Universidade do Minho
Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Vacinação COVID

Grupo 23:
Benjamim Coelho, A89616
Júlio Alves, A89468
Henrique Neto, A89618
Carolina Vila Chã, A89495
Leonardo Marreiros, A89537

30 de abril de 2022

Resumo

O foco deste trabalho é construir um caso prático que seja capaz de demonstrar as funcionalidades subjacentes à utilização da linguagem de programação em lógica `prolog`, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas. No contexto da vacinação da população portuguesa em relação à pandemia atual, foi desenvolvido um sistema com capacidade para caracterizar um universo de discurso na área que pode ser descrito em **utentes**, **centros de saúde**, **staff** que presta serviços nesses centros de saúde, serviços esses dos quais fazem parte **vacinações Covid**.

Conteúdo

1 Introdução **1**

2 Preliminares **3**
2.1 Factos 3
2.2 Regras 4
2.3 Questões 4

3 Descrição do Trabalho e Análise de Resultados **6**
3.1 Construção da Base de Conhecimento 6
3.1.1 Definição das Fases de Vacinação 7
3.2 Funcionalidades Base 8
3.2.1 Identificar pessoas não vacinadas 8
3.2.2 Identificar pessoas vacinadas 8
3.2.3 Identificar pessoas vacinadas indevidamente 10
3.2.4 Identificar pessoas não vacinadas e que são candidatas a vacinação 10
3.2.5 Identificar pessoas a quem falta a segunda toma da vacina . . . 11
3.3 Funcionalidades Adicionais 12
3.3.1 Identificar utentes por critérios de seleção 12
3.3.2 Identificar os centros de saúde por critérios de seleção 13
3.3.3 Identificar vacinações por centro de saúde 14
3.3.4 Identificar os utentes de um staff/centro de saúde 14
3.3.5 Quantidade de vacinas por centro, por staff, e por centro de saúde 15
3.3.6 Contactos próximos dos utentes 16
3.4 Predicados Auxiliares 18
3.4.1 Teste 18
3.4.2 Comprimento 18
3.4.3 Contém 18
3.4.4 Intersetam 18
3.4.5 Interseção 19
3.4.6 Pertence 19
3.4.7 ApagaElemento 19
3.4.8 ApagaRepetido 20
3.4.9 SI 20
3.4.10 Não 20

3.4.11 Soluções 21
3.4.12 Utentes_aux 21
3.4.13 Vac_aux 22
3.4.14 Vacina_aux 22
3.5 Invariantes 23
3.5.1 Evolução da Base de conhecimento 23
3.5.2 Involução da Base de conhecimento 25

4 Conclusões e Sugestões **29**

5 Anexos **30**
5.1 Base de Conhecimento 30
5.2 Entidade Data 34

1 Introdução

Com a descoberta e propagação exponencial do vírus causador de COVID-19, o mundo abrandou de uma forma que não era vista desde a crise da gripe espanhola, sensivelmente 100 anos antes. Depois de um ano de ciclos de confinamento e desconfinamento, vacinas começaram a ser produzidas e distribuídas em grande volume.

O objetivo deste projeto é, neste contexto, o desenvolvimento de um sistema de conhecimento relacional com capacidade para caraterizar um universo de discurso na área da vacinação global da população portuguesa.

O sistema deverá ter implementadas as seguintes funcionalidades básicas:

- permitir a definição de fases de vacinação, definindo critérios de inclusão de utentes nas diferentes fases (e.g., doenças crónicas, idade, profissão)
- identificar pessoas não vacinadas
- identificar pessoas vacinadas
- identificar pessoas vacinadas indevidamente
- identificar pessoas não vacinadas e que são candidatas a vacinação
- identificar pessoas a quem falta a segunda toma da vacina
- desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas

No entanto, o conhecimento a tratar pode ser extendido, pelo que se implementaram as seguintes funcionalidades extra:

- Identificar todos os centros de saúde presentes.
- Identificar um centro de saúde através do seu identificador, nome ou telemóvel.
- Identificar todos os utentes vacinados por um dado centro de saúde.
- Identificar todos os utentes pertencentes a um dado centro de saúde.
- Identificar todos os utentes vacinados por um dado clínico, através do nome ou ID do clinico.
- Saber o número de vacinas que foram dadas no total, por um determinado clínico ou num determinado centro de saúde.

- Identificar, e saber quantos são, os elementos do staff de um centro de saúde.
- Identificar os utentes de uma localidade.
- Identificar contactos frequentes entre Utentes.
- Identificar estado de risco de um utente com base nos seus contactos frequentes.

2 Preliminares

A linguagem `prolog` é do tipo declarativo, e a lógica do programa é expressa em termos de relações, onde o cálculo é feito à base da consulta dessas relações.

Existe apenas um tipo de dados em `prolog`, o *termo*. Um termo pode, no entanto, descrever objetos distintos:

- **Átomos:** Átomos são normalmente strings, dígitos ou *underscores* `"_"` e começam com uma letra minúscula. Um átomo também pode ser uma string começada com uma letra maiúscula ou com espaços em branco, mas deve ser colocado entre aspas. Além disso, átomos não tem um significado implícito, como no exemplo,

  ```
  'Jacarias' 'centrogaia@gmail.com'  atomo
  ```

  onde o programa não tem qualquer indicação de que se trata de um nome próprio.

- **Números:** podem ser *float* ou *integer*.

- **Variáveis:** indicadas utilizando uma letra maiúscula no início da palavra, como por exemplo:

  ```
  Res     DataVacina     X
  ```

As relações são definidas por cláusulas, que podem ser de três tipos: Factos, Regras, e Questões.

2.1 Factos

Os factos expressam algo que é sempre verdadeiro. Factos são declarações que descrevem propriedades de objetos ou as suas relações. Imaginemos que queremos codificar que, por exemplo, a Manuela Figueiredo faz parte da staff de vacinação. Isto traduz-se em factos do Prolog terminados com um ponto final:

```
staff(1,1,'Manuela Figueiredo','manuelaf@gmail.com').
```

Neste exemplo, é a combinação dos valores entre os parêntesis que definem um elemento do staff. Esta coleção de factos e, posteriormente, de regras, constitui uma **base de conhecimento**. Transcreve o conhecimento de uma situação particular num formato lógico. Ao adicionar mais factos a esta base de conhecimento, expressamos outras propriedades, como um centro de saúde onde o Staff trabalha:

```
centro_saude(1, 'Centro Freamunde', 'Rua das Couves',
↪  100300400, 'centrofreamunde@gmail.com').
```

Em `Prolog`, **predicados** constituem relações entre objetos. Os objetos envolvidos numa relação são chamados **argumentos** e o número de argumentos de uma relação é a **aridade**.

2.2 Regras

As regras expressam algo que é verdadeiro, dependente da veracidade das condições, e são utilizadas para expressar novas relações com base em outras relações.

```
pertence(X,[X|L]).
pertence(X,[Y|L]):- pertence(X,L).
```

Neste caso, a regra `pertence` estabelece uma relação entre uma variável `X` e uma lista `[Y|L] ou [X|L]`.

Do modo como a primeira linha está escrita, indica ao programa que a variável e a cabeça da lista, sendo ambas representadas por `X`, têm que ser o mesmo elemento, ou seja, a variável `X` *unifica* com a cabeça de lista `X`.

Na segunda linha, ao escrever `X` para a variável, e `Y` para a cabeça de lista, estamos a indicar ao sistema que estas duas variáveis não podem unificar.

2.3 Questões

Uma questão é uma solicitação para provar ou recuperar informações da base de conhecimento, por exemplo, se um facto for verdadeiro. O Prolog responde que sim se puder provar, ou seja, sim se o facto estiver no banco de dados, ou não, se não puder: se o facto estiver ausente.

```
% Sabendo que:
% centro_saude(2, 'Centro Gaia', 'Rua do Hospital', 100300401,
%               'centrogaia@gmail.com').

% Interrogamos o sistema:
?- centro_saude(X, 'Centro Gaia', Y, 100300401, Z).

% Resposta:
```

```
X = 2,
Y = 'Rua do Hospital',
Z = 'centrogaia@gmail.com'.
```

Um termo contendo variáveis como `centro_saúde(X, 'Centro Gaia', 100300401, Y)` pode unificar com um facto compatível, como `centro_saúde(2, 'Centro Gaia', 'Rua do Hospital', 100300401, 'centrogaia@gmail.com')`, com as substituições `X = 2`, Y = 'Rua do Hospital' e Z = 'centrogaia@gmail.com'.

De modo semelhante, se a pergunta não conter variáveis que o sistema possa procurar, a resposta seria, neste caso,

```
% Interrogamos o sistema:
?- centro_saude(2, 'Centro Gaia', 'Rua do Hospital', 100300401,
|  'centrogaia@gmail.com').

% Resposta:
true.
```

Caso um dos campos estivesse incorretamente escrito, ou não existisse quer o próprio objeto quer a sua relação para com os outros, o sistema responderia `false`.

3 Descrição do Trabalho e Análise de Resultados

3.1 Construção da Base de Conhecimento

Os dados a utilizar para este projeto foram criados pelo grupo, onde nenhuma das informações das entidades (utentes, centros de saúde, entre outros) tem qualquer base em dados reais.

Devem seguir a seguinte estrutura:

- utente: #Idutente, Nº Segurança_Social, Nome, Data_Nasc, Email, Telefone, Morada, Profissão, [Doenças_Crónicas], #CentroSaúde $\rightsquigarrow$ {V,F}
- centro_saúde: #Idcentro, Nome, Morada, Telefone, Email $\rightsquigarrow$ {V,F}
- staff: #Idstaff, #Idcentro, Nome, email $\rightsquigarrow$ {V,F}
- vacinação_covid: #Staff, #utente, Data, Vacina, Toma $\rightsquigarrow$ {V,F}

Exemplos de entidades presentes na Base de Conhecimento:

```
utente(3,3,'Paulo',data(20,05,1985),'paulo@gmail.com',966123456,'Rua do
        fim','Pintor',[],3).

centro_saude(3, 'Centro Maia', 'Rua de Maio', 100300402,
        'centromaia@gmail.com').

staff(3,3,'Cecilia Rego','cila@gmail.com').

vacinacaoCovid(3,2, data(20,3,2021), 'Pfizer', 1).
```

Pode consultar a totalidade da Base de Conhecimento no anexo (5.1).

Cada entidade tem um `id`, que é um número único que a identifica dentro do seu grupo. Ou seja, por exemplo, tanto um utente como um membro do staff podem ter o id de 1, mas dois utentes não podem partilhar um id.

Para além destas entidades, foi também criada a entidade `data`, cujos argumentos representam o dia, mês, e ano.

```
data(20,3,2021)
```

Pode consultar o código de controlo do conteúdo de data no anexo (5.2).

Foi criadas também outras duas entidades, profissionalSaude, que representa uma ocupação profissional na área da saúde, e profissionalSeguranca, que representa uma ocupação profissional na área da segurança.

```
profissionalSaude('Médico').
profissionalSeguranca('Militar').
```

3.1.1 Definição das Fases de Vacinação

Para a definição das fases de vacinação, foram usados vários critérios: idade, ocupação, e doenças crónicas. Estes critérios foram baseados nos critérios utilizados no mundo real. É de salientar que a ordem pela qual estas fases estão definidas foi escolhida de modo a tirar proveito do processamento sequencial (top-down) caraterístico do prolog, isto é, as fases cuja datas de início são mais antigas aparecem primeiro do que as mais recentes.

Deste modo, ao verificar que um dado utente está incluído numa fase de vacinação, sabemos que essa é a primeira em que ele está incluído e que todas as fases a seguir vão também incluí-lo, pelo que seria descenessário fazer essa mesma verificação. Assim, ao definir as fases pela ordem explicada acima, tornamos este processo mais eficiente.

```
% fase_vacinacao (ID_Utente, Data de Inicio da Fase)

%Profissionais de saude e de segurança são vacinados a partir de dezembro
fase_vacinacao(Id,data(10,12,2020)):-
            profissaoUtente(Id,Prof),
            profissionalSaude(Prof).
fase_vacinacao(Id,data(10,12,2020)):-
            profissaoUtente(Id,Prof),
            profissionalSeguranca(Prof).

%Pessoas com mais de 70 anos, ou 50 anos com pelo menos uma doença de
%risco começam a partir de fevereiro
fase_vacinacao(Id,data(1,2,2021)):- dataUtente(Id,data(_,_,Ano)), Ano=<1950.
fase_vacinacao(Id,data(1,2,2021)):-
            doencasUtente(Id,Doencas),
            dataUtente(Id,data(_,_,Ano)),
```

```
            intersetam(Doencas,['Asma','Bronquite','Insuficiencia
            cardiaca','Doenca coronaria','Insuficiencia renal']),
            Ano=<1970.

%Bombeiros, Professores e pessoas com mais de 50 anos começam a ser
%vacinados em abril
fase_vacinacao(Id,data(1,4,2021)):-
            profissaoUtente(Id,Prof),
            pertence(Prof,['Bombeiro','Bombeira','Professor','Professora']).
fase_vacinacao(Id,data(1,4,2021)):- dataUtente(Id,data(_,_,Ano)),Ano=<1970.

%Crianças ate aos 10 anos e Pessoas acima dos 30 começam a partir de junho.
fase_vacinacao(Id,data(1,6,2021)):- dataUtente(Id,data(_,_,Ano)),Ano=<1990.
fase_vacinacao(Id,data(1,6,2021)):- dataUtente(Id,data(_,_,Ano)),Ano>=2010.
```

3.2 Funcionalidades Base

3.2.1 Identificar pessoas não vacinadas

Uma das funcionalidades básicas é obter a lista dos utentes que ainda não foram vacinados. Para isso, criámos os seguintes predicados que percorrem todos os utentes da nossa base de conhecimento e para cada um, verificam se já foi vacinado. No caso positivo, adiciona-os à lista resultante.

```
utente_naoVacinado(X):-utente(X,_,_,_,_,_,_,_,_,_,_),
            -vacinacaoCovid(_,_,_,_,_).

naoVacinados(R):-solucoes(X,utente_naoVacinado(X),R).
```

3.2.2 Identificar pessoas vacinadas

Outras funcionalidades básicas é saber quais os utentes que receberam pelo menos uma dose e aqueles que já receberam as duas duas. Para isso, criámos os seguintes predicados. O primeiro percorre todos utentes e caso encontre informação de vacinação da primeira dose adiciona-os à lista resultante. O segundo também percorre todos os utentes mas verifica a existência de informação de vacinação tanto da primeira dose como da segunda, adicionando os casos que verificam estas condições à lista resultante.

```
%Receberam a primeira dose
vacinados1Toma(R):-solucoes(X,vacinacaoCovid(_,X,_,_,1),R).

% Receberam todas as doses e estão vacinados
vacinados(R):-solucoes(X,vacinacaoCovid(_,X,_,_,2),R).
```

3.2.3 Identificar pessoas vacinadas indevidamente

Para implementarmos esta funcionalidade é importante definir bem o que significa ser indevidamente vacinado. Neste caso, decidímos que um utente foi indevidamente vacinado se:

- Não estiver incluído em nenhuma das fases de vacinação;
- Está incluído numa fase de vacinação mas a data de vacinação é mais antiga que o ínicio da mesma;

Deste modo, criámos os seguintes predicados que verificam estas duas condições para todos os utentes, guardando os indevidamente vacinados na lista resultante.

```
indevidamente_vacinados_aux([],[]).
indevidamente_vacinados_aux([(ID,Data)|T],[ID|Resultado]):-
            indevidamente_vacinado(ID,Data),
            indevidamente_vacinados_aux(T,Resultado).
indevidamente_vacinados_aux([_|T],R):- indevidamente_vacinados_aux(T,R).

indevidamente_vacinados(R):-
            solucoes((X,Data),
            vacinacaoCovid(_,X,Data,_,1),Lista),
            indevidamente_vacinados_aux(Lista,R).

indevidamente_vacinado_aux([],_).
indevidamente_vacinado_aux([H|_],D):- dataAnterior(D,H).
indevidamente_vacinado_aux(_,_):- fail.

%Uma vacinação indevida implica que a data de vacinação (D),
%aconteça antes das fases a que está aceite
indevidamente_vacinado(Id,DataVacina):-
            solucoes(Datafase,fase_vacinacao(Id,Datafase),Lista),
            indevidamente_vacinado_aux(Lista,DataVacina).
```

3.2.4 Identificar pessoas não vacinadas e que são candidatas a vacinação

Para as fases de vacinação serem eficientes e executadas corretamente, é preciso garantir que todos os candidatos dessas mesmas fases são vacinados. Deste modo,

criámos os seguintes predicados que, para cada utente, verifica se é candidato a uma fase de vacinação e, no caso positivo, verifica se ainda não foi vacinado.

```
candidatosVacinacao(R):- naoVacinados(N),candidatosVacinacaoAux(N,R).

candidatosVacinacaoAux([],[]).
candidatosVacinacaoAux([H|T],[H|RES]):-
            fase_vacinacao(H,_),
            candidatosVacinacaoAux(T,RES).
candidatosVacinacaoAux([_|T],N):- candidatosVacinacaoAux(T,N).
```

3.2.5 Identificar pessoas a quem falta a segunda toma da vacina

Para não haver desperdícios de vacinas, é preciso proceder à segunda vacinação no período correto de modo a maximizar a probabilidade de sucesso da mesma. Deste modo, é importante conseguirmos identificar os utentes que já tomaram a primeira vacina mas que ainda não tomaram a segunda. Assim, o predicado seguinte verifica estas condições para todos os utentes e adiciona à lista resultante os casos válidos.

```
tomaInc(IdU):- vacinacaoCovid(_,IdU,_,_,1), -vacinacaoCovid(_,IdU,_,_,2).

falta2toma(Res):- solucoes(IdU,tomaInc(IdU),Res).
```

3.3 Funcionalidades Adicionais

3.3.1 Identificar utentes por critérios de seleção

Os predicados abaixo permitem obter todas as informações dos utentes, segundo determinados parâmetros de pesquisa. Neste caso, é possível obter a informação dos utentes através do nome, id, data de nascimento ou profissão.

Este predicado permite, dado uma lista de ids de utentes, obter a informação dos utentes correspondentes.

```
% Extensão do predicado listaIDUtente:
% Lista de Ids, Resultado -> {V,F}
listaIDUtente([],[]).
listaIDUtente([HID|TID],[UTS|T]) :-
            solucoes(utente(HID,NSS,N,DN,E,T,M,P,DC,CS),
            utente(HID,NSS,N,DN,E,T,M,P,DC,CS),[UTS]),
            listaIDUtente(TID,T).
```

Os predicados seguintes, dado um id, nome, data de nascimento, ou profissão, fazem corresponder o utente referente. Nos casos em que o resultado da procura der mais do que um utente, por exemplo, quando se procura utente por profissão ou nome, são apresentados todos os casos que satisfazem a condição imposta, isto é, todos os utentes com o nome ou profissão inserido, por exemplo.

```
% Extensão do predicado utentePorId:
% Id de utente, Resultado -> {V,F}
utentePorId(IdU,R) :-
            solucoes((IdU,NSS,N,DN,E,T,M,P,DC,CS),
            utente(IdU,NSS,N,DN,E,T,M,P,DC,CS),R).

% Extensão do predicado utentePorNome:
% Nome de utente, Resultado -> {V,F}
utentePorNome(N,R) :-
            solucoes((IdU,NSS,N,DN,E,T,M,P,DC,CS),
            utente(IdU,NSS,N,DN,E,T,M,P,DC,CS),R).

% Extensão do predicado utentePorDataNasc:
% Data de nascimento de utente, Resultado -> {V,F}
```

```
utentePorDataNasc(D,M,A,R) :-
            solucoes((Id,NSS,N,data(D,M,A),E,T,P,DC,CS),
            utente(Id,NSS,N,data(D,M,A),E,T,_,P,DC,CS),R).

% Extensão do predicado utentePorProf:
% Profissão do utente, Resultado -> {V,F}
utentePorProf(P,R):-
            solucoes((IdU,N,P),utente(IdU,_,N,_,_,_,_,_,P,_,_),R).
```

3.3.2 Identificar os centros de saúde por critérios de seleção

Os seguintes predicados permitem obter o resultado de um conjunto de perguntas relacionadas com os centros de saúde como o nome de todos os centros de saúde, centro de saúde dado id, nome ou contacto.

O predicado seguinte permite obter todos os nomes dos centros de saúde, excluindo os repetidos, uma vez que um centro de saúde poderia, eventualmente, ter várias localizações com o mesmo nome.

```
% Extensão do predicado todosCentrosSaude:
% Resultado -> {V,F}
todosCentrosSaude(R) :-
            solucoes(I,centro_saude(_,I,_,_,_),L),
            apagaRepetido(L,R).
```

Os predicados seguintes, dado um id, nome ou contacto, obtém-se toda a informação desse centro de saúde. Nos casos em que o resultado da procura der mais do que um centro de saúde, por exemplo, quando se procura centro por nome, são apresentados todos os casos que satisfazem a condição imposta, isto é, todos os centros de saúde com o nome inserido.

```
% Extensão do predicado centroSaudePorId:
% Id do centro de saúde, Resultado -> {V,F}
centroSaudePorId(IdC,R):-
            solucoes((IdC,X,Y,Z,W),
            centro_saude(IdC,X,Y,Z,W),R).

% Extensão do predicado centroSaudePorNome:
```

```
% Nome do centro de saúde, Resultado -> {V,F}
centroSaudePorNome(N,R):-
            solucoes((X,N,Y,Z,W),
            centro_saude(X,N,Y,Z,W),R).

% Extensão do predicado centroSaudePorTelemovel:
% Contacto do centro de saúde, Resultado -> {V,F}
centroSaudePorTelemovel(C,R):-
            solucoes((X,Y,Z,C,W),centro_saude(X,Y,Z,C,W),R).
```

3.3.3 Identificar vacinações por centro de saúde

Este predicado permite obter informações acerca de todas as vacinações efetuadas em cada centro de saúde, dado o nome do centro de saúde.

```
% Extensão do predicado vacinacoesPorCentroSaude:
% Nome do centro de saúde, Resultado -> {V,F}
vacinacoesPorCentroSaude(NCS,R):-
            solucoes((NU,D,V,T),
            vacina_aux(_,_,_,D,V,T,NCS,NU),R).
```

O predicado `centrosSaudeComVac` permite verificar quais dos centros de saúde já efetuaram vacinações, isto é, aqueles em que já existe pelo menos um registo de uma vacinação.

```
% Extensão do predicado centrosSaudeComVac:
% Resultado -> {V,F}
centrosSaudeComVac(R):-
            solucoes((NCS),vacina_aux(_,_,_,_,_,_,_,NCS,_),Y),
            apagaRepetido(Y,R).
```

3.3.4 Identificar os utentes de um staff/centro de saúde

Uma informação importante de obter para casos de contágio, é a informação dos utentes vacinados por um staff. Ora, dado o nome ou id de um staff, os predicados `utentesStaff` ou `utentesStaffID` essa informação é possível obter.

```
% Extensão do predicado utentesStaff:
% Nome do staff, Resultado -> {V,F}
```

```
utentesStaff(NS,R):-
            solucoes(NU,utentes_aux(_,_,_,_,NU,NS),Y),
            apagaRepetido(Y,R).

% Extensão do predicado utentesStaffID:
% Id do staff, Resultado -> {V,F}
utentesStaffID(IdS,R):-
            solucoes(NU,utentes_aux(_,IdS,_,_,NU,_),Y),
            apagaRepetido(Y,R).
```

Dado o nome ou id de um centro de saúde, é possível obter quais os utentes que já o frequentaram para obter vacinação. É precisamente isso que os dois próximos predicados se encarregam de traduzir. É utilizado o predicado `apagaRepetido` uma vez que um utente pode visitar um centro de saúde mais do que uma vez para tomar a segunda dose da vacina.

```
% Extensão do predicado utentesCentroSaude:
% Nome do centro de saúde, Resultado -> {V,F}
utentesCentroSaude(NCS,R):-
            solucoes(NU,utentes_aux(_,_,_,NCS,NU,_),Y),
            apagaRepetido(Y,R).

% Extensão do predicado utentesCentroSaudeID:
% Id do centro de saúde, Resultado -> {V,F}
utentesCentroSaudeID(IdC,R):-
            solucoes(NU,utentes_aux(IdC,_,_,_,NU,_),Y),
            apagaRepetido(Y,R).
```

3.3.5 Quantidade de vacinas por centro, por staff, e por centro de saúde

Este predicado indica o número de vacinas de um determinado tipo (Pfizer, Astrazeneca, Sputnik) que já foram administradas.

```
% Extensão do predicado quantidadeVac:
% Nome da vacina, Resultado -> {V,F}
quantidadeVac(V,R):- solucoes(V,vacinacaoCovid(_,_,_,V,_),L),
                                        comprimento(L,R).
```

Este predicado permite obter o número de vacinas administradas por um staff dado o seu nome.

```
% Extensão do predicado quantidadeStaff:
% Nome do staff, Resultado -> {V,F}
quantidadeStaff(NS,R):- solucoes(NS,vac_aux(_,_,NS,_,_),L),
                                                    comprimento(L,R).
```

Este predicado permite obter o número de vacinas administradas por um staff dado o seu id.

```
% Extensão do predicado quantidadeStaffID:
% Id do staff, Resultado -> {V,F}
quantidadeStaffID(IdS,R):- solucoes(IdS,vac_aux(_,IdS,_,_,_),L),
                                                    comprimento(L,R).
```

Este predicado permite obter o número de vacinas administradas num centro de saúde dado o seu nome.

```
% Extensão do predicado quantidadeCentro:
% Nome do centro de saúde, Resultado -> {V,F}
quantidadeCentro(NC,R):- solucoes(NC,vac_aux(_,_,_,_,NC),L),
                                                    comprimento(L,R).
```

Este predicado permite obter o número de vacinas administradas num centro de saúde dado o seu id.

```
% Extensão do predicado quantidadeCentroID:
% Id do centro de saúde, Resultado -> {V,F}
quantidadeCentroID(IdC,R):- solucoes(IdC,vac_aux(_,_,_,IdC,_),L),
                                                    comprimento(L,R).
```

3.3.6 Contactos próximos dos utentes

Achámos interessante guardar na nossa base de conhecimento os IDs dos utentes que estão em contacto próximo/frequente com cada utente.

Registar contactos frequentes dos utentes

```
contatoFrequente(1,2).
contatoFrequente(1,11).
```

```
contatoFrequente(2,3).
contatoFrequente(1,5).
```

Obter a lista dos contactos próximos de um dado utente

```
getContatosFrequentes(X,R):-solucoes(Y,contatoFrequente(X,Y),R).
```

Verificar se os contactos próximos de um utente estão vacinados

Verificar se todos os contactos próximos de um dado utente já completaram o processo de vacinaçao, isto é, já procederam à primeira e segunda toma de uma vacina. Neste caso, podemos considerar o utente em questão como "protegido"uma vez que o seu círculo social encontra-se imune.

```
estaProtegidoAux([]).
estaProtegidoAux([H|T]):-vacinacaoCovid(_,H,_,_,'2'),estaProtegidoAux(T).

estaProtegido(Id):-getContatosFrequentes(Id,R), estaProtegidoAux(R).
```

3.4 Predicados Auxiliares

3.4.1 Teste

Este predicado itera os elementos de uma lista recebida como argumento. O critério de paragem é atingido quando a lista fica vazia, isto é, quando não possui elementos.

```
%Extensao do meta predicado teste: lista -->{V,F}
teste([]).
teste([R|LR]) :-
                R, teste(LR).
```

3.4.2 Comprimento

Este predicado recebe como argumento uma lista e permite obter o seu número de elementos.O critério de paragem é atingido quando a lista fica vazia, isto é, quando a lista fica vazia o seu comprimento é 0. De forma recursiva, este predicado vai retirando a cabeça da lista à medida que incrementa por 1 e, de seguida, repete o procedimento para a cauda, até que a lista que sem elementos para analisar.

```
%Extensao do predicado comprimento: lista , Comprimento --> {V,F}
comprimento([],0).
comprimento([H|T],N):-comprimento(T,R), N is R+1.
```

3.4.3 Contém

Este predicado recebe como argumento duas listas e verifica se a segunda está contida na primeira. Para cada elemento da segunda lista, é verificado se este pertence à primeira e assim sucessivamente para o resto da lista, de forma recursiva.

```
%Extensao do predicado contem: Lista, Sublista -->{V,F}
contem(_,[]).
contem([],_):- fail.
contem(L,[H|T]) :- pertence(H,L), contem(L,T).
```

3.4.4 Intersetam

Este predicado permite verificar se duas listas se intersetam, ou seja, se duas listas tem pelo menos um elemento em comum.

```
%Extensao do predicado contem: Lista, Lista -->{V,F}
intersetam([],_) :- fail.
intersetam(_,[]) :- fail.
intersetam(L,[H|T]):- pertence(H,L),!.
intersetam(L,[_|T]) :- intersetam(L,T).
```

3.4.5 Interseção

Este predicado permite computar o conjuto de interseção de duas listas.

```
intersecao([], _, []).
intersecao([H|T1], L2, [H|T3]) :- pertence(H, L2), !, intersecao(T1, L2, T3).
intersecao([_|T1], L2, L3) :- intersecao(T1, L2, L3).
```

3.4.6 Pertence

O predicado `pertence` tem o objetivo de verificar se determinado elemento está contido numa lista.

Este predicado pode ser separado em duas partes: a primeira corresponde à pesquisa do elemento quando ele se encontra na cabeça da lista e a segunda corresponde à pesquisa quando o elemento se encontra na cauda e, por método recursivo, obtém-se uma resposta.

Do mesmo modo, `-pertence` faz o contrário, isto é, verifica se um elemento não pertence a uma lista.

```
%Extensao do predicado pertence: Elemento, lista --> {V,F}
pertence(X,[X|L]).
pertence(X,[Y|L]):- pertence(X,L).

-pertence(X,R) :- nao(pertence(X,R)).
```

3.4.7 ApagaElemento

O predicado `apagaElemento` tem como intuito eliminar todas as ocorrências de um elemento X da lista.

```
%Extensao do predicado apagaElemento: Elemento, lista, resultado --> {V,F}W
apagaElemento(X,[],[]).
```

```
apagaElemento(X,[X|Y],L):- apagaElemento(X,Y,L).
apagaElemento(X,[H|T],[H|R]):- X \= H, apagaElemento(X,T,R).
```

3.4.8 ApagaRepetido

Para o predicado `ApagaRepetido`, também se utiliza o predicado anteriormente explicado. Este serve para eliminar resultados que estejam repetidos de forma a que apareçam uma única vez.

O critério de paragem acontece quando a lista está vazia, em que o retorno é uma lista vazia.

Na situação em que a lista não é vazia, o predicado elimina os elementos repetidos da lista que recebe como argumento através de um processo recursivo.

```
%Extensao do predicado apagaRepetido: Lista, resultado --> {V,F}
apagaRepetido([],[]).
apagaRepetido([H|T],[H|Y]):-apagaElemento(H,T,NL), apagaRepetido(NL,Y).
```

3.4.9 SI

Este predicado constitui um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes ao sistema.

```
% Extensao do meta-predicado si: Questao,Resposta -> {V,F}
si( Questao,verdadeiro ) :-
    Questao.
si( Questao,falso ) :-
    -Questao.
si( Questao,desconhecido ) :-
    nao( Questao ),
    nao( -Questao ).
```

3.4.10 Não

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

3.4.11 Soluções

O predicado solucoes tem como intuito listar todas as possíveis soluções como resposta a uma pergunta.

Este predicado é constituído por 3 argumentos, onde F corresponde ao argumento que se pretende obter, Q corresponde à questão a ser feita, ie, o teorema ao qual se tenciona provar dada uma pergunta e, por fim, S que é a lista que armazena numa lista o conjuntos de soluções.

```
% Extensão do predicado solucoes: Facto, Questão, Solução -> {V,F}
solucoes(F,Q,S) :- findall(F,Q,S).
```

Os seguintes predicados permitem relacionar informação dos factos da base de conhecimento através dos id's existentes: id de utente, id de staff, id de centro de saúde.

O facto vacinacaoCovid não tem informação do centro de saúde ou informação sobre o utente ou o staff que administrou a vacina, no entanto, tem os ids de utente e staff. A partir do id de staff é possível cruzar esta informação com a informação do facto `Staff` correspondente para obter o id do centro de saúde e, por sua vez, é possível relacionar este id com o facto `centro_saude` para obter a informação do centro de saúde onde foi administrada a vacina e assim sucessivamente para os outros casos. Foi este o raciocínio usado para criar estes predicados.

Estes predicados ao contrário dos anteriores, não adicionam informação nova. Apesar disso, a sua função é de grande importância, pois permitem aceder a informação a partir de apenas um predicado em vez de ser necessário utilizar entre 3 e 4 predicados diferentes no interior de um predicado `solucoes`.

3.4.12 Utentes_aux

O predicado `utentes_aux` relaciona os predicados `vacinacaoCovid` , `staff`, `centro_saude` e `utente` através do id de staff (IdS), do id de utente (IdU), do id do centro (IdC) e do id do utente (IdU).

Este concede acesso às seguintes informações: id (IdC) e nome (NCS) do centro de saúde; id (IdS) e nome (NS) do staff; id (IdU) e nome do utente (NU).

```
utentes_aux(IdC,IdS,IdU,NCS,NU,NS):-vacinacaoCovid(IdS,IdU,_,_,_),
                                    staff(IdS,IdC,NS,_),
                                    centro_saude(IdC,NCS,_,_,_),
                                    utente(IdU,_,NU,_,_,_,_,_,_,_).
```

3.4.13 Vac_aux

O predicado `vac_aux` interliga os predicados `vacinacaoCovid` , `staff` e `centro_saude` através do id de staff (IdS) e do id do centro (IdC).

Desta forma, facilmente se acede às seguintes informações: id (IdU) e nome (NU) do utente; id (IdP) e nome (NP) do prestador; id (IdS), id do centro em que trabalha (IdC) e nome (NS) do staff; id (IdC) e nome (NCS) do centro de saúde.

```
vac_aux(V,IdS,NS,IdC,NCS):-vacinacaoCovid(IdS,_,_,V,_),
                           staff(IdS,IdC,NS,_),
                           centro_saude(IdC,NCS,_,_,_).
```

3.4.14 Vacina_aux

O predicado `vacina_aux` associa os predicados `vacinacaoCovid` , `staff`, `centro_saude` e `utente` através do id de staff (IdS), do id de utente (IdU), do id do centro (IdC) e do id do utente (IdU).

Com esta informação consulta-se as informações: id do staff administrador (IdS), id do utente(IdU), data (D), designação da vacina (V) e fase de toma (T) da vacinacaoCovid; id (IdS), id do centro em que trabalha (IdC) e nome (NS) do staff; id (IdC) e nome (NCS) do centro de saúde; id (IdU) e nome do utente (NU).

```
vacina_aux(IdC,IdS,IdU,D,V,T,NCS,NU):- vacinacaoCovid(IdS,IdU,D,V,T),
                                       staff(IdS,IdC,_,_),
                                       centro_saude(IdC,NCS,_,_,_),
                                       utente(IdU,_,NU,_,_,_,_,_,_,_).
```

3.5 Invariantes

3.5.1 Evolução da Base de conhecimento

Para inserirmos nova informação na nossa base de conhecimento é importante haver controlo de consistência, pelo que criámos o predicado evolucao. Este predicado procura todos os invariantes que se aplicam ao tipo de informação que estamos a tentar inserir e verifica as suas condições. Se todos os invariantes forem verificados a informação é inserida na base de conhecimento com sucesso.

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

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

Invariantes estruturais

De modo a garantir que não há informação repetida/duplicada na nossa base de conhecimento, criámos os seguintes invariantes que garantem que uma informação específica só pode existir uma única vez.

```
%Invariante estrutural: Nao permitir a insercao de conhecimento repetido
+utente(Id, NSS,Nome,D,E,T,M,P,Doencas,CentroS) ::
        (solucoes( (Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS),
        (utente(Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS)), S),
        comprimento(S,N),
        N==1).

%Invariante estrutural:nao permitir a insercao de conhecimento repetido
+centro_saude(Id,Nome,Morada,Tel,Email) ::
        (solucoes( (Id,Nome,Morada,Tel,Email),
        (centro_saude(Id,Nome,Morada,Tel,Email)), S),
        comprimento(S,N),
        N==1).

%Invariante estrutural:nao permitir a insercao de conhecimento repetido
```

```
+staff(Id, Centro,Nome,Email) ::
        (solucoes( (Id,Centro,Nome,Email), (staff(Id,Centro,Nome,Email)), S),
        comprimento(S,N),
        N==1).
```

Invariantes referenciais

Quando queremos inserir informação de um novo staff na nossa base de conhecimento, temos de garantir que o centro ao qual ele pertence já está na nossa base de conhecimento. O invariante seguinte verifica se o centro de saúde inserido já existe.

```
%Invariante : O centro de saúde da staff existe.
+staff(Id, Centro,Nome,Email) ::
        (solucoes( (Centro,Nome,Morada,Tel,Email),
        (centro_saude(Centro,Nome,Morada,Tel,Email)), S),
        comprimento(S,N),
        N>0).
```

Não é permitido registar a vacinação de uma dose mais do que uma vez, ou seja, um utente não pode ter tomado a primeira nem segunda doses de vacina mais que uma vez. O seguinte invariante garante esta condição sempre que uma nova vacinacaoCovid é inserida na nossa base de conhecimento.

```
%Invariante : Não pode ter tomada a mesma fase da vacina mais que uma vez
+vacinacaoCovid(S, U, D, V, T) ::
        (solucoes((S,U,D,V,T), vacinacaoCovid(S,U,D,V,T) , L),
        comprimento(L,N), N==1).
```

Também não é permitido um utente ser vacinado com a segunda dose sem ter sido vacinado com a primeira anteriormente. Deste modo, o seguinte invariante garante que, sempre que uma vacinacaoCovid da segunda dose é inserida na nossa base de conhecimento, já existe informação da vacinação da primeira dose.

```
%Invariante : So pode ter tomado a 2 dose se tomou a primeira
+vacinacaoCovid(S,U,D,V,2)::
        (solucoes((S,U,D,V), vacinacaoCovid(S,U,D,V,'1'), Primeiros),
        comprimento(Primeiros,N),
        N==1).
```

Um utente não pode ser vacinado com diferentes modelos de vacinas em dife-

rentes doses. Ora, o invariante a seguir só é aplicado quando é inserida informação de uma vacinação da segunda dose. Nestes casos, verifica se a vacinação da primeira dose desse mesmo utente é do mesmo modelo que da segunda.

```
%Invariante : A vacina da segunda toma tem de ser o mesmo modelo
+vacinacaoCovid(S, U, D, V, 2) ::
        (solucoes((S,U,D,V,'1'), vacinacaoCovid(S,U,D,V,T) , L),
        nth0(0,L,(_,_,_,C,_)),C==V).
```

De momento todas as vacinas são administradas em 2 doses. Assim, só é permitida a inserção de informação de vacinações cujo números de dose são 1 ou 2. O invariante seguinte verifica esta condição.

```
%Invariante : A toma da vacina so pode ser 1 ou 2
+vacinacaoCovid(_,_,_,_,T)::(T>0,T<3).
```

O trabalhador que administrou a vacina tem de existir na nossa base de conhecimento.

```
%Invariante : O trabalhador que administrou a vacina tem de estar registado
+vacinacaoCovid(Staff,_,_,_,_)::
        (solucoes( (Staff,Centro,Nome,Email),
        (staff(Staff,Centro,Nome,Email)), L),
        comprimento(L,N),
        N>0).
```

O utente que recebeu a vacina tem de existir na nossa base de conhecimento

```
%Invariante : O Utente que rebeu a vacina tem de estar registado
+vacinacaoCovid(_,Ut,_,_,T)::
            (solucoes( (Ut,NSS,Nome,D,E,T,M,P,Doencas,CentroS),
            (utente(Ut,NSS,Nome,D,E,T,M,P,Doencas,CentroS)), _),
            comprimento(_,N),
            N>0).
```

3.5.2 Involução da Base de conhecimento

Ao remover informação da nossa base de conhecimento temos de nos certificar que não há dependências. Por exemplo, não podemos remover um utente antes de remover as vacinações que o envolvem, senão o ID que o referencia na vacinação deixará

de ser válido. Deste modo definimos os predicados seguintes para verificarmos todos os invariantes sempre que removemos informação.

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

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

Invariantes estruturais

O invariante seguinte indica que um utente pode ser removido desde que esteja na base de conhecimento.

```
%permite remover se existir na base de conhecimento:
-utente(Id, NSS,Nome,D,E,T,M,P,Doencas,CentroS) ::
                        (solucoes( (Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS),
                        (utente(Id,NSS,Nome,D,E,T,M,P,Doencas,CentroS)), S),
                        comprimento(S,N),
                            N==0).
```

O invariante seguinte indica que um centro de saúde pode ser removido desde que esteja na base de conhecimento.

```
%permite remover se existir na base de conhecimento:
-centro_saude(Id,Nome,Morada,Tel,Email) ::
                (solucoes( (Id,Nome,Morada,Tel,Email),
                (centro_saude(Id,Nome,Morada,Tel,Email)), S),
                comprimento(S,N),
                N==0).
```

O invariante seguinte indica que um elemento do staff pode ser removido desde que esteja na base de conhecimento.

```
%permite remover se existir na base de conhecimento:
-staff(Id, Centro,Nome,Email) ::
        (solucoes( (Id,Centro,Nome,Email), (staff(Id,Centro,Nome,Email)), S),
```

```
        comprimento(S,N),
        N==0).
```

O invariante seguinte indica que uma vacinação pode ser removida desde que esteja na base de conhecimento.

```
%permite remover se existir na base de conhecimento:
-vacinacaoCovid(S, U, D, V, T) ::
        (solucoes((S,U,D,V,T), vacinacaoCovid(S,U,D,V,T) , L),
        comprimento(L,N),
        N==0).
```

Invariantes referenciais

O invariante seguinte indica que um utente que esteja presente na base de conhecimento pode ser removido, se e só se, não tiver sido vacinado.

```
%Não permite remover utentes com registos de vacinação
-utente(Id, NSS,Nome,D,E,T,M,P,Doencas,CentroS) ::
        (nao( vacinacaoCovid(_,Id,_,_,_))).
```

O invariante seguinte indica que o registo de uma vacina que seja referente à primeira toma por parte de um utente não pode ser removida se o registo da segunda toma por parte do mesmo paciente ainda esteja presente na base de conhecimento.

```
%Não permite remover a primeira toma vacina se a segunda esteja na
%base de conhecimento
-vacinacaoCovid(S, U, D, V, 1) :: (nao( vacinacaoCovid(_,Id,_,_,2))).
```

O invariante seguinte indica que um centro de saúde que esteja presente na base de conhecimento pode ser removido, se e só se, não houver nenhum elemento do staff presente na base de conhecimento que esteja associado ao centro de saúde em questão.

```
%Não permite remover um centro de saúde se existir staff associada
-centro_saude(Id,_,_,_,_) ::
        (solucoes((Staff,Id,Nome,Email), (staff(Staff,Id,Nome,Email)), S),
        comprimento(S,N),
        N==0).
```

4 Conclusões e Sugestões

Com este trabalho prático, acreditamos ter criado um sistema de representação de conhecimento e raciocínio em que a informação nele presente é coerente. Acreditamos também que que as funcionalidades que foram por nós adicionadas vão não só ao encontro do tema do projeto, como também seriam úteis num contexto de aplicação no mundo real.

A nossa maior dificuldade foi trabalhar com o `prolog`, não só porque o nosso conhecimento e experiência com esta linguagem ainda não são muito extensos, mas também porque é uma linguagem de programação com uma sintaxe e com um funcionamento bastante diferentes dos das linguagens com que estamos habituados a trabalhar.

Este trabalho permitiu, no entanto, aumentar o nosso nível de experiência para com o `prolog`, possibilitando a consolidação dos conhecimentos lecionados durante as aulas da UC.

Para fases futuras do trabalho, sugerimos tornar o controlo de informação do sistema ainda mais realista, assim como implementar funcionalidades que permitam ao administrador do sistema ter um maior controlo sobre a informação existente no sistema.

5 Anexos

5.1 Base de Conhecimento

```
%-----------------------------------------------------------------------
% BASE DE CONHECIMENTO
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% utente(Nº Utente, Nº Segurança Social, Nome, Data de Nascimento, Email,
%Telefone, Morada, Profissão, [Doenças_Crónicas], #centroSaúde)
%-----------------------------------------------------------------------
utente(1,1 , 'Jacarias',data(13,04,2000) , 'jacarias@gmail.com', 780100300,
            'Rua do Moinho','Empreiteiro', ['Asma'], 1).
utente(2,2 , 'Maria',data(1,12,1992) , 'maria@gmail.com', 420100300,
            'Rua das Duas Igrejas','Medica', ['Bronquite'], 2).
utente(3,3,'Paulo',data(20,05,1985),'paulo@gmail.com',966123456,
            'Rua do fim','Pintor',[],3).
utente(4,4,'Clara',data(3,01,2004),'clara@gmail.com',123456789,
            'Rua do inicio','Estudante',[],4).
utente(5,5,'Agostinho',data(16,07,1982),'agostinho@gmail.com',111111111,
            'Rua da Uva','Desempregado',['Asma','Bronquite'],5).
utente(6,6,'Fred',data(19,09,1991),'fred@gmail.com',222222222,
            'Rua dos Frangos','Engenheiro',[],6).
utente(7,7,'Carlos',data(23,10,1943),'carlos@gmail.com',333333333,
            'Travessa do Emigrante','Reformado',['Insuficiencia Renal'],4).
utente(8,8,'Jose',data(6,6,1930),'jose@gmail.com',444444444,
            'Travessa do Emigrante','Reformado',['Alzheimer'],4).
utente(9,9,'Josefa',data(10,11,1949),'josefa@gmail.com',555555555,
            'Lugar do Travesso','Reformado',[],7).
utente(10,10,'Alexandre',data(5,5,1995),'alex@gmail.com',666666666,
            'Rua dos Encalhados','Engenheiro',[],8).
utente(11,11,'Diana',data(13,03,2002),'diana@gmail.com',777777777,
            'Lugar dos cabaneiros','Estudante',[],8).
utente(12,12,'Ines',data(13,11,1999),'ines@gmail.com',888888888,
            'Rua do Cansado','Estudante',[],7).
```

```
utente(13,13,'Afonso',data(6,6,2006),'afonso@gmail.com',999999999,
            'Lugar da Lapa','Estudante',[],4).
utente(14,14,'Irineu',data(9,9,1970),'irineu@gmail.com',101010101,
            'Rua do Saber','Bibliotecario',[],6).
utente(15,15,'Marcelo',data(2,12,1998),'marcelo@gmail.com',929292929,
            'Praça da Sabedoria','Estudante',[],1).
utente(16,16,'Leonardo',data(5,12,2000),'leonardo@gmail.com',929292928,
            'Praça da Sabedoria','Estudante',[],1).

dataUtente(Id,D):-utente(Id,_,_,D,_,_,_,_,_,_).
doencasUtente(Id,D):-utente(Id,_,_,_,_,_,_,_,D,_).
profissaoUtente(Id,P):-utente(Id,_,_,_,_,_,_,P,_,_).
% ------------------------------------------------------------
% centro_saúde(Idcentro, Nome, Morada, Telefone, Email)
% ------------------------------------------------------------
centro_saude(1, 'Centro Freamunde', 'Rua das Couves', 100300400,
            'centrofreamunde@gmail.com').
centro_saude(2, 'Centro Gaia', 'Rua do Hospital', 100300401,
            'centrogaia@gmail.com').
centro_saude(3, 'Centro Maia', 'Rua de Maio', 100300402,
            'centromaia@gmail.com').
centro_saude(4, 'Hospital de Braga', 'Rua da Universidade', 100300403,
            'hospitalbraga@gmail.com').
centro_saude(5, 'Centro Vizela', 'Rua das Vuvuzelas', 100300404,
            'centrovizela@gmail.com').
centro_saude(6, 'Hospital de Guimaraes', 'Rua dos Conquistadores',
            100300405, 'hospitalguimaraes@gmail.com').
centro_saude(7, 'Hospital do Porto', 'Rua das Tripas', 100300406,
            'hospitalporto@gmail.com').
centro_saude(8, 'Centro Barcelos', 'Rua dos Galos', 100300407,
            'centrobarcelos@gmail.com').
centro_saude(9, 'Posto de Saúde de Dume', 'Rua da Dor', 987654321,
            'saudedume@gmail.com').

% ------------------------------------------------------------
% staff (Idstaff, #Idcentro, Nome, email)
```

```
% ------------------------------------------------------------
staff(1,1,'Manuela Figueiredo','manuelaf@gmail.com').
staff(2,2,'Alberto Dias','berto@gmail.com').
staff(3,3,'Cecilia Rego','cila@gmail.com').
staff(4,4,'Ernesto Honesto','nestinho@gmail.com').
staff(5,5,'Margarida Flor','magui@gmail.com').
staff(6,6,'Simplicio Sousa','simples@gmail.com').
staff(7,7,'Carla Carlos','carla@gmail.com').
staff(8,1,'Manuel Pessegueiro','manelp@gmail.com').


% ------------------------------------------------------------
% vacinação_Covid (Staff, Utente, Data, Vacina, Toma)
% ------------------------------------------------------------
vacinacaoCovid(1,1, data(20,3,2021), 'Astrazeneca', 1).
vacinacaoCovid(2,1, data(20,4,2021), 'Astrazeneca', 2).
vacinacaoCovid(3,2, data(20,3,2021), 'Pfizer', 1).
vacinacaoCovid(4,2, data(20,4,2021), 'Pfizer', 2).
vacinacaoCovid(5,4, data(21,3,2021), 'Astrazeneca', 1).
vacinacaoCovid(6,3, data(18,3,2021), 'Astrazeneca', 1).
vacinacaoCovid(7,3, data(18,5,2021), 'Astrazeneca', 2).
vacinacaoCovid(7,5, data(21,3,2021), 'Sputnik', 1).
vacinacaoCovid(2,5, data(21,4,2021), 'Sputnik', 2).
vacinacaoCovid(2,6, data(21,3,2021), 'Pfizer', 1).
vacinacaoCovid(4,6, data(21,4,2021), 'Pfizer', 2).
vacinacaoCovid(8,15, data(21,4,2021), 'Pfizer', 1).
vacinacaoCovid(8,15, data(30,4,2021), 'Pfizer', 2).
vacinacaoCovid(8,16, data(21,4,2021), 'Pfizer', 2).

-vacinacaoCovid(Staff, Utente, Data, Vacina, Toma) :-
            nao(vacinacaoCovid(Staff, Utente, Data, Vacina, Toma)).

% ------------------------------------------------------------------
%Factos para ajudar a definir fases de vacinação.
% ------------------------------------------------------------------
```

```
% profissionalSaúde (Profissão)
profissionalSaude('Médico').
profissionalSaude('Médico').
profissionalSaude('Enfermeiro').
profissionalSaude('Enfermeira').
profissionalSaude('Assistente Saúde').
% profissionalSegurança (Profissão)
profissionalSeguranca('Policia').
profissionalSeguranca('Militar').


contatoFrequente(1,2).
contatoFrequente(1,11).
contatoFrequente(2,3).
contatoFrequente(1,5).
```

5.2 Entidade Data

```
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(D, 2, A) :-
    A >= 0,
        A mod 4 =:= 0,
        D >= 1,
        D =< 29.
data(data(D, M, A)) :- data(D, M, A).

% Extensão do predicado dataAnterior:
% DataAnterior , DataSeguinte --> {V,F}

dataAnterior(data(_,_,A),data(_,_,Ano)):- A < Ano.
dataAnterior(data(_,M,A),data(_,Mes,Ano)):- A == Ano, M < Mes.
dataAnterior(data(D,M,A),data(Dia,Mes,Ano)):- A == Ano, M == Mes, D < Dia.
```