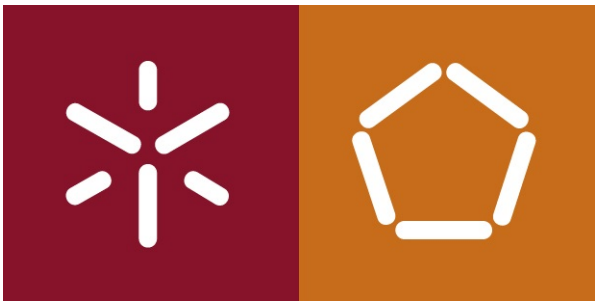
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Sistema de Representação de Conhecimento e Raciocínio

Programação em Lógica - Grupo 11

Trabalho realizado por:	*Número*
Adriana Martins Gonçalves	A75119
Bruno Alexandre Martins Carvalho	A89476
Eduardo Jorge Santos Teixeira	A84707
Fábio Fernandes Silva	A82331
Tiago Miguel Carvalho e Cunha	A87978

1 *Resumo*

No âmbito da unidade curricular de **Sistemas de Representação de Conhecimento e Raciocínio**, foi-nos proposto a elaboração de um trabalho prático cuja finalidade é desenvolver um sistema capaz de caracterizar um universo de discurso na área da vacinação global da população portuguesa no contexto atual.

A realização deste problema terá de ser desenvolvido através da utilização da linguagem de programação *PROLOG*, linguagem essa que foi abordada ao longo de toda a unidade curricular.

Ao longo do relatório iremos explicar as decisões e abordagens que tomamos ao longo do desenvolvimento do projeto, de forma a obter o resultado final desejado.

Conteúdo

1 ***Resumo*** **1**

2 ***Introdução*** **3**

3 ***Preliminares*** **4**

4 ***Descrição do trabalho e análise de resultados*** **5**
4.1 Evolução e Involução do Conhecimento 5
4.2 Invariantes 5
4.3 Base de conhecimento Extra : Profissões de Risco e Fase 7
4.4 Sobre a Base de conhecimento 8
4.5 Apresentação e Análise de Resultados 8
4.5.1 Fases de vacinação 8
4.5.2 Identificar Utentes não Vacinados 11
4.5.3 Identificar Utentes Vacinados 11
4.5.4 Identificar Utentes vacinados indevidamente 12
4.5.5 Identificar Utentes não vacinados candidatos a vacinação 13
4.5.6 Identificar Utentes a quem falta a segunda toma da vacina 14
4.5.7 Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas 15

5 ***Conclusões e Sugestões*** **16**

6 ***Referências*** **17**

7 ***Anexos*** **18**
7.1 Predicados Auxiliares 18
7.2 Base de Conhecimento Inicial 20

2 *Introdução*

No âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, foi-nos proposto o desenvolvimento de um projeto, divido em 2 partes, utilizando como linguagem de programação o *PROLOG*, sendo esta uma linguagem lógica e recomendada na representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas.

Esta primeira fase tem como objetivo organizar toda a informação relativa à vacinação da população portuguesa.

Para uma melhor organização de toda a informação, consideramos como base de conhecimento um **Utente**, um **Centro de Saúde**, o **Staff** e a **Vacinação** com adição de **Profissões de Risco** e **Fase**.

3 *Preliminares*

Tal como referido anteriormente, todo este trabalho se baseia em programar em *PROLOG* a organização da vacinação em Portugal através das quatro fontes de conhecimento identificadas que passamos a explicar cada uma delas.

Começando pela primeira fonte, o **Utente** tem associado um ID, um Nome, uma Data de Nascimento, um Email, um Telefone, uma Morada, uma Profissão, um conjunto de Doenças Crónicas associadas ao próprio e o ID do Centro de Saúde.

Seguidamente, o **Centro de Saúde** é composto por um ID, um Nome, uma Morada, um Telefone e um Email.

O **Staff** tem associado um ID de cada funcionário, o ID do Centro de Saúde, um Nome e um Email.

A **Vacinação**, composta pelo Staff, pelos Utentes, por uma Data, por uma Vacina, pela Toma e por uma Fase. Esta última (Fase) foi adicionada por nós, visto ser necessário saber a que fase da vacina nos estamos a referir, de forma a obtermos um correto resultado das *queries*.

Assim, consideramos que o panorama pode ser caracterizado da seguinte forma:

- **utente :** #Idutente, Nº Segurança_Social, Nome, Data_Nasc, Email, Telefone, Morada, Profissão, [Doenças_Crónicas], #Idcentro -> {V,F}
- **centro_saude :** #Idcentro, Nome, Morada, Telefone, Email -> {V,F}
- **staff :** #Idstaff, #Idcentro, Nome, email -> {V,F}
- **vacinacao_Covid :** #Idtaff, #utente, Data, Vacina, Toma,Fase -> {V,F}

No que diz respeito ao **utente**, este tem como identificador único o ID do Utente, de forma a que cada ID seja único. Tem ainda o ID do Centro de Saúde em que este está registado.

Relativamente ao **centro_saude** tem como identificador único o ID do Centro. Assim, não é possível que exista mais do que um Centro de Saúde com o mesmo ID.

Quanto ao **staff** tem como identificador único o ID do Staff, ou seja o ID do funcionário. Tem ainda o ID do Centro de Saúde onde este funcionário trabalha.

A **vacinacao_Covid** será o mais importante visto que irá ter os componentes a avaliar .

4 *Descrição do trabalho e análise de resultados*

4.1 Evolução e Involução do Conhecimento

Em Prolog, para introduzir e remover informação da base de conhecimento, é necessária a criação de algumas regras que permitam controlar a entrada e saída de informação. A evolução do conhecimento é a regra que assegura que, na introdução de um elemento, este não é repetido, ou seja, este não se encontra já na base de conhecimento. Quanto à involução, é uma regra que garante que um elemento não existente na base de conhecimento não é removido desta. Ambas as regras são testadas através do teste que é feito aos invariantes, pois se alguma destas falhar, a inserção/remoção também falhará.

A implementação dos invariantes é, portanto, essencial para o controlo da informação na base de conhecimento.

```
% Evolução - insercao de novo Termo
evolucao(Termo):- findall(Invariante,+Termo::Invariante,Lista),
    insercao(Termo),teste(Lista).


insercao(Termo):- assert(Termo).
insercao(Termo):- retract(Termo), !, fail. %cut operator para

%Involucao - remocao de termo

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

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

% testa se todos os predicados são verdadeiros
teste([]).
teste([R|Lr]):- R, teste(Lr).
```

Figura 1: Evolução e Involução da base de conhecimento

4.2 Invariantes

Relativamente do invariante da Figura 2, referente aos **Utentes**, vemos que este é composto por uma expressão de inserção, através de *+utente*, uma expressão de remoção de ocorrências,

através de *-utente* e o invariante referencial ambos permitem que o ID de cada utente seja único e um número inteiro e sendo assim, o que o identifica inequivocamente.

```
%Invariante Estrutural: para nao permitir inserção de ocorrências de conhecimento repetido a nivel de utente
+utente(IDU,_,_,_,_,_,_,_,_,IDCENTRO) :: (findall( (IDU,IDCENTRO), utente(IDU,_,_,_,_,_,_,_,_,IDCENTRO),S),
                comprimento(S,N), N == 1).

%Invariante Estrutural: Não remover utente que nao esteja na Base de Conhecimento
-utente(IDU,_,_,_,_,_,_,_,_,IDCENTRO) :: (findall( (IDU,IDCENTRO), (utente(IDU,_,_,_,_,_,_,_,_,IDCENTRO)),S),
                comprimento(S,N),
                N ==0).

%Invariante Referencial - ID sao inteiros
+utente(IDU,_,_,_,_,_,_,_,_,IDCENTRO) :: (
        integer(IDU),
    integer(IDCENTRO)
).
```

Figura 2: Invariante relativo aos *Utentes*

O mesmo sucede ao invariante da Figura 3, referente aos **Centros de Saúde**, em que tem uma expressão de inserção *+centro_saude*, uma expressão de remoção *-centro_saude* e ainda um invariante referencial.Ambos que permitem que cada centro de saúde possua apenas um ID único que o identifique e que este seja um inteiro.

```
%Invariante Estrutural: para permitir inserção de ocorrências de conhecimento repetido a nivel de centro_saúde
+centro_saude(IDCENTRO,_,_,_,_) :: (findall( (IDCENTRO), (centro_saude(IDCENTRO,_,_,_,_)),S),
                comprimento(S,N),
                N == 1).

%Invariante Estrutural: Não remover centro_Saude que nao esteja na Base de Conhecimento
-centro_saude(IDCENTRO,_,_,_,_) :: (findall( (IDCENTRO), (centro_saude(IDCENTRO,_,_,_,_)),S),
                comprimento(S,N),
                N ==0).
%Invariante Referencial - ID inteiro
+centro_saude(IDCENTRO,_,_,_,_) :: (
    integer(IDCENTRO)
).
```

Figura 3: Invariante relativo aos *Centros de Saúde*

O invariante relativo ao **Staff** segue a mesma linha dos invariantes anteriores, com *+staff* para inserção de novas ocorrências, *-staff* para remoção de staff e um invariante referencial.Ambos permitem que cada funcionário do staff possua um ID único e seja um inteiro.

```
%Invariante Estrutural: para permitir inserção de ocorrências de conhecimento repetido a nivel de staff
+staff(IDS,IDCENTRO,_,_) :: (findall( (IDS,IDCENTRO), (staff(IDS,IDCENTRO,_,_)),S),
                comprimento(S,N),
                N ==1).
%Invariante Estrutural: Não remover staff que nao esteja na Base de Conhecimento
-staff(IDS,IDCENTRO,_,_) :: (findall( (IDS,IDCENTRO), (staff(IDS,IDCENTRO,_,_)),S),
                comprimento(S,N),
                N ==0).

%Invariante Referencial - IDs sao inteiros
+staff(IDS,IDCENTRO,_,_) :: (
        integer(IDS),
    integer(IDCENTRO)
).
```

Figura 4: Invariante relativo ao *Staff*

Por último, o invariante relativo à **Vacinação** possui *+vacinacao_Covid* que permite inserir ocorrências de conhecimento repetido ao nível da vacinação, *-vacinacao_Covid* para não remover uma vacinação que não esteja na base de conhecimento, e um invariante referencial de forma a que o número de Toma apenas seja 1 ou 2.

```
%Invariante Estrutural: para permitir inserção de ocorrências de conhecimento repetido a nivel de vacinacao_Covid
+vacinacao_Covid(ID,IDSTAFF,_,_,T,F) :: (findall( (ID,IDSTAFF,T,F), (vacinacao_Covid(ID,IDSTAFF,_,_,T,F)),S),
                    comprimento(S,N),
                N == 1).
%Invariante Estrutural: Não remover vacinacao que nao esteja na Base de Conhecimento
-vacinacao_Covid(ID,IDSTAFF,_,_,T,F) :: (findall( (ID,IDSTAFF,T,F), (vacinacao_Covid(ID,IDSTAFF,_,_,T,F)),S),
                    comprimento(S,N),
                    N == 0).

%Invariante Referencial
+vacinacao_Covid(_,_,_,_,T,_) :: (
    T=1; %operador de disjuncao -> ;
    T=2
).
```

Figura 5: Invariante relativo à *Vacinação*

4.3 Base de conhecimento Extra : Profissões de Risco e Fase

Após a analise cuidada do enunciado e vendo o que foi proposto desenvolver nesta fase, decidimos adicionar novos conhecimentos aos já existentes: as **Profissões de Risco** e a **Fase**.

Posto isto, as bases de conhecimento **Profissões de Risco** e **Fase** têm a seguinte estrutura:

- **profissoes_risco :** [Profissão] -> {V,F}
- **fase :** #Idutente,Fase -> {V,F}

Nos invariantes relativos à **fase** achamos importante que existisse um invariante de inserção *+fase* que permite a inserção de ocorrências de conhecimento repetido e, ao contrário dos invariantes anteriores, este não possui um invariante de remoção. Por último possui um invariante de referência relativos ao ID do Utente e ao ID da fase de vacinação, enquanto que o ID de Utente não é repetido, ID da fase apenas possui os valores em baixo representados, ou seja, o número de fases que consideramos,4.

```
%Invariante Estrutural: para permitir insercao de ocorrências de conhecimento repetido a nível de fase
+fase(IDU,FASE) :: (findall( (IDU,FASE), (fase(IDU,FASE)),S),
                comprimento(S,N),
                N ==1).

%Invariante Referencial - IDs sao inteiros
+fase(IDU,IDF) :: (
        integer(IDU),
    IDF = 1;
    IDF=2;
    IDF=3;
    IDF=4
).
```

Figura 6: Invariante relativo à *Fase*

Em **profissoes_Risco** iremos ter todas as profissões que consideramos ser de risco e com prioridade na vacinação. Este conhecimento tem impacto apenas no registo de utentes na fase 3. Tratamos deste conhecimento como se fosse conhecimento fixo e inalterável

4.4 Sobre a Base de conhecimento

A nossa base de conhecimento é importante para conseguirmos testar todas as funcionalidades do sistema, visto que este foi criado de raiz pelo nosso grupo. Assim, esta base procura abranger todos os casos omissos à primeira vista, daí termos tido especial atenção nesta parte. A nossa base de conhecimento inicial encontra-se em anexo.

4.5 Apresentação e Análise de Resultados

Para cada um dos subtópicos em baixo representados foram dados exemplos sobre a Base de Conhecimento que referimos nos anexos.

4.5.1 Fases de vacinação

De acordo com o requerido no enunciado, decidimos dividir as fases de vacinação em 4 fases. Na primeira fase serão vacinadas todas as pessoas de idade igual ou superior a 65 anos, na segunda, pessoas que possuem doenças crónicas, na terceira, pessoas em profissões de risco e na quarta e última fase, todas as pessoas que não pertencem a nenhum dos critérios das fases anteriores. Aquando da insercao de um novo termo na nossa base, termo esse Utente, ao registar será feita a análise de qual a fase a que esse utente pertence e se é válido.

Assim, temos os seguintes predicados para o registo de pessoas nas diversas fases de vacinação:

```
registaUtente(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro):- registaUtenteFase1(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro),
                                                                       !.

registaUtenteFase1(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro):-verificaIdade(X/Y/Z),
                                                                          evolucao(utente(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro)),
                                                                          evolucao(fase(ID,1)),
                                                                          write('Utente registado em Fase1').


verificaIdade(X/Y/Z):- A is 2021-Z,
                       A >= 65.
```

Figura 7: Registo de uma pessoa na fase 1 de vacinação

```
registaUtente(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro):- registaUtenteFase2(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro),
                                                                       !.

registaUtenteFase2(ID,Seg,Nome,Data,Email,Tel,Mor,Prof,Doenca,IDCentro):- comprimento(Doenca,N),
                                                                          N>=1,
                                                                          evolucao(utente(ID,Seg,Nome,Data,Email,Tel,Mor,Prof,Doenca,IDCentro)),
                                                                          evolucao(fase(ID,2)),
                                                                          write('Utente registado em Fase2').
```

Figura 8: Registo de uma pessoa na fase 2 de vacinação

```
registaUtente(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro):- registaUtenteFase3(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro),
                                                                       !.

registaUtenteFase3(ID,Seg,Nome,Data,Email,Tel,Mor,Prof,Doenca,IDCentro):- profissoes_risco(L),
                                                                          pertence(Prof,L),
                                                                          evolucao(utente(ID,Seg,Nome,Data,Email,Tel,Mor,Prof,Doenca,IDCentro)),
                                                                          evolucao(fase(ID,3)),
                                                                          write('Utente registado em Fase3').
```

Figura 9: Registo de uma pessoa na fase 3 de vacinação

```
registaUtente(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro):- registaUtenteFase4(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro),
                                                                       !.

registaUtenteFase4(ID,Seg,Nome,Data,Email,Tel,Mor,Prof,Doenca,IDCentro):- evolucao(utente(ID,Seg,Nome,Data,Email,Tel,Mor,Prof,Doenca,IDCentro)),
                                                                          evolucao(fase(ID,4)),
                                                                          write('Utente registado em Fase4').
```

Figura 10: Registo de uma pessoa na fase 4 de vacinação

Em seguida apresentamos o predicado de registar um utente e também exemplos de registos de utentes nas várias fases de vacinação:

```
registaUtente(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro):- registaUtenteFase1(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro),
                                                                       !.
registaUtente(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro):- registaUtenteFase2(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro),
                                                                       !.
registaUtente(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro):- registaUtenteFase3(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro),
                                                                       !.
registaUtente(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro):- registaUtenteFase4(ID,Seg,Nome,X/Y/Z,Email,Tel,Mor,Prof,Doenca,IDCentro),
                                                                       !.
```

Figura 11: Predicado de registo de utentes nas fases de vacinação.

No seguinte exemplo o utente é registado na fase 3, pois pertence ao grupo de utentes com profissões de risco.

```
?- registaUtente(12,'1444',mario,20/04/1978,'mario@mail.com','91243769',chaves,enfermeiro,[],3).
Utente registado em Fase3
true.

?- utente(12,'1444',mario,20/04/1978,'mario@mail.com','91243769',chaves,enfermeiro,[],3).
true.
```

Figura 12: Exemplo 1 do registo de utentes.

No exemplo que se segue o utente é registado na fase 2, porque, apesar de ser um utente com profissão de risco, este pertence, também, ao grupo de doentes crónicos e por isso não será registado na fase 3, mas sim na fase 2.

```
?- registaUtente(13,'1460',paulo,22/06/1978,'roa@mail.com','912422110',chaves,medico,[asma],3).
Utente registado em Fase2
true.
```

Figura 13: Exemplo 2 do registo de utentes.

Para além disto, decidimos que era necessário também registar o staff, os centros de saúde e a vacinação, de forma a que as informações destes sejam também guardadas na base de conhecimento.

Portanto, temos:

```
%Registar Staff
registaStaff(IDS,IDC,Nome,Email):- evolucao(staff(IDS,IDC,Nome,Email)),
                                   write('Staff Registado').
%Registar Centro_saude
registaCentro(IDC,Nome,Morada,Telefone,Email):-evolucao(centro_saude(IDC,Nome,Morada,Telefone,Email)),
                                               write('Centro de Saude Registado').


%Registar Vacinacao_Covid
registaVac(ID,IDSTAFF,_,_,T,F):-evolucao(vacinacao_Covid(ID,IDSTAFF,_,_,T,F)),
                                 write('Vacinacao_Covid Registada').
```

Figura 14: Registo de uma pessoa da staff, de um centro de saúde ou de uma vacina.

4.5.2 Identificar Utentes não Vacinados

Para identificar utentes não vacinados recorremos a um *findAll* para selecionar todos os utentes, daí selecionamos também os utentes vacinados e aos dois aplicamos o predicado auxiliar eliminarComuns, para que sejam eliminados os utentes que já foram vacinados. Com isto só sobram os utentes não vacinados, por isso agrupamos simplesmente estes utentes.

```
identificaUtentesNaoVacinados(L):- findall((ID),utente(ID,Seg,Nome,Data,Email,Tel,Mor,Prof,Doenca,IDCentro),U),
                                   utentesIdVacinados(N),
                                   eliminarComuns(N,U,X),
                                   agrupaUtentesID(X,L).
```

Figura 15: Identificação dos utentes não vacinados.

Temos, na figura que se segue, este predicado a ser executado e o resultado do mesmo com a nossa base de conhecimento:

```
?- identificaUtentesNaoVacinados(L).
L = [(2, '2341', monteiro, 7/5/1999, 'mach@mail.com', '93566278', porto, taxista, [], 3), (3, '3412', torgal, 1/9/1989, 'dsaw@mail.com', '93436988', guimaraes
, medico, [], 2), (5, '1124', ze, 10/2/1972, 'opisa@mail.com', '91244012', chaves, camionista, [bronquite], 3), (9, '2564', roberta, 11/12/1973, 'opisa@mail.
com', '91244536', lisboa, medica, [asma], 3), (10, '7072', david, 13/3/1960, 'opisa@mail.com', '91243671', felgueiras, enfermeiro, [hipertensao], 3), (12, '1
444', mario, 20/4/1978, 'mario@mail.com', '91243769', chaves, enfermeiro, [], 3), (13, '1460', paulo, 22/6/1978, 'roa@mail.com', '912422110', chaves, medico,
[asma], 3)].
```

Figura 16: Exemplo do predicado para identificar utentes nao vacinados.

4.5.3 Identificar Utentes Vacinados

Para identificar utentes vacinados foi bastante mais simples pois foram utilizados dois predicados. O predicado utentesIdVacinados retorna os ID's dos utentes vacinados que possuem pelo menos uma toma no registo. Isto é feito recorrendo ao *findAll* que seleciona os utentes que levaram pelo menos a primeira toma da vacina. O segundo predicado auxiliar, dada uma lista de ID's vai agrupar os utentes pelos seus ID's. Aplicando estes dois predicados, um após o outro, vamos obter a identificação dos utentes vacinados. É de notar que nem todos estes utentes podem estar bem vacinados, ou seja terem a primeira e segunda toma.

```
identificaUtentesVacinados(L):- utentesIdVacinados(N),
                                agrupaUtentesID(N,L).

utentesIdVacinados(L):-findall((IDU),vacinacao_Covid(_,IDU,_,_,1,_),L1),
                       findall((IDU), vacinacao_Covid(_,IDU,_,_,2,_),L2),
                       concatenar(L1,L2,X),
                       remove_duplicados(X,L).

agrupaUtentesID([],[]).
agrupaUtentesID([H|T],[X|Xs]):- agrupaUtentesID(T,Xs),
                                listaUtentes(H,[X]).
```

Figura 17: Identificação dos utentes vacinados.

Na figura seguinte, apresentamos um exemplo do predicado anterior que identifica os utentes vacinados sem uma ordem específica.

```
?- identificaUtentesVacinados(L).
L = [(6, '1134', margarida, 8/12/1953, 'opisa@mail.com', '91244536', lisboa, policia, [asma], 3),  (11, '4102', manuel, 20/1/1948, 'opisa@mail.com', '91240116'
, chaves, policia, [], 3),  (1, '1234', eduardo, 10/1/1999, 'eduardo@mail.com', '93436278', fafe, estudante, [asma, diabetes], 1),  (4, '4123', mariana, 19/3/1
979, 'ds@mail.com', '91236278', braga, tenista, [asma, hipertensao], 1),  (7, '1572', daniela, 15/3/1970, 'opisa@mail.com', '91243671', felgueiras, medica, [hi
pertensao], 3),  (8, '1002', martim, 24/1/1978, 'opisa@mail.com', '91240116', chaves, enfermeiro, [], 3)].
```

Figura 18: Identificação dos utentes vacinados.

4.5.4 Identificar Utentes vacinados indevidamente

Em primeiro lugar, é de notar que se entende por pessoa vacinada indevidamente aquela que é vacinada na fase errada, ou seja, um utente que está, por exemplo, na fase 3 de vacinação e é vacinada na fase 2 ou na fase 4. A partir desta ideia, começamos por selecionar todos os IDs dos utentes e a respetiva fase em que foram vacinados e removemos os duplicados. De seguida, listamos todos os factos fases corretos. Com estas duas listas removemos/eliminamos os pares válidos e corretos(ID,$Fase$), deixando apenas uma lista com os IDs e fases dos utentes indevidamente vacinados. Por fim, agrupamos todos os ID's indevidamente vacinados.

```
identificaUtentesIndevidoVac(L):-findall((IDU,FASE) ,vacinacao_Covid(_,IDU,_,_,_,FASE),L1),
                                 remove_duplicados(L1,X),%Par com idutente e fase
                                 listaFases(L2),%elementos a retirar
                                 eliminarComuns(L2,X,I), %I lista de pares (id,fase) indevidos
                                 listFaseToListId(I,IDs),
                                 agrupaUtentesID(IDs,L). %transformar pares em lista de id para dps passar para lista de utentes
listaUtentes(Id,L):-findall((Id,Seg,Nome,Data,Email,Tel,Mor,Prof,Doenca,IDCentro),utente(Id,Seg,Nome,Data,Email,Tel,Mor,Prof,Doenca,IDCentro),L).


listaFases(L):-findall((IDU,FASE), fase(IDU,FASE),L).



listFaseToListId([],L).
listFaseToListId([(ID,F)],L):- append([ID],[],L).
listFaseToListId( [(ID,F)|T] ,L):- append([ID],N,L),
                                   listFaseToListId(T,N).
```

Figura 19: Identificação dos utentes indevidamente vacinados.

Além disto decidimos criar outro predicado que indica os utentes bem vacinados, para provar mais uma vez o nosso raciocínio acerca de como um utente é ou não devidamente vacinado. Em relação a este predicado decidimos selecionar os utentes indevidamente vacinados e, a partir da lista de todos os utentes e respetivas fases de vacinação, retirar os indevidamente vacinados, restando apenas os que foram devidamente vacinados. Por fim, retornamos apenas esses utentes.

```
identificaUtentesBemVac(L):- findall((IDU), vacinacao_Covid(_,IDU,_,_,2,_),L1),
                             identificaUtentesIndevidoVac(L2),
                             listFaseToListId(L2,X),
                             eliminarComuns(X,L1,N),
                             agrupaUtentesID(N,L).
```

Figura 20: Identificação dos utentes devidamente vacinados.

Mostramos ainda o exemplo deste predicado na seguinte figura.

```
?- identificaUtentesIndevidoVac(L).
L = [(8, '1002', martim, 24/1/1978, 'opisa@mail.com', '91240116', chaves, enfermeiro, [], 3), (7, '1572', daniela, 15/3/1970, 'opisa@mail.com', '91243671', fe
lgueiras, medica, [hipertensao], 3)] ■
```

Figura 21: Exemplo dos utentes indevidamente vacinados.

4.5.5 Identificar Utentes não vacinados candidatos a vacinação

Neste tópico fizemos questão de criar um predicado que escolhida uma fase pelo utilizador, para essa fase, obtemos todos os utentes que não estão vacinados e que fazem parte dessa fase. Em primeiro lugar foi criado um outro predicado que retorna a lista de IDs dos utentes por fase de vacinação. Após essa listagem fomos obter os IDs de todos os utentes que pertencem

a essa fase, vacinados e não vacinados. Por último os IDs são filtrados todos os que não foram vacinados e pertencem a essa própria fase, sendo depois agrupados em Utentes.

```
identificaFase(L,N):-  utentesIdVacinadosFase(X,N), %tenho os utentes vacinados
                       findall((ID), fase(ID,N),L1), %utentes na fase N
                       eliminarComuns(X,L1,L2),
                       agrupaUtentesID(L2,L).


utentesIdVacinadosFase(L,N):-findall((IDU),vacinacao_Covid(_,IDU,_,_,1,N),L1),
                       findall((IDU), vacinacao_Covid(_,IDU,_,_,2,N),L2),
                       concatenar(L1,L2,X),
                       remove_duplicados(X,L).
```

Figura 22: Utentes candidatos à vacinação.

Assim, apresentamos na imagem que se segue um exemplo dos utentes candidatos a serem vacinados.

```
|    indentificaFase(L,2).
Correct to: "identificaFase(L,2)"? yes
L = [(5, '1124', ze, 10/2/1972, 'opisa@mail.com', '91244012', chaves, camionista, [bronquite], 3), (7, '1572', daniela, 15/3/1970, 'opisa@mail.com', '91243671
', felgueiras, medica, [hipertensao], 3), (9, '2564', roberta, 11/12/1973, 'opisa@mail.com', '91244536', lisboa, medica, [asma], 3), (10, '7072', david, 13/3
/1960, 'opisa@mail.com', '91243671', felgueiras, enfermeiro, [hipertensao], 3), (13, '1460', paulo, 22/6/1978, 'roa@mail.com', '912422110', chaves, medico, [a
sma], 3)]
```

Figura 23: Exemplo dos utentes candidatos à vacinação.

4.5.6 Identificar Utentes a quem falta a segunda toma da vacina

Para esta funcionalidade começamos por selecionar todos os utentes com a primeira toma da vacina e todos os utentes com a segunda toma da vacina. Após a seleção, eliminamos os comuns, ou seja, os utentes que tinham a primeira e a segunda toma da vacina, deixando apenas aqueles que tinham a primeira toma, mas não a segunda. Por fim, identificamos retirámos os Utentes com vacinação indevida e agrupamos os Utentes desses IDs.

```
identificaFaltaToma2(L):-findall((IDU),vacinacao_Covid(_,IDU,_,_,1,_),L1),
                       findall((IDU), vacinacao_Covid(_,IDU,_,_,2,_),L2),
                       eliminarComuns(L2,L1,L),
                       agrupaUtentesID(X,T), %utentes devidamente e indevidamente vacinados a quem falta 2ªa toma
                       identificaUtentesIndevidoVac(I), %utentes indevidademente vacinados
                       eliminarComuns(I,T,L).
```

Figura 24: Identificação dos utentes os quais falta a 2ª toma da vacina.

Deste modo, apresentamos um exemplo deste predicado na figura que se segue.

```
|    identificaFaltaToma2(L).
L = [(1, '1234', eduardo, 10/1/1999, 'eduardo@mail.com', '93436278', fafe, estudante, [asma, diabetes], 1)]
```

Figura 25: Exemplo dos utentes os quais falta a 2ª toma da vacina.

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

Através de Invariantes mencionados e representados em cima podemos obter um sistema de inserção e remoção adequado ao nosso projeto. O sistema de inferência destacado em baixo faz com que exista resposta às questões de raciocínio do nosso trabalho.

```
% Sistema de inferência: Resposta -> {V,F}
si( Questao,verdadeiro ) :- Questao.
si( Questao,falso ) :- -Questao.
si(Questao,desconhecido):- nao(Questao), %se conhecimento nao existe então é desconhecido
                           nao(-Questao).
%Predicado que se não for possivel ver que questao é verdadeiro entao é falso
nao(Questao):-
    Questao,!,fail.
nao(Questao).
```

Figura 26: Sistema de inferência.

```
?- si(staff(1,4,guilherme,'ti@mail.com'),Q).
Q = verdadeiro .
```

Figura 27: Exemplo de pergunta e resposta para um elemento de staff existente.

```
?- si(staff(3,4,vanessa,'vamosne@mail.com'),Q).
Q = desconhecido.
```

Figura 28: Exemplo de pergunta e resposta para um elemento de staff não existente.

Visto que *staff(3,4,vanessa,'vamosne@mail.com')* não pertence à nossa Base de Conhecimento não conseguimos dizer se o facto é verdadeiro ou falso, deste modo facto toma o valor de desconhecido.

5 *Conclusões e Sugestões*

Durante a elaboração desta primeira fase do trabalho prático da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, foram vários os desafios com que nos deparamos na tentativa de obter um resultado final o mais fidedigno e consistente possível.

Para obter estes resultados, tentamos seguir os ensinamentos lecionados nas aulas práticas, tendo estes permitido uma maior consolidação de conhecimentos no que diz respeito à linguagem de programação lógica *PROLOG*.

Através da realização deste projeto ficou demonstrado o nosso conhecimento em relação aos temas abordados.

Assim, concluímos que nosso desempenho ao longo deste trabalho foi positivo, visto que conseguimos encontrar forma de ultrapassar os desafios apresentados durante o desenvolvimento do mesmo. Este trabalho permitiu ainda aumentar a motivação para a próxima fase do projeto, pois teremos a oportunidade de aperfeiçoar e aprofundar os conhecimentos obtidos ao longo desta unidade curricular.

6 *Referências*

Para o desenvolvimento deste trabalho utilizamos como referência os conhecimentos adquiridos durante as aulas teóricas e praticas da UC, bem como todos os materiais disponibilizados na *blackboard* tal como slides e apontamentos.

7 *Anexos*

7.1 Predicados Auxiliares

Ao longo do desenvolvimento desta fase do projeto, foi necessária a criação de funções auxiliares de forma a simplificar a abordagem a tomar e assim obtermos mais facilmente os resultados que nos foram propostos.

- **pertence :** O predicado ***pertence*** verifica se um elemento pertence ou não a uma lista.

```
%Perdicado pertence em forma de funcao pertence -> {V,F}
pertence(X,[X|T]).
pertence(X,[H|T]):-pertence(X,T).
```

Figura 29: Predicado auxiliar pertence

- **teste :** O predicado ***teste*** testa se todos os predicados são verdadeiros.

```
% testa se todos os predicados são verdadeiros
teste([]).
teste([R|Lr]):- R, teste(Lr).
```

Figura 30: Predicado auxiliar teste

- **concatenar :** O predicado ***concatenar*** concatena duas listas, isto é, retorna uma única lista que é a junção das duas outras listas fornecidas.

```
%Predicado que faz concat de T e Lista para se tornar Xs
concatenar([],L,L).
concatenar(L,[],L).
concatenar([H | T], L, [H | R]) :- concatenar(T,L,R).
```

Figura 31: Predicado auxiliar concatenar

- **remove_duplicados :** O predicado ***remove_ duplicados*** remove de uma lista todos os elementos repetidos resultado numa lista sem elementos duplicados.

```
%Predicado que remove elementos duplicados

remove_duplicados(LI,LF):-remove(LI,[],LF).
```

Figura 32: Predicado auxiliar remove_duplicados

- **remove :** O predicado ***remove*** elimina um determinado elemento de uma lista, este é auxiliar de **remove_duplicados** .

```
%Se não pertencer adiciona à cabeça da nossa lista vazia
remove([],L,L).
remove([H|T],L,X):- pertence(H,L),
                    remove(T,L,X).
remove([H|T],L,X):- remove(T,[H|L],X).
```

Figura 33: Predicado auxiliar remove

- **comprimento :** O predicado ***comprimento*** tem como retorno o valor do comprimento de uma lista.

```
% Extensão do Predicado comprimento: ListaElem, Comp -> {V,F}
comprimento([],0).
comprimento([X|L], C) :- comprimento(L, N), C is N+1.
```

Figura 34: Predicado auxiliar comprimento

- **eliminarComuns :** O predicado ***eliminarComuns*** elimina elementos comuns de uma lista quando comparada com outra.

```
%Extensao do Predicado para eliminar elementos comuns de uma lista noutra skrskr
%(lista de itens a retirar , lista onde se retira , lista return)
eliminarComuns([],[],L).
eliminarComuns(N,[],[]).
eliminarComuns([],N,N).
eliminarComuns([H|T],N,L):-removeTodasOcorrenciasElemento(H,N,L1),
                           eliminarComuns(T,L1,L).
```

Figura 35: Predicado auxiliar eliminarComuns

- **removeTodasOcorrenciasElemento :** O predicado ***removeTodasOcorrenciasElemento*** remove todas as ocorrências de um dado elemento numa lista.

```
%Extensao do Predicado para remover todas as ocorrencias de uma elemento
removeTodasOcorrenciasElemento(X,[],[]).
removeTodasOcorrenciasElemento(X,[X|T],L):- removeTodasOcorrenciasElemento(X,T,L),!.
removeTodasOcorrenciasElemento(X,[H|T],[H|L]):- removeTodasOcorrenciasElemento(X,T,L). %adicionar
```

Figura 36: Predicado auxiliar removeTodasOcorrenciasElemento

7.2 Base de Conhecimento Inicial

```
utente(1,'1234',eduardo,10/1/1999,'eduardo@mail.com','93436278',fafe,estudante,[asma,diabetes],1).%FASE2
utente(2,'2341',monteiro,07/5/1999,'mach@mail.com','93566278',porto,taxista,[],3).%FASE4
utente(3,'3412',torgal,01/9/1989,'dsaw@mail.com','93436988',guimaraes,medico,[],2).%FASE4
utente(4,'4123',mariana,19/3/1979,'ds@mail.com','91236278',braga,tenista,[asma,hipertensao],1).%FASE2
utente(5,'1124',ze,10/2/1972,'opisa@mail.com','91244012',chaves,camionista,[bronquite],3).%FASE2

utente(6,'1134',margarida,08/12/1953,'opisa@mail.com','91244536',lisboa,policia,[asma],3).%FASE 1
utente(7,'1572',daniela,15/03/1970,'opisa@mail.com','91243671',felgueiras,medica,[hipertensao],3). %FASE2
utente(8,'1002',martim,24/1/1978,'opisa@mail.com','91240116',chaves,enfermeiro,[],3).%FASE 3

utente(9,'2564',roberta,11/12/1973,'opisa@mail.com','91244536',lisboa,medica,[asma],3).%FASE 2
utente(10,'7072',david,13/03/1960,'opisa@mail.com','91243671',felgueiras,enfermeiro,[hipertensao],3).%FASE2
utente(11,'4102',manuel,20/01/1948,'opisa@mail.com','91240116',chaves,policia,[],3).%FASE1
```

```
centro_saude(1,'Alto_Ave',povoa,'234145167','alto@mail.com').
centro_saude(2,'Centro saude',fafe,'254458167','saude@mail.com').
centro_saude(3,'Chaves Centro',chaves,'234145006','ch@mail.com').
centro_saude(4,'Centro_L',lisboa,'254171267','za@mail.com').
centro_saude(5,'Riba Centro',lisboa,'250878167','riza@mail.com').
```

```
staff(1,1,rita,'rti@mail.com').
staff(2,1,gilberto,'gilberto@mail.com').
staff(3,1,monica,'saledge@mail.com').
staff(1,2,joao,'juanito@mail.com').
staff(2,2,andreia,'zeca@mail.com').
staff(3,2,belo,'li4@mail.com').
staff(1,3,rui,'salmon@mail.com').
staff(2,3,ze,'salmonze@mail.com').
staff(3,3,ana,'cena@mail.com').
staff(1,4,guilherme,'ti@mail.com').
staff(2,4,carolina,'carol@mail.com').
```

```
vacinacao_Covid(5,6,14/02/21,pfeizer,1,1).
vacinacao_Covid(5,6,21/04/21,pfeizer,2,1).
vacinacao_Covid(1,11,01/03/21,pfeizer,1,1).
vacinacao_Covid(1,11,01/04/21,pfeizer,2,1).
vacinacao_Covid(3,1,21/03/21,pfeizer,1,2).
vacinacao_Covid(3,4,21/03/21,pfeizer,1,2).
vacinacao_Covid(3,4,21/04/21,pfeizer,2,2).
vacinacao_Covid(2,7,21/03/21,pfeizer,1,3). %utente vacinado indevidamente para dar exemplo utente 7 pertence a fase 2 e não 3
vacinacao_Covid(2,8,22/03/21,pfeizer,1,2). %utente vacinado indevidamente para dar exemplo utente 8 pertence a fase 3 e não 2
vacinacao_Covid(2,8,22/04/21,pfeizer,2,2). %utente vacinado indevidamente para dar exemplo utente 8 pertence a fase 3 e não 2



profissoes_risco([medico,medica,enfermeiro,enfermeira,auxiliar_limpeza,auxiliar_lar,professor,auxiliar_escola,policia]).



fase(6,1).
fase(11,1).
fase(1,2).
fase(4,2).
fase(5,2).
fase(7,2).
fase(9,2).
fase(10,2).
fase(8,3).
fase(2,4).
fase(3,4).
```