



UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

TRABALHO PRÁTICO – Fase 1

Programação em Lógica

Sistemas de Representação de Conhecimento e Raciocínio
2º Semestre – 2020/21

Braga, abril de 2021

Autores:

(Grupo 1)

Ana Filipa Pereira A89589

Carolina Santejo A89500

Raquel Costa A89464

Sara Marques A89477

RESUMO

O presente relatório é relativo à primeira fase do trabalho prático da cadeira de Sistemas de Representação de Conhecimento e Raciocínio, que consistiu na resolução de um exercício relacionado com a área de vacinação de utentes durante a pandemia.

Este documento pretende apresentar de forma clara e objetiva todas as decisões tomadas para resolver cada uma das funcionalidades, as soluções obtidas, bem como a justificação dessas mesmas decisões.

É de realçar que além das funcionalidades exigidas no enunciado, o grupo adicionou algumas funcionalidades extra que considerou pertinente.

Todo o projeto foi desenvolvido utilizando a linguagem *Prolog*.

Índice

Introdução	7
Principais Objetivos	7
Preliminares	8
Arquitetura do Trabalho	9
Descrição do Trabalho e Análise dos Resultados	10
Base de Conhecimento.....	10
I. Utente	10
II. Centro de Saúde	11
III. Staff Médico	11
IV. Vacinações Covid.....	11
Adição e Remoção de Conhecimento	12
Evolução do Sistema.....	12
I. Utentes	13
II. Centros de Saúde	14
III. Staff	15
IV. Vacinação Covid	15
Predicados Auxiliares	17
I. Predicado <i>soluções</i>	17
II. Predicado <i>dataValida</i>	17
III. Predicado <i>elementosNaoComuns</i>	18
IV. Predicado <i>nomeUtentes</i>	18
V. Predicado <i>nomeCS</i>	18
VI. Predicado <i>sort</i>	19
VII. Predicado <i>nao</i>	19
VIII. Predicado <i>comprimento</i>	19
IX. Predicado <i>pertence</i>	20
X. Predicado <i>listaAtoms</i>	20
Funcionalidades Básicas	21
I. Definição das Fases de vacinação	21
II. Identificação de pessoas não vacinadas	23
III. Identificação de pessoas vacinadas.....	23
IV. Identificação de pessoas vacinadas indevidamente	24

V.	Identificação de pessoas não vacinadas que são candidatas a vacinação.....	25
VI.	Identificação de pessoas a quem falta a segunda toma da vacina	27
VII.	Desenvolvimento de um sistema de inferência	27
	Funcionalidades Adicionais	28
I.	Determinar todos os utentes que não tomaram nem a 1ª nem a 2ª dose da vacina	28
II.	Determinar o(s) membro(s) do staff com mais vacinas aplicadas	29
III.	Determinar quais os Centros de Saúde que permitiram vacinações indevidas.....	30
IV.	Determinar o nº de vacinações em cada mês.....	31
	Main/Menu	33
	Conclusão	35
	Bibliografia	36

Índice de Figuras

Figura 1 - Diagrama estrutural	9
Figura 2 - Predicados representativos dos Utentes	10
Figura 3 - Predicados representativos dos Centros de Saúde	11
Figura 4 - Predicados representativos do staff	11
Figura 5 - Predicados representativos das vacinações	11
Figura 6 - Predicado evolução	12
Figura 7 - Predicado teste + insere	12
Figura 8 - Predicado involução + remove	13
Figura 9 - Invariantes Utente	14
Figura 10 - Invariantes Centros de Saúde	14
Figura 11 - Invariantes Staff	15
Figura 12 - Invariantes Vacinação Covid	16
Figura 13 - Predicado solucoes	17
Figura 14 - Predicado dataValida	17
Figura 15 - Predicado auxiliar elementosNaoComuns	18
Figura 16 - Predicado Auxiliar nomeUtentes	18
Figura 17 - Predicado auxiliar nomeCS	18
Figura 18 - Predicado auxiliar nao	19
Figura 19 - Predicado auxiliar comprimeto	19
Figura 20 - Predicado auxiliar pertence	20
Figura 21 - Predicado auxiliar listaAtoms	20
Figura 22 - Predicado faseEmQueFoiVacinado	21
Figura 23 - Predicado faseVacinação	21
Figura 24 - faseDeveSerVacinado	22
Figura 25 - faseCorretaVacinação	22
Figura 26 - idadePrioritaria	22
Figura 27 - utentesNaoVacinados	23
Figura 28 - Resultados II	23
Figura 29 - utentesVacinados	23
Figura 30 - Resultados III	23
Figura 31 - utentesVacinadosIndevidamente	24
Figura 32 - utenteMalVacinado	24
Figura 33 - recebeuSegundaVacina	24
Figura 34 - Resultados IV	25
Figura 35 - UtentesNaoVacinadosCandidatos + verificaSeCandidato	25
Figura 36 - faseAnterior	25
Figura 37 - naoRecebeuPrimeiraVacina	26
Figura 38 - Resultados V - Fase 2	26
Figura 39 - Resultados V - Fase 3	26
Figura 40 - utentesFaltaSegundaDose	27
Figura 41 - Resultados VI	27

Figura 42 - demo	27
Figura 43 - utentesSemQualquerDose	28
Figura 44 - Resultados Extra I	28
Figura 45 - staffMaisVacinaoes + staffVacinaoesCount	29
Figura 46 - Resultados Extra II	30
Figura 47 - cs_Vacinacao_Invalida + cd_onde_foramVacinados	30
Figura 48 - Resultados Extra III	30
Figura 49 - mesesVacinaoes + tuplosComOcur	31
Figura 50 - contadorDeOcur	31
Figura 51 - transformeIntoMonth	32
Figura 52 - Resultados Extra IV	32
Figura 53 - main	33
Figura 54 - action_for	33
Figura 55 - Menu Interface	34

Introdução

Para esta primeira fase do projeto 1 da cadeira de SRCR, foi pedido o desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da vacinação global da população portuguesa no contexto da pandemia atual. Para tal, foram implementadas funcionalidades básicas tais como determinar utentes vacinados, definir fases de vacinação ou ainda determinar pessoas indevidamente vacinadas. É preciso ter em conta, que neste universo existem utentes, staff, centros de saúde e vacinações.

Além disto, e como já foi referido, foram desenvolvidas funcionalidades extra escolhidas pelo grupo.

Principais Objetivos

Este trabalho prático tem como principais objetivos, a consolidação dos conteúdos abordados na aula, bem como a familiarização com a programação em lógica e com a ferramenta *SWI-Prolog*.

Além disto, incentiva-se a criatividade do grupo, na medida em que, a partir do enunciado, se valoriza a criação de funcionalidades extras pertinentes no contexto em questão.

Preliminares

Para a realização desta primeira fase apenas é utilizado os valores lógicos verdadeiro ou falso pelo que foi necessário ter conta alguns princípios tais como:

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

Pressuposto do Mundo Fechado - toda a informação que não existe mencionada na Base de Conhecimento é considerada falsa;

Além disto, para o sucesso do projeto contribuíram os conteúdos lecionadas nas aulas teóricas e práticas de SRCR, a análise cuidada do enunciado, a divisão igualitária de tarefas e a partilha de ideias entre os elementos do grupo revelaram-se fatores bastante úteis.

Arquitetura do Trabalho

O projeto encontra-se dividido, no total, em cinco ficheiros. O *main.pl* trata-se daquele que contém um menu com todas as funcionalidades disponíveis pelo nosso sistema, de forma que o utilizador possa interagir de forma mais intuitiva com o mesmo. Este ficheiro importa o *funcionalidades.pl*, que por sua vez enquadra os predicados de consulta de todas as funcionalidades solicitadas, e ainda, algumas adicionais.

O ficheiro *baseDeConhecimento.pl*, tal como o nome indica, é aquele que tem a base de conhecimento do nosso sistema de representação de conhecimento e raciocínio. O *predicadosAuxiliares.pl* contém todos os predicados que foram usados como auxílio à elaboração e desenvolvimento das várias funcionalidades, e os respetivos predicados. Em relação ao *invariantes.pl*, este abrange todos os invariantes construídos para a adição e remoção de conhecimento.

De notar que estes últimos três ficheiros são englobados pelo *funcionalidades.pl*.

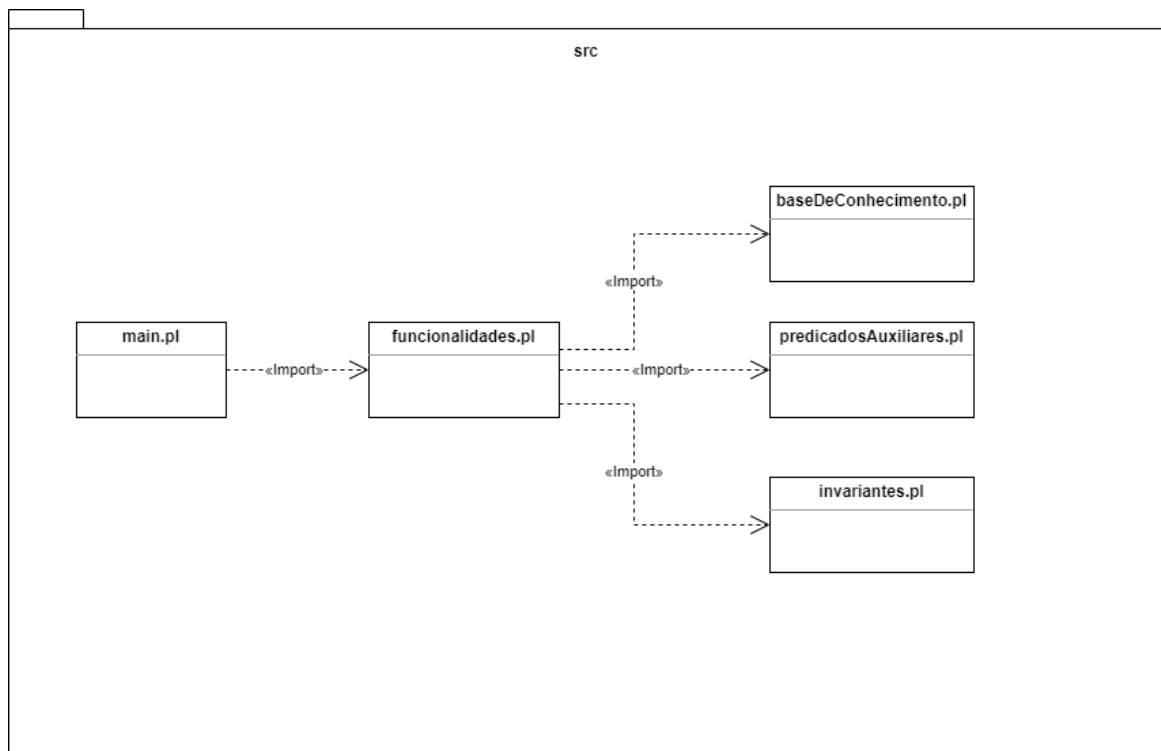


Figura 1 - Diagrama estrutural

Descrição do Trabalho e Análise dos Resultados

Base de Conhecimento

O sistema de representação de conhecimento e raciocínio construído ao longo desta fase do projeto, pretende retratar o contexto pandémico atual, mais concretamente, a implementação do plano de vacinação global da população portuguesa.

Para tal, foi considerado que o conhecimento é dado da seguinte forma:

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

I. Utente

Cada utente é caracterizado por um número identificador único, o seu número de segurança social, nome, data de nascimento, e-mail, número de telemóvel, morada, profissão, uma lista de doenças crónicas, que deverá ser vazia para aqueles que não têm doenças desse tipo, e ainda com o identificador do Centro de Saúde onde está inscrito.

```
%-----Utentes-----
utente(1,123456,'Etelvyna',dataValida(26,1,1958),'etelvyna@gmail.com',912342352,'Rua de Cima','Medico',['Insuficiência cardíaca'],1).
utente(2,123457,'Filipa',dataValida(7,7,2000),'filipa@gmail.com',926775817,'Rua de Cima','Estudante',[],2).
utente(3,123458,'Carolina',dataValida(14,12,1947),'nao tem',25334141,'Rua de Baixo','Reformado',['Doença coronária'],3).
utente(4,123459,'Raquel',dataValida(13,6,2001),'kelinhaMegusta@gmail.com',926775814,'Rua da Esquerda','Vendedor',['Colesterol elevado'],4).
utente(5,123467,'Sara',dataValida(29,1,1998),'sarocaLaroca@gmail.com',916759213,'Rua da Direita','SuperModel',[],5).

utente(6,1234564,'Mario',dataValida(12,9,1979),'mario@gmail.com',922342354,'Rua da Manteiga','Enfermeiro',[],5).
utente(7,1234574,'Possidonio',dataValida(1,1,1930),'nao tem',936775117,'Rua da Direita','Reformado',['Hipertensão arterial','Diabetes'],4).
utente(8,1234584,'Páscoa',dataValida(14,11,1947),'coelhinha_pascoa@gmail.com',25324541,'Rua Professora Doutora Etelvina','Reformado',['Diabetes'],3).
utente(9,1234594,'Anastácia',dataValida(3,3,2007),'anastacia@gmail.com',916755834,'Rua 26 de Abril','Estudante',[],1).
utente(10,1234674,'Justino',dataValida(9,10,1925),'nao tem',916648913,'Rua da Cabreira','Reformado',['Diabetes','DPOC','Insuficiência cardíaca'],2).
utente(11,1234594,'Domingos',dataValida(13,6,1959),'domingos_Adora_Segundas@gmail.com',911771814,'Rua Imaculado Jesus','Auxiliar de Saude',[],1).
utente(12,12346754,'Aurora',dataValida(5,8,1990),'aurora@gmail.com',92664913,'Rua Azul','Policial',['Obesidade'],1).

utente(13,132346754,'Paulo',dataValida(6,9,1969),'nos_bosques@gmail.com',92664913,'Rua Ativa','Guarda Florestal',[],1).
```

Figura 2 - Predicados representativos dos Utentes

II. Centro de Saúde

Um centro de saúde é representado através de um identificador único, além do seu nome, morada onde se situa, número de telefone e, ainda, e-mail.

```
%-----Centros de Saúde-----
centro_saude(1,'Santa Maria','Rua da Estrela',253214998,'santamaria@gmail.com').
centro_saude(2,'Centro Regional de Monção','Rua da Estrela',253214998,'monçãoCS@gmail.com').
centro_saude(3,'Imaculado Cristo','Rua da Estrela',253214998,'iCristo@gmail.com').
centro_saude(4,'Centro Professor Doutor Afonso Henriques','Rua dos Mouros',252234595,'afonso_MataTudo@gmail.com').
centro_saude(5,'Estrela Brilhante','Rua da Estrela',251214990,'estrela_Brilhante@gmail.com').
```

Figura 3 - Predicados representativos dos Centros de Saúde

III. Staff Médico

No caso do staff dos centros de saúde, cada um dos trabalhadores é caracterizado por identificador único, pelo identificador do centro de saúde onde trabalha, e ainda por informações pessoais como o seu nome e e-mail.

```
%-----Staff-----
staff(1,1,'Antonio','antonio@gmail.com').
staff(2,2,'Maria','maria@gmail.com').
staff(3,3,'Fátima','fatima@gmail.com').
staff(4,4,'Josefa','josefa@gmail.com').
staff(5,5,'Francisco','francisco@gmail.com').
staff(6,5,'Domingos','domingos@gmail.com').
staff(7,1,'Manuel','manuel@gmail.com').
```

Figura 4 - Predicados representativos do staff

IV. Vacinações Covid

Uma vacinação é definida através do identificador do membro do staff que a administrou, o identificador do utente que a recebeu, a data da toma, o nome da vacina, e ainda se foi a 1ª ou a 2ª dose do utente.

```
%-----Vacinações-----
vacinacao_Covid(1,10,dataValida(17,2,2021),'Astrazeneca',1).
vacinacao_Covid(5,10,dataValida(20,2,2021),'Astrazeneca',2).

vacinacao_Covid(2,1,dataValida(4,12,2020),'Pfizer',1).
vacinacao_Covid(4,1,dataValida(28,12,2020),'Pfizer',2).

vacinacao_Covid(4,11,dataValida(22,12,2020),'Pfizer',1).
vacinacao_Covid(5,11,dataValida(1,1,2021),'Pfizer',2).

vacinacao_Covid(2,8,dataValida(2,4,2021),'Pfizer',1).
vacinacao_Covid(6,3,dataValida(1,2,2021),'Astrazeneca',1).

vacinacao_Covid(7,6,dataValida(21,12,2020),'Pfizer',1).
vacinacao_Covid(2,6,dataValida(30,12,2020),'Pfizer',2).

vacinacao_Covid(5,12,dataValida(2,3,2021),'Pfizer',1). %Vacinada tarde demais
vacinacao_Covid(1,7,dataValida(1,4,2021),'Astrazeneca',1). %Vacinada tarde demais
vacinacao_Covid(3,2,dataValida(21,2,2021),'Pfizer',1). %Vacina Cedou demais
```

Figura 5 - Predicados representativos das vacinações

Adição e Remoção de Conhecimento

A adição e remoção de conhecimento é feita a partir de invariantes que especificam um conjunto de restrições que devem ser sempre verdadeiras após qualquer uma destas operações. Deste modo, sempre que algo é inserido ou removido da base de conhecimento, a consistência do sistema é verificada através dos invariantes e, se se registar algum erro ou anomalia, este irá reverter para o estado anterior à operação em questão.

É de notar que nestes invariantes foram utilizados alguns predicados auxiliares.

Evolução do Sistema

O processo de evolução do sistema consiste em um aumento de conhecimento no sistema.

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

Figura 6-Predicado evolução

```
teste([]).  
teste([H | T]) :- H, teste(T).  
  
%-----  
% Extensao do predicado insere: Termo -> {V, F}  
  
insere(Termo) :- assert(Termo).  
insere(Termo) :- retract(Termo), !, fail.
```

Figura 7 - Predicado teste + insere

O objetivo deste predicado consiste em inserir um termo na Base do Conhecimento. Para isso, insere-se o conhecimento, utilizando o predicado *insere* que por sua vez recorre ao *assert*, que é um predicado nativo do *Prolog*. Além disto, armazena-se em “S” todos os invariantes relativos ao termo em questão. Após isto, faz-se um teste à consistência do sistema, usando o predicado *teste*, para verificar se o termo inserido respeita os respetivos invariantes.

Involução do Sistema

O processo de involução do sistema é o inverso do processo de evolução e consiste na remoção de um termo da Base de Conhecimento.

```
%-----  
% Extensao do predicado involucao: Termo -> {V, F}  
  
involucao(Termo) :- Termo,  
                  solucoes(Inv, -Termo::Inv, S),  
                  remove(Termo),  
                  teste(S).  
  
%-----  
% Extensao do predicado remove: Termo -> {V, F}  
  
remove(Termo) :- retract(Termo).  
remove(Termo) :- assert(Termo), !, fail.
```

Figura 8-Predicado involução + remove

Desta forma, verifica-se primeiro se o termo de encontra na Base de Conhecimento. Se não estiver, então o processo de involução falha. Caso esteja, armazena-se em “S” todos os invariantes relativos à remoção desse termo, seguindo-se da sua remoção efetiva (usando o predicado remove que por sua vez usa o *retract*, predicado nativo do *Prolog*). Após a remoção, recorre-se ao predicado *teste*, de forma a verificar a consistência do sistema.

I. Utentes

A inserção de um novo utente na base de conhecimento deve respeitar os seguintes critérios:

- Cada utente deve ter um ID único, representado por um inteiro;
- O contato telefónico e o NIF têm de ser do tipo inteiro;
- O nome, email, morada e profissão terão de ser do *tipo atom*, ou seja, entre plicas ou estar iniciado com letra minúscula;
- A lista de doenças crónicas, tem de ser uma lista de *atoms*;
- Cada utente deve estar registado num centro de saúde já incluído na base de conhecimento, identificado por um ID único (também representado por um inteiro);
- Cada utente deve ter um NIF único;
- Cada utente deve ter uma data de nascimento válida.

Por sua vez, um utente não poderá ser removido da base de conhecimento quando existem registos de vacinações que envolvam o mesmo.

Os invariantes que traduzem todas estas restrições serão, portanto:

```
+utente(Id,SegurancaSocial,Nome,DataNasc,Email,Telefone,Morada,Profissao,ListaD,IdCentroSaude) :: (
    integer(Id),
    integer(SegurancaSocial),
    integer(Telefone),
    integer(IdCentroSaude),
    dataValida(DataNasc),
    atom(Nome),
    atom(Email),
    atom(Morada),
    atom(Profissao),
    listaAtoms(ListaD),
    solucoes(Id, utente(Id,_,_,_,_,_,_,_,_), S),
    comprimento(S, N),
    N == 1,
    solucoes(SegurancaSocial, utente(_,SegurancaSocial,_,_,_,_,_,_,_), S1),
    comprimento(S1, N1),
    N1 == 1,
    solucoes(IdCentroSaude, centro_saude(IdCentroSaude, _, _, _), S2),
    comprimento(S2, N2),
    N2 == 1
).

-utente(Id,_,_,_,_,_,_,_,_)::(
    solucoes(Id,vacinacao_Covid(_,Id,_,_),S),
    comprimento(S,N),
    N == 0
).
```

Figura 9 - Invariantes Utente

II. Centros de Saúde

A inserção de um novo centro de saúde na base de conhecimento só poderá ocorrer quando este apresenta um ID inteiro único, um contato telefónico inteiro, e um nome, morada e email do tipo *atom*.

Por sua vez, um centro de saúde só poderá ser removido se não existir nenhum utente ou membro da *staff* registado com o seu ID na base de conhecimento.

Tendo estes critérios em conta, os invariantes serão os seguintes:

```
+centro_saude(Id,Nome,Morada,Telefone,Email) :: (
    integer(Id),
    integer(Telefone),
    atom(Nome),
    atom(Morada),
    atom(Email),
    solucoes(Id,centro_saude(Id,_,_,_,_),S),
    comprimento(S,N),
    N==1
).

-centro_saude(Id,_,_,_,_) :: (
    solucoes(Id,staff(_,Id,_,_),S),
    comprimento(S,N),
    N==0
    solucoes(Id,utente(_,_,_,_,_,_,_,_,Id),S1),
    comprimento(S1,N1),
    N1==0
).
```

Figura 10 - Invariantes Centros de Saude

III. Staff

Um membro do *staff* de um centro de saúde só poderá ser inserido na base de conhecimento se possuir um ID único e o ID de um centro de saúde já incluído na base de conhecimento (sendo ambos os IDs representados por inteiros). Além disto, o nome e email têm de ser do tipo *atom*.

Por sua vez, só poderá ser removido da base de conhecimento se o seu ID não estiver associado a nenhuma vacinação.

Logo, os invariantes para estes critérios serão:

```
+staff(IdStaff,IdCentroSaude,Nome,Email) :: (  
    integer(IdStaff),  
    integer(IdCentroSaude),  
    atom(Nome),  
    atom(Email),  
    solucoes(IdStaff,staff(IdStaff,_,_),S),  
    comprimento(S,N),  
    N==1,  
    solucoes(IdCentroSaude,centro_saude(IdCentroSaude,_,_),S1),  
    comprimento(S1,N1),  
    N1==1  
).  
  
-staff(Id,_,_,_) :: (  
    solucoes(Id,vacinacao_Covid(Id,_,_,_),S),  
    comprimento(S,N),  
    N==0  
).
```

Figura 11 - Invariantes Staff

IV. Vacinação Covid

Para adicionar o registo de uma vacinação na base de conhecimento, deverão cumprir-se os seguintes critérios:

- O registo da vacinação deverá incluir os IDs (representados por inteiros) do utente e do membro do *staff* do centro de saúde envolvidos na vacinação (sendo que ambos terão de já estar inseridos na base de conhecimento);
- A vacinação deverá ocorrer numa data válida;
- O registo da vacinação deve mencionar qual o nome da vacina usada, representado por um *atom*;
- O registo da vacinação deve mencionar se esta se trata da primeira ou segunda toma da vacina;
- A segunda toma de uma vacina só poderá ocorrer (e ser adicionada à base de conhecimento) se o utente em causa já tiver recebido a primeira toma.
- A segunda dose da vacina, terá de ser da mesma marca que a primeira dose.

Por sua vez, quando um registo de vacinação é removido do sistema, apenas deve deixar de haver conhecimento do predicado.

Deste modo, os invariantes serão:

```
+vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma) :: (
    Toma == 2,
    integer(IdStaff),
    integer(IdUtente),
    dataValida(Data),
    atom(Vacina),
    solucoes(IdStaff,staff(IdStaff,_,_),S),
    comprimento(S,N),
    N==1,
    solucoes(IdUtente,utente(IdUtente,_,_,_,_,_,_),S1),
    comprimento(S1,N1),
    N1==1,
    solucoes((IdStaff,IdUtente,Data,Vacina,Toma),vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma),S2),
    comprimento(S2,N2),
    N2==1,
    naoRecebeuPrimeiraVacina(IdUtente)),
    solucoes(NomeVacina,vacinacao_Covid(_,IdUtente,_,NomeVacina,1),[H|_]),
    H == Vacina
).

+vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma) :: (
    Toma == 1,
    integer(IdStaff),
    integer(IdUtente),
    dataValida(Data),
    atom(Vacina),
    solucoes(IdStaff,staff(IdStaff,_,_),S),
    comprimento(S,N),
    N==1,
    solucoes(IdUtente,utente(IdUtente,_,_,_,_,_,_),S1),
    comprimento(S1,N1),
    N1==1,
    solucoes((IdStaff,IdUtente,Data,Vacina,Toma),vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma),S2),
    comprimento(S2,N2),
    N2==1
).

-vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma)::(
    solucoes((IdStaff,IdUtente,Data,Vacina,Toma),vacinacao_Covid(IdStaff,IdUtente,Data,Vacina,Toma),S),
    comprimento(S,N),
    N==0
).
```

Figura 12 - Invariantes Vacinacao Covid

Predicados Auxiliares

Antes de passar para a explicação das funcionalidades propriamente ditas, iremos apresentar alguns predicados auxiliares que são usados regularmente no projeto, e que foram cruciais para a resolução do mesmo.

I. Predicado *solucoes*

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

Figura 13 - Predicado solucoes

O predicado *solucoes* usa o predicado *findall* do *Prolog* e tem como objetivo procurar todas as ocorrências de “F” que satisfaçam “Q”, colocando, posteriormente, os resultados obtidos na lista “S”.

II. Predicado *dataValida*

```
dataValida(D, M, A) :-  
    A >= 1900,  
    pertence(M, [1,3,5,7,8,10,12]),  
    D >= 1,  
    D <= 31.  
dataValida(D, M, A) :-  
    A >= 1900,  
    pertence(M, [4,6,9,11]),  
    D >= 1,  
    D <= 30.  
dataValida(D, 2, A) :-  
    A >= 1900,  
    A mod 4 \= 0,  
    D >= 1,  
    D <= 28.  
dataValida(D, 2, A) :-  
    A >= 1900,  
    A mod 4 =:= 0,  
    D >= 1,  
    D <= 29.  
dataValida(dataValida(D, M, A)) :- dataValida(D, M, A).
```

Figura 14 - Predicado dataValida

O predicado *dataValida* recebe com argumentos 3 variáveis “D”, “M”, “A” que representam, respetivamente, o dia, mês e ano da data em questão. Para o predicado ser verdadeiro, foi necessário impor algumas condições para garantir a coerência, ou seja, impedir que o dia seja inválido em relação ao mês (por exemplo 31 de junho) e impedir o dia 29 de fevereiro quando o ano não é bissexto. Por outro lado, não se aceitam datas com ano anterior a 1900, uma vez que estas não são realistas.

III. Predicado *elementosNaoComuns*

```
elementosNaoComuns([],_,[]).  
elementosNaoComuns([H|T],S,L):- pertence(H,S),  
                                elementosNaoComuns(T,S,L).  
  
elementosNaoComuns([H|T],S,L):- nao(pertence(H,S)),  
                                elementosNaoComuns(T,S,Z),  
                                L = [H|Z].
```

Figura 15 - Predicado auxiliar elementosNaoComuns

O predicado *elementosNaoComuns*, recebe como argumentos as variáveis “([H|T])”, “S” e “L” que representam listas de elementos. O objetivo consiste em colocar em “L”, todos os elementos que estão em “([H|T])” e não em “S”, e vice-versa, ou seja, determinar os elementos que não são comuns entre ([H|T]) e S.

IV. Predicado *nomeUtentes*

```
nomeUtentes([],[]).  
nomeUtentes([H|T],L):- solucoes((H,Nome),utente(H,_,Nome,_,_,_,_,_),[S|_]),  
                        nomeUtentes(T,Z),  
                        L = [S|Z].
```

Figura 16 - Predicado Auxiliar nomeUtentes

O predicado *nomeUtentes* recebe como argumentos as variáveis “([H|T])” e “L”, que são listas, sendo “([H|T])” uma lista de *ids* de utentes. O objetivo consiste, em colocar em “L”, a partir dos *ids* passados, tuplos com a seguinte forma (IdUtente, NomeUtente).

V. Predicado *nomeCS*

```
nomeCS([],[]).  
nomeCS([H|T],L):- solucoes((H,Nome),centro_saude(H,Nome,_,_,_),[S|_]),  
                  nomeCS(T,Z),  
                  L = [S|Z].
```

Figura 17 - Predicado auxiliar nomeCS

Este predicado possui exatamente o mesmo objetivo que o predicado *nomeUtentes*, no entanto, é aplicado aos centros de saúde.

VI. Predicado *sort*

Este predicado é nativo do *Prolog*. Recebe 4 argumentos sendo que o primeiro é o inteiro 0 e indica que a lista pode conter elementos de qualquer tipo (inteiros, *strings*, variáveis, etc...), o segundo indica que a lista deve ser ordenada por ordem crescente e retirando os elementos repetidos, o terceiro é a lista que queremos ordenar e o último argumento é a lista onde se colocará o resultado. É de realçar que este predicado *sort*, se receber uma lista “L” cujo elementos sejam pares, por exemplo (*id*, nome), ordena os tuplos pela chave.

Além disto, existe o predicado *sort(X, L)*, que simplesmente ordena a lista “X” sem repetidos e coloca o resultado em “L”.

VII. Predicado *nao*

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

Figura 18 - Predicado auxiliar *nao*

Este predicado recebe um termo “T” como argumento, sendo que “nega” o seu valor lógico. Se o valor de “T” for *true* o *nao(T)* será *false*, mas se “T” for *false*, *nao(T)* será *true*.

VIII. Predicado *comprimento*

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

Figura 19 - Predicado auxiliar *comprimento*

Este predicado recebe como argumentos uma lista “S” e uma variável “N”, sendo que o objetivo consiste em calcular o comprimento de “S”, e guardar o valor em “N”. Para isto, usa-se o predicado *length*, já disponibilizado pelo *Prolog*.

IX. Predicado *pertence*

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

Figura 20 - Predicado auxiliar *pertence*

Este predicado recebe como primeiro argumento uma variável *X* que representa um elemento qualquer, e como segundo argumento uma lista na forma *[H|T]* em que *H* é a variável como o primeiro elemento da lista é *T* é a variável que contém a lista com os elementos da cauda. O objetivo deste predicado consiste em verificar se “*X*” está presente na lista dada.

X. Predicado *listaAtoms*

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

Figura 21 - Predicado auxiliar *listaAtoms*

Este predicado recebe como argumento uma variável que é lista. O objetivo é verificar se todos os elementos dessa lista são *atoms*. O predicado *atom* é nativo do *Prolog*.

Funcionalidades Básicas

I. Definição das Fases de vacinação

A partir dos seguintes predicados, foi possível definir as fases corretas de vacinação para cada utente, e ainda estabelecer os critérios de inclusão em cada uma delas. Para tal, tivemos como base o plano de vacinação disponibilizado pela DGS atualmente, onde são definidos grupos prioritários, por estarem mais vulneráveis à COVID-19.

Algumas das especificações que se consideraram para a definição dos critérios para verificar se um utente está apto a uma certa fase foram, a idade, a profissão, e eventuais doenças crónicas que este possa ter.

Além disso, foi considerado que cada fase tem uma data inicial e final, e cada utente que pertence a uma dada fase deve ser vacinado, tanto na 1ª como na 2ª dose, dentro desse intervalo definido. Todos aqueles que forem vacinados em fases diferentes, são considerados utentes que foram vacinados indevidamente.

O predicado *faseEmQueFoiVacinadoDose1*, recebe o ID do utente que nós queremos consultar, para tal usamos o predicado auxiliar *solucoes* de modo a verificar qual a data em que foi administrada a 1ª dose da vacina ao utente em questão, e ainda foi utilizado o predicado *faseVacinacao*. Este último irá consultar a qual das fases existentes corresponde a data obtida (Figura 23). Caso a data esteja entre 1/12/2020 ou 31/01/2021, então trata-se da fase “1A”. Se for entre 01/02/2021 ou 31/03/2021 então é a fase “1B”. No caso de a data abranger os meses de abril (/04/2021), maio (/05/2021) e junho (/06/2021), então trata-se da fase “2” do plano de vacinação, todas as datas posteriores a estas estão integradas na fase “3”. O mesmo raciocínio aplica-se ao predicado *faseEmQueFoiVacinado2*.

```
faseEmQueFoiVacinadoDose1(IdUtente,X):- solucoes(Data,vacinacao_Covid(_,IdUtente,Data,_,1),[H|_]),
                                         faseVacinacao(H,X).

faseEmQueFoiVacinadoDose2(IdUtente,X):- solucoes(Data,vacinacao_Covid(_,IdUtente,Data,_,2),[H|_]),
                                         faseVacinacao(H,X).
```

Figura 22 - Predicado faseEmQueFoiVacinado

```
faseVacinacao(dataValida(_,M,A),"1A"):- A = 2020, M = 12.
faseVacinacao(dataValida(_,M,A),"1A"):- A = 2021, M = 1.

faseVacinacao(dataValida(_,M,A),"1B"):- A = 2021, M = 2.
faseVacinacao(dataValida(_,M,A),"1B"):- A = 2021, M = 3.

faseVacinacao(dataValida(_,M,A),"2") :- A = 2021, M = 4.
faseVacinacao(dataValida(_,M,A),"2") :- A = 2021, M = 5.
faseVacinacao(dataValida(_,M,A),"2") :- A = 2021, M = 6.

faseVacinacao(dataValida(_,M,A),"3"):- A >= 2021, M > 6.
```

Figura 23 - Predicado faseVacinacao

O predicado *faseDeveSerVacinado* (Figura 24) analisa todas as condições do utente (através do seu ID) necessárias para incluí-lo numa das fases do plano de vacinação. Para tal usa-se o predicado *solucoes*, de forma a obter a data de nascimento, a profissão e as doenças que o utente possa ter. Depois faz-se uso do predicado *faseCorretaVacinação* (Figura 25) para concluir em qual fase é que o utente deverá ser incluído.

Caso a profissão do utente seja uma das seguintes: Médico, Enfermeiro, Polícia, Bombeiro ou Auxiliar de Saúde, então este deverá ser vacinado durante a fase “1A” do plano proposto.

No caso da fase “1B”, esta abarca todos os utentes, cuja idade seja superior a 80 anos, para tal aplica-se o predicado *idadePrioritaria* (Figura 26 **Erro! A origem da referência não foi encontrada.**), que dada uma data de nascimento e uma idade mínima X, irá verificar se o utente que nasceu nessa mesma data tem uma idade superior a X no ano corrente (2021). Além disso, todos os utentes com uma idade superior a 50 anos e que tenham uma das seguintes doenças: Insuficiência cardíaca, Insuficiência renal, DPOC ou então uma doença coronária, deverão ser vacinados nesta mesma fase.

Em relação à fase “2” do plano de vacinação, todos aqueles que tenham uma idade superior a 65 anos deverão ser vacinados, e ainda, as pessoas que tenham entre 50 e 65 anos que padeçam de uma ou mais doenças crónicas.

Todos os utentes que não se enquadram nestes critérios serão seleccionados na fase “3” do plano de vacinação delineado.

```
faseDeveSerVacinado(IdUtente,Y):- solucoes((IdUtente,Data,Prof,Doencas),utente(IdUtente,_,_,Data,_,_,_,Prof,Doencas,_),[H1|_]),
                                     faseCorretaVacinação(H1,Y).
```

Figura 24 - faseDeveSerVacinado

```
faseCorretaVacinação(_,_,Prof,_,"1A"):- Prof == 'Medico'.
faseCorretaVacinação(_,_,Prof,_,"1A"):- Prof == 'Enfermeiro'.
faseCorretaVacinação(_,_,Prof,_,"1A"):- Prof == 'Policia'.
faseCorretaVacinação(_,_,Prof,_,"1A"):- Prof == 'Bombeiro'.
faseCorretaVacinação(_,_,Prof,_,"1A"):- Prof == 'Auxiliar de Saude'.

faseCorretaVacinação(_,DataNasc,_,_,,"1B"):- idadePrioritaria(DataNasc,80).

faseCorretaVacinação(_,DataNasc,_,L,"1B"):- idadePrioritaria(DataNasc,50), pertence('Insuficiência cardíaca',L).
faseCorretaVacinação(_,DataNasc,_,L,"1B"):- idadePrioritaria(DataNasc,50), pertence('Doença coronária',L).
faseCorretaVacinação(_,DataNasc,_,L,"1B"):- idadePrioritaria(DataNasc,50), pertence('Insuficiência renal',L).
faseCorretaVacinação(_,DataNasc,_,L,"1B"):- idadePrioritaria(DataNasc,50), pertence('DPOC',L).

faseCorretaVacinação(_,DataNasc,_,_,,"2"):- idadePrioritaria(DataNasc,65).

faseCorretaVacinação(Id,DataNasc,_,_,,"2"):- idadePrioritaria(DataNasc,50),
                                     nao(idadePrioritaria(DataNasc,65)),
                                     nao(semDoencas(Id)).

faseCorretaVacinação(_,,"3").
```

Figura 25 - faseCorretaVacinação

```
idadePrioritaria(DataNasc,X):- getAno(DataNasc,Ano),
                               diferenca(2021,Ano,Dif),
                               Dif >= X.
```

Figura 26 - idadePrioritaria

II. Identificação de pessoas não vacinadas

Todos os utentes não vacinados são aqueles que não tomaram nenhuma dose da vacina ou então que só tomaram a primeira.

```
utentesNaoVacinados(L):- utentesVacinados(Z),
                           solucoes((Utente,Nome),utente(Utente,_,Nome,_,_,_,_,_),S),
                           elementosNaoComuns(S,Z,X1),
                           sort(0,@<,X1,L).
```

Figura 27 - utentesNaoVacinados

O predicado *utentesNaoVacinados* tem como objetivo comparar a listagem de utentes vacinados, já com a segunda dose, com a lista de utentes existentes. Através do uso do predicado auxiliar *elementosNaoComuns*, obtém-se a lista daqueles que não receberam nenhuma dose ou apenas a primeira da vacina, isto é, todos aqueles utentes que são considerados como não vacinados.

```
[(2,Filipa),(3,Carolina),(4,Raquel),(5,Sara),(7,Possidonio),(8,Páscoa),(9,Anastácia),(12,Aurora),(13,Paulo)]
```

Figura 28 - Resultados II

III. Identificação de pessoas vacinadas

Neste caso, os utentes vacinados são todos aqueles que tomaram a primeira e a segunda dose da vacina.

```
utentesVacinados(L):- solucoes(Utente,vacinacao_Covid(_,Utente,_,_,2),S),
                        nomeUtentes(S,X),
                        sort(0,@<,X,L).
```

Figura 29 - utentesVacinados

Para obter a listagem dos utentes vacinados foi desenvolvido o predicado *utentesVacinados*, que irá devolver, com o auxílio dos predicados auxiliares *solucoes* e *nomeUtentes*, os nomes das pessoas que têm uma vacinação correspondente à 2ª toma da vacina com o seu identificador. É de salientar ainda que, as pessoas que já tomaram a segunda dose da vacina, obrigatoriamente já tiveram acesso à primeira, e já lhes foi administrada.

```
[(1,Etelvyna),(6,Mario),(10,Justino),(11,Domingos)]
```

Figura 30 - Resultados III

IV. Identificação de pessoas vacinadas indevidamente

Indivíduos vacinados indevidamente são todos aqueles que tomaram a primeira ou a segunda dose da vacina na fase errada.

```
%Extensão do predicado utentesVacinadosIndevidamente: L -> {V,F}
%Funcionalidade 3

utesVacinadosIndevidamente(L):- solucoes(Utente,vacinacao_Covid(_,Utente,_,1),S),
                                utentesVacinadosIndevidamenteAux(S,X),
                                nomeUtentes(X,X1),
                                sort(0,@<,X1,L).

%-----
%Extensão do predicado utentesVacinadosIndevidamente: L,L1 ->{V,F}

utesVacinadosIndevidamenteAux([],[]).
utesVacinadosIndevidamenteAux([H|T],L):- (utenteMalVacinado(H) -> utentesVacinadosIndevidamenteAux(T,Z), L = [H|Z] ;
                                           utentesVacinadosIndevidamenteAux(T,L) ).
```

Figura 31 - utentesVacinadosIndevidamente

```
utenteMalVacinado(IdUtente):- recebeuSegundaVacina(IdUtente),
                               faseEmQueFoiVacinadoDose2(IdUtente,X),
                               faseDeveSerVacinado(IdUtente,Y),!,
                               X \== Y.

utenteMalVacinado(IdUtente):- recebeuSegundaVacina(IdUtente),
                               faseEmQueFoiVacinadoDose1(IdUtente,X1),
                               faseDeveSerVacinado(IdUtente,Y1),!,
                               X1 \== Y1.

utenteMalVacinado(IdUtente):- nao(recebeuSegundaVacina(IdUtente)),
                               faseEmQueFoiVacinadoDose1(IdUtente,X),
                               faseDeveSerVacinado(IdUtente,Y),!,
                               X \== Y.
```

Figura 32 - utenteMalVacinado

```
recebeuSegundaVacina(IdUtente):- solucoes(Data,vacinacao_Covid(_,IdUtente,Data,2),S),
                                  comprimento(S,N),
                                  N == 1.
```

Figura 33 - recebeuSegundaVacina

Para os identificar, começamos por utilizar o predicado *solucoes* de forma a colocar na lista “S” os *ids* de todos os utentes que tomaram pelo menos a primeira dose da vacina. De seguida, passamos essa lista como argumento do predicado *utesVacinadosIndevidamenteAux*, cujo objetivo é colocar numa lista “L”, todos os *ids* dos utentes mal vacinados.

Para determinar se um utente está mal vacinado, recorre-se ao predicado *utenteMalVacinado*. Neste predicado, quando uma pessoa já recebeu a segunda dose (descobre-se usando o predicado *recebeuSegundaVacina*), verifica-se se a fase em que tomou uma das doses é diferente da fase em que devia ter tomado. Quando a pessoa apenas tomou a primeira

dose, verifica-se se a tomou na fase correta. Qualquer indivíduo que não tenha tomado nenhuma vacina nunca é considerado indevidamente vacinado.

Por fim, tendo a lista dos *ids* de utentes desejados, determina-se os seus nomes, e ordena-se a lista por ordem crescente de *id*.

```
[(2,Filipa),(7,Possidonio),(12,Aurora)]
```

Figura 34 - Resultados IV

V. Identificação de pessoas não vacinadas que são candidatas a vacinação

Para a resolução desta funcionalidade o grupo decidiu que qualquer pessoa que não tenha tomado a vacina na fase em que devia é automaticamente candidato às fases posteriores e que um indivíduo vacinado numa fase anterior à que devia só poderá ser candidato à segunda dose, na fase em que deve ser vacinado.

```

utilitarios:- solucoes(Utente,utente(Utente,_,_,_),S),
              solucoes(U,vacinacao_Covid(_,U,_),S1),
              elementosNaoComuns(S,S1,Z),
              verificaSeCandidato(Z,Fase,K),!,
              nomeUtentes(K,K1),
              sort(0,@<,K1,L).

%-----
verificaSeCandidato([],_,[]).

verificaSeCandidato([H|T],Fase,L):- utenteMalVacinado(H),
                                     faseDeveSerVacinado(H,Y),
                                     faseAnterior(Fase,ListaFases),
                                     (pertence(Y,ListaFases) -> verificaSeCandidato(T,Fase,X), L = [H|X]; verificaSeCandidato(T,Fase,L)).

verificaSeCandidato([H|T],Fase,L):- naoRecebeuPrimeiraVacina(H),
                                     verificaSeCandidato(T,Fase,X),
                                     L = [H|X].

verificaSeCandidato([H|T],Fase,L):- faseAnterior(Fase,ListaFases),
                                     faseDeveSerVacinado(H,Y),
                                     (pertence(Y,ListaFases) -> verificaSeCandidato(T,Fase,X), L = [H|X];verificaSeCandidato(T,Fase,L) ).

```

Figura 35 - UtentesNaoVacinadosCandidatos + verificaSeCandidato

```
faseAnterior("3",X):- X = ["1A","1B","2","3"].
faseAnterior("2",X):- X = ["1A","1B","2"].
faseAnterior("1B",X):- X = ["1A","1B"].
faseAnterior("1A",X):- X = ["1A"].
```

Figura 36 - faseAnterior

```

naoRecebeuPrimeiraVacina(IdUtente):- solucoes(Data,vacinacao_Covid(_,IdUtente,Data,_,1),S),
                                     comprimento(S,N),
                                     N == 0.

```

Figura 37 - naoRecebeuPrimeiraVacina

O predicado *utentesNaoVacinadosCandidatos* recebe como argumentos a variável lista “L” que será onde o resultado será colocado, e a variável Fase que representa a fase atual e na qual queremos saber quem tem direito a ser vacinado.

Primeiro começa-se por determinar os *ids* de utentes presentes na base do conhecimento (resultado colocado na variável “S”) e os *ids* dos utentes que já levaram a segunda dose (resultado colocado na variável “S1”). Os elementos não comuns entre “S” e “S1” são os *ids* de utentes que não levaram nenhuma vacina ou que apenas levaram a primeira dose, e este resultado colocado na variável “Z”. A variável “Z” e a fase são passadas como argumentos do predicado *verificaSeCandidato*, que tem como objetivo colocar numa variável “K” o *id* dos candidatos a essa fase. Para isso, percorremos a lista “Z” e para cada *id* verificamos se o utente foi mal vacinado. Se tomou a primeira dose (verificado pelo predicado *nao(naoRecebeuPrimeiraDose)*) e foi indevidamente vacinado, ele só pode ser adicionado à lista se a fase em que deve ser vacinado for igual ou anterior à fase atual (passada como argumento). Se a primeira dose foi tomada corretamente, então ele será sempre candidato. Por exemplo, se a fase atual for a 2 e um individuo tomou corretamente a primeira dose na fase 1 então ele é candidato. Se tomou na fase 2, então continuam a ser candidato. As fases anteriores de uma dada fase “F” são determinadas pelo predicado *faseAnterior*.

Além disto, e ainda no predicado *verificaSeCandidato*, para um utente que ainda não tenha recebido qualquer dose da vacina, é necessário verificar se a fase na qual deve ser vacinado corresponde à atual. Se sim, então o seu *id* é adicionado à lista.

Tendo-se obtido a lista “K”, determina-se o nome dos utentes nela presentes, e ordena-se a lista por ordem crescente do *idUtente*.

```

Indique a fase ("1A". ou "1B". ou "2". ou "3".)
|: "2".
[(3,Carolina),(7,Possidonio),(8,Páscoa),(12,Aurora)]

```

Figura 38 - Resultados V - Fase 2

```

Indique a fase ("1A". ou "1B". ou "2". ou "3".)
|: "3".
[(2,Filipa),(3,Carolina),(4,Raquel),(5,Sara),(7,Possidonio),(8,Páscoa),(9,Anastácia),(12,Aurora),(13,Paulo)]

```

Figura 39 - Resultados V - Fase 3

VI. Identificação de pessoas a quem falta a segunda toma da vacina

O seguinte predicado tem como objetivo listar todos os utentes que ainda não tomaram a segunda dose da vacina, ou seja, aqueles a quem já foi administrada a primeira dose da vacina, mas que a segunda dose ainda se encontra por administrar.

```
utentesFaltaSegundaDose(L):- utentesVacinados(S),
                             solucoes(Utente,vacinacao_Covid(_,Utente,_,1),S1),
                             nomeUtentes(S1,X),
                             elementosNaoComuns(X,S,X1),
                             sort(0,@<,X1,L).
```

Figura 40 - utentesFaltaSegundaDose

Primeiramente, foi necessário consultar a listagem de utentes vacinados, e ainda apurar a listagem de utentes que já tomaram a primeira dose da vacina através do predicado auxiliar *solucoes*. De seguida, foi usado o predicado *elementosNaoComuns* de forma a comparar as duas listas obtidas: *X* que contém os tuplos (*id,nome*) dos utentes que receberam a primeira dose da vacina, e *S* que representa a lista de todos os utentes vacinados. Desta forma iremos obter a lista *X1* com todos os utentes que receberam a primeira dose mas que ainda não são considerados como vacinados porque ainda lhes falta tomar a segunda dose da vacina.

```
[(2,Filipa),(3,Carolina),(7,Possidonio),(8,Páscoa),(12,Aurora)]
```

Figura 41 - Resultados VI

VII. Desenvolvimento de um sistema de inferência

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

Figura 42 - demo

O desenvolvimento de um sistema de inferência é feito através do predicado *demo* que avalia qualquer termo de acordo com as restrições impostas. É de realçar que nesta primeira fase, um termo apenas pode ter valor lógico verdadeiro ou falso.

Funcionalidades Adicionais

Dada a diversidade do tema deste trabalho prático, o grupo decidiu implementar algumas funcionalidades extra que considerou pertinentes. No entanto, salientamos que isto são apenas alguns exemplos e que poderiam ser escolhidas muita outras funcionalidades interessantes.

I. Determinar todos os utentes que não tomaram nem a 1ª nem a 2ª dose da vacina

Como já foi referido anteriormente, considerámos que os utentes vacinados são aqueles que levaram as duas doses das vacinas e os não vacinados aqueles que tomaram a primeira dose ou nenhuma. Desta forma, o grupo considerou necessário existir uma funcionalidade que determina todos os utentes que não levaram qualquer vacina.

```
utenteSemQualquerDose(L):- solucoes(Utente,utente(Utente,_,_,_,_,_,_,_,_),S),
                           solucoes(U,vacinacao_Covid(_,U,_,_,1),S1),
                           elementosNaoComuns(S,S1,Z),
                           nomeUtentes(Z,X1),
                           sort(0,@<,X1,L).
```

Figura 43 - utentesSemQualquerDose

Este predicado coloca na lista “S” o *id* de todos os utentes na base do conhecimento e em “S1”, o *id* de todos os utentes que tomaram, no mínimo a primeira dose. Realçamos que no nosso universo, qualquer utente que tenha a segunda dose da vacina, tem de ter a primeira tomada também.

Assim, determinando os elementos não comuns entre “S” e “S1”, e usando o predicado *nomeUtentes* para determinar o nome dos indivíduos, obtemos o resultado desejado, que é colocado em “L”. Usa-se o *sort* para ordenar a lista por número de *id*.

```
[(4,Raquel),(5,Sara),(9,Anastácia),(13,Paulo)]
```

Figura 44 - Resultados Extra I

II. Determinar o(s) membro(s) do staff com mais vacinas aplicadas

```
%-----  
%Extensão do predicado staffMaisVacinaoes: X,L ->{V,F}  
%Lista do(s) elemento(s) do Staff com mais vacinaoes e respetiva frequencia  
%Funcionalidade extra 2  
  
staffMaisVacinaoes(Max,IdNomes):- solucoes(IdStaff,vacinacao_Covid(IdStaff,_,_,_),L),  
    sort(0,@<,L,Ls),  
    staffMaisVacinaoesAux(Max,Ls,IdMaxL),  
    nomeStaff(IdMaxL,IdNomes).  
  
staffMaisVacinaoesAux(0,[],[]).  
staffMaisVacinaoesAux(Max,[IdStaff],[IdStaff]):- staffVacinaoesCount(IdStaff,C),  
    Max = C.  
  
staffMaisVacinaoesAux(Max,[IdStaff|T],L):- staffVacinaoesCount(IdStaff,Count),  
    staffMaisVacinaoesAux(Max,T,Ls),  
    (Count > CountMax -> Max = Count, L = [IdStaff];  
    Count == CountMax -> Max = CountMax, L = [IdStaff|Ls]; Max = CountMax, L = Ls).  
  
%-----  
%Extensão do predicado staffVacinaoesCount: Id,C ->{V,F}  
%Conta o numero de vacinaoes por um elemento do staff  
  
staffVacinaoesCount(IdStaff,X):- solucoes(IdStaff,vacinacao_Covid(IdStaff,_,_,_),L),  
    comprimento(L,X).
```

Figura 45 - staffMaisVacinaoes + staffVacinaoesCount

O predicado *staffMaisVacinaoes* recebe como argumento duas variáveis *Max* e *IdNomes* onde serão colocados respetivamente, o número de vacinações e a lista de pares (Id, Nome) do(s) elemento(s) do *staff* com mais vacinas aplicadas.

Em primeiro lugar, é calculada a lista dos ids de todos os elementos do *staff* que vacinaram, e é feito um *sort* para ordená-los e retirar os repetidos (lista *Ls*). De seguida, é chamado o predicado auxiliar *staffMaisVacinaoesAux* que recebe a lista *Ls* e as variáveis referidas acima *Max* e *IdNomes*. Esta função chama também o predicado auxiliar *staffVacinaoesCount* que para um *IdStaff*, coloca em *L* todas as ocorrências desse *Id* na *vacinacao_Covid*. O comprimento desta lista *L* é assim colocado na variável *X* uma vez que representa a frequência de vacinações desse mesmo *Id*.

Retornando agora ao *staffMaisVacinaoesAux*, este predicado para cada *Id* da lista *Ls* recebida, calcula o seu número de vacinas aplicadas (*staffVacinaoesCount*)(variável *Count*) e compara com o valor máximo já calculado na recursividade (*CountMax*), sendo que se for maior substitui-se a lista de ids máximos existente (*Ls*) pela singular [*IdStaff*] e o valor *Max* pelo *Count* e, no caso de serem iguais em vez de substituir, adiciona-se o elemento à lista. Caso contrário, mantém-se os valores máximos já calculados (*CountMax* e *Ls*).

Se a lista recebida tiver apenas um elemento essa será a lista a retornar e o valor de *Max* é o número de ocorrências do respetivo *Id*. No caso de a lista ser vazia toma-se como valor de *Max* 0 e retorna-se a lista recebida.

```
Membro do staff: [(2,Maria),(5,Francisco)]  
Nº de vacinas: 3
```

Figura 46 - Resultados Extra II

III. Determinar quais os Centros de Saúde que permitiram vacinações indevidas

```
cs_Vacinacao_Invalida(L):- solucoes(Utente,vacinacao_Covid(_,Utente,_,1),S),  
                           utentesVacinadosIndevidamenteAux(S,X1),  
                           cs_onde_foramVacinados(X1,G),  
                           sort(0,@<,G,G1),  
                           nomeCS(G1,L).  
  
cs_onde_foramVacinados([],[]).  
cs_onde_foramVacinados([Utente|T],L):- solucoes(Staff,vacinacao_Covid(Staff,Utente,_,_),[S|_]),  
                                       solucoes(Centro,staff(S,Centro,_,_),[S1|_]),  
                                       cs_onde_foramVacinados(T,G),  
                                       L = [S1|G].
```

Figura 47 - cs_Vacinacao_Invalida + cd_onde_foramVacinados

O predicado *cs_Vacinacao_Invalida* recebe como argumento uma lista “L” que será preenchida com o *id* e nome de todos os centros de saúde que permitiram vacinações indevidas.

Realçamos que o grupo considerou que o centro de saúde onde a pessoa está registada, pode não ter sido o local onde foi vacinada. Assim, sendo foi preciso determinar os utentes vacinados indevidamente, pelo que se usou os predicados *solucoes* e *utentesVacinadosIndevidamenteAux* (já explicado anteriormente). Tendo isto, utiliza-se o predicado *cs_onde_foramVacinados* para determinar o membro do staff que efetuou a vacinação e de seguida o centro de saúde onde este trabalha, colocando-se o *id* da instituição numa lista passada como segundo argumento.

Assim sendo, retiram-se os elementos repetidos, pois podemos ter várias pessoas a ser vacinadas no mesmo local, e utiliza-se o predicado *nomeCS* para ir buscar o nome dos centros de saúde.

```
[(1,Santa Maria),(3,Imaculado Cristo),(5,Estrela Brilhante)]
```

Figura 48 - Resultados Extra III

IV. Determinar o nº de vacinações em cada mês

```
mesesVacinaoes(L):- solucoes((M,A),vacinacao_Covid(_,_,dataValida(_,M,A),_,_),S),
                      sort(0,@<,S,M1),
                      transformeIntoMonth(S,S1),
                      transformeIntoMonth(M1,M),
                      tuplosComOcur(M,S1,L).

tuplosComOcur([],_,[]).
tuplosComOcur([(M,A)|T],Y,L):- contadorDeOcur(Y,(M,A),Aux),
                                tuplosComOcur(T,Y,D),
                                L = [(M,A,Aux)|D].
```

Figura 49 - mesesVacinaoes + tuplosComOcur

```
contadorDeOcur([],_,0).
contadorDeOcur([X|T],X,Y):- contadorDeOcur(T,X,Z), Y is 1+Z.
contadorDeOcur([X1|T],X,Z):- X1\=X,contadorDeOcur(T,X,Z).
```

Figura 50 - contadorDeOcur

Para este predicado, é necessário, em primeiro lugar, determinar os meses e os anos nos quais foram aplicadas vacinas, e guardar os pares numa lista “S”. De seguida usamos o predicado *sort* para organizar os pares por ordem numérica crescente do mês e para também retirar os elementos repetidos. Este resultado fica armazenado na variável “M1”. É usado um predicado auxiliar, *transformeIntoMonth*, que recebe uma lista de pares do tipo (mês,ano) na forma numérica e preenche uma lista “L” com os pares dos meses e dos anos, mas com os meses escritos por extenso. Após isto, utiliza-se outro predicado auxiliar *tuplosComOcur*, que recebe 3 variáveis, sendo elas “X”, “Y”, “Z”, que são listas. O objetivo é colocar em Z tuplos com 3 elementos: o mês, o ano, e o número de vacinas nessa data. O cálculo do número de ocorrências de um elemento numa lista, é feito pelo predicado *contadorDeOcur*.

Realçamos que a passagem dos meses em formato numérico para o formato texto, é feita simplesmente por ser mais intuitivo e fácil de interpretar pelo utilizador.

```

transformeIntoMonth([],[]).

transformeIntoMonth([(H,A)|T],L):- H == 1,
                                   transformeIntoMonth(T,X),
                                   L = [("Janeiro",A)|X].

transformeIntoMonth([(H,A)|T],L):- H == 2,
                                   transformeIntoMonth(T,X),
                                   L = [("Fevereiro",A)|X].

transformeIntoMonth([(H,A)|T],L):- H == 3,
                                   transformeIntoMonth(T,X),
                                   L = [("Março",A)|X].
transformeIntoMonth([(H,A)|T],L):- H == 4,
                                   transformeIntoMonth(T,X),
                                   L = [("Abril",A)|X].
transformeIntoMonth([(H,A)|T],L):- H == 5,
                                   transformeIntoMonth(T,X),
                                   L = [("Maio",A)|X].
transformeIntoMonth([(H,A)|T],L):- H == 6,
                                   transformeIntoMonth(T,X),
                                   L = [("Junho",A)|X].
transformeIntoMonth([(H,A)|T],L):- H == 7,
                                   transformeIntoMonth(T,X),
                                   L = [("Julho",A)|X].
transformeIntoMonth([(H,A)|T],L):- H == 8,
                                   transformeIntoMonth(T,X),
                                   L = [("Agosto",A)|X].
transformeIntoMonth([(H,A)|T],L):- H == 9,
                                   transformeIntoMonth(T,X),
                                   L = [("Setembro",A)|X].
transformeIntoMonth([(H,A)|T],L):- H == 10,
                                   transformeIntoMonth(T,X),
                                   L = [("Outubro",A)|X].
transformeIntoMonth([(H,A)|T],L):- H == 11,
                                   transformeIntoMonth(T,X),
                                   L = [("Novembro",A)|X].
transformeIntoMonth([(H,A)|T],L):- H == 12,
                                   transformeIntoMonth(T,X),
                                   L = [("Dezembro",A)|X].

```

Figura 51 - transformeIntoMonth

```

[(Janeiro,2021,1),(Fevereiro,2021,4),(Março,2021,1),(Abril,2021,2),(Dezembro,2020,5)]

```

Figura 52 - Resultados Extra IV

Main/Menu

No ficheiro *main.pl* foi desenvolvido um predicado *main*, responsável por criar um “loop” de forma a apresentar um menu com várias funcionalidades que o utilizador poderá escolher e executar. Existem 9 funcionalidades no total a dispor do utilizador, a opção “0.” será para o caso de o utilizador querer sair do “loop” /menu.

```
main :-
    repeat,
    nl,nl,
    write('-----MENU-----'), nl,
    write('1. Consultar utentes vacinados'), nl,
    write('2. Consultar utentes não vacinados, com 0 ou 1 toma da vacina'), nl,
    write('3. Consultar utentes vacinados indevidamente'), nl,
    write('4. Consultar utentes não vacinados e que são candidatos à vacinação'), nl,
    write('5. Consultar utentes a quem falta a segunda toma da vacina'), nl,
    write('6. Consultar utentes que não tomaram qualquer dose da vacina'), nl,
    write('7. Determinar o nº de vacinações em cada mês do ano'), nl,
    write('8. Determinar centros de saúde que permitiram vacinações indevidas'), nl,
    write('9. Consultar o membro do staff com mais vacinas aplicadas'), nl,
    write('0. Exit'), nl,nl,
    write('Choose : '),
    read(Z),
    ( Z = 0 -> !, fail ; true ),
    nl,nl,
    action_for(Z),
    fail.
```

Figura 53 - main.

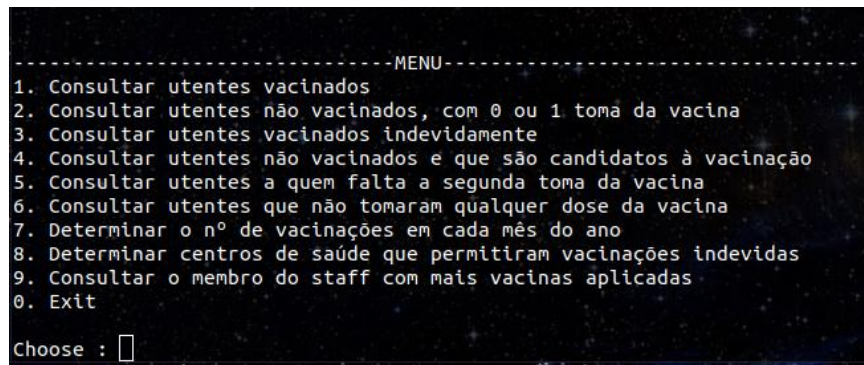
A partir do predicado *action_for*, consoante a escolha que o utilizador introduzir, é feita uma ação, ou seja, é executado um dos predicados que representam as funcionalidades do nosso sistema.

```
action_for(3) :-
    utentesVacinadosIndevidamente(L),
    write(L), nl.

action_for(4) :-
    write('Indique a fase ("1A". ou "1B". ou "2". ou "3".)'),!,nl,
    read(Fase),
    utentesNaoVacinadosCandidatos(L,Fase),
    write(L), nl.
```

Figura 54 - action_for

A interface obtida através da implementação destes predicados anteriores foi a seguinte:



```
-----MENU-----
1. Consultar utentes vacinados
2. Consultar utentes não vacinados, com 0 ou 1 tomã da vacina
3. Consultar utentes vacinados indevidamente
4. Consultar utentes não vacinados e que são candidatos à vacinação
5. Consultar utentes a quem falta a segunda toma da vacina
6. Consultar utentes que não tomaram qualquer dose da vacina
7. Determinar o nº de vacinações em cada mês do ano
8. Determinar centros de saúde que permitiram vacinações indevidas
9. Consultar o membro do staff com mais vacinas aplicadas
0. Exit

Choose : 
```

Figura 55 - Menu Interface

Desta forma, foi possível construir algo esteticamente mais agradável, e ainda fornecer ao utilizador uma opção mais intuitiva de explorar o nosso sistema e as suas funcionalidades.

Conclusão

A realização desta primeira fase do trabalho prático, permitiu ao grupo não só consolidar o conhecimento adquirido ao longo das aulas teóricas e práticas, mas também praticar a programação em lógica.

Consideramos que concluímos esta fase com sucesso, na medida em que implementamos todas as funcionalidades obrigatórias e alguns extras, tendo-se obtido em todos os casos, a solução esperada.

Como sugestão de melhoria para trabalhos futuros, salienta-se que poderá haver funcionalidades com formas mais eficientes de implementação do que aquelas apresentadas neste relatório. Além disto, e devido á diversidade do tema do trabalho, poderiam ter sido desenvolvidas mais funcionalidades adicionais.

Bibliografia

Bratko, I. (1986). *Prolog: Programming for Artificial Intelligence*.

C. A., & J. N. (2016). Documento Pedagógico. *Representação de Informação Incompleta* .

Vacinação Covid-19. (s.d.). Obtido de covid19 - DGS: <https://covid19.min-saude.pt/vacinacao/>