id = 6 trabalha.

- À semelhança deste fizemos o conhecimento impreciso para todos os atributos de *vacinacao_covid*

3.3.3 Conhecimento interdito

Neste tipo de conhecimento, o conhecimento não deve ser especificado ou conhecido. Para indicar que é proibido é necessário a atribuição de um valor nulo ao conhecimento que queremos "bloquear".

Tal como explicamos em cima, não deve ser permitido incluir informação sobre o conhecimento *interdito* por isso criamos um meta-predicado **nulo** para caracterizar estes valores.

Vamos utilizar este meta-predicado num invariante para impedir a inserção de qualquer conhecimento interdito. No entanto, este conhecimento não deixa de ser *desconhecido* e devemos por isso associar uma **exceção**.

Foi necessário criar um invariante de inserção, que não permitisse a inclusão de factos na base de conhecimento que indicassem o conhecimento que nós definimos como interdito.

Começamos por definir o conhecimento imperfeito para o predicado **utente** pretende-se que a profissão do utente 8 seja interdita, para isso atribuimos-lhe o valor *np1* ao parâmetro correspondente à profissão, para que não se pudesse representar e conhecer essa informação.

```
utente(8, 9261376, luis,  date(1997,08,18), 'luis@outlook.pt', 91272635, geres, np1, [], centro_saude_geres).
excecao(utente(ID,S,No,D,E,T,M,P,L,C)) :- utente(ID,S,No,D,E,T,M,np1,L,C).
nulo(np1).
+utente(ID,S,No,D,E,T,M,P,L,C) :: (solucoes(Profissao,(utente(8, 9261376, luis,  date(1997,08,18), 'luis@outlook.pt', 91272635, geres, Profissao, [],
                comprimento(L,N), N==0).
```

Para o **centro de saúde** pretende-se que a morada do centro de saúde 8 fosse interdita, por isso atribuimos-lhe o valor *np11* ao parâmetro que lhe é correspondente, para que não se pudesse representar e conhecer essa informação.

```
centro_saude(8, centro_saude_paredes, np11, 25262537, 'paredes@csvpc.pt').
excecao(centro_saude(ID,No,M,T,E)) :- centro_saude(ID,No, np11,T,E).
nulo(np11).
+centro_saude(ID,No,M,T,E) :: (solucoes(Morada,(centro_saude(8, centro_saude_paredes, Morada, 25262537, 'paredes@csvpc.pt'), nao(nulo(Morada))),L),
               comprimento(L,N), N==0).
```

Para o **staff** pretende-se que o id da staff *idalete* fosse interdita, por isso atribuimos-lhe o valor *np16* ao parâmetro que lhe é correspondente, para que não se pudesse representar e conhecer essa informação

```
staff(np16,5, idalete, 'idalete_esposende@csve.pt').
excecao(staff(ID,I,No,E)) :- staff(np16,I,No,E).
nulo(np16).
+staff(ID,I,No,E) :: (solucoes(IDs,(staff(IDs,5, idalete, 'idalete_esposende@csve.pt'), nao(nulo(IDs))),L),
                comprimento(L,N), N==0).
```

E por fim para a **vacinacao_covid** pretende-se que o nome da vacina que foi administrada ao utente com id 7 fosse interdita, por isso atribuimos-lhe o valor *np23* ao parâmetro que lhe é correspondente, para que não se pudesse representar e conhecer essa informação

```
vacinacao_covid(3, 7, '18-02-2021', np23,1).
excecao(vacinacao_covid(Ids,Idu,D,V,T)) :- vacinacao_covid(Ids,Idu,D,np23,T).
nulo(np23).
+vacinacao_covid(Ids,Idu,D,V,T) :: (solucoes(Vacina,(vacinacao_covid(3, 7, '18-02-2021', Vacina,1), nao(nulo(Vacina))),L),
                comprimento(L,N), N==0).
```

3.4 Manipulação de invariantes

Criamos alguns invariantes para impedir a inconsitência na base de conhecimento.

- **Invariantes relativos ao nulo interdito**

Esta invariante foi criado para o caso de haver conhecimento interdito na base de conhecimento. O invariante vai impedir a inserção da informação sobre um dado conhecimento interdito.

Criamos este tipo de invariantes para todos os predicados que enunciamos no capítulo anterior:

Para o utente:

```
+utente(ID,S,No,D,E,T,M,P,L,C) :: (solucoes(Profissao,(utente(8, 9261376, luis,  date(1997,08,18), 'luis@outlook.pt', 91272635, geres, Profissao, [],
                comprimento(L,N), N==0).
```

Para o centro_saude:

```
+centro_saude(ID,No,M,T,E) :: (solucoes(Morada,(centro_saude(8, centro_saude_paredes, Morada, 25262537, 'paredes@csvpc.pt'), nao(nulo(Morada))),L),
                comprimento(L,N), N==0).
```

Para o staff:

```
+staff(ID,I,No,E) :: (solucoes(IDs,(staff(IDs,5, idalete, 'idalete_esposende@csve.pt'), nao(nulo(IDs))),L),
                comprimento(L,N), N==0).
```

E por fim para a vacinacao_covid:

```
+vacinacao_covid(Ids,Idu,D,V,T) :: (solucoes(Vacina,(vacinacao_covid(3, 7, '18-02-2021', Vacina,1), nao(nulo(Vacina))),L),
                comprimento(L,N), N==0).
```

- **Invariante que não permite a inserção de conhecimento negativo repetido**

Para um determinado *id* pode existir mais do que um facto de conhecimento negativo mas não pode existir conhecimento negativo repetido, ou seja, o mesmo facto não pode existir mais do que uma vez.

```
+(-Q) :: (solucoes(Q, clause(-Q, true), S),
        comprimento(S,N),
        N =< 1).
```

- **Invariante que não permitem contradições**

Quando efetuamos algumas atualizações na base de conhecimento que criamos, precisamos de comprovar que não existem contradições com o conhecimento que inserimos com o que já lá estava, ou seja, que não existem dois tipos de conhecimento associados à mesma informação.

```
% Invariante nao permite que haja o mesmo conhecimento negativo e positivo ao mesmo tempo
+Q :: nao(-Q).
+(-Q) :: nao(Q).

% Invariante nao permite que haja o mesmo conhecimento negativo e desconhecido
+(-Q) :: (solucoes(Q,clause(excecao(Q), true),S),
                comprimento(S,N),
                N == 0).
```

- **Invariantes que impedem a inserção do mesmo id para conhecimento positivo**

Neste tipo de conhecimento, só podemos ter uma representação de conhecimento para cada *id* de um predicado. Por isso criamos o invariante que impede que sejam inseridas informações com o mesmo id.

```
% Não permite que existam IDs iguais nos utentes

+utente(ID,S,No,D,E,T,M,P,L,C) :: (solucoes((ID),(utente(ID,A,B,C,F,G,H,I,J,K)),S),
                                   comprimento(S,N), N =< 1).

% Não permite que existam IDs iguais nos centros de centro_saude

+centro_saude(ID,No,M,T,E) :: (solucoes((ID), (centro_saude(ID,X,Y,Z,W)),S)
                               comprimento(S,N), N =< 1).

% Não permite que existem IDs iguais no satff

+staff(ID,I,No,E) :: (solucoes((ID), (staff(ID,X,Y,Z)), S),
                     comprimento(S,N), N =< 1).
```

3.5 Evolução do conhecimento

Para resolução do problema de evolução do conhecimento, recorreu-se à criação de meta-predicados que permitissem a atualização de conhecimento já existente na base de conhecimento, a adição de novo conhecimento, ou inserir conhecimento imperfeito. Para realizar esta tarefa foram tidos em conta algunss fatores.

Dado que para atualizar conhecimento incerto ou impreciso para conhecimento positivo de forma correta, as exceções associadas ao conhecimento a atualizar deverão ser eliminadas.

3.5.1 Inserir conhecimento perfeito

Para atualizar o conhecimento existente e caso seja conhecimento perfeito, desenvolvemos o predicado **atualiza** que garante que as atualizações na base de conhecimento sejam bem feitas.

Para a **evolucao** do conhecimento, efetuamos uma atualização antes do conhecimento ser inserido:

```
evolucao( Termo ) :-
    solucoes( Invariante,+Termo::Invariante,Lista ),
    atualiza( Termo ),
    teste( Lista ).
```

Para o meta-predicado **atualiza** conseguir resolver todos os problemas que se vão encontrando.

- **Conhecimento positivo para conhecimento positivo**

Não pode existir conhecimento repedito positivo para um mesmo *id*, então o conhecimento vai ser alterado para poder pertencer à base de conhecimento.

Para inserir informação positiva, precisa-se de eliminar a base de conhecimento desatualizada e inserir a nova atualizada.

Fez-se o mesmo raciocínio para **utente**, **centro_saude** e **staff**:

```
% Conhecimento positivo -> Conhecimento positivo (utente)
atualiza(utente(ID,S,No,D,E,T,M,P,L,C)) :-
                nao(utente(ID,S,No,D,E,T,M,P,L,C)),
                nao(excecao(utente(ID,S,No,D,E,T,M,P,L,C))),
                solucoes((utente(ID,_,_,_,_,_,_,_,_,_)),
                        (utente(ID,_,_,_,_,_,_,_,_,_)),
                        R),
                remove(R),
                insere(utente(ID,S,No,D,E,T,M,P,L,C)).

% Conhecimento positivo -> Conhecimento positivo (centro_saude)
atualiza(centro_saude(ID,No,M,T,E)) :-
                nao(centro_saude(ID,No,M,T,E)),
                nao(excecao(centro_saude(ID,No,M,T,E))),
                solucoes((centro_saude(ID,_,_,_,_)),
                        (centro_saude(ID,_,_,_,_)),
                        R),
                remove(R),
                insere(centro_saude(ID,No,M,T,E)).

% Conhecimento positivo -> Conhecimento positivo (satff)
atualiza(staff(ID,I,No,E)) :-
                nao(staff(ID,I,No,E)),
                nao(excecao(staff(ID,I,No,E))),
                solucoes((staff(ID,_,_,_)),
                        staff(ID,_,_,_),
                        R),
                remove(R),
                insere(staff(ID,I,No,E)).
```

- **Conhecimento incerto/impreciso para positivo**

Apesar de existirem restrições na base de conhecimento relativo ao conhecimento desconhecido de um dado predicado, essa informação pode ser adicionada à base de conhecimento. Para isso, assumimos que para qualquer conhecimento incerto ou impreciso que seja positivo, iria ser inserido na base de conhecimento e as suas exceções eliminadas.

Criamos os meta-predicados **cImperfeito** e **solucoes** que ao inserir na base de conhecimento conhecimento imperfeito irá determinar as cláusulas que dizem que o conhecimento é imperfeito relativo à informação que queremos adicionar e depois vai eliminá-las. O conhecimento deixa de ser imperfeito e passa a ser positivo.

```
% Conhecimento incerto/impreciso -> Conhecimento positivo
atualiza(Q) :- demo(Q,desconhecido),
           solucoes(C,
                   (cImperfeito(Q,C)),
                   R),
           remove(R),
           insere(Q).
```

- **Conhecimento incerto/impreciso para conhecimento negativo**

No que toca a conhecimento incerto, este pode ser atualizado com conhecimento negativo sendo que não é necessário realizar qualquer tipo de alteração. O mesmo acontece para o conhecimento impreciso especificado por um conjunto específico de valores. No entanto, quando se trata de conhecimento impreciso definido por valores pontuais, pode acontecer

de a informação negativa que estamos a inserir se encontre expressa como informação desconhecida. Isto é, pode acontecer de o facto que pretendemos inserir como negativo, já se encontrar especificado por uma *excecao*. Neste caso é necessário eliminar essa exceçao, visto que caso não o fizéssemos a base de conhecimento deixaria de estar consistente.

```
atualiza(-Q) :- clause(excecao(Q),true),
                remove(excecao(Q)),
                insere(-Q).
```

- **Conhecimento negativo para positivo e vice-versa**

Embora possa existir conhecimento negativo, essa informação pode ser considerada positiva a qualquer momento, isto é, pode-se alterar o conhecimento negativo para positivo, desde que o negativo seja eliminado da base de conhecimento, para não haverem contradições.

```
atualiza(Q) :- clause(-Q,true),
               remove(-Q),
               insere(Q).
```

Da mesma forma que o conhecimento negativo pode virar positivo, o contrário também pode acontecer. A informação positiva pode ser negada a qualquer momento desde que seja retirada da base de conhecimento. Caso o conhecimento que se quer negar coincida com conhecimento incerto, não é possível se negar.

```
atualiza(-Q) :- solucoes(Q, excecao(Q), S),
                comprimento(S,N),
                N == 0,
                clause(Q,true),
                remove(Q),
                insere(-Q).
```

- **Inserir conhecimento positivo/negativo**

Caso a inserção de conhecimento perfeito não seja relativa a informação já existente na base de conhecimento, então não é necessário atualizar a base de conhecimento. Desta forma a única tarefa necessária será a verificação dos invariantes, que é feita pelo meta-predicado **evolucao**. Posto isto, a cláusula do meta-predicado **atualiza** limita-se a inserir a informação na base de conhecimento para que esta seja testada.

```
atualiza(-Q) :- solucoes(Q, excecao(Q), S),
                comprimento(S,N),
                N == 0,
                clause(Q,true),
                remove(Q),
                insere(-Q).
```

3.5.2 Inserir conhecimento imperfeito

Estratégia de como passamos o conhecimento imperfeito para conhecimento perfeito.

Como dissemos em secções anteriores existem três categorias de conhecimento imperfeito, o *incerto*, o *impreciso* e o *interdito* que não tem capacidade de ser evoluído e por isso não será analisado neste sub capítulo.

- **Incerto**

Para sermos capazes de transformar o conhecimento incerto em conhecimento perfeito é necessário, primeiro, remover todas as exceções referentes ao determinado conhecimento e, em seguida, atualizar o conhecimento do utente que é desconhecido.

É de referir, também, que foi criado um predicado para todos os tipos de informação de utentes, centros de saúde, staffs e vacinação covid, de modo a ser possível evoluir conhecimento incerto em toda a base de conhecimento.

Aqui vamos mostrar dois exemplos que fizemos para o predicado **utente.**

```
evolucaoIncSSutente(utente(ID,S,No,D,E,T,M,P,Do,C)) :-
                    demo(utente(ID,S,No,D,E,T,M,P,Do,C),desconhecido),
                    solucoes(excecao(utente(ID,S,No,D,E,T,M,P,Do,C))) :- utente(ID,X,No,D,E,T,M,P,Do,C),
                                                                         (utente(ID,X,No,D,E,T,M,P,Do,C),nao(nulo(X)),L),
                    removeL(L),
                    remove(utente(ID,X,No,D,E,T,M,P,Do,C)),
                    evolucao(utente(ID,S,No,D,E,T,M,P,Do,C)).


% nome UTENTE
evolucaoIncNutente(utente(ID,S,No,D,E,T,M,P,Do,C)) :-
                    demo(utente(ID,S,No,D,E,T,M,P,Do,C),desconhecido),
                    solucoes(excecao(utente(ID,S,No,D,E,T,M,P,Do,C))) :- utente(ID,S,X,D,E,T,M,P,Do,C),
                                                                         (utente(ID,S,X,D,E,T,M,P,Do,C),nao(nulo(X)),L),
                    removeL(L),
                    remove(utente(ID,S,X,D,E,T,M,P,Do,C)),
                    evolucao(utente(ID,S,No,D,E,T,M,P,Do,C)).
```

- **Impreciso**

Neste caso, de forma análoga ao anterior, é imperativo remover as exceções e adicionar o caso de conhecimento perfeito corretamente. Por outro lado, este não necessita de remover o conhecimento anterior pelo facto de ser impossível criar dois ou mais casos idênticos em que apenas um valor diferia de um para todos os outros.

É de referir, também, que foi criado um predicado para todos os tipos de informação de utentes, centros de saúde, staffs e vacinação covid, de modo a ser possível evoluir conhecimento impreciso em toda a base de conhecimento.

```
%evolucao do conhecimento impreciso UTENTE
evolucaoUtImpreciso(utente(ID,Ss,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro)):-
                demo(utente(ID,Ss,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro),desconhecido),
                solucoes(excecao(utente(ID,S,No,D,E,T,M,P,Do,C)), excecao(utente(ID,S,No,D,E,T,M,P,Do,C)),L),
                removeL(L),
                evolucao(utente(ID,Ss,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro)).

%evolucao do conhecimento impreciso CENTRO_SAUDE
evolucaoCSImpreciso(centro_saude(ID,Nome,Morada,Telefone,Email)):-
                demo(centro_saude(ID,Nome,Morada,Telefone,Email),desconhecido),
                solucoes(excecao(centro_saude(ID,No,M,T,E)), excecao(centro_saude(ID,No,M,T,E)),L),
                removeL(L),
                evolucao(centro_saude(ID,Nome,Morada,Telefone,Email)).


%evolucao do conhecimento impreciso STAFF
evolucaoSImpreciso(staff(ID,IDutente,Nome,Email)):-
                demo(staff(ID,IDutente,Nome,Email),desconhecido),
                solucoes(excecao(staff(ID,IDu,N,E)), excecao(staff(ID,IDu,N,E)),L),
                removeL(L),
                evolucao(staff(ID,IDutente,Nome,Email)).

%evolucao do conhecimento impreciso VACINACAO_COVID
evolucaoVImpreciso(vacinacao_covid(IDstaff,IDutente,Data,Vacina,Toma)):-
                demo(vacinacao_covid(IDstaff,IDutente,Data,Vacina,Toma),desconhecido),
                solucoes(excecao(vacinacao_covid(IDs,IDu,D,V,T)), excecao(vacinacao_covid(IDs,IDu,D,V,T)),L),
                removeL(L),
                evolucao(vacinacao_covid(IDstaff,IDutente,Data,Vacina,Toma)).
```

Criamos também predicados para inserir informação imprecisa de cada um dos predicados. Vamos mostrar uns exemplos para utentes, seguidos a mesma estratégia para os outros.

```
%Inserir UTENTE impreciso
% para o nº segurança social
inserirUtImpreciso(utente(ID,[],No,D,E,T,M,P,Do,C)).
inserirUtImpreciso(utente(ID,[X|Y],No,D,E,T,M,P,Do,C)) :-
            evolucao( (excecao(utente(ID,X,No,D,E,T,M,P,Do,C)))),
            inserirUtImpreciso(utente(ID,Y,No,D,E,T,M,P,Do,C)).


% para o nome

inserirUtImpreciso(utente(ID,S,[],D,E,T,M,P,Do,C)).
inserirUtImpreciso(utente(ID,S,[X|Y],D,E,T,M,P,Do,C)) :-
            evolucao( (excecao(utente(ID,S,X,D,E,T,M,P,Do,C)))),
            inserirUtImpreciso(utente(ID,S,Y,D,E,T,M,P,Do,C)).
```

3.6 Sistema de Inferência

Nesta fase tivemos que criar um meta-predicado novo que é bastante semelhante ao **demo**, o demo infere sobre a veracidade de apenas uma questão. Já o novo meta-predicado **demoC** vai deduzir a veracidade de uma composição de questões, essa composição vai ser constituída por *conjunções* e *disjunções*.

Com este novo meta-predicado vamos conseguir analisar com mais precisão o maior número de informação.

```
demoC(Q1 e Q2, R) :- demo(Q1,R1), demoC(Q2,R2), conjuncao(R1,R2,R).
demoC(Q1 ou Q2, R) :- demo(Q1,R1), demoC(Q2,R2), disjuncao(R1,R2,R).
demoC(Q,R) :- demo(Q,R).
```

Criamos os meta-predicados **conjuncao** e **disjuncao** para testar de as conjunções e disjunções do conhecimento são verdadeiras, falsas ou desconhecidas, obdecendo às regras das mesmas.

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

Tivemos que definir novos operadores o **e** e o **ou** para que o meta-predicado demoC pudesse analisar as composições.

```
:- op( 900,xfy,'e' ).
:- op( 900,xfy,'ou' ).
```

4 Conclusão

Depois da realização desta fase do Trabalho Prático consideramos que foi relativamente bem sucedida, visto que pensamos que cumprimos com todos os requisitos.

Ao longo da realização do projeto notamos uma maior facilidade de aplicação da linguagem PROLOG comparativamente à primeira fase do trabalho.

Sobre o exercício em si, sentimos que conseguimos aplicar todos os conteúdos relacionados com conhecimento negativo e imperfeito mesmo com alguma dificuldade. No entanto, é de destacar o esforço feito para que o sistema produzido seja capaz de evoluir o conhecimento imperfeito presente no seu conhecimento (retirando desta funcionalidade o conhecimento interdito, que não possui essa capacidade).