Sistemas de Representação de Conhecimento e Raciocínio

Trabalho realizado por:

João Figueiredo Martins Peixe dos Santos

Francisco Alves Andrade

Luís Filipe Cruz Sobral

Paulo Silva Sousa

A89520 João Santos

A89474 Luís Sobral

A89465 Paulo Sousa

A89513 Francisco Andrade

Grupo 14
Projeto SRCR
2020/2021
Universidade do Minho

Conteúdo

Índice de Figuras

1 Introdução

No âmbito da Unidade Curricular de Sistemas de Reconhecimento de Conhecimento e Raciocínio foi-nos proposto o desenvolvimento de um sistema de representação de conhecimento e raciocínio capaz de caracterizar um universo de discurso na área de vacinação global da população portuguesa no contexto da pandemia que atualmente vivemos.

Nesta fase continuamos o trabalho desenvolvido anteriormente, acrescentando conhecimento negativo e conhecimento imperfeito.

Tal como na fase anterior, para a construção dos mecanismos de raciocínio e representação de conhecimento será utilizada a linguagem de programação em lógica PROLOG.

2 Preliminares

Existem três pressupostos fundamentais no âmbito da programação em lógica:

- **Pressuposto dos nomes únicos**: duas constantes diferentes designam duas entidades diferentes.
- **Pressuposto do mundo fechado**: toda a informação inexistente é falsa.
- **Pressuposto do dominio fechado**: não há mais objetos no universo para além dos designados por constantes.

Na primeira fase do trabalho foram tidos em conta os pressupostos anteriores. Contudo, agora será abordada a extensão à programação em lógica, deste modo deveremos abandonar o segundo pressuposto (Pressuposto do mundo fechado). Assim, deixará de apenas existir conhecimento perfeito (conhecimento verdadeiro e falso), mas também existirá conhecimento imperfeito, ou seja, algo desconhecido.

Este conhecimento imperfeito poderá ser caracterizado de três formas diferentes:

- **Incerto**: conhecimento sobre o qual não temos qualquer informação, mas poderemos vir a ter.
- **Impreciso**: conhecimento para o qual temos um conjunto/intervalo de valores.
- **Interdito**: conhecimento sobre o qual não temos informação e no futuro também não teremos.

Agregando toda esta informação consideramo-nos habilitados a prosseguir para o desenvolvimento desta fase do trabalho.

3 Descrição do Trabalho e Resultados

Seguidamente apresentaremos o trabalho desenvolvido através da apresentação do conhecimento representado e da explicação dos predicados e meta-predicados envolvidos, cuja veracidade será comprovada pela exibição de valores experimentais.

3.1 Conhecimento negativo

Um facto é considerado falso se a informação não for verdadeira nem desconhecida, assim representaremos o conhecimento negativo. Este conhecimento será representado com base na negação forte, onde o facto será falso se não existir na base do conhecimento nem possuir uma exceção associada a si.

Desta forma elaboramos um meta-predicado, baseado em falha na prova, que verifica a existência de uma questão na base de conhecimento. Este meta-predicado é o *nao* que indicará verdadeiro caso a questão não se encontre na base de conhecimento.

```
% Extensao do meta-predicado nao: Questao -> {V,F}
nao( Questao ) :- Questao, !, fail.
nao( Questao ).
```

Figura 1: Meta-predicado *nao*

Contamos também com o meta-predicado exceção que identifica a presença de conhecimento imperfeito, este predicado será melhor analisado posteriormente. Com estes dois predicados conseguimos passar à negação forte (acima referida), uma vez que são utilizados como condições na cláusula de conhecimento negativo, como pode ser comprovado para o predicado utente (em baixo explícito). Os restantes predicados seguem o mesmo raciocínio.

```
-utente(10,12345678,'Antonio',2000,'antonio@gmail.com',965269469,'Viana Castelo','Estudante',[],
1).

%-----------------------------------------------
% Aplicação da negação forte para o predicado utente
-utente(ID, NSS, Nome, Data_Nasc, Email, Telefone, Morada, Profissao, LDoencas, Centro_Saude) :-
    nao(utente(ID, NSS, Nome, Data_Nasc, Email, Telefone, Morada, Profissao, LDoencas,
    Centro_Saude)),
    nao(excecao(utente(ID, NSS, Nome, Data_Nasc, Email, Telefone, Morada, Profissao, LDoencas,
    Centro_Saude))).
```

Figura 2: Conhecimento Negativo para o predicado *utente*

3.2 Conhecimento Imperfeito

Neste trabalho o conhecimento imperfeito é representado através de três tipos de valores nulos: incerto, impreciso e interdito. O conhecimento imperfeito será representado no nosso sistema como informação desconhecida. Para tal elaboramos um meta-predicado para podermos obter o valor da informação, predicado esse que dá pelo nome de *demo*, que indicará se o conhecimento tem valor verdadeiro, falso ou desconhecido.

```
%------------------------------ - - - - - - - - - -  -  -  -  -  -
% Extensao do meta-predicado demo: Questao,Resposta -> {V,F}
%                           Resposta = { verdadeiro,falso,desconhecido }

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

Figura 3: Predicado *demo*

3.2.1 Conhecimento Imperfeito Incerto

No conhecimento imperfeito incerto, não existe qualquer informação acerca de pelo menos um dos campos de um facto. Desta forma, esse campo terá uma label de modo a conseguir identificar o campo sem informação disponível. Para além disto, a este facto é associada uma exceção para o conhecimento representado ser identificado como desconhecido. Em baixo segue um exemplo de conhecimento imperfeito incerto.

```
utente(10,12345678,'Antonio', 2000,email_desc,965269469,'Viana Castelo','Estudante',[],1).
```

Figura 4: Facto do Utente Antonio

```
% Conhecimento imperfeito incerto para o predicado utente

excecao(utente(ID,SS,Nome,Data_Nasc,_,Telefone,Morada,Profissao,LDoencas,Centro_Saude)) :-
utente(ID,SS,Nome,Data_Nasc,email_desc,Telefone,Morada,Profissao,LDoencas,Centro_Saude).
emailIncertoUtente(10).
```

Figura 5: Conhecimento Imperfeito Incerto para o predicado *utente*

O facto e cláusula acima indicam que para o *utente antonio* se desconhece qualquer informação acerca do seu email. Como tal o campo *email* do utente tem uma label representativa, *email_ desc*. Aplicando agora o predicado *demo* verificamos que qualquer valor atribuído ao campo *email* do utente será indicada a presença de conhecimento desconhecido.

```
?- demo(utente(10,12345678,'Antonio', 2000, 'Antonio@gmail.com' ,965269469,'Viana Castelo','Estudante',[],1), R).
R = desconhecido.
```

Figura 6: Meta-predicado *demo* para demonstração de conhecimento incerto

Seguindo o raciocínio utilizado para este predicado, representamos, também, conhecimento imperfeito incerto para os predicados *staff*, *centro_ saude* e *vacinacao_ Covid*.

3.2.2 Conhecimento Imperfeito Impreciso

O conhecimento imperfeito impreciso caracteriza-se por a informação de pelo menos um campo de um facto se encontrar num determinado conjunto/intervalo de valores, não sendo, tal como o nome indica, um valor exato. Ainda, qualquer valor fora desse mesmo conjunto/intervalo é considerado falso.

Esta imprecisão pode ser referente a uma restrição de valores pontuais ou então a um conjunto de valores limitado dentro de um determinado domínio. Como tal o conhecimento imperfeito impreciso pode ser representado de duas formas diferentes. Caso se tratem de valores pontuais pode-se representar o conhecimento imperfeito através de factos, caso contrário deve ser representado por uma cláusula. Em seguida são demonstrados estes dois tipos de representação:

```
excecao(utente(14,65748392,'Tiago',1990,'t@gmail.com',925434567,'Tomar','Canalizador',[],2)).
excecao(utente(14,65748392,'Tiago',1990,'t@gmail.com',925434567,'Tomar','Eletricista',[],2)).
profissaoImprecisaUtente(14).
```

Figura 7: Conhecimento Imperfeito Impreciso para valores pontuais

A incerteza relativa à profissão do *utente Tiago* encontra-se apenas entre dois valores pontuais, 'Canalizador' e 'Eletricista'. Por outro lado, quanto à data de nascimento do *utente Nuno* sabemos que se encontra no intervalo [1960, 1965], sendo assim necessária a utilização de uma cláusula.

```
excecao(utente(15,56789123,'Nuno',Data,'n@gmail.com',912345678,'Albufeira','Mecanico',[],1))
:-  integer(Data), Data >= 1960, Data =< 1965.
dataNascimentoImprecisaUtente(15).
```

Figura 8: Conhecimento Imperfeito Impreciso para um conjunto de valores

```
?- demo(utente(15,56789123,'Nuno',1960,'n@gmail.com',912345678,'Albufeira','Mecanico',[],1), R).
R = desconhecido.

?- demo(utente(15,56789123,'Nuno',1965,'n@gmail.com',912345678,'Albufeira','Mecanico',[],1), R).
R = desconhecido.

?- demo(utente(15,56789123,'Nuno',1955,'n@gmail.com',912345678,'Albufeira','Mecanico',[],1), R).
R = falso .

?- demo(utente(15,56789123,'Nuno',1975,'n@gmail.com',912345678,'Albufeira','Mecanico',[],1), R).
R = falso .
```

Figura 9: Meta-predicado *demo* para a demonstração de conhecimento imperfeito impreciso

Novamente, através da utilização do meta-predicado *demo* podemos verificar se o conhecimento é imperfeito, neste caso impreciso, atribuindo valores pertencentes ao intervalo estipulado para a data de nascimento do *utente Nuno* verificamos que o conhecimento é desconhecido, e para valores não contemplados no intervalo o conhecimento é falso.

Seguindo o mesmo raciocínio utilizado para o predicado *utente*, representamos conhecimento imperfeito impreciso para os predicados *staff*, *centro_saude* e *vacinacao_Covid*.

3.2.3 Conhecimento Imperfeito Interdito

O conhecimento imperfeito interdito, o último tipo de conhecimento imperfeito contemplado neste trabalho, caracteriza-se pela presença de informação não especificada e que não será conhecida. Como tal, atribuímos um valor nulo ao conhecimento que pretendemos "bloquear". Para caracterizar estes valores nulos interditos criamos o meta-predicado *nulo* que, de facto, os caracteriza.

Como este conhecimento não deixa de ser desconhecido é necessário associar-lhe uma exceção.

O seguinte exemplo demonstra como este conhecimento foi representado.

```
utente(17,nss_interdito,'Evaristo',1976,'e@gmail.com',922222222,'Leixoes','Pedreiro',[],2).
```

Figura 10: Utente com número de segurança social interdito

```
excecao(utente(ID,_,Nome,Data_Nasc,Email,Telefone,Morada,Profissao,LDoencas,Centro_Saude)) :-
utente(ID,nss_interdito,Nome,Data_Nasc,Email,Telefone,Morada,Profissao,LDoencas,Centro_Saude).

nulo(nss_interdito).
+utente(_,_,_,_,_,_,_,_,_,_) :: (solucoes( NSS, (utente(17,NSS,'Evaristo',1976,'e@gmail.com',
922222222,'Leixoes','Pedreiro',[],2),
                                nao( nulo(NSS))), S),
                                comprimento(S, N),
                                N == 0).
```

Figura 11: Conhecimento imperfeito interdito para o predicado *utente*

Como o valor do número de segurança social do *utente Evaristo* é um valor interdito, foi necessária a criação de uma label que o identificasse (*nss_interdito*). Ainda, adicionamos uma exceção de modo a que o valor deste conhecimento fosse dado como desconhecido pelo predicado *demo*, como podemos verificar em baixo.

```
?- demo(utente(17,12314,'Evaristo',1976,'e@gmail.com',922222222,'Leixoes','Pedreiro',[],2), R).
R = desconhecido.
```

Figura 12: Conhecimento imperfeito interdito para o predicado *utente*

3.3 Evolução do Conhecimento

Para que fosse possível a evolução do conhecimento, procedemos à criação de meta-predicados que permitissem a atualização de conhecimento existente, adição de novo conhecimento e inserção de conhecimento imperfeito.

3.3.1 Inserção de Conhecimento Perfeito

Apesar de já estar contemplada no trabalho realizado por nós na primeira fase, passamos a explicar novamente a inserção de conhecimento perfeito, uma vez que será necessária na atualização de conhecimento imperfeito para conhecimneto perfeito. O meta-predicado por nós criado foi o *evolucao* que depois de reunir os invariantes de inserção(explicados na fase um do trabalho) insere o *Termo* na base de conhecimento caso os teste do Termo segundo os invariantes sejam passados.

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

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

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

Figura 13: Meta-predicado *evolucao*

3.3.2 Remoção de Conhecimento

Tal como na secção anterior a remoção de conhecimento está contemplada na primeira fase deste trabalho, no entanto também é essencial na atualização de conhecimento imperfeito para perfeito.

O meta-predicado por nós criado foi o *involucao* que reúne os invariantes de remoção(explicados na fase um do trabalho) e, caso estes sejam cumpridos, remove o conhecimento da base de conhecimento.

```
involucao( Termo ) :-
    solucoes( Invariante,-Termo::Invariante,Lista ),
    teste( Lista ),
    remocao( Termo ).


remocao( Termo ) :-
    retract( Termo ).
remocao( Termo ) :-
    assert( Termo ),!,fail.
```

Figura 14: Meta-predicado *involucao*

3.3.3 Atualização de Conhecimento Imperfeito Incerto para Conhecimento Perfeito

Embora na base de conhecimento existir conhecimento desconhecido, este pode ser atualizado. Aquando da atualização, como está conhecimento positivo na base de dados que consideramos correto, o conhecimento imperfeito seria introduzido na base de conhecimento como perfeito e as exceções associadas a este seriam removidas.

Desta forma passamos a explicar um caso de atualização de conhecimento imperfeito incerto para conhecimento perfeito. Através do predicado *registaUtente* procedemos a esta atualização. Primeiro passa pela identificação do conhecimento incerto e do campo crítico do predicado através de um predicado identificativo do mesmo e do ID do utente a ele associado, neste caso o predicado *emailIncertoUtente*. Em seguida procedemos à remoção do utente da base de conhecimento e à remoção do predicado identificativo do conhecimento imperfeito através do predicado *involucao*. Por último inserimos o conhecimento perfeito na base de conhecimento através do predicado *evolucao*.

```
% Adicionar Email Utente (incerto)
registaUtente(ID,Q,Nome,Data,E,T,M,P,D,C) :- emailIncertoUtente(ID),
                                             involucao(utente(ID,Q,Nome,Data,email_desc,T,M,P,D,
                                             C)),
                                             involucao(emailIncertoUtente(ID)),
                                             evolucao(utente(ID,Q,Nome,Data,E,T,M,P,D,C)).
```

Figura 15: Atualização de conhecimento imperfeito incerto para conhecimento perfeito

Em baixo, através do predicado *utenteT* verificamos a atualização com sucesso do utente.

```
?- registaUtente(10,12345678,'Antonio',2000,'ant@gmail.com',965269469,'Viana Castelo','Estudante',[],1)
|    .
true .

?- utenteT.
:- dynamic utente/10.

utente(1, 21455655, 'Luis', 1970, 'luis@gmail.com', 936696454, 'Adaufe', 'Estudante', ['Doenca coronaria', 'Doenca respiratoria'], 1).
utente(2, 31455655, 'Filipe', 1969, 'filipe@gmail.com', 936696100, 'Barcelos', 'Estudante', ['Diabetes'], 1).
utente(3, 41455655, 'Jose', 1990, 'j@gmail.com', 936696151, 'Esposende', 'Estudante', [], 1).
utente(4, 42455655, 'Diogo', 1998, 'd@gmail.com', 936696171, 'Aveiro', 'Estudante', ['Doenca coronaria'], 1).
utente(5, 43455655, 'Rui', 1936, 'r@gmail.com', 936696190, 'Braga', 'Estudante', ['Diabetes'], 1).
utente(6, 41454355, 'Paulo', 1980, 'p@gmail.com', 936738461, 'Esposende', 'Medico', [], 2).
utente(7, 75355655, 'Beatriz', 1995, 'b@gmail.com', 936973645, 'Aveiro', 'Enfermeiro', [], 2).
utente(8, 43455692, 'Maria', 1951, 'm@gmail.com', 912382248, 'Braga', 'Estudante', [], 2).
utente(9, 42487435, 'Carlos', 1983, 'c@gmail.com', 936827634, 'Aveiro', 'Estudante', ['Artroses'], 2).
utente(11, 87654321, 'Ricardo', 1983, 'ri@gmail.com', 965269477, morada_Desc, 'Padeiro', [], 1).
utente(12, 13243546, 'Bruno', 1993, email_desc, 965233333, 'Portimao', 'Ferreiro', [], 2).
utente(13, 86756453, 'Fernando', 1989, 'f@gmail.com', 967269477, morada_Desc, 'Motorista', [], 3).
utente(16, 56789321, 'Vasco', 1957, email_interdito, 912345679, 'Porto', 'Contabilista', ['Artroses'], 3).
utente(17, nss_interdito, 'Evaristo', 1976, 'e@gmail.com', 922222222, 'Leixoes', 'Pedreiro', [], 2).
utente(10, 12345678, 'Antonio', 2000, 'ant@gmail.com', 965269469, 'Viana Castelo', 'Estudante', [], 1).

true.
```

Figura 16: Verificação da atualização

3.3.4 Atualização de Conhecimento Imperfeito Impreciso para Conhecimento Perfeito

Tal como na atualização de conhecimento imperfeito incerto para conhecimento perfeito, ao passar de conhecimento imperfeito impreciso para conhecimento perfeito consideramos a informação já introduzida na base de conhecimento como correta, atualizando o facto para que seja caracterizado por conhecimento perfeito, inserindo-o na base de conhecimento e eliminando as exceções a ele associadas.

Recorrendo ao predicado identificativo do conhecimento impreciso, que segue o padrão *parâmetro do facto, Impreciso, Predicado* (ex: *profissaoImprecisaUtente*), ao predicado *removeExcecoes*, que, tal como o nome indica, remove as exceções, e aos meta-predicados *solucoes*, *excecao*, *comprimento* e *evolucao*, conseguimos atualizar o conhecimento imperfeito impreciso para conhecimento perfeito.

```
removeExcecoes([]).
removeExcecoes([H|T]) :- involucao(H), removeExcecoes(T).
```

Figura 17: Predicado *removeExcecoes*

Pegando no exemplo *registaUtente*, que atualiza a profissão de um utente, podemos demonstrar o processo levado para a atualização do conhecimento.

Primeiramente, é necessário encontrar o predicado associado ao conhecimento impreciso para profissão de um Utente. Em seguida, encontrar todas as exceções associadas ao utente, através do meta-predicado *solucoes*, e removê-las da base de conhecimento. Por fim, inserir o conhecimento perfeito na base de conhecimento através do meta-predicado *solucoes*.

```
registaUtente(ID,Q,Nome,Data,E,T,M,P,D,C) :- profissaoImprecisaUtente(ID),
                                             solucoes(excecao(utente(ID,Q,Nome,Data,E,T,M,Ps,D,
                                             C)), (excecao(utente(ID,Q,Nome,Data,E,T,M,Ps,D,C)))
                                             , R),
                                             comprimento(R,N), N>0, removeExcecoes(R),
                                             evolucao(utente(ID,Q,Nome,Data,E,T,M,P,D,C)).
```

Figura 18: Atualização de conhecimento imperfeito impreciso para conhecimento perfeito

Este raciocínio por nós desenvolvido é comprovado pela utilização do predicado *utenteT* que, através da sua utilização, conseguimos verificar a atualização do conhecimento, como podemos comprovar em baixo.

```
?- registaUtente(14,65748392,'Tiago',1990,'t@gmail.com',925434567,'Tomar','Canalizador',[],2).
true .

?- utenteT.
:- dynamic utente/10.

utente(1, 21455655, 'Luis', 1970, 'luis@gmail.com', 936696454, 'Adaufe', 'Estudante', ['Doenca coronaria', 'Doenca respiratoria'], 1).
utente(2, 31455655, 'Filipe', 1969, 'filipe@gmail.com', 936696100, 'Barcelos', 'Estudante', ['Diabetes'], 1).
utente(3, 41455655, 'Jose', 1990, 'j@gmail.com', 936696151, 'Esposende', 'Estudante', [], 1).
utente(4, 42455655, 'Diogo', 1998, 'd@gmail.com', 936696171, 'Aveiro', 'Estudante', ['Doenca coronaria'], 1).
utente(5, 43455655, 'Rui', 1936, 'r@gmail.com', 936696190, 'Braga', 'Estudante', ['Diabetes'], 1).
utente(6, 41454355, 'Paulo', 1980, 'p@gmail.com', 936738461, 'Esposende', 'Medico', [], 2).
utente(7, 75355655, 'Beatriz', 1995, 'b@gmail.com', 936973645, 'Aveiro', 'Enfermeiro', [], 2).
utente(8, 43455692, 'Maria', 1951, 'm@gmail.com', 912348248, 'Braga', 'Estudante', [], 2).
utente(9, 42487435, 'Carlos', 1983, 'c@gmail.com', 936827634, 'Aveiro', 'Estudante', ['Artroses'], 2).
utente(11, 87654321, 'Ricardo', 1983, 'ri@gmail.com', 965269477, morada_Desc, 'Padeiro', [], 1).
utente(12, 13243546, 'Bruno', 1993, email_desc, 965233333, 'Portimao', 'Ferreiro', [], 2).
utente(13, 86756453, 'Fernando', 1989, 'f@gmail.com', 967269477, morada_Desc, 'Motorista', [], 3).
utente(16, 56789321, 'Vasco', 1957, email_interdito, 912345679, 'Porto', 'Contabilista', ['Artroses'], 3).
utente(17, nss_interdito, 'Evaristo', 1976, 'e@gmail.com', 922222222, 'Leixoes', 'Pedreiro', [], 2).
utente(10, 12345678, 'Antonio', 2000, 'ant@gmail.com', 965269469, 'Viana Castelo', 'Estudante', [], 1).
utente(14, 65748392, 'Tiago', 1990, 't@gmail.com', 925434567, 'Tomar', 'Canalizador', [], 2).

true.
```

Figura 19: Validação da atualização de conhecimento

3.3.5 Inserção de Conhecimento Imperfeito Incerto

De modo a ter uma evolução de conhecimento mais completa considerámos a possibilidade de inserção de conhecimento imperfeito, neste caso incerto.

Quando pretendemos inserir conhecimento imperfeito incerto na base de conhecimento e não existe referência dessa mesma informação, necessitamos de registar também as exceções e o predicado identificativo de conhecimento imperfeito.

Como exemplo usamos o predicado *registaUtenteMoradaIncerta* que regista um Utente sem ter qualquer informação sobre a morada deste. Para tal necessitamos de utilizar outros predicados e meta-predicados como *evolucaoMoradaIncerta*, *evolucaoDesconhecido*, *assert* e *obtemInvariantes*.

```
evolucaoDesconhecido(Termo) :-
    obtemInvariantes( Termo, LI1, LI2),
    insercao(Termo),
    teste( LI1 ), teste( LI2 ).
```

Figura 20: Predicado *evolucaoDesconhecido*

```
obtemInvariantes(Termo, LI1, LI2) :-
    solucoes(Invariante, +Termo:::Invariante, LI1),
    solucoes(Invariante, +Termo::Invariante, LI2).
```

Figura 21: Predicado *obtemInvariantes*

O meta-predicado *obtemInvariantes* permite obter os invariantes de inserção de conhecimento perfeito e imperfeito. O meta-predicado *evolucaoDesconhecido* reúne os invariantes de inserção através do predicado *obtemInvariantes* e insere o Termo na base de conhecimento se os invariantes de inserção forem validados. O predicado *evolucaoMoradaIncerta* insere o conhecimento imperfeito incerto na base de conhecimento através do predicado *evolucaoDesconhecido* e insere também o predicado identificativo do conhecimento imperfeito incerto associado à morada de um Utente.

```
registaUtenteMoradaIncerta(ID, NSS, Nome, Data, Email, Telefone, Profissao, LDoencas, Centro)
:- evolucaoMoradaIncerta(utente(ID, NSS, Nome, Data, Email, Telefone, morada_Desc,Profissao,
LDoencas, Centro)).

evolucaoMoradaIncerta(utente(ID, NSS, Nome, Data, Email, Telefone, morada_Desc,Profissao,
LDoencas, Centro)) :- evolucaoDesconhecido(utente(ID, NSS, Nome, Data, Email, Telefone, Morada,
Profissao, LDoencas, Centro)), assert(moradaIncertaUtente(ID)).
```

Figura 22: Predicado que regista um Utente sem informação sobre a sua morada

Este processo pode ser comprovado pelos predicados *utenteT* e *listing(moradaIncertaUtente)*, como podemos ver de seguida.

```
?- registaUtenteMoradaIncerta(20, 12312312, 'Leandro', 1999, 'leandro@gmail.com', 912828282, 'Estudante', [], 1).
true .

?- utenteT.
:- dynamic utente/10.

utente(1, 21455655, 'Luis', 1970, 'luis@gmail.com', 936696454, 'Adaufe', 'Estudante', ['Doenca coronaria', 'Doenca respiratoria'], 1).
utente(2, 31455655, 'Filipe', 1969, 'filipe@gmail.com', 936696100, 'Barcelos', 'Estudante', ['Diabetes'], 1).
utente(3, 41455655, 'Jose', 1990, 'j@gmail.com', 936696151, 'Esposende', 'Estudante', [], 1).
utente(4, 42455655, 'Diogo', 1998, 'd@gmail.com', 936696171, 'Aveiro', 'Estudante', ['Doenca coronaria'], 1).
utente(5, 43455655, 'Rui', 1936, 'r@gmail.com', 936696190, 'Braga', 'Estudante', ['Diabetes'], 1).
utente(6, 41454355, 'Paulo', 1980, 'p@gmail.com', 936738461, 'Esposende', 'Medico', [], 2).
utente(7, 75355655, 'Beatriz', 1995, 'b@gmail.com', 936973645, 'Aveiro', 'Enfermeiro', [], 2).
utente(8, 43455692, 'Maria', 1951, 'm@gmail.com', 912348248, 'Braga', 'Estudante', [], 2).
utente(9, 42487435, 'Carlos', 1983, 'c@gmail.com', 936827634, 'Aveiro', 'Estudante', ['Artroses'], 2).
utente(11, 87654321, 'Ricardo', 1983, 'ri@gmail.com', 965269477, morada_Desc, 'Padeiro', [], 1).
utente(12, 13243546, 'Bruno', 1993, email_desc, 965233333, 'Portimao', 'Ferreiro', [], 2).
utente(13, 86756453, 'Fernando', 1989, 'f@gmail.com', 967269477, morada_Desc, 'Motorista', [], 3).
utente(16, 56789321, 'Vasco', 1957, email_interdito, 912345679, 'Porto', 'Contabilista', ['Artroses'], 3).
utente(17, nss_interdito, 'Evaristo', 1976, 'e@gmail.com', 922222222, 'Leixoes', 'Pedreiro', [], 2).
utente(10, 12345678, 'Antonio', 2000, 'ant@gmail.com', 965269469, 'Viana Castelo', 'Estudante', [], 1).
utente(14, 65748392, 'Tiago', 1990, 't@gmail.com', 925434567, 'Tomar', 'Canalizador', [], 2).
utente(20, 12312312, 'Leandro', 1999, 'leandro@gmail.com', 912828282, _, 'Estudante', [], 1).

true.

?- listing(moradaIncertaUtente).
:- dynamic moradaIncertaUtente/1.

moradaIncertaUtente(11).
moradaIncertaUtente(13).
moradaIncertaUtente(20).

true.
```

Figura 23: Validação da inserção de conhecimento imperfeito incerto

3.3.6 Inserção de Conhecimento Imperfeito Impreciso

Para a inserção de conhecimento imperfeito impreciso na base de conhecimento é necessária também a inserção de todas as exceções e predicados identificativos de conhecimento imperfeito impreciso associados ao facto em questão.

Como tal, consideramos valores imprecisos do email de um membro do staff a ser inserido para demonstrar a inserção deste conhecimento.

Primeiramente, utilizamos o predicado *registaStaffEmailImpreciso* que chama o predicado *evolucaoEmailImpreciso* para inserção das exceções e o meta-predicado *assert* para inserir o facto (*emailImprecisoStaff*), identificativo do conhecimento imperfeito impreciso relativo ao email de um membro do staff, na base de conhecimento.

O predicado *evolucaoEmailImpreciso* consiste na inserção recursiva das exceções relativas ao email de um membro do staff na base de conhecimento. Desta forma, reúne os invariantes de inserção e insere uma exceção do valor de email do staff caso os invariantes sejam validados. Por fim, cria recursividade chamando-se a si própria para introduzir as exceções dos restantes emails.

```
registaStaffEmailImpreciso(ID, Centro, Nome,[Email|T]) :- evolucaoEmailImpreciso(staff(ID,
Centro, Nome, [Email|T])), assert( emailImprecisoStaff(ID) ).

evolucaoEmailImpreciso(staff(ID, Centro, Nome, [])).
evolucaoEmailImpreciso(staff(ID, Centro, Nome,[Email|T])) :- obtemInvariantes(excecao(staff(ID,
Centro, Nome,Email)), LI1, LI2),
                    insercao(excecao(staff(ID, Centro, Nome,Email))), teste( LI1 ), teste(
                    LI2 ),
                    evolucaoEmailImpreciso( staff(ID, Centro, Nome,T)).
```

Figura 24: Predicado que regista um membro do Staff com vários valores possíveis para o seu Email

Esta inserção pode ser validada pela utilização dos predicados *staffT* e *listing(excecao)*.

```
?- registaStaffEmailImpreciso(15, 2, 'Ze', ['z@gmail.com', 'zezoca@gmail.com']).
true .
```

Figura 25: Validação da inserção de conhecimento imperfeito impreciso

```
?- listing(excecao).
:- dynamic excecao/1.

excecao(utente(ID, SS, Nome, Data_Nasc, _, Telefone, Morada, Profissao, LDoencas, Centro_Saude)) :-
    utente(ID,
           SS,
           Nome,
           Data_Nasc,
           email_desc,
           Telefone,
           Morada,
           Profissao,
           LDoencas,
           Centro_Saude).
excecao(utente(ID, SS, Nome, Data_Nasc, Email, Telefone, _, Profissao, LDoencas, Centro_Saude)) :-
    utente(ID,
           SS,
           Nome,
           Data_Nasc,
           Email,
           Telefone,
           morada_Desc,
           Profissao,
           LDoencas,
           Centro_Saude).
excecao(centro_saude(ID, _, Morada, Telefone, Email)) :-
    centro_saude(ID, nome_desc, Morada, Telefone, Email).
excecao(staff(ID, Centro, Nome, _)) :-
    staff(ID, Centro, Nome, email_desc).
excecao(utente(14, 65748392, 'Tiago', 1990, 't@gmail.com', 925434567, 'Tomar', 'Canalizador', [], 2)).
excecao(utente(14, 65748392, 'Tiago', 1990, 't@gmail.com', 925434567, 'Tomar', 'Eletricista', [], 2)).
excecao(utente(15, 56789123, 'Nuno', Data, 'n@gmail.com', 912345678, 'Albufeira', 'Mecanico', [], 1)) :-
    integer(Data),
    Data>=1960,
    Data=<1965.
excecao(centro_saude(4, 'GuardaH', 'Guarda', 999999996, 'guardah@gmail.com')).
excecao(centro_saude(4, 'GuardaH', 'Guarda', 999999996, 'hguarda@gmail.com')).
excecao(staff(6, 1, 'Bernardo', 'bernardo@gmail.com')).
excecao(staff(6, 1, 'Bernardo', 'bern@gmail.com')).
excecao(utente(ID, SS, Nome, Data_Nasc, _, Telefone, Morada, Profissao, LDoencas, Centro_Saude)) :-
    utente(ID,
           SS,
           Nome,
           Data_Nasc,
           email_interdito,
           Telefone,
           Morada,
           Profissao,
           LDoencas,
           Centro_Saude).
excecao(utente(ID, _, Nome, Data_Nasc, Email, Telefone, Morada, Profissao, LDoencas, Centro_Saude)) :-
    utente(ID,
           nss_interdito,
           Nome,
           Data_Nasc,
           Email,
           Telefone,
           Morada,
           Profissao,
           LDoencas,
           Centro_Saude).
excecao(vacinacao_Covid(2, 9, Dia, 6, 2021, 'PFizer', 1)) :-
    integer(Dia),
    Dia>=1,
    Dia=<31.
excecao(vacinacao_Covid(ID_Staff, 11, 2, 4, 2021, 'AstraZeneca', 1)) :-
    contains(ID_Staff, [1, 3]).
excecao(staff(15, 2, 'Ze', 'z@gmail.com')).
excecao(staff(15, 2, 'Ze', 'zezoca@gmail.com')).

true.
```

Figura 26: Validação da inserção de conhecimento imperfeito impreciso

3.3.7 Inserção de Conhecimento Imperfeito Interdito

Ao inserir conhecimento imperfeito interdito na base de conhecimento temos de ter em consideração a adição de outras componentes, nomeadamente exceções associadas, factos representativos do conhecimento imperfeito interdito e um invariente de inserção que impossibilita a alteração do valor interdito.

Como exemplo para esta inserção de conhecimento imperfeito interdito temos o predicado *registaUtenteNSSInterdito* que tem como cláusula a negação do predicado *integer(NSS)* e o predicado *evolucaoNSSInterdito*.

O predicado *nao(integer(NSS))* é utilizado para impedir a evolução com um número de segurança social válido, já o predicado *evolucaoNSSInterdito* tratará da evolução do conhecimento imperfeito interdito.

Este predicado obtém os invariantes de inserção para poder inserir o novo utente com NSS interdito caso os invariantes sejam validados. Em seguida, insere na base de conhecimento o predicado responsável pela caracterização de conhecimento imperfeito interdito, o predicado *nulo*, insere também a exceção associada a este novo utente. Por fim, cria um invariante que representa o conhecimento imperfeito interdito que consiste na confirmação da inexistência de utentes onde o NSS interdito introduzido seja falso

para o facto *nulo(NSS)*.

```
registaUtenteNSSInterdito(ID, NSS, Nome, Data, Email, Telefone, Morada, Profissao, LDoencas,
Centro) :- nao(integer(NSS)), evolucaoNSSInterdito(utente(ID, NSS, Nome, Data, Email, Telefone,
Morada,Profissao, LDoencas, Centro)).

evolucaoNSSInterdito(utente(ID, NSSInterdito, Nome, Data, Email, Telefone, Morada, Profissao,
LDoencas, Centro)) :-
                    obtemInvariantes(utente(ID, NSSInterdito, Nome, Data, Email, Telefone,
                    Morada, Profissao, LDoencas, Centro), LI1, LI2),
                    insercao(utente(ID, NSSInterdito, Nome, Data, Email, Telefone, Morada,
                    Profissao, LDoencas, Centro)), teste(LI1), teste(LI2),
                    assert(nulo(NSSInterdito)),
                    assert((excecao(utente(ID1, NSS1, Nome1, Data1, Email1, Telefone1, Morada1,
                    Profissao1, LDoencas1, Centro1)) :- utente(ID1, NSSinterdito, Nome1, Data1,
                    Email1, Telefone1, Morada1, Profissao1, LDoencas1, Centro1))),
                    assert((+utente(ID1, NSS1, Nome1, Data1, Email1, Telefone1, Morada1,
                    Profissao1, LDoencas1, Centro1) :: (solucoes(NSS1, (utente(ID, NSS1, Nome,
                    Data, Email, Telefone, Morada, Profissao, LDoencas, Centro), nao(nulo(NSS1))
                    ), S), comprimento(S, 0)))).
```

Figura 27: Predicado que regista um utente com valor de NSS interdito

Conseguimos validar esta inserção através dos predicados *utenteT*, *listing(excecao)* e *listing(nulo)*.

```
?- registaUtenteNSSInterdito(30, nss_nulo, 'Ulisses', 1974, 'ulisses@gmail.com', 921313711, 'Acores', 'Carpinteiro', [], 1).
true .

?- registaUtenteNSSInterdito(31, 12356700, 'Ricardo', 1974, 'riczao@gmail.com', 921313441, 'Leiria', 'Calceteiro', [], 2).
false.

?- utenteT.
:- dynamic utente/10.

utente(1, 21455655, 'Luis', 1970, 'luis@gmail.com', 936696454, 'Adaufe', 'Estudante', ['Doenca coronaria', 'Doenca respiratoria'], 1).
utente(2, 31455655, 'Filipe', 1969, 'filipe@gmail.com', 936696100, 'Barcelos', 'Estudante', ['Diabetes'], 1).
utente(3, 41455655, 'Jose', 1990, 'j@gmail.com', 936696151, 'Esposende', 'Estudante', [], 1).
utente(4, 42455655, 'Diogo', 1998, 'd@gmail.com', 936696171, 'Aveiro', 'Estudante', ['Doenca coronaria'], 1).
utente(5, 43455655, 'Rui', 1936, 'r@gmail.com', 936696190, 'Braga', 'Estudante', ['Diabetes'], 1).
utente(6, 41454355, 'Paulo', 1980, 'p@gmail.com', 936738461, 'Esposende', 'Medico', [], 2).
utente(7, 75355655, 'Beatriz', 1995, 'b@gmail.com', 936973645, 'Aveiro', 'Enfermeiro', [], 2).
utente(8, 43455692, 'Maria', 1951, 'm@gmail.com', 912348248, 'Braga', 'Estudante', [], 2).
utente(9, 42487435, 'Carlos', 1983, 'c@gmail.com', 936827634, 'Aveiro', 'Estudante', ['Artroses'], 2).
utente(11, 87654321, 'Ricardo', 1983, 'ri@gmail.com', 965269477, morada_Desc, 'Padeiro', [], 1).
utente(12, 13243546, 'Bruno', 1993, email_desc, 965233333, 'Portimao', 'Ferreiro', [], 2).
utente(13, 86756453, 'Fernando', 1989, 'f@gmail.com', 967269477, morada_Desc, 'Motorista', [], 3).
utente(16, 56789321, 'Vasco', 1957, email_interdito, 912345679, 'Porto', 'Contabilista', ['Artroses'], 3).
utente(17, nss_interdito, 'Evaristo', 1976, 'e@gmail.com', 922222222, 'Leixoes', 'Pedreiro', [], 2).
utente(10, 12345678, 'Antonio', 2000, 'ant@gmail.com', 965269469, 'Viana Castelo', 'Estudante', [], 1).
utente(14, 65748392, 'Tiago', 1990, 't@gmail.com', 925434567, 'Tomar', 'Canalizador', [], 2).
utente(20, 12312312, 'Leandro', 1999, 'leandro@gmail.com', 912828282, _, 'Estudante', [], 1).
utente(30, nss_nulo, 'Ulisses', 1974, 'ulisses@gmail.com', 921313711, 'Acores', 'Carpinteiro', [], 1).

true.

?- listing(nulo).
:- dynamic nulo/1.

nulo(email_interdito).
nulo(nss_interdito).
nulo(nss_nulo).

true.
```

Figura 28: Validação da inserção de conhecimento imperfeito interdito

3.4 Invariantes de Conhecimento Imperfeito

Devido à inserção de conhecimento imperfeito foi necessária a implementação de invariantes para controlar essa mesma inserção, de modo a não existir conhecimento repetido nem inserir conhecimento imperfeito onde este já é imperfeito. Como tal procedemos à criação de invariantes que controlassem as inserções por nós definidas, caso prosseguíssemos com a elaboração de novas inserções também criaríamos novos invariantes que seguiriam o mesmo raciocínio daqueles que passaremos a demonstrar de seguida. Como na inserção de conhecimento imperfeito são adicionadas novas exceções, também criámos invariantes de forma a controlar essa mesma adição.

- **Impedir adicionar conhecimento imperfeito a factos que já são caracterizados como imperfeitos:**
  - Utente com número de segurança social já interdito

```
+utente(ID,Q,Nome,Data,E,T,M,P,D,C) ::: nao(nulo(Q)).
```

Figura 29: Invariante de Utente com NSS interdito

– Exceção de utente com número de segurança social já interdito

```
+excecao(utente(ID,Q,Nome,Data,E,T,M,P,D,C)) ::: nao(nulo(Q)).
```

Figura 30: Invariante da exceção de Utente com NSS interdito

– Utente com morada já incerta

```
+utente(ID,Q,Nome,Data,E,T,M,P,D,C) ::: nao(moradaIncertaUtente(ID)).
```

Figura 31: Invariante de Utente com morada incerta

– Exceção de utente com morada já incerta

```
+excecao(utente(ID,Q,Nome,Data,E,T,M,P,D,C)) ::: nao(moradaIncertaUtente(ID)).
```

Figura 32: Invariante da exceção de Utente com morada incerta

– Centro de Saúde com nome já incerto

```
+centro_saude(ID, Nome, Morada, Telefone, Email) ::: nao(nomeIncertoCentro(ID)).
```

Figura 33: Invariante de Centro de Saúde com nome já incerto

– Exceção de centro de Saúde com nome já incerto

```
+excecao(centro_saude(ID, Nome, Morada, Telefone, Email)) ::: nao(nomeIncertoCentro(ID)).
```

Figura 34: Invariante da exceção do centro de Saúde com nome incerto

– Staff com Email já impreciso

```
+staff(ID, Centro, Nome, Email) ::: nao(emailImprecisoStaff(ID)).
```

Figura 35: Invariante de Staff com email impreciso

– Exceção de Staff com Email já impreciso

```
+excecao(staff(ID, Centro, Nome, Email)) ::: nao(emailImprecisoStaff(ID)).
```

Figura 36: Invariante da exceção de Staff com email impreciso

4 Conclusão

Após o término deste trabalho prático de grupo da Unidade Curricular de SRCR, atribuimos um balanço positivo ao trabalho desenvolvido.

Conseguimos complementar da melhor maneira todo o trabalho realizado durante as aulas práticas, que se revelou um desafio interessante no que diz respeito à descoberta da melhor estratégia a utilizar para a realização daquilo que era pretendido.

Com o desenrolar do projeto deparamo-nos com algumas dificuldades, que fomos ultrapassando, com maior ou menor facilidade.

Assim, consideramos que foi alcançada uma melhor consolidação da matéria lecionada nas aulas, permitindo, ainda, a exploração de novos domínios necessários à concretização do nosso projeto.

Seguimos determinados e focados em levar este projeto a bom porto.