

UNIVERSIDADE DO MINHO
DEPARTAMENTO DE INFORMÁTICA
SISTEMAS DE REPRESENTAÇÃO DE
CONHECIMENTO E RACIOCÍNIO

Trabalho Prático - Fase 2
Grupo Nº 38

Manuel Carvalho (A69856)
Pedro Pereira (A80627)
Sofia Marques (A87963)
Pedro Pereira (A89232)
José Martins (A90122)

3 de maio de 2021

Resumo

Este relatório foi realizado no âmbito da disciplina de Sistemas de Representação de Conhecimento e Raciocínio e tem como objetivo apresentar e descrever as soluções implementadas na resolução do projeto proposto pela equipa docente. De salientar que toda a implementação foi realizada com recurso à linguagem *Prolog*.

Este documento inicia-se com uma breve introdução, seguida de um conjunto de preliminares que definem o universo sobre o qual o trabalho prático foi desenvolvido. De seguida é feita a descrição do trabalho e análise de resultados, onde para além de se apresentar a base de conhecimento, também se explicam devidamente todos os predicados desenvolvidos para resolver o problema em questão e o sistema de inferência criado. Por fim, surge uma breve conclusão com a reflexão do grupo sobre o trabalho realizado.

Conteúdo

1	Introdução	2
2	Preliminares	3
3	Descrição do Trabalho e Análise de Resultados - Fase 1	4
3.1	Base de Conhecimento	4
3.1.1	utente	5
3.1.2	centroSaude	5
3.1.3	staff	5
3.1.4	vacinacao	6
3.1.5	critério	6
3.2	Predicados auxiliares	6
3.3	Adição e remoção de conhecimento	7
3.3.1	Evolução e Involução	7
3.3.2	Registo/Remoção de Utentes	8
3.3.3	Registo/Remoção de Staff	9
3.3.4	Registo/Remoção de Centros de Saúde	10
3.3.5	Registo/Remoção de Vacinação	11
3.3.6	Registo/Remoção de Critérios de Vacinação	13
3.4	Identificar pessoas não vacinadas	14
3.5	Identificar pessoas vacinadas	15
3.6	Identificar pessoas vacinadas indevidamente	16
3.7	Identificar pessoas não vacinadas e que são candidatas a vacinação	17
3.8	Identificar pessoas a quem falta a segunda toma da vacina	18
3.9	Sistema de inferência	18
4	Descrição do Trabalho e Análise de Resultados - Fase 2	20
4.1	Base de Conhecimento	20
4.1.1	utente	21
4.1.2	centroSaude	21
4.1.3	staff	21
4.1.4	excecao	22
4.2	Predicados auxiliares adicionados	22
4.3	Adição e remoção de conhecimento	22
4.3.1	Registo/Remoção de Utentes	23
4.3.2	Registo de Staff	25
4.4	Atualização de Conhecimento	25
4.4.1	Análise de resultados	25
4.5	Coerência da Base de Conhecimento	26
4.6	Predicados atualizados	26
4.7	Sistema de inferência Atualizado	27
4.7.1	Análise de resultados	28
5	Conclusão	28

Capítulo 1

Introdução

De forma a incentivar para a utilização da linguagem de programação em lógica *Prolog* e construção de mecanismos de raciocínio para a resolução de problemas, foi-nos proposto um exercício. Para este primeiro desafio foi sugerido o desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da vacinação da COVID-19. Para o efeito, considerou-se um panorama de conhecimento e implementaram-se funcionalidades que respeitam os requisitos pedidos pela equipa docente. Tudo isto será devidamente demonstrado e justificado ao longo deste relatório com a apresentação de excertos de código desenvolvido, casos práticos e seus resultados.

Capítulo 2

Preliminares

Para o desenvolvimento de uma solução para este problema, foram necessários todos os conteúdos lecionados nas aulas teóricas e práticas. Em acréscimo o grupo aprofundou o seu conhecimento em programação lógica, de forma a obter uma percepção mais clara das suas funcionalidades e de seguida focámos a nossa atenção no estudo do tema em questão, para que o nosso programa se assemelhasse à realidade. Feito tudo isto passámos então à implementação da solução.

Capítulo 3

Descrição do Trabalho e Análise de Resultados - Fase 1

3.1 Base de Conhecimento

Neste trabalho existem 5 tipos de predicados que representam o conhecimento, sendo eles apresentados a seguir:

- **utente:** #Idutente, N^o Segurança_Social, Nome, Data_Nasc, Email, Telefone, Morada, Profissão, [Doenças_Crónicas], #CentroSaúde -> {V,F}
- **centroSaude:** #Idcentro, Nome, Morada, Telefone, Email -> {V,F}
- **staff:** #Idstaff, #Idcentro, Nome, Email -> {V,F}
- **vacinacao:** #Staff, #Utente, Data, Vacina, Toma -> {V,F}
- **criterio:** #Fase, Data, [Doenças_Crónicas], [Profissões] -> {V,F}

A Base de Conhecimento apresentada será o ponto de partida de todos os testes realizados.

3.1.1 utente

Os seguintes predicados representam utentes:

```
1 utente(1,12345678901,'Rui',data(15,03,1969),'rui@gmail.com',914972655,'
    Guimaraes','Cantor',[ 'Diabetes','Insuficiencia Renal'],1).
2 utente(2,12345678902,'Manuel',data(15,03,1971),'batschecks@gmail.com'
    ,91497261,'Regua','Enfermeiro',[ 'Insuficiencia Cardiaca'],2).
3 utente(3,12345678903,'Luisa',data(25,04,1978),'luisa@gmail.com',914972656,'
    Braga','Medico',[ 'Asma','Reumatismo'],2).
4 utente(4,12345678904,'Fernando',data(30,05,2000),'fernando@gmail.com'
    ,914972657,'Porto','Estudante',[],1).
5 utente(5,12345678905,'Marta',data(30,05,1950),'marta@gmail.com',914972658,'
    Guimaraes','Reformado',[ 'Conjuntivite'],1).
6 utente(6,12345678906,'Pedro',data(3,03,2004),'pedro@gmail.com',914972659,'
    Lisboa','Estudante',[ 'DPOC'],1).
7 utente(7,12345678907,'Jose',data(1,03,2005),'jose@gmail.com',914972662,'
    Braga','Estudante',[],2).
8 utente(8,12345678908,'Goncalo',data(15,03,1941),'goncalo@gmail.com'
    ,914972663,'Barcelos','Reformado',[ 'Hipertensao Arterial'],2).
9 utente(9,12345678909,'Maria',data(15,03,1956),'maria@gmail.com',914972664,'
    Guimaraes','Militar',[],1).
10 utente(10,12345678910,'Sofia',data(15,03,1958),'sofia@gmail.com',914972665,'
    Guimaraes','Reformado',[ 'Obesidade'],1).
11 utente(11,12345678911,'Nuno',data(15,03,1957),'nuno@gmail.com',914972666,'
    Braga','Pescador',[],2).
12 utente(12,12345678912,'Joaquim',data(15,03,1959),'joaquim@gmail.com'
    ,914972667,'Alentejo','Pastor',[ 'Neoplasia Maligna Ativa'],2).
13 utente(13,123456789123,'Antonio',data(05,07,1965),'antonio@gmail.com'
    ,914972668,'Alentejo','Agricultor',[ 'Neoplasia Maligna Ativa'],2).
```

3.1.2 centroSaude

Os seguintes predicados representam centros de saúde:

```
1 centroSaude(1,'Centro de Saude de Guimaraes','Guimaraes',253519923,'
    centroSaudeGuimaraes@gmail.com').
2 centroSaude(2,'Centro de Saude de Braga','Braga',253209230,'
    centroSaudeBraga@gmail.com').
```

3.1.3 staff

Os seguintes predicados representam identidades de staff que trabalham nos centros de saúde respectivos:

```
1 staff(1,1,'Mel','mel@gmail.com').
2 staff(2,1,'Nina','nina@gmail.com').
3 staff(3,2,'Francisca','francisca@gmail.com').
4 staff(4,2,'Alberta','alberta@gmail.com').
```

3.1.4 vacinacao

Os seguintes predicados representam as vacinas já administradas:

```
1 vacinacao(1,1,'10-03-2021','Pfizer',1).
2 vacinacao(1,1,'07-04-2021','Pfizer',2).
3 vacinacao(3,2,'15-03-2021','Pfizer',1).
4 vacinacao(3,2,'08-03-2021','Pfizer',2).
5 vacinacao(4,3,'10-03-2021','AstraZeneca',1).
6 vacinacao(4,3,'07-04-2021','AstraZeneca',2).
7 vacinacao(1,4,'17-03-2021','AstraZeneca',1).
8 vacinacao(2,10,'10-03-2021','Pfizer',1).
9 vacinacao(2,10,'07-04-2021','Pfizer',2).
10 vacinacao(3,7,'10-03-2021','Pfizer',1).
11 vacinacao(1,9,'10-03-2021','Pfizer',1).
12 vacinacao(1,9,'07-04-2021','Pfizer',2).
13 vacinacao(4,8,'10-03-2021','AstraZeneca',1).
14 vacinacao(4,8,'07-04-2021','AstraZeneca',2).
```

3.1.5 criterio

Os seguintes predicados representam os critérios para as diferentes fases de vacinação:

```
1 criterio(1, data(01,01,1971),['Insuficiencia Cardiaca',
2   'Insuficiencia Renal', 'Doenca Coronaria', 'DPOC'], []).
3 criterio(1, data(01,01,2021),[], ['Medico', 'Bombeiro', 'Militar']).
4 criterio(2, data(01,01,1971),['Diabetes', 'Neoplasia Maligna Ativa',
5   'Doenca Renal Cronica', 'Insuficiencia Hepatica',
6   'Hipertensao Arterial', 'Obesidade'], []).
```

3.2 Predicados auxiliares

1. **member1**: Dado um elemento e uma lista verifica se esse elemento pertence à lista.
2. **memberList**: Dadas duas listas verifica se algum elemento da primeira lista faz parte da segunda lista.
3. **subtrair**: Retira à primeira lista os elementos presentes na segunda lista.
4. **eliminaRepetidos**: Dada uma lista, elimina todos os seus elementos repetidos.
5. **eliminaRepAux**: Predicado auxiliar da **eliminaRepetidos**.
6. **dataA**: Dada uma data (D,M,A) retorna o ano (A).
7. **dataM**: Dada uma data (D,M,A) retorna o mês (M).
8. **dataD**: Dada uma data (D,M,A) retorna o dia (D).

3.3 Adição e remoção de conhecimento

A adição e remoção de conhecimento é realizada à custa de invariantes que especificam um conjunto de restrições que devem ser verdadeiras após uma inserção ou remoção de conhecimento. A inserção de conhecimento pode ser vista como uma evolução do sistema em termos de conhecimento, da mesma forma que a remoção pode ser vista como uma involução. De salientar que cada uma dessas operações são sempre efetuadas, porém caso alguma provoque uma anomalia no sistema, esta perde o seu efeito e o sistema volta ao estado anterior a essa operação.

Antes de descrevermos de forma detalhada os processos de evolução e involução do sistema, assim como todos os invariantes envolvidos, faz sentido mencionar e apresentar os quatro predicados desenvolvidos que são muito importantes para o funcionamento destes processos.

Sendo assim apresenta-se o predicado **solucoes**:

```
1 solucoes (X,Y,Z):- findall(X,Y,Z) .
```

De seguida apresentam-se os predicados de remoção e inserção (**remove** e **insercao** respetivamente), que fazem uso dos mecanismos de asserção do *Prolog*:

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

No caso da inserção, a primeira cláusula (*assert*) será sempre verificada, e caso nos encontremos na segunda, é porque não foi possível inserir o termo por violar algum invariante. Por esta razão, deverá ser removido o termo adicionado (com recurso a *retract*), sendo que de seguida é aplicado um *cut* (!) de forma a que o valor lógico da expressão seja falso, e assim, a evolução do conhecimento seja falsa também.

```
1 remove(Termo) :- retract(Termo) .
2 remove(Termo) :- assert(Termo) , ! , fail .
```

Na remoção o raciocínio é o mesmo que na inserção, trocando-se como é evidente a ordem do *assert* e *retract*.

Por último é importante referir o predicado **teste** responsável pelo teste dos invariantes:

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

3.3.1 Evolução e Involução

De forma a que a nossa base de conhecimento pudesse evoluir dinamicamente, foi necessário desenvolver dois predicados base: **evolucao** e **involucao**. Tal como dito anteriormente o primeiro representa a adição de conhecimento, enquanto que o segundo representa a subtração de conhecimento.

```
1 evolucao( Termo ) :-
2     solucoes( Invariante , +Termo::Invariante , Lista ) ,
3     insercao( Termo ) ,
4     teste( Lista ) .
```

```
1 involucao( Termo ) :-
2     solucoes( Invariante , -Termo::Invariante , Lista ) ,
3     remove( Termo ) ,
4     teste( Lista ) .
```

3.3.2 Registo/Remoção de Utentes

Registo

Para registar novos utentes é necessário verificar que não existe mais nenhum com o mesmo ID e nº de Segurança Social. Verificámos também se o mês da data de nascimento é válido. Por este motivo foram desenvolvidos os seguintes invariantes:

Verificar que o ID não se repete:

```
1 +utente(Id, -, -, -, -, -, -, -, -, -) ::  
2   (solucoes((Id), utente(Id, -, -, -, -, -, -, -, -, -), R), length(R, 1)).
```

Verificar que o nº de Segurança Social não se repete:

```
1 +utente(-, Ns, -, -, -, -, -, -, -, -) ::  
2   (solucoes(Id, utente(-, Ns, -, -, -, -, -, -, -, -), R), length(R, 1)).
```

Verificar que o mês da data de nascimento é válido:

```
1 +utente(-, -, -, -, D, -, -, -, -, -) :: (dataM(D, R), R > 0, R < 13).
```

Predicado para o registo de utentes:

```
1 regUtente(Id, Ns, N, D, E, T, M, P, DC, C) :- evolucao(utente(Id, Ns, N, D, E, T, M, P, DC, C)).
```

Remoção

Para remover um utente é necessário verificar se este já foi vacinado, pois em caso afirmativo este não pode ser removido da base de conhecimento, uma vez que isso levaria à inconsistência da mesma. Posto isto foi desenvolvido o seguinte invariante:

```
1 -utente(Id, -, -, -, -, -, -, -, -) ::  
2   (solucoes(Id, vacinacao(-, Id, -, -, -), L), length(L, 0)).
```

Predicado para a remoção de utentes:

```
1 removeUtente(Id) :-  
2   utente(Id, Ns, N, D, E, T, M, P, DC, C), involucao(utente(Id, Ns, N, D, E, T, M, P, DC, C)).
```

Análise de Resultados

```
?- regUtente(10,123456789000,'Luis',data(05,07,1965),'luis@gmail.com',914972668,'Alentejo','Agricultor',[],2).%ID repetido  
false.  
?- regUtente(14,12345678910,'Luis',data(05,07,1965),'luis@gmail.com',914972668,'Alentejo','Agricultor',[],2).%Nº de segurança social repetido  
false.  
?- regUtente(14,12345678000,'Luis',data(05,13,1965),'luis@gmail.com',914972668,'Alentejo','Agricultor',[],2).%Mes invalido  
false.  
?- regUtente(14,12345678000,'Luis',data(05,07,1965),'luis@gmail.com',914972668,'Alentejo','Agricultor',[],3).%Centro de saude nao registado  
false.  
?- regUtente(14,123456789000,'Luis',data(05,07,1965),'luis@gmail.com',914972668,'Alentejo','Agricultor',[],2).%Registo valido  
true.
```

Figura 3.1: Teste dos invariantes de registo de utente

```

?- removeUtente(1).%Remoção de utente ja vacinado
false.

?- removeUtente(12). %Remoção de utente nao vacinado
true .

```

Figura 3.2: Teste do invariante de remoção de utente

3.3.3 Registo/Remoção de Staff

Registo

Tal como no registo de utentes, para registar novos membros do staff é necessário verificar que não existe mais nenhum com o mesmo ID. Para tal criámos o seguinte invariante:

```

1 +staff(Id,-,-,-) :: ( solucoes (Id, staff (Id,-,-,-),R), length(R,1) ) .

```

Predicado para o registo de staff:

```

1 regStaff (IDS, IDU, N, M) :- evolucao ( staff (IDS, IDU, N, M) ) .

```

Remoção

Para remover o staff é necessário garantir que não existem registos de vacinações realizadas por este, sendo assim desenvolvido um invariante para cumprir essa função:

```

1 -staff (IDS, -,-,-) ::
2   ( solucoes (IDS, vacinacao (IDS, -,-,-,-), L), length(L,0) ) .

```

Predicado para a remoção de staff:

```

1 removeStaff (IDS) :-
2   staff (IDS, ICS, Nome, Email), involucao ( staff (IDS, ICS, Nome, Email) ) .

```

Análise de Resultados

```

?- regStaff(4,2,'Guilherme','guilherme@gmail.com'). %ID repetido
false.

?- regStaff(5,2,'Guilherme','guilherme@gmail.com'). %Registo valido
true .

```

Figura 3.3: Teste do invariante de registo de staff

```
?- removeStaff(3). %Staff com vacinações registadas
false.

?- removeVacinacao(2,2). %Vacinação do staff Id 3
true .

?- removeVacinacao(2,1). %Vacinação do staff Id 3
true .

?- removeVacinacao(7,1). %Vacinação do staff Id 3
true .

?- removeStaff(3). %Staff sem vacinações registadas
true .
```

Figura 3.4: Teste do invariante de remoção de staff

3.3.4 Registo/Remoção de Centros de Saúde

Registro

Seguindo mais uma vez o mesmo raciocínio do registo dos utentes e staff, para registar novos centros de saúde é necessário verificar que não existem já outros com o mesmo ID. Posto isto foi elaborado o seguinte invariante:

```

1 +centroSaude(Id, -, -, -, -) ::
2   (solucoes(Id, centroSaude(Id, -, -, -, -), R), length(R, 1)).

```

Predicado para o registo de Centros de Saúde:

```

1 regCentroSaude (ID,N,L,Nu,M):−evoluc o ( centroSaude (ID,N,L,Nu,M) ) .

```

Remoção

Para remover um centro de saúde é necessário verificar que não existe staff nem utentes que pertençam a esse centro de saúde. Para isso foi criado o seguinte invariante:

$$\begin{aligned} &1 \text{ --centroSaude}(\text{CS}, -, -, -, -) :: \\ &2 \quad (\text{solucoes}(\text{Id}, (\text{staff}(\text{Id}, \text{CS}, -, -); \text{utente}(\text{Id}, -, -, -, -, -, -, -, -, \text{CS})), \text{L}), \\ &\quad \text{length}(\text{L}, 0)). \end{aligned}$$

Predicado para a remoção de Centros de Saúde:

```

1 removeCentroSaude(ID):-
2     centroSaude(ID,N,L,Nu,M) ,   involucao(centroSaude(ID,N,L,Nu,M)).

```

Análise de Resultados

```
?- regCentroSaude(2,'Centro de Saude de Paços de Ferreira','Paços de Ferreira',255123123,'centroSaudePaços@gmail.com'). %ID repetido
false.
?- regCentroSaude(3,'Centro de Saude de Paços de Ferreira','Paços de Ferreira',255123123,'centroSaudePaços@gmail.com'). %Registo valido
true .
```

Figura 3.5: Teste do invariante de registo de centros de saúde

```
?- removeCentroSaude(1). %Centro de Saude com vacinações registadas
false.
?- removeVacinacao(1,2). %Vacinacao registada com staff do Centro de Saude ID 1
true .
?- removeVacinacao(1,1). %Vacinacao registada com staff do Centro de Saude ID 1
true .
?- removeVacinacao(4,1). %Vacinacao registada com staff do Centro de Saude ID 1
true .
?- removeVacinacao(10,2). %Vacinacao registada com staff do Centro de Saude ID 1
true .
?- removeVacinacao(10,1). %Vacinacao registada com staff do Centro de Saude ID 1
true .
?- removeVacinacao(9,2). %Vacinacao registada com staff do Centro de Saude ID 1
true .
?- removeVacinacao(9,1). %Vacinacao registada com staff do Centro de Saude ID 1
true .
?- removeStaff(1). %Staff registado com Centro de Saude ID 1
true .
?- removeStaff(2). %Staff registado com Centro de Saude ID 1
true .
?- removeUtente(1). %Utente registado com Centro de Saude ID 1
true .
?- removeUtente(4). %Utente registado com Centro de Saude ID 1
true .
?- removeUtente(5). %Utente registado com Centro de Saude ID 1
true .
?- removeUtente(6). %Utente registado com Centro de Saude ID 1
true .
?- removeUtente(9). %Utente registado com Centro de Saude ID 1
true .
?- removeUtente(10). %Utente registado com Centro de Saude ID 1
true .
?- removeCentroSaude(1). %Centro de saude sem vacinações, staff nem utentes registados
true .
```

Figura 3.6: Teste do invariante de remoção de centros de saúde

3.3.5 Registo/Remoção de Vacinação

Registo

Para registar novas vacinações é necessário ter em atenção algumas das suas características pelo que foi desenvolvida uma série de invariantes:

Invariante que garante que não é introduzida uma vacinação com o mesmo número de utente e mesma toma da vacina:

```

1 +vacinacao(-,IDU,-,-,NT)::
2   (solucoes((IDU,NT),vacinacao(-,IDU,-,-,NT),R),length(R,1)).

```

Invariante que garante que o número de toma é válido(>0 e <3):

```

1 +vacinacao(-,-,-,-,NT)::(NT>0,NT<3).

```

Invariante que garante que a toma das vacinas é feita por ordem e com a mesma vacina

```

1 +vacinacao(-,IDU,-,NV,2)::vacinacao(-,IDU,-,NV,1).

```

Invariante que garante que a toma da vacina é feita no centro de saúde do utente e que o staff responsável pela realização da vacinação também pertence a esse centro de saúde:

```

1 +vacinacao(IDS,IDU,-,-,-)::
2   (solucoes((CS),(utente(IDU,-,-,-,-,-,-,-,CS),staff(IDS,CS,-,-)),R),
   length(R,1)).

```

Predicado para o registo de vacinação:

```

1 regVacinacao(IDS,IDU,DN,NV,NT):- evolucao(vacinacao(IDS,IDU,DN,NV,NT)).

```

Remoção

Para remover o registo de vacinação é necessário garantir que a primeira toma não é removida se existir uma segunda toma registada:

```

1 -vacinacao(-,IDU,-,-,1)::nao(vacinacao(-,IDU,-,-,2)).

```

Predicado para a remoção de Vacinação:

```

1 removeVacinacao(IDU,NT):-
2   vacinacao(IDS,IDU,DN,NV,NT), involucao(vacinacao(IDS,IDU,DN,NV,NT)).

```

Análise de Resultados

```

?- regVacinacao(1,4,'17-03-2021','AstraZeneca',1). %Toma repetida
false.

?- regVacinacao(1,4,'17-03-2021','AstraZeneca',3). %Toma invalida
false.

?- regVacinacao(1,5,'17-03-2021','AstraZeneca',2). %Toma 2 antes da toma 1
false.

?- regVacinacao(1,4,'17-03-2021','Pfizer',2). %Toma 2 com vacina diferente
false.

?- regVacinacao(3,4,'17-03-2021','AstraZeneca',2). %Vacinação com staff de centro de saude diferente do utente
false.

?- regVacinacao(1,4,'17-03-2021','AstraZeneca',2). %Vacinação valida
true.

```

Figura 3.7: Teste do invariante de registo de vacinação

```

?- removeVacinao(1,1). %Remoção da 1ª toma antes da remoção da 2ª toma
false.

?- removeVacinao(1,2). %Remoção da 2ª toma
true .

?- removeVacinao(1,1). %Remoção da 1ª toma
true .

```

Figura 3.8: Teste do invariante de remoção de vacinação

3.3.6 Registo/Remoção de Critérios de Vacinação

Registo

Predicado para o registo de Critérios:

```

1 regCriterio(Fase, data(D, M, A), Doenca, Profissao) :-
2     evolucao(criterio(Fase, data(D, M, A), Doenca, Profissao)).

```

Remoção

Predicado para a remoção de Critérios:

```

1 removeCriterio(Fase, data(D, M, A), Doenca, Profissao) :-
2     involucao(criterio(Fase, data(D, M, A), Doenca, Profissao)).

```

Análise de Resultados

```

?- mostrarCritérios(C).
C = [(1, data(1, 1, 1971), ['Insuficiencia Cardiaca', 'Insuficiencia Renal', 'Doenca Coronaria', 'DPOC'], []), (1, data(1, 1, 2021), [], ['Medico', 'Bombeiro', 'Militar']), (2, data(1, 1, 1971), ['Diabetes', 'Neoplasia Maligna Ativa', 'Doenca Renal Cronica'...], [])].

?- utenteFV(5,F).
F = 3 .

?- regCriterio(2,data(01,01,1951),['Conjuntivite'],[]).
true .

?- mostrarCritérios(C).
C = [(1, data(1, 1, 1971), ['Insuficiencia Cardiaca', 'Insuficiencia Renal', 'Doenca Coronaria', 'DPOC'], []), (1, data(1, 1, 2021), [], ['Medico', 'Bombeiro', 'Militar']), (2, data(1, 1, 1971), ['Diabetes', 'Neoplasia Maligna Ativa', 'Doenca Renal Cronica'...], []), (2, data(1, 1, 1951), ['Conjuntivite'], [])].

?- utenteFV(5,F).
F = 2 .

```

Figura 3.9: Teste do registo de critérios

```

?- mostrarCriterios(C).
C = [(1, data(1, 1, 1971), ['Insuficiencia Cardiaca', 'Insuficiencia Renal', 'Doenca Coronaria', 'DPOC'], []), (1, data(1, 1, 2021),
[], ['Medico', 'Bombeiro', 'Militar']), (2, data(1, 1, 1971), ['Diabetes', 'Neoplasia Maligna Ativa', 'Doenca Renal Cronica' [...],
[]])].

?- utenteFV(1,F).
F = 1 .

?- removeCriterio(1, data(01,01,1971),['Insuficiencia Cardiaca', 'Insuficiencia Renal', 'Doenca Coronaria', 'DPOC'], []).
true .

?- mostrarCriterios(C).
C = [(1, data(1, 1, 2021), [], ['Medico', 'Bombeiro', 'Militar']), (2, data(1, 1, 1971), ['Diabetes', 'Neoplasia Maligna Ativa', 'Do
enca Renal Cronica', 'Insuficiencia Hepatica' [...], []])].

?- utenteFV(1,F).
F = 2 .

```

Figura 3.10: Teste da remoção de critérios

3.4 Identificar pessoas não vacinadas

Para identificar os utentes que não foram ainda vacinados, quer com a primeira, quer com a segunda toma, foi necessário elaborar dois predicados:

```

1 utenteNV(Id) :- utentesNV(R), member1(Id, R).
2
3 utentesNV(R):-
4     solucoes(Id, utente(Id,_,_,_,_,_,_,_,_,_) ,L) ,
5     solucoes(X, vacinacao( _,X,_,_,_) ,LI) ,
6     subtrair(L,LI,R) .

```

O primeiro predicado serve para verificar se um dado utente já foi ou não vacinado. Por sua vez o segundo predicado dá-nos todos os utentes que não foram vacinados.

Era também necessário identificar quais os utentes que não tomaram a segunda dose, independentemente de terem tomado a primeira ou não, e para isso foram criados mais dois predicados:

```

1 utenteNV2(Id) :- utentesNV2(R), member1(Id, R).
2
3 utentesNV2(R):-
4     solucoes(Id, utente(Id,_,_,_,_,_,_,_,_,_), L),
5     solucoes(X, vacinacao(_,X,_,_,2), LI),
6     subtrair(L, LI, R).

```

Em semelhança com os predicados anteriores, o primeiro predicado verifica se um dado utente já tomou ou não a segunda dose e o segundo dá como resultado todos os utentes que não a tomaram.

Análise de Resultados

```
?- utenteNV(1). %2 doses tomadas
false.

?- utenteNV(12). %nenhuma dose tomada
true .

?- utenteNV(4). %1 dose tomada
false.

?- utentesNV(L). %utentes sem nenhuma dose tomada
L = [5, 6, 11, 12, 13].
```

Figura 3.11: Teste dos predicados `UtenteNV` e `UtentesNV`


```

?- utenteNV2(1). %2 doses tomadas
false.

?- utenteNV2(12). %nenhuma dose tomada
true .

?- utenteNV2(4). %1 dose tomada
true .

?- utentesNV2(L). %utentes sem a segunda dose tomada
L = [4, 5, 6, 7, 11, 12, 13].

```

Figura 3.12: Teste dos predicados UtenteNV2 e UtentesNV2

3.5 Identificar pessoas vacinadas

Relativamente às pessoas já vacinadas, é possível verificar a lista de pessoas a quem já foram administradas pelo menos 1 vacina e também a lista de pessoas que tomaram as 2 doses. Para estes 2 casos é também possível, dado um ID, verificar se o utente já foi vacinado.

```

1 utentesV(R):- solucoes(X,vacinacao(_,X,-,-,-), R1), eliminaRepetidos(R1, R).
2
3 utenteV(Id):-vacinacao(_,Id,-,-,-), !.

```

```

1 utentesV2(R):-
2     solucoes(X,(vacinacao(_,X,-,-,1),vacinacao(_,X,-,-,2)), R1),
3     eliminaRepetidos(R1, R).
4
5 utenteV2(Id):-vacinacao(_,Id,-,-,1), vacinacao(_,Id,-,-,2).

```

Análise de Resultados

```

?- utenteV(1). %2 doses tomadas
true.

?- utenteV(12). %nenhuma dose tomada
false.

?- utenteV(4). %1 dose tomada
true.

?- utentesV(L). %utentes com qualquer dose tomada
L = [8, 9, 7, 10, 4, 3, 2, 1].

```

Figura 3.13: Teste dos predicados UtenteV e UtentesV

```

?- utenteV2(1). %2 doses tomadas
true .

?- utenteV2(12). %nenhuma dose tomada
false.

?- utenteV2(4). %1 dose tomada
false.

?- utentesV2(L). %utentes com 2 doses tomadas
L = [8, 9, 10, 3, 2, 1].

```

Figura 3.14: Teste dos predicados UtenteV2 e UtentesV2

3.6 Identificar pessoas vacinadas indevidamente

De maneira a identificar os utentes vacinados indevidamente, foi necessário determinar se as diferentes fases de vacinação já tinham sido terminadas. Tendo isso em conta foi desenvolvido o predicado abaixo apresentado, que dado um número de uma fase, nos indica se esta está terminada ou não:

```
1 faseTerminada(1) :-
2     solucoes(ID, (vacinacao(_,ID,_,_,1), vacinacao(_,ID,_,_,2),
3     utenteFV(ID,1)), LV),
4     fase1Vacinacao(L1),
5     subtrair(L1, LV, R),
6     length(R, 0).
7
8 faseTerminada(2) :-
9     solucoes(ID, (vacinacao(_,ID,_,_,1), vacinacao(_,ID,_,_,2),
10    utenteFV(ID,2)), LV),
11    fase2Vacinacao(L1),
12    subtrair(L1, LV, R),
13    length(R, 0).
14
15 faseTerminada(3) :-
16    solucoes(ID, (vacinacao(_,ID,_,_,1), vacinacao(_,ID,_,_,2),
17    utenteFV(ID,3)), LV),
18    fase3Vacinacao(L1),
19    subtrair(L1, LV, R),
20    length(R, 0).
```

Para além disso foi preciso identificar qual a fase de vacinação de um utente, criando-se um predicado que associa o ID de um utente à sua devida fase:

```
1 utenteFV(Id,1):- cumpreCritérios(Id, 1).
2 utenteFV(Id,2):- cumpreCritérios(Id, 2), nao(utenteFV(Id,1)).
3 utenteFV(Id,3):- solucoes(I, (utente(I,_,_,_,_,_,_,_,_,_,_),
4     nao(utenteFV(I,2)), nao(utenteFV(I,1))), L),
5     member1(Id, L).
```

Como podemos verificar é usado como predicado auxiliar o *cumpreCritérios* que associa o ID de um utente a uma fase e verifica se este cumpre os critérios necessários para ser vacinado nessa fase:

```
1 cumpreCritérios(Id, F) :-
2     utente(Id,_,_,D,_,_,_,P,DC,-),
3     criterio(F, data(DiaC, MesC, AnoC), Doenca, Profissao),
4     dataA(D,A),A=<=AnoC, (memberList(DC, Doenca) ; member1(P, Profissao)).
```

Finalmente pudemos definir o predicado principal:

```
1 vacinacaoIndevida(R) :-
2     nao(faseTerminada(1)),
3     solucoes(ID, (vacinacao(_, ID, -, -, -), (utenteFV(ID, 2); utenteFV(ID, 3))), R1),
4     eliminaRepetidos(R1, R).
5 vacinacaoIndevida(R) :-
6     faseTerminada(1), nao(faseTerminada(2)),
7     solucoes(ID, (vacinacao(_, ID, -, -, -), utenteFV(ID, 3)), R1),
8     eliminaRepetidos(R1, R).
9 vacinacaoIndevida([]).
```

Análise de Resultados

```
?- fase1Vacinacao(L).
L = [9, 3, 2, 1].

?- faseTerminada(1). %Todos os utentes da fase 1 ja foram vacinados (9,3,2,1)
true.

?- fase2Vacinacao(L).
L = [13, 12, 10, 8].

?- faseTerminada(2). %Nem todos os utentes da fase 2 ja foram vacinados (13,12,8)
false.

?- vacinacaoIndevida(L). %Listagem de utentes que nao sao da fase 2 e que ja foram vacinados
L = [7, 4].

?- utenteFV(4,Fase). %Saber fase do utente 4
Fase = 3.

?- utenteV(4). %Saber se utente 4 ja foi vacinado
true.
```

Figura 3.15: Teste gerais de vacinação indevida

3.7 Identificar pessoas não vacinadas e que são candidatas a vacinação

Para ser possível identificar os candidatos a vacinação (utentes da fase ainda não terminada mais prioritária que ainda não tiveram a primeira toma), foram precisos os predicados **faseTerminada** e **utenteFV** (definidos no 3.6). Com a ajuda destes foi então possível desenvolver o seguinte predicado:

```
1 candidatosVac(R) :-
2     nao(faseTerminada(1)),
3     solucoes(ID, (utenteNV(ID), utenteFV(ID, 1)), L),
4     eliminaRepetidos(L, R).
5 candidatosVac(R) :-
6     faseTerminada(1),
7     nao(faseTerminada(2)),
8     solucoes(ID, (utenteNV(ID), utenteFV(ID, 2)), L),
9     eliminaRepetidos(L, R).
10 candidatosVac(R) :-
11     faseTerminada(2),
12     solucoes(ID, (utenteNV(ID), utenteFV(ID, 3)), L),
13     eliminaRepetidos(L, R).
```

Considerou-se neste predicado que os utentes que já tiveram a primeira toma da vacina não são considerados candidatos uma vez que apenas esperam a segunda toma da vacina.

Análise de Resultados

```
?- faseTerminada(1). %1ª fase de vacinação ja terminada
true.

?- faseTerminada(2). %2ª fase de vacinação ainda nao terminada
false.

?- candidatosVac(L). %Utentes pertencentes a fase 2 que ainda nao tiveram a primeira toma
L = [13, 12] .

?- utenteFV(13,2). %Utente 13 pertence a fase 2
true .

?- utenteNV(13). %Utente 13 ainda nao foi vacinado
true .
```

Figura 3.16: Teste relativos ao predicado candidatosVac

3.8 Identificar pessoas a quem falta a segunda toma da vacina

De maneira a identificar as pessoas que tomaram a primeira dose da vacina mas não a segunda foi desenvolvido o seguinte predicado:

```
1 segundaToma(R):-
2     solucoes(X,vacinacao(_,X,_,_,1),L1),
3     solucoes(X2,vacinacao(_,X2,_,_,2),L2),
4     subtrair(L1,L2,R).
```

Análise de Resultados

```
?- segundaToma(L). %Listagem de utentes que tomaram a 1ª dose mas nao a 2ª
L = [4, 7].

?- utenteV(4). %utente 4 vacinado
true.

?- utenteV2(4). %utente 4 vacinado com 2 tomas
false.

?- regVacinacao(1,4,'08-04-2021','AstraZeneca',2). %registo da toma da 2ª dose do utente 4
true .

?- segundaToma(L).
L = [7].
```

Figura 3.17: Teste relativos ao predicado segundaToma

3.9 Sistema de inferência

De modo a conseguirmos verificar a veracidade de um determinado termo foi utilizado o predicado **si**. Este retorna verdade caso exista uma prova explícita na base de conhecimento em que o parâmetro Questão seja verdadeiro, falso se existir na base de conhecimento uma negação da Questão.

```
1 si(Questao, verdadeiro) :- Questao.
2 si(Questao, falso) :- nao(Questao).
```

De forma a aumentar a eficiência do nosso Sistema de Inferência foi também criado um predicado que permite responder a várias perguntas ao mesmo tempo, designado por **siLista**:

```
1 siLista([], []).
2 siLista([Questao|Qs],[R|Rs]):- si(Questao,R), siLista(Qs,Rs).
```

Análise de Resultados

```
?- si(faseTerminada(1), R).
R = verdadeiro .

?- si(faseTerminada(2), R).
R = falso.

?- siLista([faseTerminada(1),faseTerminada(2)], R).
R = [verdadeiro, falso] .

?- si(utenteFV(1,1),R).
R = verdadeiro .

?- si(utenteFV(1,2),R).
R = falso.

?- si(utenteFV(1,3),R).
R = falso.
```

Figura 3.18: Testes do sistema de inferência

Capítulo 4

Descrição do Trabalho e Análise de Resultados - Fase 2

4.1 Base de Conhecimento

Nesta fase adicionamos mais alguns predicados que representam conhecimento, sendo estes apresentados a seguir:

- **excecao**: Utente $\rightarrow \{V,F\}$
- **excecao**: Staff $\rightarrow \{V,F\}$
- **excecao**: CentroSaude $\rightarrow \{V,F\}$

Estes diferentes predicados com o nome **excecao** dão-nos a possibilidade de inserir na base de conhecimento, conhecimento imperfeito, podendo este ser do tipo incerto, impreciso ou interdito. Em acréscimo, foram também elaborados predicados que permitem adicionar à base de conhecimento informação negativa (relativa aos utentes e staff), sendo eles apresentados a seguir:

```
1 -utente(Id,NS,N,DN,E,T,M,P,DC,CS):- nao(utente(Id,NS,N,DN,E,T,M,P,DC,CS)),
    nao(excecao(utente(Id,NS,N,DN,E,T,M,P,DC,CS))).
2 -staff(ID,IDC,N,E):- nao(staff(ID,IDC,N,E)), nao(excecao(staff(ID,IDC,N,E))).
```

4.1.1 utente

Relativamente aos utentes foi criado conhecimento imperfeito para os campos de morada, data de nascimento e número de telemóvel, permitindo adicionar à base de conhecimento utentes com uma morada desconhecida (conhecimento imperfeito do tipo incerto), utentes com registo de várias datas de nascimento (conhecimento imperfeito do tipo impreciso) e ainda utentes com um número de telemóvel inacessível (conhecimento do tipo interdito). Em acréscimo a isto, foi também inserido conhecimento negativo (negação forte).

```
1 % conhecimento imperfeito incerto
2 utente(16,123456789126, 'Rita', data(08,12,2002), 'rita@gmail.com', 914972682,
   morada_incerta, 'Estudante', ['Diabetes'], 2).
3
4 % conhecimento imperfeito impreciso
5 excecacao(utente(18,123456789128, 'Marcia', data(07,07,1921), 'marcia@gmail.com',
   914972684, 'Fafe', 'Bombeiro', [], 1)).
6 excecacao(utente(18,123456789128, 'Marcia', data(07,07,1920), 'marcia@gmail.com',
   914972684, 'Fafe', 'Bombeiro', [], 1)).
7
8 % conhecimento imperfeito interdito
9 utente(20,1234567891290, 'Ricardo', data(09,09,1999), 'ricardo@gmail.com',
   telemovel_inderdito, 'Lisboa', 'Esteticista', [], 2).
10
11 %conhecimento negativo (negação forte)
12 -utente(14,123456789124, 'Bernardo', data(05,07,2008), 'bernardo@gmail.com',
   914972680, 'Barcelos', 'Estudante', [], 2).
13 -utente(15,123456789125, 'Lara', data(05,10,2010), 'lara@gmail.com', 914972681, '
   Guimaraes', 'Estudante', ['Diabetes'], 1).
```

4.1.2 centroSaude

Em relação ao centro de saúde foi elaborado conhecimento imperfeito para o campo do email, permitindo adicionar centros de saúde que possuam um email desconhecido (conhecimento imperfeito do tipo incerto).

```
1 %conhecimento imperfeito incerto
2 centroSaude(3, 'Centro de Saude de Vizela', 'Vizela', 253245678, email_incerto).
```

4.1.3 staff

Em relação ao staff foi elaborado conhecimento imperfeito para o campo do email, permitindo adicionar elementos do staff que possuam um email desconhecido (conhecimento imperfeito do tipo incerto) ou com um email inacessível (conhecimento imperfeito do tipo interdito).

```
1 %conhecimento imperfeito incerto
2 staff(7,1, 'Silvia', email_incerto).
3
4 %conhecimento imperfeito interdito
5 staff(10,1, 'Magda', email_interdito).
6
7 %conhecimento negativo negacao forte
8 -staff(5,1, 'Simone', 'simone@gmail.com').
9 -staff(6,2, 'Simao', 'simao@gmail.com').
```

4.1.4 excecao

Com a ajuda deste predicado foi possível criar várias exceções para os diferentes tipos de conhecimento imperfeito, que foram mencionados anteriormente. Posto isto, em baixo são apresentados os casos em que este predicado foi aplicado:

```
1 %excecao morada_incerta (utente)
2 excecao(utente(Id,Ns,No,DNa,Em,Te,-,Pr,DCs,CSa)):-utente(Id,Ns,No,DNa,Em,Te,
   morada_incerta,Pr,DCs,CSa).
3
4 %excecao email_incerto (utente)
5 excecao(utente(Id,Ns,No,DNa,-,Te,M,Pr,DCs,CSa)):-utente(Id,Ns,No,DNa,
   email_incerto,Te,M,Pr,DCs,CSa).
6
7 %excecao telemovel_interdito (utente)
8 excecao(utente(Ids,Ns,No,DNa,Em,-,Mo,Pr,DCs,CS)):-utente(Ids,Ns,No,DNa,Em,
   telemovel_interdito,Mo,Pr,DCs,CS).
9 nulo(telemovel_interdito).
10
11 %excecao email_incerto (staff)
12 excecao(staff(Id,IdS,N,-)):-staff(Id,IdS,N,email_incerto).
13
14 %excecao email_interdito (staff)
15 excecao(staff(Id,IdS,N,-)):-staff(Id,IdS,N,email_interdito).
16 nulo(email_interdito).
```

4.2 Predicados auxiliares adicionados

1. **maiorData**: Dadas duas datas como argumento retorna a mais antiga.
2. **maxData**: Dada uma lista de datas retorna a mais antiga, utilizando o predicado acima como auxiliar.
3. **removeL**: Dada uma lista como argumento remove todos os seu elementos
4. **escrever**: Dada uma lista, escreve cada elemento da mesma linha a linha.

4.3 Adição e remoção de conhecimento

Nesta fase foi necessário expandir o conjunto de invariantes de adição conhecimento, uma vez que agora a base de conhecimento não só contém conhecimento positivo mas também conhecimento negativo e imperfeito. No entanto não foi necessário elaborar mais invariantes de remoção. Sendo assim, vão ser apresentados abaixo só os invariantes que foram acrescentados nesta fase, bem como os que foram modificados.

4.3.1 Registo/Remoção de Utentes

Registo

Invariantes para verificar que o ID não se repete:

```
1 %conhecimento positivo
2 +utente(Id,_,_,_,_,_,_,_,_,_)::
3   (solucoes((Id),(utente(Id,_,_,_,_,_,_,_,_,_)),R),(length(R,1);
4   (solucoes((Id),excecao(utente(Id,_,_,_,_,_,_,_,_,_)),R2),
5   length(R2, L2), L2 >=1)).
6
7 %conhecimento negativo
8 +(-utente(Id,_,_,_,_,_,_,_,_,_))::
9   (solucoes((Id),(-utente(Id,_,_,_,_,_,_,_,_,_));
10   -excecao(utente(Id,_,_,_,_,_,_,_,_,_)),R),
11   length(R,1)).
```

Invariantes para verificar que o nº de Segurança Social não se repete:

```
1 %conhecimento positivo
2 +utente(_,Ns,_,_,_,_,_,_,_,_)::
3   (solucoes(Ns,(utente(_,Ns,_,_,_,_,_,_,_,_)),R),
4   length(R,1);
5   (solucoes(Ns,excecao(utente(_,Ns,_,_,_,_,_,_,_,_)),R2),
6   length(R2, L2), L2 >=1)).
7
8 %conhecimento negativo
9 +(-utente(_,Ns,_,_,_,_,_,_,_,_))::
10   (solucoes(Ns,(-utente(_,Ns,_,_,_,_,_,_,_,_));
11   -excecao(utente(Id,_,_,_,_,_,_,_,_,_)),R),
12   length(R,1)).
```

Invariantes para verificar que o mês da data de nascimento é válido:

```
1 %conhecimento positivo
2 +utente(_,_,_,_,D,_,_,_,_,_)::(dataM(D,R),R>0,R<13).
3
4 %conhecimento negativo
5 +(-utente(_,_,_,_,D,_,_,_,_,_))::(dataM(D,R),R>0,R<13).
```

Invariantes para verificar que o centro de saúde no qual o utente está a ser registado existe:

```
1 %conhecimento positivo
2 +utente(_,_,_,_,_,_,_,_,_,CS)::centroSaude(CS,_,_,_,_,_).
3
4 %conhecimento negativo
5 +(-utente(_,_,_,_,_,_,_,_,_,CS))::centroSaude(CS,_,_,_,_,_).
```

Predicado para o registo de utentes com várias datas de nascimento (para inserir conhecimento imperfeito do tipo impreciso):

```

1 regUtente(Id,Ns,N,D,E,T,M,P,DC,C, Datas):-
2     evolucao(utente(Id,Ns,N,D,E,T,M,P,DC,C), Datas).

```

De notar também que foi feita uma evolução para este tipo de casos, que é apresentada a seguir:

```

1 evolucao(utente(Id,NS,N,DN,E,T,M,P,DC,CS),ListaDatas):-
2     cicloevolucao(utente(Id,NS,N,_,E,T,M,P,DC,CS),[DN|ListaDatas]).
3
4 cicloevolucao(utente(Id,NS,N,_,E,T,M,P,DC,CS),[H|Tail]):-
5     evolucao(excecao(utente(Id,NS,N,H,E,T,M,P,DC,CS))),
6     cicloevolucao(utente(Id,NS,N,_,E,T,M,P,DC,CS),Tail).
7 cicloevolucao(_,[]).

```

Remoção

Predicado para a remoção de utentes com várias datas de nascimento:

```

1 removeUtente(Id):- excecao(utente(Id,Ns,N,D,E,T,M,P,DC,C)), involucao(utente
    (Id,Ns,N,D,E,T,M,P,DC,C),data_imp).

```

De notar também que foi feita uma involução para este tipo de casos, que é apresentada a seguir:

```

1 involucao(utente(Id,NS,N,_,E,T,M,P,DC,CS),data_imp):-
2     solucoes((excecao(utente(Id,NS,N,D,E,T,M,P,DC,CS))),excecao(utente(Id,NS
    ,N,D,E,T,M,P,DC,CS)),L),
3     removeL(L).

```

Análise de Resultados

```

?- faselVacinacao(L).
L = [18, 20, 9, 3, 1].

?- regUtente(19,122678902,'Lucia',data(15,03,1969),'lucia@gmail.com',91557261,'Portimao','Enfermeiro',['Insuficiencia Cardiaca'],2,[data(15,03,1968),data(15,03,1970)]).
true.

?- faselVacinacao(L).
L = [19, 18, 20, 9, 3, 1].

?- regVacinacao(3,19,'2-05-2021','Pfizer',1).
true.

?- segundaToma(L).
L = [4, 7, 19].

```

Figura 4.1: Teste de registo de utente com conhecimento impreciso

```

?- excecao(utente(18,_,_,D,_,_,_,_,_)).
D = data(7, 7, 1921) ;
D = data(7, 7, 1920) ;
false.

?- removeUtente(18).
true.

?- excecao(utente(18,_,_,D,_,_,_,_,_)).
false.

```

Figura 4.2: Teste de remoção de utente com conhecimento impreciso

4.3.2 Registo de Staff

Registo

Verificar que o ID não se repete (só foi necessário acrescentar o conhecimento negativo):

```
1 %conhecimento negativo
2 +(-staff(Id,-,-,-)) :: (solucoes(Id,-staff(Id,-,-,-),R),length(R,1)).
```

4.4 Atualização de Conhecimento

Com estes novos predicados, temos a possibilidade de atualizar a informação de utentes e membros do staff inseridos previamente na base de conhecimento.

Atualizar utente com morada incerta:

```
1 atualizarMoradaUtente(Id,NM) :-
2     utente(Id,NS,N,DN,E,T,M,P,DC,CS),
3     demo(utente(Id,NS,N,DN,E,T,NM,P,DC,CS),R),
4     R = desconhecido,
5     nao(nulo(M)),
6     remove(utente(Id,NS,N,DN,E,T,M,P,DC,CS)),
7     evolucao(utente(Id,NS,N,DN,E,T,NM,P,DC,CS)).
```

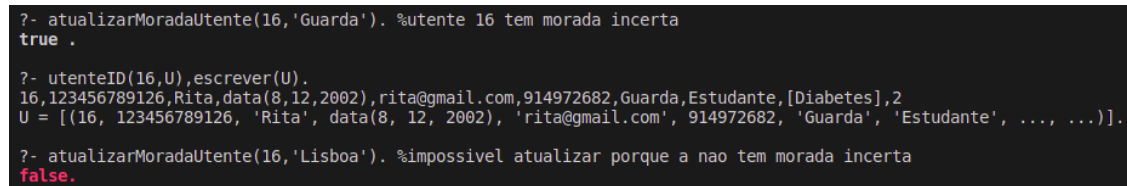
Atualizar utente com email incerto:

```
1 atualizarEmailUtente(Id,NE):-
2     utente(Id,NS,N,DN,E,T,M,P,DC,CS),
3     demo(utente(Id,NS,N,DN,NE,T,M,P,DC,CS),R),
4     R = desconhecido,
5     nao(nulo(E)),
6     remove(utente(Id,NS,N,DN,E,T,M,P,DC,CS)),
7     evolucao(utente(Id,NS,N,DN,NE,T,M,P,DC,CS)).
```

Atualizar staff com email incerto:

```
1 atualizarEmailStaff(ID,NE):-
2     staff(ID,IDS,N,E),
3     demo(staff(ID,IDS,N,NE),R),
4     R = desconhecido,
5     nao(nulo(E)),
6     remove(staff(ID,IDS,N,E)),
7     evolucao(staff(ID,IDS,N,NE)).
```

4.4.1 Análise de resultados



```
?- atualizarMoradaUtente(16,'Guarda'). %utente 16 tem morada incerta
true .

?- utenteID(16,U),escrever(U).
16,123456789126,Rita,data(8,12,2002),rita@gmail.com,914972682,Guarda,Estudante,[Diabetes],2
U = [(16, 123456789126, 'Rita', data(8, 12, 2002), 'rita@gmail.com', 914972682, 'Guarda', 'Estudante', ..., ...)].

?- atualizarMoradaUtente(16,'Lisboa'). %impossivel atualizar porque a nao tem morada incerta
false.
```

Figura 4.3: Teste da atualização de utente com morada incerta

```

?- regUtente(19,122678902,'Lucia',data(15,03,1969),email_incerto,91557261,'Portimao','Enfermeiro',['Insuficiencia Cardiaca'],2).
true .

?- utente(19,_,_,_,Email,_,_,_,_).
Email = email_incerto.

?- atualizarEmailUtente(19,'novoemail').
true .

?- utente(19,_,_,_,Email,_,_,_,_).
Email = novoemail.

?- atualizarEmailUtente(19,'novoemail2').
false.

```

Figura 4.4: Teste da atualização de utente com email incerto

```

?- atualizarEmailStaff(10,'novoemail'). %staff com email interdito
false.

?- atualizarEmailStaff(7,'novoemail'). %staff com email incerto
true .

?- staff(7,CS,NM,EM).
CS = 1,
NM = 'Silvia',
EM = novoemail.

```

Figura 4.5: Teste da atualização de staff com email incerto

4.5 Coerência da Base de Conhecimento

De forma a garantir que a base de conhecimento é congruente, foram implementados um conjunto de invariantes, que serão mencionados de seguida:

Invariante que garante que não existe conhecimento perfeito repetido:

```

1 +T :: ( solucoes (T,T,R) , length (R,1) ) .

```

Invariante que garante que não existe conhecimento negativo repetido:

```

1 +(-T) :: ( solucoes (T,-T,R) , length (R,1) ) .

```

Invariante que garante que não existe conhecimento positivo a contradizer o conhecimento negativo e vice-versa.

```

1 +T :: nao(-T) .
2 +(-T) :: nao(T) .

```

Invariante que garante que não existe conhecimento imperfeito repetido

```

1 +(execacao(T)) :: ( solucoes (T,execacao(T),R) , length (R,1) ) .

```

4.6 Predicados atualizados

Tal como em casos anteriores, devido a ter sido inserido conhecimento imperfeito na base de conhecimento foi necessário atualizar alguns dos predicados que tínhamos implementado na fase anterior.

```

1 utenteID(Id, R) :- solucoes ((Id,NS,N,D,E,T,M,P,DC,C), (utente(Id,NS,N,D,E,T,
M,P,DC,C);execacao(utente(Id,NS,N,D,E,T,M,P,DC,C))), R) .

```

```

1 idUtentes(L) :- solucoes(Id, (utente(Id, -, -, -, -, -, -, -, -); excecacao(utente(
    Id, -, -, -, -, -, -, -, -))), L1), eliminaRepetidos(L1, L).

```

```

1 utenteFV(Id, 3) :- solucoes(I, ((utente(I, -, -, -, -, -, -, -, -); excecacao(utente(I,
    -, -, -, -, -, -, -, -))), nao(utenteFV(I, 2)), nao(utenteFV(I, 1))), L),
    member1(Id, L).

```

```

1 fase3Vacinacao(R) :- solucoes(Id, (utente(Id, -, -, -, -, -, -, -, -); excecacao(utente(
    Id, -, -, -, -, -, -, -, -))), L), eliminaRepetidos(L, L2), fase12Vacinacao(
    L2), subtrair(L2, L12, R).

```

```

1 cumpreCritérios(Id, F) :-
2     (utente(Id, -, -, -, -, -, -, -, -, P, DC, -); excecacao(utente(Id, -, -, -, -, -, -, -, -, P, DC, -))),
3     criterio(F, DataC, Doenca, Profissao),
4     dataUtente(Id, DataU), maiorData(DataU, DataC, DataU), (memberList(DC,
    Doenca) ; member1(P, Profissao)).
5
6 dataUtente(Id, data(D, M, A)) :-
7     solucoes(data(Dia, Mes, Ano), (utente(Id, -, -, -, -, -, -, -, -, -, -, -, -)
    ; excecacao(utente(Id, -, -, -, -, -, -, -, -, -, -, -, -))), L),
8     maxData(L, data(12, 12, 2022), data(D, M, A)).

```

4.7 Sistema de inferência Atualizado

Com a inserção do conhecimento imperfeito, foi necessário atualizar o nosso sistema de inferência. Porém, em vez de atualizar o sistema elaborado na primeira fase decidimos criar um novo, com o nome de *demo*. Este retorna verdade caso exista uma prova explícita na base de conhecimento em que o parâmetro Questão seja verdadeiro, falso se existir na base de conhecimento uma negação da Questão e desconhecido se existir na base de conhecimento algum tipo de conhecimento imperfeito.

```

1 demo(Questao, verdadeiro) :- Questao.
2 demo(Questao, falso) :- ~Questao.
3 demo(Questao, desconhecido) :- nao(Questao), nao(~Questao).

```

Tal como na primeira fase, foi criado um predicado que permite responder a várias perguntas ao mesmo tempo, de forma a aumentar a eficiência do nosso Sistema de Inferência, designado por **demoLista**:

```

1 demoLista([], []).
2 demoLista([Questao | Qs], [R | Rs]) :- demo(Questao, R), demoLista(Qs, Rs).

```

4.7.1 Análise de resultados

```
?- demo(utente(15,123456789125,'Lara',data(05,10,2010),'lara@gmail.com',914972681,'Guimaraes','Estudante',['Diabetes'],1),R). %negacao forte
R = falso .

?- demo(utente(18,123456789128,'Marcia',data(07,07,1921),'marcia@gmail.com',914972684,'Fafe','Bombeiro',[],1),R).%data imprecisa
R = desconhecido.

?- demo(utente(18,123456789128,'Marcia',data(07,07,1923),'marcia@gmail.com',914972684,'Fafe','Bombeiro',[],1),R). %data imprecisa (nao uma das possibilidades)
R = falso .

?- demo(utente(18,123456789128,'Marcia',data(07,07,1920),'marcia@gmail.com',914972684,'Fafe','Bombeiro',[],1),R). %data imprecisa
R = desconhecido.

?- demo(staff(7,1,'Silvia','novoemail'),R). %email incerto
R = desconhecido.

?- demo(staff(10,1,'Magda','novoemail'),R). %email interdito
R = desconhecido.

?- demo(utente(1,12345678901,'Rui',data(15,03,1969),'rui@gmail.com',914972655,'Guimaraes','Cantor',['Diabetes','Insuficiencia Renal'],1),R). %utente registado
R = verdadeiro .
```

Figura 4.6: Teste do sistema de inferência com conhecimento imperfeito

Capítulo 5

Conclusão

No final deste trabalho prático, o grupo conclui que foi desenvolvido com sucesso um sistema de caracterização de um universo na área de vacinação da COVID-19, que inclui diversos tipos de conhecimento desde *utentes*, *staff*, *centro de saúde e vacinação* sugeridos pela equipa docente a *critérios* definidos por nós.

Neste projeto foram implementadas todas as funcionalidades necessárias para a representação de conhecimento perfeito (positivo e negativo) e imperfeito (incerto, impreciso e interdito). Foram desenvolvidos vários invariantes importantes para uma correta manipulação da base de conhecimento tanto para a adição como para a remoção do mesmo. Relativamente à problemática da evolução e da involução de conhecimento garantimos que todos os casos foram abrangidos de forma a construir um sistema fiável e funcional para qualquer tipo de conhecimento. Foram ainda desenvolvidos sistemas de inferência de forma a demonstrar a eficiência e veracidade do trabalho desenvolvido. Assim dito, consideramos que foram cumpridos todos os requisitos mínimos estabelecidos no enunciado, tendo sido desenvolvidos e documentados vários predicados para cumprir essas expectativas.

Podemos então, com êxito, afirmar que consolidamos e exercitamos os nossos conhecimentos em programação lógica atingindo, portanto, o objectivo deste projeto.