Universidade do Minho
Escola de Engenharia

UNIVERSIDADE DO MINHO

TRABALHO PRÁTICO I

SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

(2º SEMESTRE 20/21)

A89533 - Ana Luísa Carneiro

A89612 - Ana Rita Peixoto

A89506 - Luís Miguel Pinto

A89574 - Pedro Almeida Fernandes

Braga

Abril - 2021

RESUMO

O seguinte relatório pretende descrever como foi desenvolvido o sistema de representação de conhecimento e raciocínio referente ao tópico tão em voga nos dias de hoje que é a área da vacinação global da população portuguesa no contexto pandémico em que nos inserimos.

Para a realização deste projeto fomos incentivados a utilizar a linguagem de programação em lógica PROLOG. Esta ferramenta foi fundamental para a concretização da base de conhecimentos assim como do sistema de inferência.

A primeira fase do trabalho visa desenvolver um conjunto de funcionalidades que permitem ao utilizador ter acesso a informação sobre as etapas de vacinação, utentes e outro conhecimento referente à área de vacinação contra o COVID-19 em Portugal. Para isso implementou-se uma base de conhecimento completa e descritiva que será utilizada como forma de representar a realidade da vacinação em Portugal. De seguida implementou-se um conjunto de funcionalidades básicas e extras que permitiram obter informação detalhada e organizada, relacionada com os conhecimentos implementados. Por fim, implementou-se um conjunto de invariantes que permitem a evolução e involução da base de conhecimento, isto é, um conjunto de regras que impeçam ou permitam a inserção ou remoção de conhecimento.

Por último realçamos que, tal como nos foi proposto, foi adicionada uma análise dos resultados obtidos e alguns extras de forma a enriquecer o trabalho, tornando-o desta forma mais representativo e próximo do sistema real de vacinação português.

ÍNDICE

ÍNDICE FIGURAS

INTRODUÇÃO

O trabalho prático I tem como objetivo construir um sistema de representação de conhecimento e raciocínio capaz de caracterizar um universo na área da vacinação global da população portuguesa no contexto da pandemia do COVID-19.

No âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas é nos proposto a implementação de diversas funcionalidades que visam demostrar e caracterizar o plano de vacinação portuguesa contra o covid através do uso da linguagem de programação em lógica, PROLOG.

Além das necessidades básicas propostas, decidimos implementar um conjunto de funcionalidades e características extras como forma de tornar o sistema criado mais completo e personalizado.

No presente relatório apresentamos explicações detalhadas de todas as etapas e funcionalidades que sustentaram esta fase do projeto, acompanhadas por código e imagens de forma a ilustrar o raciocínio usado e manter uma documentação exemplificativa do projeto.

PRELIMINARES

Antes de iniciar uma leitura mais aprofundada sobre o presente relatório, qualquer leitor deverá ter como base alguns conceitos genéricos, mas fundamentais para a sua inteira compreensão. Assim sendo, mesmo o leitor, mas ingénuo na temática em causa deverá ser capaz de compreender os conceitos discutidos após a leitura deste capítulo.

Comecemos pelo princípio, numa vertente mais técnica o trabalho realizado está assente em 5 princípios fundamentais:

- **Pressuposto do Mundo Fechado** – toda a informação que não existe mencionada na base de dados é considerada falsa;

- **Pressuposto dos Nomes Únicos** – duas constantes diferentes (que definam valores atómicos ou objetos) designam, necessariamente, duas entidades diferentes do universo de discurso;

- **Pressuposto do Domínio Fechado** – não existem mais objetos no universo de discurso para além daqueles designados por constantes na base de dados.

- **Valores lógicos** – Apesar de, numa teoria lógica clássica, o conhecimento ser equacionado em termos do que se prova ser verdadeiro, o que se prova ser falso e o que representa informação sobre a qual não se pode ser conclusivo, num programa em lógica, as respostas às questões colocadas são, apenas, de dois tipos: verdadeiro ou falso. Isto deve-se ao facto de um programa em lógica apresentar várias limitações em termos da representação de conhecimento (não permite representar explicitamente informação negativa ou disjuntiva), para além do facto de que, em termos de uma semântica operacional se aplicar, automaticamente, o PMF a todos os predicados.

- **Negação forte/fraca –** dois tipos de negação: a negação por falha, característica dos programas em lógica tradicionais, representada pelo termo não, e a negação forte ou clássica, como forma de identificar informação negativa, ou falsa, representada pela conectiva '¬'.

Numa faceta agora voltada para a temática da vacinação faseada para a COVID-19 achamos por bem esclarecer alguns dos seus conceitos, desde o que são as fases de vacinação, os critérios de ilegibilidade de cada fase, bem como as vacinas adotadas pelo SNS. As fases de vacinação servem para agilizar o seu processo, combatendo assim a excessiva procura face ao inventário disponível na sociedade. Para cada fase são definidos critérios que tem em conta a suscetibilidade do sistema imunitário de cada pessoa baseando-se em fatores como idade ou doenças crónicas ou por outro lado o grau de transmissão a que pessoa está exposta. As principais vacinas adotadas pela DGS, AstraZeneca e Pfizer, necessitam de duas tomas por parte do utente para que a vacina tenha a máxima eficácia.

Para finalizar, mencionar apenas que caso procure um aprofundamento dos tópicos supramencionados ou qualquer outro esclarecimento sobre sistemas de representação de conhecimento e raciocínio, deverá ler o capítulo da Bibliografia que contempla todos os materiais utilizados por nós para o desenvolvimento do projeto.

DESCRIÇÃO DO TRABALHO

De modo a melhor organizar e estruturar o trabalho, decidimos dividir o projeto criando diferentes ficheiros PROLOG que agregam componentes semelhantes do sistema:

- **BaseConhecimento.pl:** onde estão presentes todos os predicados e conhecimento que sustenta o sistema.
- **Funcionalidades.pl:** neste ficheiro estão implementadas todas as funcionalidades propriamente ditas, sendo elas obrigatórias ou extra, traduzidas na forma de regras.
- **RegrasAuxiliares.pl:** este documento agrega todas as regras que serviram de auxílio à implementação das regras principais.
- **Invariantes.pl:** contém todos os invariantes necessários para o sistema.
- **EvolucaoInvolucao.pl:** possui os mecanismos de evolução e involução do conhecimento.

Nesta secção será feita uma abordagem em detalhe de cada um destes ficheiros, de modo a analisar e expor os diferentes elementos que os constituem.

BASE DE CONHECIMENTO

Tal como proposto no enunciado, foi desenvolvido um sistema de representação de conhecimento e raciocinio com capacidade para caraterizar a vacinação da população portuguesa contra o coronavírus.

Para tal, foi necessário adicionar conhecimento referente aos utentes, centros de saúde, *staff,* à vacinação propriamente dita, às fases de vacinação e também registos de cancelamento de vacinas. Nas diferentes subsecções será abordado cada um destes tópicos em detalhe.

UTENTE

Os utentes deste sistema são caraterizados através de diversos elementos: ID de utente, Nº segurança social, nome, data de nascimento, *email*, telefone, morada, profissão, doenças crónicas e o centro de saúde onde decorreu a vacinação.

Os pressupostos anteriormente referidos traduzem-se em linguagem PROLOG da seguinte forma:

```
% utente: #Idutente, N° Segurança_Social, Nome, Data_Nasc, Email,
% Telefone, Morada, Profissão, [Doenças_Crónicas], #CentroSaúde -> {V,F}

utente(1,1234,'alice',data(1950,2,12), 'alice@gmail.com',935852315,
'Praceta Conde Arnoso 22 2635-240 FRANCOS', padeiro,[asma,bronquite], 1).
utente(2,1234,'paulo',data(2000,5,12), 'paulo@gmail.com',934209315,
'Rua Portas Água 2 6120-772 MATADOURO', carteiro,[rinite], 1).
utente(3,1234,'josefa',data(1926,3,20), 'josefa@gmail.com',930052314,
'Largo Prazeres 34 6100-689 QUINTA DO ROSÁRIO', peixeiro,[diabetes], 1).
utente(4,1234,'tiago',data(1993,3,22), 'tiagoaf@gamil.com',965850111,
'R Capela 118 2410-433 LEIRIA', informatico,[], 1).
utente(5,1234,'juliana',data(1965,5,23), 'juliana@gamil.com',960350178,
'Estrada Logo Deus 110 3720-568 IGREJA', medico,[], 1).
utente(6,1234,'alice almeida rodrigues',data(2000,8,19), 'alicearod@gamil.com',936950825,
'R Portela 94 3600-202 CASTRO DAIRE', militar,[], 1).
utente(7,1234,'miguel carvalho',data(1994,9,4), 'miguelcarvalho@gamil.com',935621465,
'R Alto Eira 110 2745-758 QUELUZ', policia,[], 1).
utente(8,1234,'diana batista',data(2003,3,27), 'dianabati@gamil.com',965836215,
'R Nossa Senhora Fátima 61 3400-444 GALIZES', estudante,[osteoartrite], 1).
utente(9,1234,'pedro albino',data(1961,1,3), 'albinopedro@gamil.com',925856215,
'R Tomé B Queirós 118 4525-221 CANEDO', carpinteiro,[obesidade, 'doenca coronaria'], 1).
utente(10,1234,'albertina maria',data(1959,8,24), 'mariaalbertina@gamil.com',925856565,
'Quinta Beloura 21 2714-521 PORTELA', costureira,[diabetes], 1).
utente(11,1234,'rita abreu',data(1954,2,10), 'ritaabreu@gamil.com',965256565,
'R Cimo Povo 47 4615-355 BORBA DE GODIM', farmaceutica,[], 1).
utente(12,1234,'alberto andrade',data(1929,9,30), 'andradealberto@gamil.com',965216165,
'Avenida Almirante Reis 81 2580-467 CARREGADO', reformado,[], 1).
utente(13,1234,'aline correia rodrigues',data(1969,5,19), 'jalberto@gamil.com',216950825,
'R Pescador Bacalhoeiro 78 4495-441 PÓVOA DE VARZIM', informatico,[dpoc], 1).
utente(14,1234,'Roberto Miguel',data(1974,5,19), 'robMig@gmail.com',216950825,
'R São Vicente 75 2070-580 VALE DA PINTA', staff,[], 1).
utente(15,1234,'Ana Tomas',data(1972,6,19), 'anaTom@gmail.com',216950825,
'R Luís Camões 18 5210-263 PARADELA', staff,[], 1).
utente(16,1234,'Patricia Filipa',data(1969,5,30), 'pf@gmail.com',216950825,
'R São Bento 52 4765-714 OLIVEIRA SÃO MATEUS', staff,[], 2).
utente(17,1234,'Joana Sousa',data(1968,2,22), 'js@gmail.com',216950825,
'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2).
```

Figura 1: Predicados Utente

CENTRO DE SAÚDE

De modo a complementar o sistema, é importante manter registo dos centros de saúde onde decorrem as vacinações. Cada centro de saúde pode ser identificado por informações como o seu ID, o nome, morada, telefone e *email.* A implementação destes registos na base de conhecimento traduziu-se em:

```
%------------------------------ - - - - - - - - - - - - - - - -
% centro_saude: #Idcentro, Nome, Morada, Telefone, Email -> {V,F}

centro_saude(1,centroSaude1, 'Largo Prazeres 41 6150-114 CASAL DA RIBEIRA', 251423567, 'centro1Saude@gmail.com').
centro_saude(2,centroSaude2, 'R Casal Borga 103 2150-065 QUINTA DA BROA', 250763183, 'centro2Saude@gmail.com').
```

Figura 2: Predicados centro de saúde

STAFF

O *staff* corresponde às pessoas responsáveis por vacinar os utentes. Como tal, e de modo a careterizar um indivíduo com este estatuto, foram armazenadas informações como o seu ID, o ID do centro de saúde onde trabalha, o seu nome e o seu *email.* Em PROLOG, a implementação tomou o seguinte rumo:

```
%--------------------------------- - - - - - - - - - - - - - - - -
% staff: #Idstaff, #Idcentro, Nome, email -> {V,F}

staff(1,1, 'Roberto Miguel','robMig@gmail.com').
staff(2,1,'Ana Tomas','anaTom@gmail.com').
staff(3,2, 'Patricia Filipa','pf@gmail.com').
staff(4,2,'Joana Sousa','js@gmail.com').
```

Figura 3: Predicados *staff*

VACINAÇÃO COVID

Em relação à vacinação contra o covid, foi necessário armazenar o registo das vacinações tendo em conta o ID do *staff* que vacinou o utente, o ID do próprio utente, a data da toma, a vacina em questão e qual foi a toma correspondente àquele registo. A seguinte figura ilustra o povoamento efetuado dos registos de vacinação:

```
%--------------------------------- - - - - - - - - - - - - - - - -
% vacinação_Covid: #Staf, #utente, Data, Vacina, Toma -> {V,F}

vacinacao_Covid(1,5,data(2021,2,10),astrazeneca,1).
vacinacao_Covid(1,8,data(2021,3,13),pfizer,1).
vacinacao_Covid(1,13,data(2021,3,13),pfizer,1).
vacinacao_Covid(1,13,data(2021,4,10),pfizer,2).
vacinacao_Covid(1,3,data(2021,2,19),astrazeneca,1).
vacinacao_Covid(2,3,data(2021,2,20),astrazeneca,2).
vacinacao_Covid(2,14,data(2021,1,14),pfizer,1).
vacinacao_Covid(1,15,data(2021,1,10),pfizer,1).
vacinacao_Covid(4,16,data(2021,1,22),pfizer,1).
vacinacao_Covid(3,17,data(2021,1,20),pfizer,1).
```

Figura 4: Predicados vacinação covid

FASES DE VACINAÇÃO

Para as fases de vacinação, foi decido implementar 3 fases cada uma com critérios de inclusão de utentes distintos, tal como está descrito no site oficial da DGS (ver Referências).

Para a fase 1, os critérios de inclusão são os utentes com mais de 80 ou aqueles com mais de 50 anos que possuem um conjunto de doenças tais como, insuficiência cardíaca e insuficiência renal. Além disso, todos os utentes que sejam profissionais de saúde e lares, militares e funcionários de segurança pública também estão incluídos nesta fase.

Para a segunda fase, os critérios de inclusão envolvem todos os utentes que sejam maiores de 65 anos ou utentes com mais de 50 anos que tenham sido diagnosticados com diabetes, neoplasia, entre outros. Além disso, é nesta fase que são vacinados todos os professores.

Na última fase são vacinados todos os restantes utentes que não foram vacinados nas fases anteriores. Em cada fase foram determinadas datas de início e fim da vacinação, de forma que sejam sequenciais.

Para a implementação das fases em PROLOG decidiu-se criar um novo predicado designado por *fase*. Este conhecimento é caracterizado por uma data de início e fim, pela lista das doenças que fazem parte dos critérios de inclusão, uma lista com as profissões incluídas na fase e também duas idades, uma delas relacionada com as doenças da lista e a outra relacionada com a idade mínima para tomar a vacina nessa fase. Para caracterizar as datas de início e fim decidiu-se descrevê-las através do conhecimento *data* que é caracterizada por um ano, mês e dia. Na figura abaixo encontra-se a representação das fases em PROLOG.

```
%-------------------------------- - - - - - - - - - - - - - - -
% fase: #fase, dataInicio, dataFim, lista profissoes, lista doenças, idadeDoenca, idade -> {V,F}

fase(1,data(2020,12,11),data(2021,4,11),
    [medico, enfermeiro, militar, 'auxiliar lar', 'auxiliar saude', policia, staff],
    ['insuficiencia cardiaca', 'doenca coronaria', 'insuficiencia renal', dpoc],
    50,80).

fase(2,data(2021,4,12),data(2021,7,12),
    [professor],
    [diabetes, neoplasia, 'doenca renal cronica', 'insuficiencia hepatica','hipertensao arterial',obesidade],
    50,65).
```

Figura 5: Fases de vacinação

EXTRA – CANCELA VACINA

De modo a completar o sistema implementado, considerou-se conveniente ter em conta o caso em que um utente rejeita a vacina, ou por outras palavras, efetua ao cancelamento da mesma. Para isso, foram adicionados à base de conhecimento novos predicados que denotam esse mesmo propósito, através da identificação do utente que cancelou a vacinação e da data em que o efetuou, tal como é visível na figura 6.

```
%---------------------------------- - - - - - - - - - - - - - - - -
% cancela_vacina: #utente, DataCancelamento -> {V,F}

cancela_vacina(15,data(2021,1,9)).
cancela_vacina(1,data(2021,2,10)).
```

Figura 6: Predicado *cancela_vacina*

FUNCIONALIDADES

As funcionalidades implementadas visam caracterizar e descrever todos os conceitos e fundamentos na área da vacinação do COVID-19 em Portugal. Através da linguagem PROLOG, implementou-se um conjunto de necessidades básicas que se encontram, de seguida, detalhadas e fundamentadas com conceitos tanto da área da vacinação como de sistemas de conhecimento.

O ficheiro que agrega estas funcionalidades é inicialmente composto por um conjunto de definições iniciais, que tem como propósito corrigir eventuais avisos fornecidos pelo PROLOG. Em particular, o primeiro comando permite que seja impressa a totalidade de uma lista. Estas definições são visíveis na figura 7.

```
%------------------------------- - - - - - - - - - - -  -  -  -  -  -
% Definicoes iniciais
:- set_prolog_flag(answer_write_options, [max_depth(0)]).
:- set_prolog_flag( discontiguous_warnings,off ).
:- set_prolog_flag( single_var_warnings,off ).

:- style_check(-singleton).
```

Figura 7: Definições Iniciais

Além disso, sendo este o ficheiro principal a ser carregado pelo PROLOG, é também importante mencionar que neste ponto são efetuados os *includes* de todos os outros ficheiros constituintes do sistema.

```
%------------------------------- - - - - - - - - - - -  -  -  -  -  -
% Includes
:- include('BaseConhecimento.pl').
:- include('regrasAuxiliares.pl').
:- include('EvolucaoInvolucao.pl').
:- include('Invariantes.pl').
```

Figura 8: Includes

VACINADOS

Uma funcionalidade implementada neste sistema foi a consulta das pessoas que já foram vacinadas, ou seja, todos os utentes que possuem registo de toma de 2 doses da vacina. De forma a concretizar a ideia previamente exposta, recorreu-se à função *solucoes* na regra *totalVacinados*, em que o resultado será uma lista de par (ID Utente, Nome) e será utilizada outra regra *vacinado* para verificar se um dado utente se encontra vacinado. Esta regra é composta por diferentes conjunções, que denotam a existência do registo de utente na base de conhecimento e a existência de registos de vacinação correspondentes à 1ª e 2ª fases.

```
%------------------------------- - - - - - - - - - - - - - - - - -
% Total de Vacinados - com 2 doses: Lista -> {V,F}

totalVacinados(L) :- solucoes((Id,X),vacinado(Id,X),L).

vacinado(Id,X) :- utente(Id,_,X,_,_,_,_,_,_,_),
                  vacinacao_Covid(_,Id,_,_,1),
                  vacinacao_Covid(_,Id,_,_,2).
```

Figura 9: Regra *totalVacinados*

NÃO VACINADOS

De modo a denotar quais os utentes que não se encontram vacinados, foi necessário verificar quais ainda não tomaram nenhuma dose da vacina. Para concretizar tal feito, criou-se a regra *totalNaoVacinados*. Esta regra recorre à função *solucoes* cujo resultado denota uma lista de pares (ID Utente, Nome) e verifica se cada utente da base de conhecimento não se encontra vacinado, i.e, não tomou nenhuma dose da vacina, através da regra naoV*acinado*. Para a sua implementação foi necessário consultar os utentes presentes na BC e verificar se não existem registos da primeira e segunda toma da vacina, tal como se encontra representado de seguida:

```
%------------------------------- - - - - - - - - - - - - - - - - -
% Total Não Vacinados - sem nenhuma dose: Lista -> {V,F}

totalNaoVacinados(L) :- solucoes((Id,X),naoVacinado(Id,X),L).

naoVacinado(Id,X) :- utente(Id,_,X,_,_,_,_,_,_,_),
                     nao(vacinacao_Covid(_,Id,_,_,_)).
```

Figura 10: Regra *totalNaoVacinados*

VACINADOS INDEVIDAMENTE

Esta funcionalidade permite conhecer quais os utentes que foram vacinados indevidamente, isto é, quais os utentes vacinados numa fase que não lhe correspondia, os utentes que levaram a segunda dose da vacina antes do tempo, as pessoas que cancelaram a vacina e posteriormente foram vacinadas com a 1 dose ou se cancelaram após tomar a 1 e de seguida tomaram a 2 dose. Decidimos considerar que todas as vacinas administradas em Portugal são de 2 doses e que entre cada dose têm de passar cerca de 28 dias, tal como está descrito no site oficial da DGS.

Esta funcionalidade foi implementada em PROLOG através da regra *vacinasIndevidas* que vai determinar, com o uso da função *solucoes* do PROLOG e da regra *utenteIndevido*, todos

os utentes vacinados indevidamente devolvem uma lista de pares com o nome e o seu ID de utente.

Para a criação da regra *utenteIndevido* foram implementados três critérios específicos, um que analisa se a vacina foi administrada dentro da fase respetiva, outro que determina se a segunda fase foi administrada dentro dos dias corretos e uma última que determina se o utente cancelou a vacina e posteriormente foi-lhe administrada essa vacina.

Para o primeiro critério vamos verificar em que fase a primeira dose da vacina foi administrada através da regra *obterFase*. Para verificar se o utente está incluído nessa fase ou não utilizamos a regra *confirmaFase.* Caso o utente não tenha nenhum critério de inclusão para essa fase então o utente recebeu uma vacina indevida.

No que toca ao segundo critério verificamos se um utente tem as duas doses da vacina. Para verificar se a segunda dose da vacina foi administrada dentro do tempo correto utilizamos a regra *confirmaToma*. Caso o utente não tenha a segunda toma correta então o utente recebeu uma vacina indevida.

No terceiro e último critério verificamos se o utente cancelou a sua vacinação e em que data o fez através do predicado cancela_vacina. Caso a data de cancelamento tenha sido antes da administração da primeira toma da vacina e caso essa data de cancelamento tenha disso depois da administração da primeira toma e antes da segunda toma então o utente foi vacinado indevidamente. Na figura 11 encontra-se a representação da funcionalidade em PROLOG de todos os critérios.

```
% Utentes com vacinas indevidas: Lista -> {V,F}

vacinasIndevidas(L) :- solucoes((Id,X),utenteIndevido(Id,X),L).

%- caso seja vacinado na fase errada
utenteIndevido(Id,X):- utente(Id,_,X,Data,_,_,_,Profissao,Doenca,_),
                       vacinacao_Covid(_,Id,DataVacina1,_,1),
                       obtemFase(DataVacina1,Fase),
                       nao(confirmaFase(Fase,Id,Profissao,Doenca)).

%- caso seja vacinado fora do intervalo tempo
utenteIndevido(Id,X):- utente(Id,_,X,Data,_,_,_,Profissao,Doenca,_),
                       vacinacao_Covid(_,Id,DataVacina1,_,1),
                       vacinacao_Covid(_,Id,DataVacina2,_,2),
                       nao(confirmaToma(DataVacina1,DataVacina2)).

%- caso tenha cancelado a vacina e depois tomou a 1 dose
utenteIndevido(Id,X):- utente(Id,_,X,_,_,_,_,_,_,_),
                       cancela_vacina(Id,DataCancela),
                       vacinacao_Covid(_,Id,DataVacina1,_,1),
                       dataAntes(DataCancela,DataVacina1).

%- caso tenha tomado a 1 dose cancelado a vacina e depois tomou 2 dose
utenteIndevido(Id,X):- utente(Id,_,X,_,_,_,_,_,_,_),
                       cancela_vacina(Id,DataCancela),
                       vacinacao_Covid(_,Id,DataVacina2,_,2),
                       dataAntes(DataCancela,DataVacina2).
```

Figura 11: Vacinados indevidamente

CANDIDATOS

Em adição às funcionalidades anteriormente referidas, foi também implementada uma regra que permite identificar quais as pessoas que não estão vacinadas e são candidatas à vacinação numa dada fase. Para isso foi implementada a regra *utentesCandidatos* que recorre à função *solucoes* agrupando os resultados numa lista de par (ID, Nome). A regra *utentesCandidatos* recorre a *candidatos*, que verifica individualmente para cada candidato se pertence àquela fase. A regra *candidatos* verifica quais os utentes na BC e, para esses, se não cancelaram a vacina, se não foram previamente vacinados e se a fase em questão corresponde à fase em que o utente se insere.

```
% Utentes candidatos ás fases: Lista -> {V,F}

utentesCandidatos(Fase,L) :- solucoes((Id,X),candidatos(Fase,X,Id),L).
candidatos(Fase,X,Id) :- utente(Id,_,X,Data,_,_,_,Profissao,Doenca,_),
                         nao(cancela_vacina(Id,DataCancela)),
                         nao(vacinacao_Covid(_,Id,_,_,_)),
                         confirmaFase(Fase,Id,Profissao,Doenca).
```

Figura 12: Regra *candidatos*

FALTA SEGUNDA TOMA

Para responder a este tópico, foi necessário implementar a regra *utentesSegundaToma* que recorre à função *solucoes,* cujo resultado é uma lista de par (ID, Nome), com auxílio da regra *candidatosSegunda*. Esta última efetua a verificação de diversas conjunções, tais como a existência de utente, registo da primeira toma da vacina e não haver registo da segunda toma.

```
%--------------------------------- - - - - - - - - - - - - - - - - - -
% utentes candidatos á segunda toma - so com 1 dose: Lista -> {V,F}

utentesSegundaToma(L) :- solucoes((Id,X),candidatosSegunda(X,Id),L).
candidatosSegunda(X,Id) :- utente(Id,_,X,Data,_,_,_,Profissao,Doenca,_),
                           vacinacao_Covid(_,Id,_,_,1),
                           nao(vacinacao_Covid(_,Id,_,_,2)).
```

Figura 13: Regra *utentesSegundaToma*

SISTEMA DE INFERÊNCIA

O sistema de inferência aparece neste projeto com o propósito de estender a programação em lógica, a partir da adição de um novo valor lógico: o desconhecido. No entanto, não se revelou âmbito de estudo nesta fase, daí a implementação de um SI mais simplificado que apenas lida com os valores verdadeiro e falso. O SI implementado responde à questão que lhe é transmitida, indicando **verdadeiro** no caso em que esse conhecimento possa ser inferido a partir da base de conhecimento, ou **falso**, caso contrário. Neste último caso, recorremos ao meta-predicado *nao* para representar a negação no SI.

```
% Sistema de inferencia: Questão, ValorLógico -> {V,F}

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

Figura 14: Sistema de inferência

EXTRA - VACINAS NUM DADO CENTRO DE SAÚDE

De modo a consultar quais os utentes que foram vacinados num dado centro de saúde, foi implementada uma regra *vacinasCentro* que recorre à função *solucoes* agrupando o resultado numa lista de par (Nome, ID). Esta regra recorre a uma outra auxiliar *vacinaCentroUtente* que verifica se um dado utente está na BC e se esse utente foi vacinado num centro de saúde por um elemento do *staff* que se encontra empregado nesse mesmo centro.

```
%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Vacinas dadas num centro de saude: Centro, Lista -> {V,F}

vacinasCentro(Centro, L) :- solucoes((X,Id),vacinaCentroUtente(Centro,X,Id),L).

vacinaCentroUtente(Centro,X,Id) :- utente(Id,_,X,Data,_,_,_,_,_,_),
                                   vacinacao_Covid(IdStaff,Id,_,_,_),
                                   staff(IdStaff,Centro,_,_).
```

Figura 15: Funcionalidade Extra - *vacinasCentro*

EXTRA – NÚMERO VACINAS NUM DADO CENTRO DE SAÚDE

Para calcular o número de vacinas efetuadas num dado centro de saúde, utilizou-se a regra anteriormente referida, mas em vez de fornecer uma lista com informações dos utentes lá vacinados, fornece uma variável com o comprimento dessa mesma lista. Desta forma, conseguimos ter acesso à informação pretendida.

```
%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Numero de vacinas dadas num centro de saude: Centro, Numero -> {V,F}

nrVacinasCentro(Centro, N) :- solucoes((X,Id),
                                vacinaCentroUtente(Centro,X,Id),L),
                                comprimento(L,N).
```

Figura 16: Funcionalidade Extra - *nrVacinasCentro*

EXTRA – VACINAS CANCELADAS

Um dos extras considerados para este projeto, foi a oportunidade de consultar quais os utentes que rejeitaram a toma da vacina. Para isso, recorreu-se à função *solucoes* que irá agrupar os resultados numa lista de par (Nome, ID Utente) e que recorre a outra regra *utenteCancelado*. Esta última verifica se existe informação na BC relativa ao cancelamento da vacina daquele utente, e se esse mesmo utente está presente na BC.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Utentes que nao querem ser vacinados: Lista -> {V,F}

vacinasCanceladas(L) :- solucoes((X,Id),utenteCancelado(X,Id),L).

utenteCancelado(X,Id) :- cancela_vacina(Id, Data), utente(Id,_,X,_,_,_,_,_,_,_).
```

Figura 17 : Funcionalidade Extra - *vacinasCanceladas*

REGRAS AUXILIARES

Com o propósito de agrupar as diversas regras que serviram de auxílio às funcionalidades do sistema, criou-se o documento "regrasAuxiliares.pl". Em cada subsecção presente neste tópico é possível consultar informação explicativa de cada regra auxiliar.

OBTER FASE

Nesta regra determinamos qual a fase em que a vacina foi administrada através da data de vacinação do utente e das datas de início e fim de fase. Assim, a vacina foi fornecida durante uma certa fase caso a data de vacinação seja antes da data de fim de fase e caso a data de início de fase seja antes da data de vacinação.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Determina fase segundo data de vacinação: DataVacinacao, Fase -> {V,F}

obtemFase(Data,1) :- fase(1,Data0,Data1,_,_,_,_),
                     dataAntes(Data0,Data),
                     dataAntes(Data,Data1).
obtemFase(Data,2) :- fase(2,Data0,Data1,_,_,_,_),
                     dataAntes(Data0,Data),
                     dataAntes(Data,Data1).
obtemFase(Data,3) :- nao(obtemFase(Data,1)),
                     nao(obtemFase(Data,2)).
```

Figura 18: Regra *obtemFase*

DATA ANTES

De modo a verificar se uma data precede outra, implementou-se a regra *dataAntes* que avalia os campos ano, mês e dia de cada uma das datas e efetua uma comparação. Na figura 19 está representada a implementação da regra *dataAntes*.

```
dataAntes(date(Y0,M0,D0),date(Y1,M1,D1)) :- (Y0-Y1 < 0 ;
                                             Y0-Y1 =:= 0, M0-M1 < 0;
                                             Y0-Y1 =:= 0, M0-M1 =:= 0, D0-D1 =< 0).
```

Figura 19: Regra *dataAntes*

CONFIRMAR FASE

A implementação desta regra teve como objetivo determinar se um utente possui os critérios suficientes para receber a vacina na fase 1, 2 ou 3.

Para a fase 1 verifica-se se o utente tem uma profissão que esteja incluída no leque de profissões para esta fase através da função *member* do PROLOG. Além disso verificou-se se as doenças do utente fazem parte do conjunto de doenças incluídas nesta fase e se a idade deste condiz com o espetro de idades reservadas para esta fase. De forma a cumprir com este objetivo recorreu-se às regras *idade* e *verificaElem,* que determinam a idade do utente e se este tem pelo menos uma doença no conjunto de doenças incluídas na fase, respetivamente.

Para a fase 2 verificamos os mesmos parâmetros como na fase 1, contudo também verificamos se o utente não está confirmado na fase 1. Caso este não esteja confirmado na fase 1 então há possibilidade de estar incluído na fase 2. Para a fase 3 verificamos se o utente foi confirmado na fase 1 e na fase 2. Caso o utente não esteja incluído nestas 2 fases então este está automaticamente incluído na fase 3. Assim, obtemos a implementação da regra *confirmaFase*, tal como está na imagem abaixo.

```
confirmaFase(1,Id,Profissao,Doenca) :- fase(1,_,_,LP,LD,ID,I),
                                       member(Profissao,LP),!.
confirmaFase(1,Id,Profissao,Doenca) :- fase(1,_,_,LP,LD,ID,I),
                                       idade(Id,Idade),
                                       Idade>= I, !.
confirmaFase(1,Id,Profissao,Doenca) :- fase(1,_,_,LP,LD,ID,I),
                                       idade(Id,Idade),
                                       Idade>= ID,
                                       verificaElem(LD,Doenca).

confirmaFase(2,Id,Profissao,Doenca) :- fase(2,_,_,LP,LD,ID,I),
                                       nao(confirmaFase(1,Id,Profissao,Doenca)),
                                       member(Profissao,LP),!.
confirmaFase(2,Id,Profissao,Doenca) :- fase(2,_,_,LP,LD,ID,I),
                                       nao(confirmaFase(1,Id,Profissao,Doenca)),
                                       idade(Id,Idade),
                                       Idade>= I,!.
confirmaFase(2,Id,Profissao,Doenca) :- fase(2,_,_,LP,LD,ID,I),
                                       nao(confirmaFase(1,Id,Profissao,Doenca)),
                                       idade(Id,Idade),
                                       Idade>= ID,
                                       verificaElem(LD,Doenca).

confirmaFase(3,Id,Profissao,Doenca) :- nao(confirmaFase(2,Id,Profissao,Doenca)),
                                       nao(confirmaFase(1,Id,Profissao,Doenca)).
```

Figura 20: Implementação da regra *confirmaFase*

CONFIRMAR TOMA

Na regra *confirmaToma* determinamos se o número de dias entre a 1ª toma e a 2ª está entre 26 e 31 dias. Para cumprir com este objetivo consultamos uma segunda regra, *calculaDias,* que calcula o número de dias entre o início do ano e a data em que foi vacinado a 1º e a 2º dose. De seguida, verificamos se a diferença entre os dias obtidos está entre 26 e 31.

```
confirmaToma(DataVacina1,DataVacina2) :- calculaDias(DataVacina1,Res1),
                                         calculaDias(DataVacina2,Res2),
                                         Res2-Res1 >= 26 , Res2-Res1 =< 31.
```

Figura 21: Regra *confirmaToma*

CALCULAR DIAS

Para cumprir com este objetivo implementamos uma segunda regra, *calculaDias,* que calcula o número de dias entre o início do ano e a data em que foi vacinado a 1º e a 2º dose. Todos estes números servem para se poderem considerar todas as nuances provenientes dos anos.

```
calculaDias(date(Y,M,D),Res):- Res is (((Y*1461)/4)+((M*153)/5)+D).
```

Figura 22: Regra *calculaDias*

CALCULAR IDADE

Na regra *idade* calculamos a idade do utente através da sua data de nascimento e da data atual do sistema. Esta regra verifica um conjunto de conjunções, que denotam a existência do utente, o cálculo da data atual e a obtenção da idade propriamente dita. Esta implementação traduz-se na seguinte imagem:

```
%--------------------------------
% Determina a idade de um utente: #IdUtente, Idade -> {V,F}

idade(Id,I) :- utente(Id,_,_,D,_,_,_,_,_,_),
               dataAtual(Key, Value),
               obterIdade(D,Value,I).
```

Figura 23: Regra *idade*

DATA ATUAL

Para determinar a data atual implementou-se a regra *dataAtual* que através de funções do PROLOG determina a data atual sobre o formato date (ano, mês, dia), armazenando o seu valor no campo *value*.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Obtém data atual: Key, Value -> {V,F}

dataAtual(Key, Value) :- get_time(Stamp),
                         stamp_date_time(Stamp, DateTime, local),
                         date_time_value(Key, DateTime, Value).
```

Figura 24: Regra *dataAtual*

OBTER IDADE

Na regra *obterIdade* comparamos o ano atual com a ano de nascimento do utente obtendo a idade do utente. Esta implementação traduz-se em linguagem PROLOG da seguinte forma:

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Obtém a idade de um utente atraves de duas datas: DataUtente, DataAtual -> {V,F}

obterIdade(data(Y0,_,_),date(Y1,_,_), X) :- X is Y1-Y0.
```

Figura 25: Regra *obterIdade*

VERIFICAR ELEMENTO

A regra *verificaElem* serviu de auxílio à implementação da regra *confirmaFase.* Assim, determinamos se existe pelo menos um elemento na lista de doenças do utente que também esteja no conjunto de doenças da respetiva fase. Na figura 26 encontra-se implementado a regra apresentada.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Verifica se existe pelo menos um elemento de uma lista na outra: ListaUtente, ListaDoenças -> {V,F}

verificaElem([X|H],LD) :- member(X,LD).
verificaElem([X|H],LD) :- verificaElem(H,LD).
```

Figura 26: Regra *verificaElem*

META-PREDICADO NÃO

Consideramos conveniente implementar a extensão do meta-predicado *não* de modo a poder efetuar a negação de predicados. Deste modo, se o predicado *Questao* possuir o valor **Falso**, a regra *não* irá opor este valor de verdade. O mesmo acontece caso o valor de verdade de *Questao* seja **Verdade**.

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

nao( Questao ) :-
    Questao, !, fail.
nao( Questao ).
```

Figura 27: Meta-predicado *nao*

META-PREDICADO SOLUÇÕES

Uma das funções muito utilizadas e relevantes do PROLOG é o *findall.* Como tal, implementou-se a regra *solucoes* que recorre à função anteriormente referida e procede de modo equivalente.

```
%------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do meta-predicado solucoes: Elemento, Questao, Lista -> {V,F}

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

Figura 28: Meta-predicado *solucoes*

META-PREDICADO COMPRIMENTO

No sistema implementado foi necessário recorrer frequentemente à função *length.* Deste modo, considerou-se conveniente efetuar a sua conversão para o meta-predicado *comprimento* que tira partido de todas as suas funcionalidades e procede de modo análogo.

```
%------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do meta-predicado comprimento: Lista, Comprimento -> {V,F}

comprimento(L,N):- length(L,N).
```

Figura 29: Meta-predicado *comprimento*

VERIFICAR CENTRO DE SAÚDE

De modo a verificar se o centro de saúde onde o utente se insere coincide com o centro de saúde onde o *staff* trabalha, implementou-se a regra auxiliar *verificaCentro*. Esta regra recorre a duas conjunções que verificam a existência na BC de registos de utente e *staff* e a igualdade dos centros de saúde. A sua implementação foi necessária para auxiliar a criação de um invariante.

```
%---------------------------------- - - - - - - - - - - - - - - - - - -
% Verifica se o centro do Utente é o mesmo do staff: Id, Staff -> {V,F}

verificaCentro(Id,Staff) :- utente(Id,_,_,_,_,_,_,_,Centro), staff(Staff,Centro,_,_).
```

Figura 30: Regra *verificaCentro*

VALIDAR DATA

Com o objetivo de verificar se uma dada *data* possui um valor válido, implementou-se a regra *validaData*. Esta regra recorre à regra *data* para verificar a validade da mesma. Nesta última é efetuada a validação propriamente dita, isto é, restrição do valor dos meses e do ano a determinados valores. A sua implementação foi necessária para auxiliar a criação de um invariante. Na seguinte figura está representada a implementação:

```
%---------------------------------- - - - - - - - - - - - - - - - - - -
% Verifica se data é valida: Data -> {V,F}

validaData(data(Ano,Mes,Dia)):- data(Ano,Mes,Dia).

data(Ano,2,Dia) :- integer(Ano), integer(Dia), Dia >=1, Dia =< 29.
data(Ano,Mes,Dia) :- integer(Ano), integer(Mes), integer(Dia), Mes >= 1, Mes =< 12, Dia >=1, Dia =< 31.
```

Figura 31: Regra *validaData*

VALIDAR LISTA

De modo a validar uma lista implementou-se a função *validaLista* que, de forma muito simplificada e elementar, valida a existência e estrutura de uma lista. A sua implementação foi necessária para auxiliar a criação de um invariante.

```
%---------------------------------- - - - - - - - - - - - - - - - - - -
% Verifica se a variavel é uma lista: Lista -> {V,F}

validaLista([]).
validaLista([H|T]).
```

Figura 32: Regra *validaLista*

EXTRA - INVARIANTES

Um tópico extra implementado no sistema foi a existência de invariantes que controlam e restringem a informação a ser inserida ou removida no sistema.

Analogamente às secções anteriores, a presente contém uma descrição detalhada acerca dos elementos constituintes deste documento, "Invariantes.pl".

Este ficheiro em linguagem PROLOG é iniciado com algumas definições iniciais relevantes para a evolução ou involução do conhecimento. Estas definições traduzem a possibilidade de adicionar ou remover novos predicados à BC declarando esses predicados como dinâmicos.

```
%------------------------------ - - - - - - - - - - - - - - - - -
% Definicoes iniciais

:- op( 900,xfy,'::' ). %- criação de um novo operador para o prolog
:- dynamic utente/10.
:- dynamic staff/4.
:- dynamic centro_saude/5.
:- dynamic vacinacao_Covid/5.
:- dynamic cancela_vacina/2.
```

Figura 33: Invariantes - Definições Iniciais

INVARIANTE UTENTE

Para começar, foi necessário implementar invariantes para o predicado utente. Para este predicado apenas se considerou a inserção de novo conhecimento, ou seja, é possível adicionar novos utentes, mas não faz sentido excluir utentes da base de conhecimento, uma vez que consideramos manter o registo de todos os utentes.

No momento de inserção é necessário verificar diversos detalhes acerca dos parâmetros do predicado, isto é, as variáveis que representam o ID Utente, número de segurança social e ID Centro de saúde devem corresponder ao tipo de dados *integer.* Para as variáveis que denotam o nome de utente, email e profissão, deverá ser considerado o tipo de dados *atom.* No caso da data de nascimento de utente, foi necessário recorrer à regra *validaData* que trata de verificar se o valor inserido é correto. Por fim, para a inserção da lista de doenças do utente, utilizou-se a regra *validaLista* de forma a verificar a validade da mesma.

Desta forma, cumprindo as restrições anteriormente referidas, conseguimos garantir que o conhecimento a ser inserido na BC contém valores válidos e adequados ao contexto de utente. O invariante que permite identificar os tipos de dados está representado na seguinte figura:

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Estrutural para o perdicado utente que permite identificar os tipos das variaveis

+utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro) ::(
            integer(Id),
            integer(SS),
            atom(Nome),
            validaData(Data),
            atom(Email),
            atom(Profissao),
            validaLista(Doenca),
            integer(Centro)).
```

Figura 34: Invariante Utente 1

Além disso, também foi relevante acrescentar o invariante que permite limitar a inserção de novos conhecimentos. Este invariante verifica se o número de ocorrências de um dado utente encontradas na base de conhecimento é igual a 1.

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Estrutural para o predicado utente que permite limitar a insercao de novos conhecimentos

+utente(Id,_,_,_,_,_,_,_,_,_) :: (solucoes(Id,utente(Id,_,_,_,_,_,_,_,_,_),S), comprimento( S,N ), N == 1).
```

Figura 35: Invariante Utente 2

INVARIANTE CENTRO SAÚDE

No caso do invariante implementado para o centro de saúde considerou-se apenas o caso de inserção de novo conhecimento relativo a este predicado. Inicialmente, foi criado um invariante que verifica os tipos de dados do centro de saúde a ser inseridos e a sua validade. Assim, os valores ID Centro e Telefone deverão ser representados por um *integer* e os valores nome, morada e email do centro de saúde deverão ser representados pelo tipo de dados *atom.* Na seguinte figura está representada esta implementação:

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Estrutural para o predicado centro_saude que permite identificar os tipos das variaveis

+centro_saude(ID,Nome,Morada,Telefone,Email) :: (
    integer(ID),
    atom(Nome),
    atom(Morada),
    integer(Telefone),
    atom(Email)).
```

Figura 36: Invariante Centro de Saúde 1

De forma a permitir a inserção de conhecimento único e impedir repetições, foi criado um invariante que garante que o número de ocorrências de um dado centro de saúde na BC é igual a 1. Na figura de seguida está exposto o raciocínio anterior:

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Estrutural para o predicado centro_saude que permite limitar a insercao de novos conhecimentos

+centro_saude(Id,_,_,_,_) :: (solucoes(Id,centro_saude(Id,_,_,_,_),S), comprimento( S,N ), N == 1 ).
```

Figura 37: Invariante Centro de Saúde 2

INVARIANTE STAFF

Para o predicado *staff* considerou-se a possibilidade de adição de conhecimento deste tipo. De modo a concretizar tal feito, começou-se por verificar os tipos de dados constituintes deste predicado e posteriormente a restrição que impede a existência de conhecimento repetido, isto é, dois elementos do *staff* não podem possuir o mesmo ID.

Os tipos de dados que representam o ID Staff e o ID Centro de saúde devem ser do tipo *integer*. No caso dos variáveis que denotam o nome do centro de saúde e o email, devem constituir elementos do tipo *atom.* Na seguinte figura está representado o invariante que constitui as restrições anteriormente referidas:

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Estrutural para o predicado staff que permite identificar os tipos das variaveis

+staff(Idstaff, Idcentro, Nome, Email) :: (
    integer(Idstaff),
    integer(Idcentro),
    atom(Nome),
    atom(Email)).
```

Figura 38: Invariante Staff 1

Para garantir que não existe conhecimento repetido na base de conhecimento é necessário efetuar uma procura por todas as ocorrências de um dado ID de Centro na base de conhecimento e verificar se apenas existe uma ocorrência. O raciocínio referido traduz-se em PROLOG da seguinte forma:

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Estrutural para o predicado staff que permite limitar a insercao de novos conhecimentos

+staff(Id,_,_,_) :: (solucoes(Id,staff(Id,_,_,_),S), comprimento( S,N ), N == 1 ).
```

Figura 39: Invariante Staff 2

INVARIANTE VACINAÇÃO COVID

Quanto ao predicado *vacinacao_Covid* foi considerado importantíssimo proceder-se à adição de conhecimento deste tipo. Para isso primeiro foi necessário verificar o seu tipo de dados e de seguida verificar se as restrições que este contém foram cumpridas antes de ser adicionado.

Deste modo tanto para representar ao ID do Staff que dá a vacina, como o ID do Utente que a vai receber e ainda o número da toma, são representados por um *integer*. Para identificar o nome da vacina foi usado o tipo de dados *atom.* Por fim no caso da data da vacina, foi necessário recorrer à regra *validaData* que trata de verificar se uma data inserida é válida.

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Estrutural para o vacinacao_Covid staff que permite identificar os tipos das variaveis

+vacinacao_Covid(IDStaff,IDUtente,Data,Vacina,Toma) :: (
    integer(IDStaff),
    integer(IDUtente),
    validaData(Data),
    atom(Vacina),
    integer(Toma)).
```

Figura 40: Invariante Vacinação 1

De modo a limitar a adição deste predicado, foram ainda realizados mais dois invariantes, um para verificar a primeira toma e um outro para a segunda. Na primeira toma é preciso ter em atenção se o utente a receber a vacina não tem já um registo desta primeira toma. Quanto à segunda toma, não só é necessário verificar se este já tomou a segunda dose, mas também verificar se a vacina a tomar foi a mesma que a tomada numa primeira toma. Em ambos os critérios verificamos também se o utente que vai ser vacinado pertence aos conjuntos de utentes presentes na BC.

```
%---------------------------------- - - - - - - - - - - - - - - - - -
% % Invariante Estrutural para o predicado vacinacao_Covid que permite limitar a insercao de novos conhecimentos

+vacinacao_Covid(Staff,Id,_,Vacina,1) :: (solucoes(Id,utente(Id,_,_,_,_,_,_,_,_,_),S), comprimento( S,N ), N == 1,
                                          solucoes(Id,vacinacao_Covid(_,Id,_,_,1),S), comprimento( S,N ), N == 1,
                                          verificaCentro(Id,Staff)). % tem de ser vacinado no centroSaude respetivo

+vacinacao_Covid(Staff,Id,_,Vacina,2) :: (solucoes(Id,utente(Id,_,_,_,_,_,_,_,_,_),S), comprimento( S,N ), N == 1,
                                          solucoes(Id,vacinacao_Covid(_,Id,_,Vacina,1),S1), comprimento( S1,N1 ), N1 == 1,
                                          solucoes(Id,vacinacao_Covid(_,Id,_,Vacina,2),S2), comprimento( S2,N2 ), N2 == 1,
                                          verificaCentro(Id,Staff)).
```

Figura 41: Invariante Vacinação 2

INVARIANTE CANCELA VACINA

O último invariante implementado foi para o controlo da adição e remoção de conhecimento do predicado *cancelaVacina*. Neste caso, considerou-se conveniente efetuar a adição e remoção de conhecimento, sendo que um utente pode rejeitar a vacina e posteriormente mudar de ideias.

Assim, de modo a implementar este pressuposto, começou-se por implementar um invariante que verifica o tipo de dados inserido. Os parâmetros fornecidos ao predicado *cancelaVacina* são os parâmetros ID de Utente e Data de cancelamento. Deste modo, prodeceu-se à verificação do valor ID que deverá corresponder a um *integer* e à validação da data de modo a constituir uma data válida, recorrendo à função *validaData.* O procedimento para a remoção de conhecimento ocorreu de forma análoga. Nas seguintes figuras encontram-se representados os invariantes que controlam os tipos de dados, em linguagem PROLOG:

```
%------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Estrutural para o predicado cancela_vacina que permite identificar os tipos das variaveis

+cancela_vacina(Id, Data) :: (integer(Id), validaData(Data)).
```

Figura 42: Invariante +*cancela_vacina* 1

```
%------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Referencial para o predicado cancela_vacina que permite identificar os tipos das variaveis - na remocao

-cancela_vacina(Id, Data) :: (integer(Id), validaData(Data)).
```

Figura 43: Invariante -*cancela_vacina* 1

Para adicionar um registo de cancelamento de vacina propriamente dito, foi necessário efetuar a verificação de diversas condições: verificar se existe um único utente com o ID fornecido, verificar que apenas existe um registo de cancelamento de vacina e que ainda não foram administradas a 1 e 2 doses da vacina. Para remover um registo de cancelamento foi necessário implementar um invariante estrutural que garante a existência do utente em questão e a inexistência daquele mesmo registo na base de conhecimento. A figuras 44 e 45 representam o invariante de adição e o de remoção, respetivamente.

```
%------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Estrutural para o predicado cancela_vacina que permite limitar a insercao de novos conhecimentos

+cancela_vacina(Id, Data) :: (solucoes(Id,utente(Id,_,_,_,_,_,_,_,_),S), comprimento( S,N ), N == 1,
                              solucoes(Id,cancela_vacina(Id,_),S0), comprimento( S0,N0 ), N0 == 1,
                              solucoes(Id,vacinacao_Covid(_,Id,_,Vacina,1),S1), comprimento( S1,N1 ), N1 == 0,
                              solucoes(Id,vacinacao_Covid(_,Id,_,Vacina,2),S2), comprimento( S2,N2 ), N2 == 0).
```

Figura 44: Invariante +*cancela_vacina* 2

```
%------------------------------- - - - - - - - - - - - - - - - - -
% Invariante Referencial para o predicado cancela_vacina que permite limitar a remocao de novos conhecimentos

-cancela_vacina(Id,Data) :: (solucoes(Id,utente(Id,_,_,_,_,_,_,_,_),S), comprimento( S,N ), N == 1,
                             solucoes(Id,cancela_vacina(Id,_),S1), comprimento( S1,N1 ), N1 == 0).
```

Figura 45: Invariante -*cancela_vacina* 2

EXTRA - EVOLUÇÃO E INVOLUÇÃO

Para que seja possível adicionar e remover conhecimento à base de conhecimento, consideramos conveniente implementar processos que permitem a evolução e a involução do mesmo. Nesta secção serão abordadas em detalhe as regras que sustentam este princípio.

O conteúdo relativo a este tópico, tal como foi referido anteriormente, encontra-se atribuído a um ficheiro individual "EvolucaoInvolucao.pl". Este ficheiro contém inicialmente uma definição inicial com o propósito de permitir a definição de invariantes. Na seguinte figura é possível observar a definição inicial deste documento:

```
%------------------------------- - - - - - - - - - - - - - - - - -
% Definicoes iniciais

:- op( 900,xfy,'::' ).
```

Figura 46: EvolucaoInvolucao - Definições Iniciais

EVOLUÇÃO

O predicado *evolucao* é aquele que vai tratar de adicionar informações à base de conhecimento. Para a sua implementação, o procedimento passou por efetuar uma procura por invariantes correspondentes a um dado *Termo*, agrupando os resultados numa *Lista*, seguida da inserção desse mesmo termo e da invocação da regra *teste* à *Lista*. A procura é feita com recurso à regra *solucoes* (secção Regras Auxiliares). A inserção recorre à regra *insercao* que efetua um *assert* ou *retract* do termo. A regra *teste* é abordada posteriormente nesta secção. Na seguinte figura estão representadas as regras *evolucao* e *insercao*:

```
%------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a evolucao do conhecimento

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

insercao(Termo) :- assert(Termo).
insercao(Termo) :- retract(Termo), !, fail. %- impede o backtracking (!) e falha (fail)
```

Figura 47: Evolução

INVOLUÇÃO

No caso da involucao, esta procede com o objetivo contrário à evolução, ou seja, possui o propósito de remover conhecimento. De modo a traduzir este pressuposto em PROLOG, criou-se a regra *involucao* que começa por verificar o *Termo* recebido como argumento, de seguida efetua uma procura, recorrendo à regra *solucoes*, pelos invariantes associados a esse *Termo* agrupando os resultados numa *Lista.* O próximo passo é efetuar a remoção do *Termo* da base de conhecimento através da regra *remocao* e finalmente efetuar o teste à *Lista.* Esta implementação traduz-se em PROLOG da seguinte forma:

```
%------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a involucao do conhecimento

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

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

Figura 48: Involução

TESTE AO INVARIANTE

Esta regra tem como objetivo efetuar testes ao invariante presente na lista fornecida como argumento. Na figura 49 está representada a sua implementação.

```
%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Testes ao invariante: Lista -> {V,F}

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

Figura 49: Teste ao invariante

ANÁLISE DE RESULTADOS

Nesta secção apresentamos o resultado das diversas funcionalidades implementadas no projeto. Cada funcionalidade representada foi aplicada á base de conhecimento presente neste relatório, como forma de demonstrar a consistência entre os resultados obtidos e o pretendido.

TOTAL VACINADOS

De seguida conseguimos ver o resultado obtido a partir da funcionalidade *totalVacinados*, onde aparecem todos as pessoas totalmente vacinadas, ou seja, com a primeira e segunda tomas efetuadas tendo em conta a BC carregada.

```
?- totalVacinados(L).
L = [(3,josefa),(13,aline correia rodrigues)].
```

Figura 50: Resultado *totalVacinados*

TOTAL NÃO VACINADOS

Na figura seguinte está representado o resultado obtido da funcionalidade *totalNaoVacinados*. No seu resultado é apresentada uma lista de toda a gente que ainda não se encontra totalmente vacinada, ou seja, que não tomou nenhuma vacina.

```
?- totalNaoVacinados(L).
L = [(1,alice),(2,paulo),(4,tiago),(6,alice almeida rodrigues),(7,miguel carvalho),
     (9,pedro albino),(10,albertina maria),(11,rita abreu),(12,alberto andrade)].
```

Figura 51: Resultado *totalNaoVacinados*

VACINAS INDEVIDAS

Tendo em conta os critérios de vacinação indevida anteriormente referidos e analisando o *output* da figura 52 verifica-se que, para a utente 8 a vacinação foi indevida pois a utente pertence à fase 3 e foi vacinada na fase 1. Para a utente 3 esta foi vacinada sem ter respeitado o espaçamento de 28 dias entre a toma das doses. Finalmente, as utentes 15 e 13 foram consideradas indevidas porque cancelaram a toma das doses e posteriormente foram vacinadas.

```
?- vacinasIndevidas(L).
L = [(8,diana batista),(3,josefa),(15,Ana Tomas),(13,aline correia rodrigues)].
```

Figura 52: Resultado *vacinasIndevidas*

UTENTES CANDIDATOS

De modo a obter as pessoas candidatas a uma determinada fase foi necessário criar a funcionalidade *utentesCandidatos*. Assim sendo, nas seguintes figuras 53 a 55 estão apresentados os resultados das invocações desta funcionalidade para as diversas fases. No caso da utente 6, esta está incluída na primeira fase pois, tal como o utente 7, exercem profissões mais suscetíveis de contraírem o covid. Também o utente 9, que foi diagnosticado com doença coronária, e o utente 12, que tem mais de 80 anos, ficam incluídos nesta fase. Já na segunda fase, estão incluídas as utentes 10 e 11, pois estão inseridas nas categorias de doenças de risco e de maiores de 65 anos, respetivamente. Finalmente para a terceira fase, apenas sobram os utentes 2 e 4, visto que são os únicos que não cumprem com nenhum critério especial estabelecido para as fases anteriores.

```
?- utentesCandidatos(1,L).
L = [(alice almeida rodrigues,6),(miguel carvalho,7),(pedro albino,9),(alberto andrade,12)].
```

Figura 53: Resultado *utentesCandidatos* fase 1

```
?- utentesCandidatos(2,L).
L = [(albertina maria,10),(rita abreu,11)].
```

Figura 54: Resultado *utentesCandidatos* fase 2

```
?- utentesCandidatos(3,L).
L = [(paulo,2),(tiago,4)].
```

Figura 55: Resultado *utentesCandidatos* fase 3

UTENTES SEGUNDA TOMA

Com o resultado desta funcionalidade, apresentado na figura abaixo, conseguimos ter acesso a informação dos utentes que ainda necessitam de tomar a segunda toma.

```
?- utentesSegundaToma(L).
L = [(juliana,5),(diana batista,8),(Roberto Miguel,14),(Patricia Filipa,16),(Joana Sousa,17)].
```

Figura 56: Resultado *segundaToma*

SISTEMA DE INFERÊNCIA

Nas figuras abaixo encontra-se representado o funcionamento do sistema de inferência, denominado por *si*, através do uso da regra *utentesSegundaToma.* Como podemos ver pelas imagens, quando aplicamos um predicado com valor lógico falso, figura 58, o sistema de inferência consegue de facto inferir que esse é o valor lógico. O funcionamento é análogo para o valor logico verdadeiro, figura 57. Podemos ainda observar que o sistema é consistente na medida em que se infere corretamente que o valor lógico é falso, irá também inferir que é incorreto o valor lógico para o mesmo predicado ser verdadeiro.

```
?- si(utentesSegundaToma([(juliana,5),('diana batista',8),('Roberto Miguel',14),
     ('Patricia Filipa',16),('Joana Sousa',17)]),verdadeiro).
true .
```

Figura 57: Sistema Inferência – *utentesSegundaToma*

```
?- si(utentesCandidatos(1,[('alice almeida rodrigues',6)]),falso).
true.
```

Figura 58: Sistema de Inferência - *utentesCandidatos* 1

```
?- si(utentesCandidatos(1,[('alice almeida rodrigues',6)]),verdadeiro).
false.
```

Figura 59: Sistema de Inferência - *utentesCandidatos* 2

VACINAS NUM CENTRO DE SAÚDE

Nas figuras seguintes encontra-se o *output* da funcionalidade *vacinasCentro* que determina, para cada centro de saúde, as vacinas que foram administradas aos seus utentes. Em particular, na figura 60, a utente 3 e a utente 13 estão representadas duas vezes uma vez que ambas receberam as duas doses da vacina.

```
?- vacinasCentro(1,L).
L = [(josefa,3),(josefa,3),(juliana,5),(diana batista,8),(aline correia rodrigues,13),
     (aline correia rodrigues,13),(Roberto Miguel,14),(Ana Tomas,15)].
```

Figura 60: *vacinasCentro 1*

```
?- vacinasCentro(2,L).
L = [(Patricia Filipa,16),(Joana Sousa,17)].
```

Figura 61: *vacinasCentro* 2

NÚMERO DE VACINAS NUM CENTRO DE SAÚDE

Na figura 62, encontra-se o resultado da funcionalidade *nrVacinasCentro* que determina o número de vacinas que foram administradas no respetivo centro de saúde. Podemos confirmar que o resultado se encontra de acordo com os dados presentes na BC referentes a *vacinacao_Covid*.

```
?- nrVacinasCentro(1,N).
N = 8.

?- nrVacinasCentro(2,N).
N = 2.
```

Figura 62: *nrVacinasCentro*

VACINAS CANCELADAS

Na imagem abaixo encontra-se o resultado da funcionalidade *vacinasCanceladas* que lista os utentes que cancelaram vacinas. Recorrendo à BC é possível confirmar este resultado a partir da observação do predicado *cancela_vacina*.

```
?- vacinasCanceladas(L).
L = [(Ana Tomas,15),(alice,1),(aline correia rodrigues,13)].
```

Figura 63: *vacinasCanceladas*

EVOLUÇÃO

Na imagem seguinte temos o resultado da evolução do conhecimento *staff*. Como podemos ver quando adicionamos um novo predicado *staff* este fica inserido na BC. Contudo, quando tentamos adicionar um novo *staff* cujo Id já existe, o sistema de evolução devolve uma mensagem de erro, isto é, *false*. Analogamente, este funcionamento é aplicado aos restantes invariantes, mas com critérios de inserção distintos para além do Id único, tal como já foi previamente mencionado.

```
?- findall(Id,staff(Id,_,_,_),L).
L = [1, 2, 3, 4].

?- evolucao(staff(5,1, 'Roberto Miguel','robMig@gmail.com')).
true .

?- findall(Id,staff(Id,_,_,_),L).
L = [1, 2, 3, 4, 5].

?- evolucao(staff(5,1, 'Roberto Miguel','robMig@gmail.com')).
false.

?- findall(Id,staff(Id,_,_,_),L).
L = [1, 2, 3, 4, 5].
```

Figura 64: Evolução - Análise de Resultados

INVOLUÇÃO

Para a involução do conhecimento, o funcionamento e resultado é semelhante à evolução. Quando tentamos retirar um predicado inexistente na BC então o sistema de involução devolve uma mensagem de erro. Quando o predicado a remover é existente na BC então o sistema de involução retira esse predicado da BC. Na figura 65, está um exemplo ilustrativo da involução do conhecimento para o predicado *cancela_vacina*.

```
?- findall(Id,cancela_vacina(Id,_),L).
L = [15,1].

?- involucao(cancela_vacina(15,data(2021,1,9))).
true .

?- findall(Id,cancela_vacina(Id,_),L).
L = [1].

?- involucao(cancela_vacina(15,data(2021,1,9))).
false.

?- findall(Id,cancela_vacina(Id,_),L).
L = [1].
```

Figura 65: Involução - Análise de Resultados

CONCLUSÃO E SUGESTÕES

Dada por concluída a 1ª fase do projeto, consideramos relevante efetuar uma análise crítica do trabalho realizado, apontando aspetos positivos e negativos, resumindo os resultados finais e dando ênfase a possíveis evoluções futuras do trabalho.

Notamos que existiram aspetos positivos a realçar, entre eles a existência de diversas funcionalidades extras que complementam o sistema, o controlo da informação existente através de invariantes e a possibilidade de efetuar adição e remoção de conhecimento.

No que toca às dificuldades sentidas, consideramos que a maior de todas foi associar os tipos de dados do PROLOG, nomeadamente datas e listas. No entanto, apesar de ter requirido atenção extra, revelou-se um problema ultrapassado.

Consideramos que o resultado final se revelou bastante satisfatório dado que alberga todas as funcionalidades propostas e ainda vários extras. Numa versão futura do trabalho, poderemos ainda implementar um leque abrangente de funcionalidades adicionais, como por exemplo guardar registos de utentes que foram contaminados e suas implicações no sistema ou até mesmo impor um limite mínimo de idade para a vacinação.

Para concluir, consideramos que houve um balanço positivo do trabalho realizado dado que as dificuldades sentidas foram superadas e obtivemos um sistema completo de acordo com o proposto.

REFERÊNCIA

REFERÊNCIAS BIBLIOGRÁFICAS

[Analide, 2010] ANALIDE, NEVES, José,
"Representação de informação completa",
Documento pedagógico, Departamento de Informática, Universidade do Minho, Portugal, 2010.

[Analide, 2011] ANALIDE, Cesar, Novais, Paulo, Neves, José,
"Sugestões para a Elaboração de Relatórios",
Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.

REFERÊNCIAS ELETRÓNICAS

Anon., 2021. *Vacinação.* [Online]
Available at: https://covid19.min-saude.pt/vacinacao/
[Acedido em Abril 2021].

LISTA DE SIGLAS E ACRÓNIMOS

BC Base de Conhecimento

DGS Direção Geral de Saúde

SNS Serviço Nacional de Saúde

SI Sistema de Inferência

PMF Pressuposto Mundo Fechado