# UNIVERSIDADE DO MINHO

## Programação Lógica e Invariantes

MIEI

Sistemas de Representação de Conhecimento e Raciocínio

(2º Semestre - 2020/2021)


A85489 Simão Monteiro

A89557 Pedro Veloso

A89587 Carlos Preto

A93785 Ricardo Gomes


Março 2021

# Resumo

De forma a consolidar os conhecimentos até agora obtidos nesta Unidade Curricular, foi desenvolvido um pequeno exercício prático, um Sistema de Representação de Conhecimento e Raciocínio com capacidade para lidar com um caso bastante atual que é a vacinação da população contra o vírus Covid-19. Foram exigidas algumas funcionalidades base e ainda adicionadas algumas outras que fazem sentido dado o tema apresentado.

# Índice

# Índice de figuras

# Introdução

No âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio foi proposto o desenvolvimento de um sistema capaz de lidar com alguns problemas relacionados com a vacinação da população para combater o Covid-19.

Este trabalho foi desenvolvido na linguagem de programação em lógica PROLOG e implementa funcionalidades como a definição de fases de vacinação tendo em conta alguns critérios, verificar se algum utente foi vacinado indevidamente, entre outras que irão ser discutidas posteriormente neste relatório.

Para além destas funcionalidades base, foram também desenvolvidas funcionalidades adicionais, e alguns predicados auxiliares que se vão encontrar em anexo no final do relatório, que se acharam relevantes para o caso prático em questão.

# Preliminares

Para o desenvolvimento deste trabalho o conhecimento adquirido até ao momento nas aulas mostrou-se essencial para a sua correta realização.

Também a consulta da página da DGS e os próprios noticiários que são transmitidos pela televisão nacional diariamente, tiveram um papel essencial no desenvolvimento deste trabalho dado que, informações relevantes como por exemplo, os critérios de vacinação, podem ser obtidas a partir destas fontes.

# Descrição do Trabalho

## 1.1. Base de conhecimento

O sistema desenvolvido carateriza o cenário da vacinação da população, posto isto considerou-se que o conhecimento é dado do seguinte modo:

**utente:** #Idutente, Nº Seguranca Social, Nome, Data_Nasc, Email, Telefone, Morada, Profissão, [Doenças_Crónicas], #CentroSaúde ⇝ { 𝕍, 𝔽 }

**centro_saúde:** #Idcentro, Nome, Morada, Telefone, Email ⇝ { 𝕍, 𝔽 }

**staff:** #Idstaff, #Idcentro, Nome, email ⇝ { 𝕍, 𝔽 }

**vacinação_Covid:** #Staf, #utente, Data, Vacina, Toma ⇝ { 𝕍, 𝔽 }

**profissaoPrioritaria:** Profissao ⇝ { 𝕍, 𝔽 }

**doença_risco:** Doenca ⇝ { 𝕍, 𝔽 }

### 1.1.1. Utente

O conhecimento expresso em "utente", representa as entidades que vão ser incluídas nos planos de vacinação posteriormente.

Assim, um utente é caraterizado pelo seu identificador único (Id_utente), pelo número de segurança social, nome, data de nascimento, email, telefone, morada, profissão, lista de doenças crónicas e ainda o identificador do seu centro de saúde.

Os seguintes predicados, apresentados na figura 1, representam exemplos de utentes conhecidos.

```
utente(1 , 11111111111, 'Simao', data(14,12,1972), 'simao@mail.com', 930551476, 'Largo Coronel Brito Gorjao - Mafra','Gestor',
       ['Asma'], 2).
utente(2 , 22222222222, 'Telmo', data(27,12,1967), 'telmo@mail.com', 961773477, 'Praca Jose Maximo da Costa - Lourinha',
       'Engenheiro Informatico', ['Colesterol','Hipertensao'], 1).
utente(3 , 33333333333, 'Tiago', data(6,11,2000), 'tiago@mail.com', 968895013, 'Rua dos 3 Vales - Almeirim','Motorista', [], 1).
utente(4 , 44444444444, 'Vasco', data(23,6,1953), 'vasco@mail.com', 961773477, 'Largo Doutor Dario Gandra Nunes - Amadora',
       'Consultor', ['Cancro'], 2).
utente(5 , 55555555555, 'Raul', data(9,12,1953), 'raul@mail.com', 918851034, 'Rua Real Fabrica do Vidro - Amadora','Professor', [], 2).
utente(6 , 66666666666, 'Ana', data(12,12,1933), 'ana@mail.com', 935465241, 'R. Prof. Francisco Gentil - Barreiro','Coach', [], 3).
utente(7 , 77777777777, 'Augusta', data(17,12,1982), 'augusta@mail.com', 966148612, 'Rua S. Tomas de Aquino - Braga','Medico', ['Asma'], 4).
utente(8 , 88888888888, 'Carlota', data(7,12,1980) , 'carlota@mail.com', 930568014, 'Largo Coronel Brito Gorjao - Coimbra',
       'Enfermeiro', [], 5).
utente(9 , 99999999999, 'Cristina', data(22,12,1980), 'cristina@mail.com', 930570152, 'Rua Dr. Antonio Jose de Almeida - Covilha',
       'Medico', [], 6).
utente(10, 10101010101, 'Eva', data(3,12,1962), 'eva@mail.com', 935465241, 'Rua Assis Leao - Evora','Enfermeiro', [], 7).
utente(11, 11111111111, 'Rui', data(29,6,1973), 'rui@mail.com', 998773477, 'Avenida Doutor Dario Gandra Nunes - Porto',
       'Mecânico', ['Cancro'], 2).
```

*Figura 1: Conhecimento Utente*

### 1.1.2. Centro de Saúde

Tal como o utente, também um centro de saúde é conhecido pelo seu identificador único (Id_centro), para além do seu nome, morada, telefone e email. É nos diferentes centros de saúde que os utentes vão levar as diferentes tomas das vacinas. Os seguintes predicados, expostos na próxima figura, representam exemplos do conhecimento sobre centros de saúde.

```
centroSaude(1, 'Posto de Analises Clinicas do Almeirim', 'Rua General Humberto Delgado', 243592604, 'posto.almeirim@synlab.pt').
centroSaude(2, 'Posto de Analises Clinicas da Amadora' , 'Largo Doutor Dario Gandra Nunes', 914155759, 'laboratorio.amadora@synlab.pt').
centroSaude(3, 'Posto de Analises Clinicas do Barreiro', 'Avenida do Bocage', 910728308, 'posto.venteira@synlab.pt').
centroSaude(4, 'Posto de Analises Clinicas de Braga', 'Rua Ambrosio Santos', 935465241, 'posto.ambrosio.santos@synlab.pt').
centroSaude(5, 'Posto de Analises Clinicas de Coimbra', 'Praceta Professor Robalo Cordeiro', 239701512, 'laboratorio.coimbra@synlab.pt').
centroSaude(6, 'Posto de Analises Clinicas da Covilha', 'Av. Infante D. Henrique', 275313383, 'posto.covilha@synlab.pt').
centroSaude(7, 'Posto de Analises Clinicas de Evora', 'Praceta Horta do Bispo', 266759590, 'laboratorio.evora@synlab.pt').
```

*Figura 2: Conhecimento Centro de Saúde*

### 1.1.3. Staff

O staff, representa as entidades que operam num centro de saúde. São os elementos do staff que estão responsáveis por dar a vacina aos utentes e o seu conhecimento é expresso através de um identificador do centro em que trabalha, um identificador pessoal, nome e email.

Os seguintes predicados representam exemplos de staff:

```
staff(1, 1, 'Tatiana', 'tatiana@staff.com').
staff(2, 2, 'Augusta', 'augusta@staff.com').
staff(3, 3, 'Carlota', 'carlota@staff.com').
staff(4, 4, 'Cristina', 'cristina@staff.com').
staff(5, 5, 'Eva', 'eva@staff.com').
staff(6, 6, 'Rui', 'rui@staff.com').
staff(7, 7, 'Vitor', 'vitor@staff.com').
```

*Figura 3: Conhecimento Staff*

### 1.1.4. Vacinação Covid

Na vacinação covid é expresso o conhecimento angariado sobre a evolução da vacinação.

Desta forma, deve ser conhecida informação sobre o staff que vacinou o utente, manifestada sobre a forma do identificador do staff. Também deve ser conhecido o utente que tomou a vacina, a data em que a toma foi dada e o nome da vacina que o utente tomou. Uma vez que determinadas vacinas, para realizar efeito devem ser tomadas mais que uma vez, deve ser conhecido o número de vezes que o utente já realizou a toma.

De seguida apresentam-se alguns predicados, que representam exemplos do conhecimento vacinação covid:

```
%Fase1
vacinacaoCovid(3, 7, data(1,2,2021), 'astrazeneca', 1).
vacinacaoCovid(4, 8, data(1,1,2021), 'astrazeneca', 1).
vacinacaoCovid(6, 9, data(1,1,2021), 'astrazeneca', 1).
vacinacaoCovid(7, 10, data(1,1,2021), 'astrazeneca', 1).

vacinacaoCovid(3, 7, data(20,2,2021), 'astrazeneca', 2).
vacinacaoCovid(4, 8, data(20,1,2021), 'astrazeneca', 2).
vacinacaoCovid(6, 9, data(20,1,2021), 'astrazeneca', 2).
vacinacaoCovid(7, 10, data(20,1,2021), 'astrazeneca', 2).

%Fase2
vacinacaoCovid(2, 1, data(10,3,2021), 'pfizer', 1).
vacinacaoCovid(2, 1, data(30,3,2021), 'pfizer', 2).
vacinacaoCovid(9, 11, data(20,3,2021), 'astrazeneca', 1).

%Fase3
vacinacaoCovid(5, 2, data(20,10,2021), 'astrazeneca', 1).
vacinacaoCovid(1, 3, data(20,1,2021), 'astrazeneca', 1).
```

*Figura 4: Conhecimento vacinação contra covid*

### 1.1.5. Profissão Prioritária

Sendo a covid uma doença que se transmite muito facilmente, é fundamental proteger pessoas que, devido às suas profissões estão em maior risco de contágio. Assim considerou-se relevante ter como base de conhecimento as profissões que devido à sua importância ou risco, são consideradas prioritárias na fase de vacinação.

Os seguintes predicados representam exemplos dessas profissões:

```
profissaoPrioritaria('Medico').
profissaoPrioritaria('Enfermeiro').
```

*Figura 5: Conhecimento profissão prioritária*

### 1.1.6. Doença de Risco

Devido à grande taxa de mortalidade característica da covid, que se acentua nas pessoas mais debilitadas, sendo pela idade ou devido ao seu historial clínico, considerou-se fundamental ter conhecimento das doenças que poderão influenciar nos sintomas associados à covid.

Os predicados, que constam na figura 6, representam exemplos de doenças de risco:

```
doenca_risco('Asma').
doenca_risco('Cancro').
```

*Figura 6: Conhecimento doença de risco*

## 1.2. Adição e remoção de conhecimento

De forma a adicionar e remover conhecimento da base de conhecimentos é necessário recorrer a mecanismos de programação lógica como invariantes que sob um conjunto de restrições devem resultar em verdade após uma inserção/remoção de conhecimento. A correta adição de conhecimento pode ser vista como uma evolução do mesmo, visto que a base de conhecimento fica com mais informação do que tinha anteriormente, pelo mesmo raciocínio se pode dizer que a remoção de conhecimento pode ser vista como uma involução deste.

Antes da explicação dos processos de adição e remoção de conhecimento é de notar dois predicados que se revelam muito importantes para o funcionamento destes processos.

O primeiro é o predicado *soluções*, presente na figura seguinte:

```
solucoes(X,P,S) :- findall(X,P,S).
```

*Figura 7: Predicado - Soluções*

Este predicado é importante pois permite encontrar todas as ocorrências de "X" que satisfazem o predicado "P", colocando por fim o resultado em "S".

O segundo predicado, utilizado na complementação do predicado *soluções,* chama-se *comprimento*:

```
comprimento( [],0 ).
comprimento( [_|L],N ) :-
    comprimento( L,N1 ),
    N is N1+1.
```

*Figura 8: Predicado - Comprimento*

Como o próprio nome sugere este predicado calcula o comprimento de uma lista "L" e coloca o seu resultado em N.

A utilização dos dois predicados em conjunto permite, por exemplo, impedir a adição de conhecimento repetido.

### 1.2.1. Evolução

A evolução do sistema dá-se quando a base de conhecimento do sistema ganha informação nova. Assim, para adicionar novo conhecimento deve-se recorrer ao predicado *evolucao*:

```
evolucao(Termo) :- solucoes(Inv, +Termo::Inv, S),
                   insere(Termo),
                   teste(S).
```

*Figura 9: Predicado - Evolução*

Ao utilizar este predicado, é possível garantir que o conhecimento angariado respeita uma série de invariantes. O conjunto de invariantes, obtido através do predicado *solucoes*, permite que o conhecimento representado respeite um conjunto de regras, sendo desta forma coerente. Para facilmente identificar um invariante relativo a uma inserção de um termo, coloca-se o símbolo '+' a preceder o nome desse termo.

O predicado *insere*, é responsável por inserir novo conhecimento, enquanto o predicado *teste*, realiza testes sobre todos os invariantes relativos ao "Termo". Contudo, como se observa na figura 9, o predicado insere primeiro o conhecimento e só depois testa a sua coerência. Assim, caso o predicado *teste* falhe, é necessário retirar o termo da base de conhecimento, como se pode observar na figura seguinte:

```
%------------------------------ - - - - - - - -
% Extensao do predicado teste: Lista -> {V, F}

teste([]).
teste([H | T]) :- H, teste(T).

%------------------------------ - - - - - - - -
% Extensao do predicado insere: Termo -> {V, F}

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

%------------------------------ - - - - - - - -
% Extensao do predicado remove: Termo -> {V, F}

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

*Figura 10: Predicado – Teste/Insere/Remove*

### 1.2.2. Involução

A involução do sistema dá-se quando a base de conhecimento do sistema perde informação.

Para que isto seja possível, sem perder a coerência do conhecimento, criou-se o predicado *involução*, representado de seguida:

```
involucao(Termo) :- Termo,
                    solucoes(Inv, -Termo::Inv, S),
                    remove(Termo),
                    teste(S).
```

*Figura 11: Predicado - Involução*

Neste caso o predicado *remove* elimina conhecimento. Ao colocar a variável "Termo" como condição para a remoção, pretende-se verificar a existência do conhecimento a remover. Posteriormente, realiza-se, de modo semelhante à evolução, a procura por invariantes relativos à remoção do conhecimento dado. Para identificar o invariante

coloca-se o símbolo '-', a preceder o nome desse termo. Posteriormente remove-se o elemento e testa-se a coerência. Caso o conhecimento fique incoerente volta-se a inserir o conhecimento, como observado na figura 10.

### 1.2.3. Utentes (Invariantes)

Na adição de um novo utente deve-se ter em conta as seguintes restrições:

- O identificador tem de ser um número;
- O número de segurança social tem de ter 11 dígitos;
- A data de nascimento tem de ser válida;
- O número de telefone tem de ter 9 dígitos;
- A idade não pode ser superior a 150 anos. Considera-se que inserir utentes com idade superior a 150 anos é impossível uma vez que não existe ninguém com essas características;
- Não pode existir um utente com o mesmo identificador.

```
+utente(ID,SS,_,DA,_,TS,_,_,_,PS) :: (integer(ID),
                                      SS >= 10000000000,
                                      SS =< 99999999999,
                                      data(DA),
                                      TS >= 100000000,
                                      TS =< 999999999,
                                      idade(DA, IDADE),
                                      IDADE > 0,
                                      IDADE =< 150,
                                      integer(PS),
                                      solucoes(ID,(utente(ID,_,_,_,_,_,_,_,_,_)),S ),
                                      comprimento( S,C ),
                                                    C == 1
                                     ).
```

*Figura 12: Invariante adição Utente*

Em relação à remoção, considera-se que se o utente estiver registado como vacinado, não pode ser removido. Com este invariante pretende-se garantir a coerência de conhecimento, não permitindo existir o conhecimento de ID's de utentes, sem conhecer toda a informação do utente.

```
-utente(ID,_,_,_,_,_,_,_,_,_) :: (solucoes(ID,(vacinacaoCovid(_,ID,_,_,_)),S ),
                                   comprimento( S,N ),
                                   N == 0
                                  ).
```

*Figura 13: Invariante remoção Utente*

Para realizar a remoção de um utente que já se encontre vacinado, deve ser primeiramente removida a informação sobre a sua vacinação e só depois pode remover-se o conhecimento relativo ao utente.

### 1.2.4. Centro de Saúde (Invariantes)

Para adicionar conhecimento de um centro de saúde, deve-se considerar as seguintes restrições:

- O identificador tem de ser um inteiro;
- O número de telefone tem de ter 9 dígitos;
- Não pode existir um centro de saúde com o mesmo identificador.

```
+centroSaude(ID,_,_,T,_) :: (integer(ID),
                             T >= 100000000,
                             T =< 999999999,
                             solucoes(ID,(centroSaude(ID,_,_,_,_)),S ),
                             comprimento( S,C ),
                                          C == 1
                            ).
```

*Figura 14: Invariante adição Centro de Saúde*

Para remover o conhecimento de um centro de saúde, deve-se garantir que não existe nenhum staff associado ao centro.

```
-centroSaude(ID,_,_,_,_) :: (solucoes(ID,(staff(_,ID,_,_)),S ),
                             comprimento( S,C ),
                                          C == 0
                            ).
```

*Figura 15: Invariante remoção Centro de Saúde*

### 1.2.5. Staff (Invariantes)

Para adicionar um staff, é necessário o conhecimento respeitar as seguintes restrições:

- O identificador do centro de saúde e do staff tem de ser um inteiro;
- O staff não pode ter o mesmo identificador que outro staff;
- Não pode estar associado a um centro de saúde desconhecido.

```
+staff(ID,IC,_,_) :: (integer(ID),
                      integer(IC),
                      centroSaude(IC,_,_,_,_),
                      solucoes(ID,(staff(ID,_,_,_)),S ),
                      comprimento( S,C ),
                                   C == 1
                     ).
```

*Figura 16: Invariante adição Staff*

Para remover do conhecimento um staff, o mesmo não pode estar conhecido como ativo na sua profissão, ou seja, não podem ser removidos staffs que já tenham realizado vacinações.

```
-staff(ID,_,_,_) :: (solucoes(ID,(vacinacaoCovid(ID,_,_,_,_)),S ),
                     comprimento( S,C ),
                     C == 0
                    ).
```

*Figura 17: Invariante remoção Staff*

### 1.2.6. Vacinação covid (Invariantes)

Na adição de conhecimento relativo à vacinação contra a covid, deve-se ter em conta que as seguintes restrições devem ser cumpridas:

- Não podem ser dados repetidos;
- Staff e utente têm que ser conhecidos;
- Data da vacinação tem de ser válida;
- A toma deve respeitar o número mínimo e máximo, ou seja: $Toma \in [0,2]$

```
+vacinacaoCovid(IS,IU,D,V,T) :: (integer(IS),
                                 integer(IU),
                                 utente(IU,_,_,_,_,_,_,_,_,_),
                                 staff(IS,_,_,_),
                                 data(D),
                                 integer(T),
                                 T >= 0,
                                 T < 3,
                                 solucoes(IS,(vacinacaoCovid(IS,IU,D,V,T)),S ),
                                 comprimento( S,C ),
                                              C == 1
                                ).
```

*Figura 18: Invariante adição Vacinação contra Covid*

Para a informação referente à vacinação, não se impõe qualquer restrição para a remoção de conhecimento.

### 1.2.7. Profissão prioritária (Invariantes)

Para profissões prioritárias apenas é permitido adicionar conhecimento que ainda não seja conhecido, não impondo qualquer restrição à remoção.

```
+profissaoPrioritaria(P) :: (solucoes(P,(profissaoPrioritaria(P)),S ),
                             comprimento( S,C ),
                                          C == 1
                            ).
```

*Figura 19: Invariante adição Profissão Prioritária*

### 1.2.8. Doença de risco (Invariantes)

Para o conhecimento referente a doenças consideradas mais perigosas em caso de contrair a covid, não pode ser adicionado conhecimento já adquirido. Contudo pode ser removido qualquer conhecimento.

```
+doenca_risco(D) :: (solucoes(D,(doenca_risco(D)),S ),
                     comprimento( S,C ),
                                    C == 1
                    ).
```

*Figura 20: Invariante adição Doença Risco*

## 1.3. Implementação das funcionalidades base

Nesta secção serão mostradas e explicadas as resoluções feitas pelo grupo para responder às funcionalidades bases.

**Nota:** Considerou-se que para cada toma da vacina seria adicionada uma nova *vacinacaoCovid,* isto é, se existir um utente que já levou as duas tomas da vacina, existirão duas *vacinacaoCovid* associadas a esse utente, uma para a primeira toma e outra para a segunda, tal como apresentado no exemplo a seguir.

```
vacinacaoCovid(3, 7, data(1,2,2021), 'astrazeneca', 1).
vacinacaoCovid(4, 8, data(1,1,2021), 'astrazeneca', 1).
vacinacaoCovid(6, 9, data(1,1,2021), 'astrazeneca', 1).
vacinacaoCovid(7, 10, data(1,1,2021), 'astrazeneca', 1).

vacinacaoCovid(3, 7, data(20,2,2021), 'astrazeneca', 2).
vacinacaoCovid(4, 8, data(20,1,2021), 'astrazeneca', 2).
vacinacaoCovid(6, 9, data(20,1,2021), 'astrazeneca', 2).
vacinacaoCovid(7, 10, data(20,1,2021), 'astrazeneca', 2).
```

*Figura 21: Exemplo de conhecimento semelhante da Vacinação contra o Covid*

### 1.3.1. Definição de Fases de Vacinação

Para resolver esta funcionalidade consideraram-se, de modo a simplificar, apenas a existência de 3 fases de vacinação.

```
% Extensao do predicado pessoasCandidatas: Fase, Lista_Pessoas -> {V,F}

pessoasCandidatas(1, X) :- fase1(X).
pessoasCandidatas(2, X) :- fase2(X).
pessoasCandidatas(3, X) :- fase3(X).
```

*Figura 22: Predicado – Pessoas Candidatas*

Para a primeira fase de vacinação, convocou-se todas as pessoas que trabalham com profissões prioritárias.

```
%-------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado pessoasCandidadatasFase1: ID_Utente, Nome_Utente -> {V, F}
% Verifica se um utente apresenta as caracteristicas para a fase 1 da vacinação

pessoasCandidadatasFase1(ID,N) :- utente(ID,_,N,_,_,_,_,P,_,_),
                                  profissaoPrioritaria(P).

%-------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado fase1: Lista -> {V, F}
% Verifica se na lista estão todos os utentes candidatos à fase 1 de vacinação
% por conseguinte informa todas as pessoas candidatas à fase 1

fase1(X):- solucoes((ID,N), pessoasCandidadatasFase1(ID,N), X).
```

*Figura 23: Predicado – Fase 1*

Para a segunda fase de vacinação considerou-se todas as pessoas com idade superior a 80 anos ou pessoas que tenham alguma doença de risco.

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado pessoasCandidadatasFase2: ID_Utente, Nome_Utente -> {V, F}
% Verifica se um utente apresenta as caracteristicas para a fase 2 da vacinação

pessoasCandidadatasFase2(ID,N) :- utente(ID,_,N,DA,_,_,_,_,_,_),
                                  idade(DA, IDADE),
                                  IDADE >= 80,
                                  fase1(F1),
                                  nao(pertence((ID,N),F1)).
pessoasCandidadatasFase2(ID,N) :- utente(ID,_,N,_,_,_,_,_,D,_),
                                  doenteRisco(D),
                                  fase1(F1),
                                  nao(pertence((ID,N),F1)).

%---------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado fase2: Lista -> {V, F}
% Verifica se na lista estão todos os utentes candidatos à fase 2 de vacinação
% por conseguinte informa todas as pessoas candidatas à fase 2

fase2(X):- solucoes((ID,N), pessoasCandidadatasFase2(ID,N), X).
```

*Figura 24: Predicado – Fase 2*

Por fim, na terceira fase, inclui-se todos os utentes, que não estão escalados nas fases anteriores.

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado pessoasCandidadatasFase3: ID_Utente, Nome_Utente -> {V, F}
% Verifica se um utente apresenta as caracteristicas para a fase 3 da vacinação

pessoasCandidadatasFase3(ID,N) :- utente(ID,_,N,_,_,_,_,_,_,_),
                                  fase1(F1),
                                  fase2(F2),
                                  nao(pertence((ID,N),F1)),
                                  nao(pertence((ID,N),F2)).

%---------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado fase2: Lista -> {V, F}
% Verifica se na lista estão todos os utentes candidatos à fase 3 de vacinação
% por conseguinte informa todas as pessoas candidatas à fase 3

fase3(X):- solucoes((ID,N), pessoasCandidadatasFase3(ID,N), X).
```

*Figura 25: Predicado – Fase 3*

### 1.3.2. Identificar pessoas não vacinadas

Para a resolução desta funcionalidade, considera-se como não vacinado, caso o seu identificador não esteja associado a nenhuma *vacinaoCovid.*

```
utenteNaoVacinado(ID,N) :- utente(ID,_,N,_,_,_,_,_,_,_),
                                                nao(vacinacaoCovid(_, ID, _, _, _)).
utenteNaoVacinado(ID,N) :- utente(ID,_,N,_,_,_,_,_,_,_),
                                                vacinacaoCovid(_, ID, _, _, 0).

%---------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado pessoasNaoVacinadas: Lista_Pessoas -> {V,F}
% verifica se todas as pessoas na lista não estão vacinadas
% por conseguinte informa todas as pessoas que não estão vacinadas

pessoasNaoVacinadas(X) :- solucoes((ID,N), utenteNaoVacinado(ID,N), X).
```

*Figura 26: Predicado – Pessoas Não Vacinadas*

### 1.3.3. Identificar pessoas vacinadas

Uma pessoa é considerada vacinada, somente se já tiver tomado as 2 doses da vacina. Assim para encontrar todas as pessoas vacinadas deve-se realizar o seguinte predicado:

```
utenteVacinado(ID,N) :- utente(ID,_,N,_,_,_,_,_,_,_),
                                        vacinacaoCovid(_, ID, _, _, T),
                                                T >= 2.

%---------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do predicado pessoasVacinadas: Lista_Pessoas -> {V,F}
% verifica se todas as pessoas na lista estão totalmente vacinadas
% por conseguinte informa todas as pessoas que estão totalmente vacinadas

pessoasVacinadas(X) :- solucoes((ID,N), utenteVacinado(ID,N), X).
```

*Figura 27: Predicado – Pessoas Vacinadas*

### 1.3.4. Identificar pessoas vacinadas indevidamente

Para esta funcionalidade, admite-se que um utente foi vacinado indevidamente se pertencer a uma fase posterior à que está a decorrer, isto é, um utente é considerado vacinado de forma indevida se pertencer por exemplo à fase 3 e ainda houver pessoas da fase 2 ou 1 por serem vacinadas.

```
%---------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do predicado utenteVacinadoIndevidamente: ID_Utente , Nome_Utente, Fase -> {V,F}
% Verifica se um utente foi bem vacinado, considerando a fase de vacinação atual

utenteVacinadoIndevidamente(ID, N, 1) :- utente(ID,_,N,_,_,_,_,_,_,_),
                                         vacinacaoCovid(_, ID, _, _, _),
                                         fase1(F1),
                                         nao(pertence((ID,N),F1)).
utenteVacinadoIndevidamente(ID, N, 2) :- utente(ID,_,N,_,_,_,_,_,_,_),
                                         vacinacaoCovid(_, ID, _, _, _),
                                         fase3(F3),
                                         pertence((ID,N),F3).
utenteVacinadoIndevidamente(ID, N, 2) :- utente(ID,_,N,_,_,_,_,_,_,_),
                                         vacinacaoCovid(_, ID, D, _, _),
                                         fase1(F1),
                                         nao(pertence((ID,N),F1)),
                                         ultimaDataFase(1,UltData),
                                         antes(D,UltData).
utenteVacinadoIndevidamente(ID, N, 3) :- utente(ID,_,N,_,_,_,_,_,_,_),
                                         vacinacaoCovid(_, ID, D, _, _),
                                         fase3(F3),
                                         pertence((ID,N),F3),
                                         ultimaDataFase(2,UltData),
                                         antes(D,UltData).

%---------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do predicado listaVacinadosIndividamente: Fase, Lista_Pessoas, Acumulador, Resultado -> {V,F}
% Constroi uma lista com as pessoas vacinadas indevidamente

listaVacinadosIndividamente(_,[],A,A).
listaVacinadosIndividamente(FaseAtual, [(ID,N)|Resto], A, R) :- utenteVacinadoIndevidamente(ID,N, FaseAtual), listaVacinadosIndividamente(FaseAtual, Resto, [(ID,N)|A], R).
listaVacinadosIndividamente(FaseAtual, [_|Resto], A, R) :- listaVacinadosIndividamente(FaseAtual, Resto, A, R).

%---------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do predicado pessoasVacinadasIndevidamente: Lista -> {V,F}
% Verifica dada uma lista de utentes, se todos eles foram vacinados indevidamente
% por conseguinte informa todos os utentes vacinados indevidamente

pessoasVacinadasIndevidamente(X) :- solucoes((ID,N),utente(ID,_,N,_,_,_,_,_,_,_), L),faseAtual(FaseAtual), listaVacinadosIndividamente(FaseAtual,L,[],X).
```

*Figura 28: Predicado – Pessoas Vacinadas Indevidamente*

### 1.3.5. Identificar pessoas não vacinadas e candidatas a vacinação

Para identificar as pessoas não vacinadas e candidatas a vacinação procura-se por todos os utentes que ainda não foram vacinados e que pertencem à fase de vacinação que está a decorrer. Se estas condições se verificarem, então o utente é candidato.

```
utenteNaoVacinadoCandidato(FaseAtual, ID,N) :- utente(ID,_,N,_,_,_,_,_,_,_),
                                               pessoasCandidatas(FaseAtual,R),
                                               pertence((ID,N),R),
                                               nao(vacinacaoCovid(_,ID,_,_,_)).

%------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do predicado pessoasNaoVacinadasCandidatas: Lista -> {V,F}
% Verifica se todas as pessoas na lista são candidatas nao vacinadas à fase atual de vacinação
% por conseguinte informa todas as pessoas não vacindas candidatas

pessoasNaoVacinadasCandidatas(X) :- faseAtual(FaseAtual), solucoes((ID, N), utenteNaoVacinadoCandidato(FaseAtual, ID,N), X).
```

*Figura 29: Predicado – Pessoas Não Vacinadas Candidatas*

### 1.3.6. Identificar pessoas a quem falta a segunda toma da vacina

Para identificar os utentes a quem falta tomar a segunda toma da vacina, verificam-se todos os utentes que já tomaram a primeira toma, mas que ainda falta tomar a segunda.

```
pessoasFaltaSegundaFaseVacinacao(X) :- solucoes((ID,N),( utente(ID,_,N,_,_,_,_,_,_,_),
                                                         vacinacaoCovid(_,ID,_,_,1),
                                                         nao(vacinacaoCovid(_,ID,_,_,2))
                                                                                        ), X).
```

*Figura 30: Predicado – Pessoas Falta a 2ª Toma*

## 1.4. Funcionalidades Adicionais

### 1.4.1. Fase de vacinação

Esta funcionalidade consiste num um predicado que determina a fase da vacinação que está a decorrer no momento.

```
faseAtual(N) :- fase1(F1), faseConcluida(F1), fase2(F2), faseConcluida(F2), !, N is 3.
faseAtual(N) :- fase1(F1), faseConcluida(F1), !, N is 2.
faseAtual(N) :- N is 1.
```

Para determinar a fase atual de vacinação é necessário determinar todas as fases que já foram concluídas, desta forma criou-se o predicado *faseConcluida* que verifica se todos os utentes escalados para uma fase já foram vacinados com as duas tomas da vacina.

```
%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do predicado faseConcluida: Lista -> {V,F}
% Verifica se todas as pessoas na lista já tomaram a 2 toma da vacina

faseConcluida([]).
faseConcluida([(ID,_)|T]) :- vacinacaoCovid(_, ID, _, _, Toma), Toma =:= 2, faseConcluida(T).
```

*Figura 31: Predicado – Fase Concluída*

### 1.4.2. Término de uma fase

Esta funcionalidade permite saber a última data de vacinação de uma determinada fase.

```
ultimaDataFase([],A,A).
ultimaDataFase([(ID,_)|T],A,R) :- vacinacaoCovid(_, ID, D, _, _), depois(D,A), ultimaDataFase(T,D,R).
ultimaDataFase([(ID,_)|T],A,R) :- vacinacaoCovid(_, ID, D, _, _), antes(D,A), ultimaDataFase(T,A,R).

%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do predicado ultimaDataFase: Fase, Data -> {V,F}
% Verifica se a data dada corresponde á data da ultima vacina dada nessa faseAtual
% por conseguinte calcula a data da ultima vacina de uma fase

ultimaDataFase(N,R) :- pessoasCandidatas(N,P), faseConcluida(P), ultimaDataFase(P,data(1,1,1),R).
```

*Figura 32: Predicado – Última Data Fase*

### 1.4.3. Quem tomou determinada vacina

Com interesses estatísticos, criou-se o predicado *pessoasQueTomaramXvacina* que permite determinar todas as pessoas que tomaram uma determinada vacina.

```
pessoasQueTomaramXvacina(X,R) :- solucoes((ID,N),(utente(ID,_,N,_,_,_,_,_,_),vacinacaoCovid(_,ID,_,X,_)),R).
```

*Figura 33: Predicado – Pessoas que Tomaram X vacina*

### 1.4.4. Verificar que staff deu vacinas, numa fase

Funcionalidade que permite verificar quais elementos do staff deram pelo menos uma vacina a algum utente, numa fase de vacinação escolhida.

```
% Extensao do predicado staffPessoa: Lista, Acumulador, Resultado -> {V,F}
% Constroi uma lista com todos os elementos do staff que deram vacinas numa determinada fase

staffPessoa([],A,A).
staffPessoa([(ID1,_)|T], A, R) :- vacinacaoCovid(ID,ID1,_,_,_), staff(ID,_,N,_), nao(pertence((ID,N),A)), !, staffPessoa(T,[(ID,N)|A],R).
staffPessoa([_|T], A, R) :- staffPessoa(T,A,R).

%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do predicado staffAtivoFase: Fase, Lista_Staff -> {V,F}
% Verifica se todos os elementos do staff dado deram vacinas na fase indicada
% por conseguinte determinada todos os elementos do staff que deram vacinas na fase

staffAtivoFase(Fase,R) :- pessoasCandidatas(Fase,P), staffPessoa(P,[],R).
```

*Figura 34: Predicado – Staff ativo numa fase*

### 1.4.5. Verificar se os pares utente-staff estão corretamente associados

Funcionalidade que verifica se um dado utente foi vacinado por um dado staff.

```
utenteVacinadoStaff(IDStaff, NStaff, IDUtente, NUtente) :- staff(IDStaff,_,NStaff,_),
                                                           utente(IDUtente,_,NUtente,_,_,_,_,_,_,_),
                                                           vacinacaoCovid(IDStaff,IDUtente,_,_,_).
```

*Figura 35: Predicado – Utente Vacinado por Staff*

Utilizando o predicado pessoasVacinadasStaff também é possivel obter todos os pares existentes.

```
pessoasVacinadasStaff(X) :- solucoes((NStaff,NUtente), utenteVacinadoStaff(_,NStaff,_,NUtente), X).
```

*Figura 36: Predicado – Pessoas Vacinadas por um Staff*

### 1.4.6. Determina em que centro de saúde um utente foi vacinado

Com a criação desta funcionalidade pretende-se conhecer o identificador do centro de saúde, onde um utente foi vacinado. O utente pode ser indicado, informando tanto o nome, como o seu identificador.

```
lugarOndeVacinado(ID, R) :- integer(ID), !, vacinacaoCovid(IDStaff,ID,_,_,_), staff(IDStaff,R,_,_).
lugarOndeVacinado(Nome, R) :- utente(ID,_,Nome,_,_,_,_,_,_,_), vacinacaoCovid(IDStaff,ID,_,_,_), staff(IDStaff,R,_,_).
```

*Figura 37: Predicado – Lugar onde foi Vacinado*

# Conclusão

Com a realização do trabalho desenvolveu-se um sistema capaz de lidar com problemas relacionados à vacinação da covid-19.

Utilizando a linguagem de programação em lógica PROLOG, desenvolveu-se um sistema capaz de inferir conhecimento relativo a fases de vacinação, onde com o conhecimento obtido seleciona-se utentes para três fases de vacinação, diferentes. Para a primeira fase de vacinação são selecionados os utentes que devido à sua atividade profissional colocam-se todos os dias em risco de contrair a doença. Na segunda fase considera todos os utentes que ou possuem uma doença capaz de agravar os sintomas da covid ou que possuem uma idade superior a 80 anos. Para a terceira fase são selecionados todos os outros utentes.

Visou-se, também neste trabalho a criação de conhecimento sobre pessoas que já foram vacinas, ou sobre pessoas que ainda não foram vacinadas, até mesmo sobre pessoas que foram vacinadas, mas de forma indevida.

Alem deste conhecimento já referido também se implementou algum considerado adicional ao trabalho pedido, tal como: determinar a fase atual que a vacinação se encontra, determinar as pessoas que tomaram uma vacina, entre outro conhecimento já referido.

Por fim, considera-se que apesar da base de conhecimento não ser muito extensa desenvolveu-se um trabalho que utilizando o mecanismo de evolução, também desenvolvido, pode ser aprofundado, adicionado assim mais utentes, ou mesmo doenças de risco que levaria à mudança das pessoas selecionadas para a fase 2, por exemplo.

# Anexos

### *1.1. Predicado: não*

```
%-------------------------------- - - - - - - - - - -
% Extensao do predicado nao: Termo -> {V, F}

nao(T) :- T, !, fail.
nao(_).
```

### *1.2. Predicado: pertence*

```
%-------------------------------- - - - - - - - - - -
% Extensao do predicado pertence: X, Lista -> {V, F}

pertence(X, [X|_]).
pertence(X, [H|T]) :- X \= H, pertence(X, T).
```

### *1.3. Predicado: data*

```
% Extensao do predicado data: Dia, Mes, Ano -> {V, F}
data(D, 2, A) :-
    A >= 0,
        A mod 4 =:= 0,
        D >= 1,
        D =< 29.
data(D, M, A) :-
        A >= 0,
    pertence(M, [1,3,5,7,8,10,12]),
        D >= 1,
        D =< 31.
data(D, M, A) :-
        A >= 0,
    pertence(M, [4,6,9,11]),
        D >= 1,
        D =< 30.
data(D, 2, A) :-
        A >= 0,
    A mod 4 =\= 0,
        D >= 1,
        D =< 28.
data(data(D, M, A)) :- data(D, M, A).
```

### 1.4. Predicado: Antes e depois

```
% Extensao do predicado depois: Data, Data -> {V, F}
% Verifica se uma data vem depois de outra

depois(data(_,_,A1), data(_,_,A2)) :-
        A1 > A2.
depois(data(_,M1,A1), data(_,M2,A2)) :-
        A1 >= A2,
        M1 > M2.
depois(data(D1,M1,A1), data(D2,M2,A2)) :-
        A1 >= A2,
        M1 >= M2,
        D1 > D2.

%------------------------------ - - - - - - - - - -  -  -  -  -   -
% Extensao do predicado antes: Data, Data -> {V,F}
% Testa se uma data ocorre antes de uma outra data

antes(data(D1,M1,A1), data(D2,M2,A2)) :- nao(depois(data(D1,M1,A1), data(D2,M2,A2))).
```

### 1.5. Predicado: Idade

```
% Extensao do predicado idade: Data, Idade -> {V,F}
% Verifica/Calcula a idade de uma pessoa

idade(data(_,_,A), IDADE) :- data_atual(data(_,_,AA)),
                                                    IDADE is AA-A.
```

### 1.6. Predicado: doente de risco

```
% Extensao do predicado doenteRisco: Lista_Doentes -> {V,F}
% Verifica se alguma das doenças presentes na lista é considerada de risco

doenteRisco([]) :- !, fail.
doenteRisco([H|_]) :- doenca_risco(H).
doenteRisco([_|T]) :- doenteRisco(T).
```

### 1.7. Predicado: fase concluída

```
% Extensao do predicado faseConcluida: Lista -> {V,F}
% Verifica se todas as pessoas na lista já tomaram a 2 toma da vacina

faseConcluida([]).
faseConcluida([(ID,_)|T]) :- vacinacaoCovid(_, ID, _, _, Toma), Toma =:= 2, faseConcluida(T).
```

### 1.8. Predicado: Teste

```
% Extensao do predicado teste: Lista -> {V, F}

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