Sistemas de Representação de Conhecimento e Raciocínio

Trabalho realizado por:

João Figueiredo Martins Peixe dos Santos

Francisco Alves Andrade

Luís Filipe Cruz Sobral

Paulo Silva Sousa

A89520 João Santos

A89474 Luís Sobral

A89465 Paulo Sousa

A89513 Francisco Andrade

Grupo 14
Projeto SRCR
2020/2021
Universidade do Minho

Conteúdo

Índice de Figuras

1 Introdução

No âmbito da Unidade Curricular de Sistemas de Reconhecimento de Conhecimento e Raciocínio foi-nos proposto o desenvolvimento de um sistema de representação de conhecimento e raciocínio capaz de caracterizar um universo de discurso na área de vacinação global da população portuguesa no contexto da pandemia que atualmente vivemos.

Este sistema implementará várias funcionalidades de modo a que o universo anteriormente referido esteja bem caracterizado, como a identificação de pessoas vacinadas e não vacinadas.

Para a contrução dos mecanismos de raciocínio e representação de conhecimento será utilizada a linguagem de programção em lógica PROLOG.

2 Preliminares

Tal como referido anteriormente, este sistema, que pretende representar um universo de discurso na área da vacinação global da população portuguesa no contexto COVID que estamos a viver, existem quatro fontes de conhecimento principais, nas quais se vai basear todo o seu cenario.

Primeiramente temos o utente, a única personagem interveniente neste sistema, à qual está associado um ID, Número da Segurança Social, Nome, Data de nascimento, email, telefone, morada, profissão, lista de doenças e um ID do centro de saúde.

Seguidamente temos o centro de saúde, que tem associado a si um ID, nome, morada, telefone e email.

Posteriormente, temos o staff ao qual está associado um ID, um ID do centro onde trabalha, nome e email.

Por último temos a vacinação covid, que basicamente faz a junção do utente com o staff, ou seja, diz-nos quando, qual vacina e qual toma foi administrada por um determinado staff a um determinado utente. Assim, tem associado a si o ID do utente em questão, o ID do staff que administrou a vacina, o nome da vacina, a data e se é a primeira ou segunda toma. Deste modo, temos as seguintes defniçõoes iniciais:

- utente: (ID Utente, SegurançaSocial, Nome, DataNasc, Email, Telefone, Morada, Profissão, ListaDoencas, CentroSaúde) -> V,F
- centro_saude: (ID Centro, Nome, Morada, Telefone, Email) -> V,F
- staff: (ID staff, ID Centro, Nome, Email) -> V,F
- vacinacao_Covid: (ID staff, ID Utente, Data, Vacina, Toma) -> V,F

Tanto o ID do utente como o do centro de saúde e o do staff têm de ser únicos e identificadores deles mesmos. Em relação à vacinação, visto que estes são a associação entre utentes e staff, é possivel haver para o mesmo ID de staff vários utentes, uma vez que um determinado staff pode administrar uma vacina a vários utentes diferentes, sendo que um utente apenas recebe duas tomas da vacina.

Posto isto, depois de todas estas consideraçõoes iniciais, encontramo-nos habilitados a prosseguir com a elaboração do trabalho propriamente dito.

3 Descrição do Trabalho e Resultados

3.1 Base de conhecimento incial

É necessário começar com uma base de conhecimento inicial, que premita testar a informação sem que esta tenha que ser, anteriormente, inserida. Ao preenchê-la, tivemos em consideração as definições das fontes de conhecimento. Também quisemos abranger casos mais especificos, por exemplo, inserir utentes com os critérios necessários para cada uma das 3 fases de vacinação.

Outro cuidado que tivemos foi na inserção das vacinas Covid. Inserimos utentes sem vacina, com só a primeira toma ou com as duas tomas da vacina. Testamos, também, o caso de ocorrer vacinação inválida, ou seja, ao utente ter sido administrada apenas a segunda dose da vacina.

```
% Extensao do predicado utente : ID Utente, Segurança_Social, Nome, Data_Nasc, Email, Telefone, Morada, Profissão, LDoencas, CentroSaúde -> {V,F}
utente(1,21455655,'Luís',1970,'luis@gmail.com',936696454,'Adaúfe','Estudante',['Doença coronária','Doença respiratória'],1).
utente(2,31455655,'Filipe',1969,'filipe@gmail.com',936696100,'Barcelos','Estudante',['Diabetes'],1).
utente(3,41455655,'José',1990,'j@gmail.com',936696151,'Esposende','Estudante',[],1).
utente(4,42455655,'Diogo',1998,'d@gmail.com',936696171,'Aveiro','Estudante',['Doença coronária'],1).
utente(5,43455655,'Rui',1936,'r@gmail.com',936696190,'Braga','Estudante',['Diabetes'],1).
utente(6,41454355,'Paulo',1980,'p@gmail.com',936738461,'Esposende','Medico',[],2).
utente(7,75355655,'Beatriz',1995,'b@gmail.com',936973645,'Aveiro','Enfermeiro',[],2).
utente(8,43455692,'Maria',1951,'m@gmail.com',93718248,'Braga','Estudante',[],2).
utente(9,42487435,'Carlos',1983,'c@gmail.com',936827634,'Aveiro','Estudante',['Arteroses'],2).
```

Figura 1: Conhecimento inicial de Utentes

```
% Extensao do predicado 'centro_saude' : ID Centro, Nome, Morada, Telefone,
% Email -> {V,F}
centro_saude(1,'BragaH','Braga','999999999','bragah@gmail.com').
centro_saude(2,'GuimaraesH','GuimaraesH','999999998','guimaraesh@gmail.com').
```

Figura 2: Conhecimento inicial de Centros de Saúde

```
% Extensao do predicado 'staff' : ID staff, ID Centro, Nome, Email -> {V,F}
staff(1,1,'Luís','luis@gmail.com').
staff(2,2,'Paulo','paulo@gmail.com').
staff(3,1,'Francisco','francisco@gmail.com').
staff(4,2,'João','joao@gmail.com').
```

Figura 3: Conhecimento inicial de Staff

```
% Extensao do predicado 'vacinacao_Covid' : ID staff, ID Utente, Data, Vacina, Toma -> {V,F}
% Utente 1 - 2 vacinas fase 1
vacinacao_Covid(1, 1, 11, 1, 2021, 'AstraZeneca', 1).
vacinacao_Covid(1, 1, 11, 4, 2021, 'AstraZeneca', 2).
vacinacao_Covid(3, 2, 30, 5, 2021, 'Moderna', 1).
vacinacao_Covid(2, 3, 12, 5, 2021, 'PFizer', 1).
vacinacao_Covid(1, 4, 11, 11, 2021, 'Moderna', 1).
vacinacao_Covid(1, 4, 14, 12, 2021, 'Moderna', 2).
vacinacao_Covid(4, 6, 12, 2, 2021, 'PFizer', 1).
vacinacao_Covid(4, 6, 12, 3, 2021, 'PFizer', 2).
vacinacao_Covid(2, 7, 1, 3, 2021, 'AstraZeneca', 1).
```

Figura 4: Conhecimento inicial de Vacinação Covid

3.2 Invariantes

Para o correto funcionamento da nossa base de conhecimento é necessária a implementação de invariantes para o controlo da informação presente na mesma. Assim, implementamos dois tipos de invariantes: invariantes associados à inserção e invariantes associados à remoção de conhecimento.

Em relação aos invariantes associados à inserção, encontramos um conjunto de invariantes que impede a repetição de conhecimento, ou seja, um utente, staff, centro ou vacinação repetido. Para isto, verificamos se após a inserção, no caso do utente, o ID do predicado, número de segurança social, telefone e email são únicos e, caso contrário, remove o mesmo da base de conhecimento.

Para o centro de saúde aferimos o ID, telefone e email, e para o staff, averiguamos o ID e o email.

Relivamente à vacinação, para um dado utente não é permitida a inserção de mais do que uma vacinação para a mesma fase.

```
% Invariante:  nao permite a insercao de conhecimento repetido
+utente(ID,_,_,_,_,_,_,_,_,_)::
    (solucoes(ID,utente(ID,_,_,_,_,_,_,_,_,_),Lista),
    comprimento(Lista,N), N==1).

+utente(_,SS,_,_,_,_,_,_,_,_)::
    (solucoes(SS,utente(_,SS,_,_,_,_,_,_,_,_),Lista),
    comprimento(Lista,N), N==1).

+utente(_,_,_,_,E,_,_,_,_,_)::
    (solucoes(E,utente(_,_,_,_,E,_,_,_,_,_),Lista),
    comprimento(Lista,N), N==1).

+utente(_,_,_,_,_,T,_,_,_,_)::
    (solucoes(T,utente(_,_,_,_,_,T,_,_,_,_),Lista),
    comprimento(Lista,N), N==1).

+centro_saude(ID,_,_,_,_)::
    (solucoes(ID,centro_saude(ID,_,_,_,_),Lista),
    comprimento(Lista,N), N==1).

+centro_saude(_,_,_,T,_)::
    (solucoes(T,centro_saude(_,_,_,T,_),Lista),
    comprimento(Lista,N), N==1).

+centro_saude(_,_,_,_,E)::
    (solucoes(E,centro_saude(_,_,_,_,E),Lista),
    comprimento(Lista,N), N==1).

+staff(ID,_,_,_)::
    (solucoes(ID,staff(ID,_,_,_),Lista),
    comprimento(Lista,N), N==1).

+staff(_,_,_,E)::
    (solucoes(E,staff(_,_,_,E),Lista),
    comprimento(Lista,N), N==1).

+vacinacao_Covid(_,ID_Utente,_,_,_,_,Toma)::
    (solucoes((ID_Utente,Toma),vacinacao_Covid(_,ID_Utente,_,_,_,_,Toma),Lista),
    comprimento(Lista,N), N==1).
```

Figura 5: Invariantes que não permitem a insersão de conhecimento repetido

Adicionalmente, criamos também três invariantes, que garantem que todos os ID inseridos são números naturais.

```
% Invariante: IDs têm que ser números naturais
+utente(ID,_,_,_,_,_,_,_,_,_,_):: natural(ID).
+centro_saude(ID,_,_,_,_):: natural(ID).
+staff(ID,_,_,_):: natural(ID).
```

Figura 6: Invariantes que obrigam os ids a ser números naturais

O outro tipo de invariantes é referente à remoção. Ao remover conhecimento é essencial controlar a existência do mesmo.

```
% Invariante:  nao permite a remoção de conhecimento inexistente
-utente(ID,_,_,_,_,_,_,_,_,_,_)::
    (solucoes(ID,utente(ID,_,_,_,_,_,_,_,_,_,_),Lista),
    comprimento(Lista,N), N==1).

-centro_saude(ID,_,_,_,_)::
    (solucoes(ID,centro_saude(ID,_,_,_,_),Lista),
    comprimento(Lista,N), N==1).

-staff(ID,_,_,_)::
    (solucoes((ID,_,_,_),staff(ID,_,_,_),Lista),
    comprimento(Lista,N), N==1).

-vacinacao_Covid(_,ID_Utente,_,_,_,_,Toma)::
    (solucoes((ID_Utente,Toma),vacinacao_Covid(_,ID_Utente,_,_,_,_,Toma),Lista),
    comprimento(Lista,N), N==1).
```

Figura 7: Invariantes que não permitem a remoção de conhecimento repetido

Para além disso, existem também vários invariantes associados à vacinação contra o covid. A data tem que ser válida, o ID do utente e do staff têm que ser válidos, apenas podem haver duas tomas por utente. Um utente só pode receber a segunda fase de vacinação caso já tenha recebido a primeira.

Em relação ao utente, existe um invariante para verificar se o centro de saúde a que está associado é válido.

```
% Invariante: nao admitir vacinas associadas a uma data inválida
+vacinacao_Covid(ID_Staff,ID_Utente,Dia,Mes,Ano,Vacina,Toma):: data(Ano, Mes, Dia).

% Invariante: nao admitir vacinas associadas um staff ou utente inválido
+vacinacao_Covid(ID_Staff,ID_Utente,Dia,Mes,Ano,Vacina,Toma):: (utente(ID_Utente,Q,No,Da,E,T,M,P,D,C),staff(ID_Staff,IdCentro,Nome,Email)).

% Invariante: nao admitir mais de duas tomas da vacina
+vacinacao_Covid(_,_,_,_,_,_,Toma):: contains(Toma,[1,2]).

% Invariante: nao admitir segunda toma da vacina diferente da primeira
+vacinacao_Covid(_,ID_Utente,_,_,_,Vacina,2):: vacinacao_Covid(_,ID_Utente,_,_,_,Vacina,1).

% Invariante: nao admitir staff e utentes de centros de saude diferentes
+vacinacao_Covid(ID_Staff,ID_Utente,_,_,_,_,_):: mesmoCentro(ID_Staff, ID_Utente).

% Invariante: nao admitir utentes associados um centro de saúde inválido
+utente(_,_,_,_,_,_,_,_,_,_,ID_Centro):: (centro_saude(ID_Centro,_,_,_,_)).

% Invariante: nao admitir utentes com telefone inválido
+utente(_,_,_,_,_,_,Tel,_,_,_,_):: telefone(Tel).

% Invariante: Telefone tem que ser válido
+centro_saude(_,_,_,Tel,_):: telefone(Tel).
```

Figura 8: Restantes Invariantes

Por fim, é crucial manter a consistência da base de conhecimento. Para tal, não deverá ser possivel remover conhecimento que possui referência noutro lado, sendo que poderá acontecer na remoção de um utente ou staff ao qual existe uma vacinação covid associada ou na remoção de um centro de saúde ao qual existem utentes associados.

3.3 Predicados de evolução e involução de conhecimento

Para que o funcionamento da nossa base de conhecimento fosse o esperado foram inseridos uma série de invariantes para controlar o conhecimento que esta iria conter. Desta forma, os invariantes irão garantir que não existirão problemas na remoção ou inserção de informação. Para tal, foram criados os predicados evolução e involução.

```
% Extensao do predicado que permite a evolução do conhecimento
evolucao( Termo ) :-
    solucoes( Invariante,+Termo::Invariante,Lista ),
    insercao( Termo ),
    teste( Lista ).

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

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

Figura 9: Predicado que permite a evolução de conhecimento

```
% Extensao do predicado que permite a involucao do conhecimento
involucao( Termo ) :-
    solucoes( Invariante,-Termo::Invariante,Lista ),
    teste( Lista ),
    remocao( Termo ).


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

Figura 10: Predicado que permite a involução de conhecimento

Estes predicados, no momento de inserção ou remoção, permitem manter a consistência da base de conhecimento testando os invariantes e impedindo qualquer ação caso não se verifiquem.

3.4 Predicados de adição e remoção de conhecimento

3.4.1 Predicados de adição de conhecimento

Os seguintes predicados permitem inserir um utente, staff, centro de saúde ou vacinação covid:

```
registaUtente(ID,Q,Nome,Data,E,T,M,P,D,C) :- evolucao(utente(ID,Q,Nome,Data,E,T,M,P,D,C)).

registaCentro(ID,N,M,T,E) :- evolucao(centro_saude(ID,N,M,T,E)).

registaStaff(ID,IdCentro,Nome,Email) :- evolucao(staff(ID,IdCentro,Nome,Email)).

registaVacina(ID_Staff,ID_Utente,Dia,Mes,Ano,Vacina,Toma) :- evolucao(vacinacao_Covid(ID_Staff,ID_Utente,Dia,Mes,Ano,Vacina,Toma)).
```

Figura 11: Predicados de adição de conhecimento

Estes predicados, basicamente, recebem os parâmetros de um predicado (utente, staff, centro ou vacinação), e redirecionam para o predicado evolucao(T), que insere na base de conhecimento de forma controlada.

3.4.2 Predicados de remoção de conhecimento

Por outro lado, existem também predicados que permitem remover o conhecimento anteriormente inserido.

De modo a remover a informação do utente, utilizamos duas funções que nos permitem remover as vacinas associadas a este.

```
removeUtente(ID) :- vacinasUtente(ID,R), removeVacinasUtente(R), involucao(utente(ID,_,_,_,_,_,_,_,_,_)).

vacinasUtente(ID,R) :- solucoes((ID,Toma), (vacinacao_Covid(_,ID,_,_,_,_,Toma)), R).

removeVacinasUtente([]).
removeVacinasUtente([(ID,Toma)|T]) :- involucao(vacinacao_Covid(_,ID,_,_,_,_,Toma)), removeVacinasUtente(T).
```

Figura 12: Predicados de remoção de conhecimento de um utente

Para remover o conhecimento de um centro, removemos também o conhecimento dos utentes e staff que estão relacionados com este. Por sua vez, removemos também as vacinas associadas a estes utentes e membros do staff.

```
removeCentro(ID) :- staffCentro(ID,L), removeStaffCentro(L), utentesCentro(ID,L2), removeUtentesCentro(L2), involucao(centro_saude(ID,_,_,_,_)).

staffCentro(ID,R) :- solucoes(ID_Staff, (staff(ID_Staff,ID,_,_)), R).

utentesCentro(ID,R) :- solucoes(ID_Utente, (utente(ID_Utente,_,_,_,_,_,_,_,_,ID)), R).

removeStaffCentro([]).
removeStaffCentro([ID|T]) :- vacinasStaff(ID,R), removeVacinasStaff(R), involucao(staff(ID,_,_,_)), removeStaffCentro(T).

removeUtentesCentro([]).
removeUtentesCentro([ID|T]) :- vacinasUtente(ID,R), removeVacinasUtente(R), involucao(utente(ID,_,_,_,_,_,_,_,_,_)), removeUtentesCentro(T).
```

Figura 13: Predicados de remoção de conhecimento de um centro de saúde

De modo a remover a informação de um membro do staff, eliminados também as vacinas associadas a este.

```
removeStaff(ID) :- vacinasStaff(ID,R), removeVacinasStaff(R), involucao(staff(ID,_,_,_)).

vacinasStaff(ID,R) :- solucoes((ID_Utente, Toma), (vacinacao_Covid(ID,ID_Utente,_,_,_,_,Toma)), R).

removeVacinasStaff([]).
removeVacinasStaff([(ID,Toma)|T]) :- involucao(vacinacao_Covid(_,ID,_,_,_,_,Toma)), removeVacinasStaff(T).
```

Figura 14: Predicados de remoção de conhecimento de um membro do staff

Para remover a vacinação utilizamos o seguinte predicado.

```
removeVacina(ID_Utente,Toma) :- involucao(vacinacao_Covid(_,ID_Utente,_,_,_,_,Toma)).
```

Figura 15: Predicados de remoção de conhecimento de uma vacinação

Estes predicados utilizam o predicado involucao(T), que permite remover de forma controlada qualquer conhecimento presente na base de conhecimento.

3.5 Predicados de listagem de conhecimento

3.5.1 Permitir a definição de fases de vacinação

A definição das diferentes fases de vacinação é elaborada pelos predicados seguintes.

Desde logo, esta definição é iniciada através de predicados que permitem a identificação dos vários utentes pertencentes às diversas fases, isto é, identificar os utentes que pertencem à primeira fase, à segunda e à terceira.

```
% Extensão do predicado que permite identificar utentes que pertencem à primeira fase de vacinação
% 'utentes1fase': Resultado -> {V, F}
utentes1fase(R) :- solucoes(ID, (utente(ID,_,_,A,_,_,_,P,L,_), criterios1fase(A,P,L)), L), removeRepetidos(L,R).

% Extensão do predicado que permite identificar utentes que pertencem à segunda fase de vacinação
% 'utentes2fase': Resultado -> {V, F}
utentes2fase(R) :- solucoes(ID, (utente(ID,_,_,A,_,_,_,P,L,_), criterios2fase(A,P,L)), L), removeRepetidos(L,R).

% Extensão do predicado que permite identificar utentes que pertencem à terceira fase de vacinação
% 'utentes3fase': Resultado -> {V, F}
utentes3fase(R) :- solucoes(ID, (utente(ID,_,_,A,_,_,_,P,L,_), criterios3fase(A,P,L)), L), removeRepetidos(L,R).
```

Figura 16: Predicados para permitir identificar os utentes das diferentes fases

```
Output

?- utentes1fase(R).
R = [1, 5, 6, 7].

?- utentes2fase(R).
R = [2, 8].

?- utentes3fase(R).
R = [3, 4, 9].
```

Em seguida, é possível indicar os utentes que foram vacinados nas diferentes fases através dos próximos predicados.

```
% Extensão do predicado que permite identificar utentes que foram vacinados na primeira fase
% 'vacinados1fase': Resultado -> {V, F}
vacinados1fase(R) :- solucoes(ID, (vacinacao_Covid(_,ID,_,M,_,_,1),M<5), R).

% Extensão do predicado que permite identificar utentes que foram vacinados na segunda fase
% 'vacinados2fase': Resultado -> {V, F}
vacinados2fase(R) :- solucoes(ID, (vacinacao_Covid(_,ID,_,M,_,_,1),M>4,M<10), R).

% Extensão do predicado que permite identificar utentes que foram vacinados na terceira fase
% 'vacinados3fase': Resultado -> {V, F}
vacinados3fase(R) :- solucoes(ID, (vacinacao_Covid(_,ID,_,M,_,_,1),M>9), R).
```

Figura 17: Predicados para permitir identificar os utentes vacinados nas diferentes fases

Output

```
?- vacinados1fase(R).
R = [1, 6].

?- vacinados2fase(R).
R = [2, 3].

?- vacinados3fase(R).
R = [4].
```

Ainda, como seria de esperar, predicados que permitem reconhecer os vários critérios de vacinação que possibilita a identificação dos utentes a ser vacinados em cada fase.

```
% Extensão do predicado que permite identificar criterios da 1ª fase
% 'criterios1fase': Resultado -> {V, F}
criterios1fase(Ano,Profissao,LDoencas) :- 2021 - Ano >= 80.
criterios1fase(Ano,Profissao,LDoencas) :- profissaoRisco(Profissao).
criterios1fase(Ano,Profissao,LDoencas) :- listaDoencas(LDoencas), 2021 - Ano >= 50.

% Extensão do predicado que permite identificar criterios da 2ª fase
% 'criterios2fase': Resultado -> {V, F}
criterios2fase(Ano,Profissao,LDoencas) :- nao(profissaoRisco(Profissao)), nao(listaDoencas(LDoencas)),2021 - Ano >= 65, 2021 - Ano < 80.
criterios2fase(Ano,Profissao,LDoencas) :- nao(profissaoRisco(Profissao)), nao(listaDoencas(LDoencas)),listaDoencas2(LDoencas), 2021 - Ano >= 50, 2021 - Ano < 65.

% Extensão do predicado que permite identificar criterios da 3ª fase
% 'criterios3fase': Resultado -> {V, F}
criterios3fase(Ano,Profissao,LDoencas) :- nao(profissaoRisco(Profissao)), 2021 - Ano < 50.
criterios3fase(Ano,Profissao,LDoencas) :- nao(profissaoRisco(Profissao)), nao(listaDoencas(LDoencas)),nao(listaDoencas2(LDoencas)), 2021 - Ano >= 50, 2021 - Ano < 65.
```

Figura 18: Predicados para identificar os criterios de seleção de casa fase de vacinação

Os próximos predicados permitem a identificação de doenças(de risco 1 ou risco 2), de profissões de risco e se uma determinada lista de doenças(de um utente a ser vacinado) contém alguma das doenças de risco.

```
% Extensão do predicado que permite identificar se uma lista contém uma doença de risco 1
% 'listaDoencas': Resultado -> {V, F}
listaDoencas([H|_]) :- doencaRisco(H).
listaDoencas([_|T]) :- listaDoencas(T).

% Extensão do predicado que permite identificar se uma lista contém uma doença de risco 2
% 'listaDoencas2': Resultado -> {V, F}
listaDoencas2([H|_]) :- doencaRisco2(H).
listaDoencas2([_|T]) :- listaDoencas2(T).

% Extensão do predicado que permite identificar se uma doença é de risco 1
% 'doencaRisco': Resultado -> {V, F}
doencaRisco('Insuficiência cardíaca').
doencaRisco('Doença coronária').
doencaRisco('Insuficiência renal').
doencaRisco('Doença respiratória').

% Extensão do predicado que permite identificar se uma doença é de risco 2
% 'doencaRisco2': Resultado -> {V, F}
doencaRisco2('Diabetes').
doencaRisco2('Insuficiência hepática').
doencaRisco2('Hipertensão arterial').
doencaRisco2('Obesidade').

% Extensão do predicado que permite identificar se uma profissão é de risco
% 'profissaoRisco': Resultado -> {V, F}
profissaoRisco('Medico').
profissaoRisco('Enfermeiro').
profissaoRisco('Militar').
profissaoRisco('Segurança').
profissaoRisco('Residente de lar de idosos').
```

Figura 19: Predicados para permitir identificar os casos de risco

3.5.2 Identificar as pessoas não vacinadas

Através da utilização do predicado *solucoes* conseguimos obter todos os utentes não vacinados, cujo ID de utente não esteja representado na lista de utentes vacinados.

```
% Extensão do predicado que permite identificar utentes não vacinados
% 'utentesNVacinados': Resultado -> {V, F}
utentesNVacinados(R) :- solucoes(ID,(utente(ID,_,_,_,_,_,_,_,_,_),utentesVacinados(LVacinados),nao(contains(ID,LVacinados))),R).
```

Figura 20: Predicado para identificar os utentes não vacinados

```
Output

?- utentesNVacinados(R).
R = [5, 8, 9].
```

3.5.3 Identificar as pessoas vacinadas

Tal como procedemos na identificação das pessoas não vacinadas, utilizamos o predicado *solucoes* para obter os utentes vacinados, verificando se o ID deste corresponde ao ID do predicado *vacinacao_Covid*. Ainda, removemos as repetições(que acontecem caso os utentes tenham sido vacinados em mais que uma fase), de modo a não ser apresentada informação duplicada.

```
% Extensão do predicado que permite identificar utentes vacinados
% 'utentesVacinados': Resultado -> {V, F}
utentesVacinados(R) :- solucoes(ID,vacinacao_Covid(_,ID,_,_,_,_,_),L), removeRepetidos(L,R).
```

Figura 21: Predicado para identificar os utentes vacinados

```
Output

?- utentesVacinados(R).
R = [1, 2, 3, 4, 6, 7].
```

3.5.4 Identificar as pessoas vacinadas indevidamente

A identificação de pessoas indevidamente vacinadas é *utentesInd1*, *utentesInd2* e *utentesInd3*, que correspondem aos utentes indevidamente vacinados (isto é, vacinados na fase errada) das fases 1, 2 e 3, respetivamente.

Primeiramente, juntamos as listas de pessoas vacinadas nas três fases. De seguida, verificamos que utentes foram indevidamente vacinados da primeira fase. Através do predicado *solucoes* conseguimos verificar que utentes foram indevidamente vacinados, comparando o ID dos utentes destinados à fase 1 com o ID dos utentes vacinados nas fases 2 e 3.

Da mesma forma se processa a identificação dos utentes indevidamente vacinados nas fases 2 e 3, comparando os ID dos utentes destinados a uma determinada fase com o ID dos utentes vacinados nas outras duas fases.

```
% Extensão do predicado que permite identificar utentes vacinados indevidamente
% 'utentesVacinadosInd': Resultado -> {V, F}
utentesVacinadosInd(R) :- utentesInd1(L1),utentesInd2(L2),concat(L1,L2,L),utentesInd3(L3),concat(L, L3, R).

utentesInd1(R) :- solucoes(ID, (utente(ID,_,_,_,_,_,_,_,_,_),utentes1fase(UF1),vacinados2fase(VF1),contains(ID,UF1),contains(ID,VF1)), L2),
                  solucoes(ID, (utente(ID,_,_,_,_,_,_,_,_,_),utentes1fase(UF2),vacinados3fase(VF2),contains(ID,UF2),contains(ID,VF2)), L3),
                  concat(L2,L3,R).
utentesInd2(R) :- solucoes(ID, (utente(ID,_,_,_,_,_,_,_,_,_),utentes2fase(UF1),vacinados1fase(VF1),contains(ID,UF1),contains(ID,VF1)), L1),
                  solucoes(ID, (utente(ID,_,_,_,_,_,_,_,_,_),utentes2fase(UF2),vacinados3fase(VF2),contains(ID,UF2),contains(ID,VF2)), L3),
                  concat(L1,L3,R).
utentesInd3(R) :- solucoes(ID, (utente(ID,_,_,_,_,_,_,_,_,_),utentes3fase(UF1),vacinados1fase(VF1),contains(ID,UF1),contains(ID,VF1)), L1),
                  solucoes(ID, (utente(ID,_,_,_,_,_,_,_,_,_),utentes3fase(UF2),vacinados2fase(VF2),contains(ID,UF2),contains(ID,VF2)), L2),
                  concat(L1,L2,R).
```

Figura 22: Predicados para identificar os utentes vacinados indevidamente

```
Output

?- utentesVacinadosInd(R).
R = [3].
```

3.5.5 Identificar as pessoas não vacinadas que são candidatas

Através do mesmo processo utilizado na secção anterior, conseguimos obter as pessoas não vacinadas que são candidatas à vacinação. Utilizando o predicado *solucoes*, verificamos se o ID de um utente não vacinado é o mesmo que o ID de um utente pertencente a uma das fases.

```
% Extensão do predicado que permite identificar utentes não vacinados que são candidatos
% 'utentesNVacinadosCan': Resultado -> {V, F}
utentesNVacinadosCan(R) :- solucoes(ID, (utente(ID,_,_,_,_,_,_,_,_,_),utentesNVacinados(LNV),utentes1fase(L1),utentes2fase(L2),concat(L1,L2,L),contains(ID,L),contains(ID,LNV)), R).
```

Figura 23: Predicados para identificar os utentes não vacinados mas que são candidatos

```
Output

?- utentesNVacinadosCan(R).
R = [5, 8].
```

3.5.6 Identificar as pessoas a quem falta a toma da segunda vacina

Com a verificação da inexistência do ID de um utente na lista de pessoas a quem já foi administrada a segunda dose e utilizando o predicado *solucoes*, conseguimos identificar as pessoas que ainda não tomaram a segunda dose da vacina.

```
% Extensão do predicado que permite identificar utentes que só tomaram a 1ª toma da vacina
% 'utentes1toma': Resultado -> {V, F}
utentes1toma(R) :- solucoes(ID,(vacinacao_Covid(_,ID,_,_,_,_,1),utentes2toma(L),nao(contains(ID,L))),R).

% Extensão do predicado que permite identificar utentes que tomaram a 2ª toma da vacina
% 'utentes2toma': Resultado -> {V, F}
utentes2toma(R) :- solucoes(ID,vacinacao_Covid(_,ID,_,_,_,_,2),R).
```

Figura 24: Predicados para identificar os utentes que não tomaram a segunda dose

```
Output

?- utentes1toma(R).
R = [2, 3].
```

3.5.7 Sistema de inferência

```
inferencia(Questao, verdadeiro) :- Questao.
inferencia(Questao, falso) :- -Questao.
```

Figura 25: Predicados para construir o sistema de inferência

```
Output

?- inferencia(utentes1toma(R),L).
R = [2, 3],
L = verdadeiro .
```

3.6 Predicados extra de listagem de conhecimento

3.6.1 Identificar a fase de vacinação de uma pessoa

O predicado *faseUtente* permite identificar a fase à qual um utente com um determinado ID pertence. Verifica se o ID do utente está presente nas listas de utentes relativas a cada fase.

```
faseUtente(ID, 'Primeira Fase') :- utentes1fase(L), contains(ID, L).
faseUtente(ID, 'Segunda Fase') :- utentes2fase(L), contains(ID, L).
faseUtente(ID, 'Terceira Fase') :- utentes3fase(L), contains(ID, L).
```

Figura 26: Predicado que permite identificar a fase de vacinação de cada utente

```
Output

?- faseUtente(1,R).
R = 'Primeira Fase'.
```

3.6.2 Identificar o nome do centro de saúde de um membro do staff

O predicado *centroStaff* possibilita a obtenção do nome do centro de saúde onde trabalha um membro do staff.

```
centroStaff(ID, Nome) :- staff(ID,ID_Centro,_,_), centro_saude(ID_Centro,Nome,_,_,_).
```

Figura 27: Predicado que permite identificar o centro de saude de um membro do staff

```
Output

?- centroStaff(1,R).
R = 'BragaH'.
```

3.6.3 Identificar as vacinas administradas por um membro do staff

O predicado *vacinaStaff* identifica as vacinas administradas por um determinado membro do staff através do predicado solucoes,comparando o ID do staff com o ID do staff na vacina. Em seguida removem-se os repetidos para não haver múltipla informação com o mesmo significado.

```
vacinasAdminStaff(ID_Staff,R) :- solucoes(V, vacinacao_Covid(ID_Staff,_,_,_,_,V,_), L), removeRepetidos(L,R).
```

Figura 28: Predicado para identificar as vacinas administrados por um membro do staff

```
Output

?- vacinasAdminStaff(1,R).
R = ['AstraZeneca', 'Moderna'].
```

3.6.4 Identificar a frequência de cada vacina

O predicado *frequenciaVacinas* calcula o número de vezes que cada vacina foi administrada. Inicialmente, através do predicado solucoes, é obtida a lista com todas as vacinas administradas. De seguida, removemos a informação repetida na lista. Por fim, utilizamos o predicado *vacina_lista* que devolve uma lista com as vacinas e o número de vezes que foram administradas. Este predicado utiliza outro, *ocorrencia_vacina*, que calcula, efetivamente, o número de ocorrências de cada vacina, através de um acumulador.

```
frequenciaVacinas(R) :- solucoes(V, vacinacao_Covid(_,_,_,_,_,V,_), L), removeRepetidos(L,L2), vacinaLista(L2,L,R).

vacinaLista([],_,[]).
vacinaLista([H|T],V,[(H,Q)|L]) :- ocorrenciaVacina(H,V,0,Q),vacinaLista(T,V,L).

ocorrenciaVacina(_, [], Acc, Acc).
ocorrenciaVacina(X, [X|T], OldAcc, Res) :- NewAcc is OldAcc + 1, ocorrenciaVacina(X, T, NewAcc, Res).
ocorrenciaVacina(X, [_|T], Acc, Res) :- ocorrenciaVacina(X, T, Acc, Res).
```

Figura 29: Predicado que identifica a frequencia de uso de cada vacina

```
Output

?- frequenciaVacinas(R).
R = [('Moderna', 3),  ('PFizer', 3),  ('AstraZeneca', 3)].
```

3.7 Predicados auxiliares

3.7.1 Predicado *nao*

O predicado *nao* representa a negação de uma questão, ou seja, nega o seu parâmetro.

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

Figura 30: Extensão do predicado *nao*

3.7.2 Predicado *contains*

O predicado *contains* permite verificar se um determinado elemento pertence a uma lista.

```
% Extensão do predicado que permite verificar se um elemento percente a uma lista:
% 'contains':  Elemento, Conjunto -> {V,F}
contains(E,[E|T]).
contains(E,[Y|T]) :- E\=Y, contains(E,T).
```

Figura 31: Extensão do predicado *contains*

3.7.3 Predicado *natural*

O predicado *natural* verifica se um determinado número é natural.

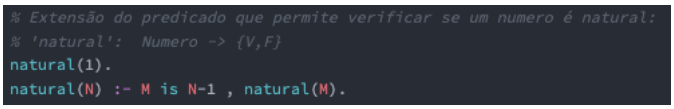

Figura 32: Extensão do predicado *natural*

3.7.4 Predicado *add*

O predicado *add* adiciona um elemento a uma lista caso este ainda não exista na mesma.

Figura 33: Extensão do predicado *add*

3.7.5 Predicado *concat*

O predicado *concat* resulta numa lista obtida através da concatenação de duas listas fornecidas.

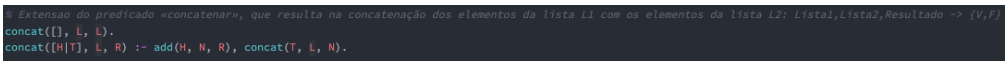

Figura 34: Extensão do predicado *concat*

3.7.6 Predicado *solucoes*

O predicado *solucoes* obtém uma lista com todos todas as ocorrências de X em Y, o memso que findall.

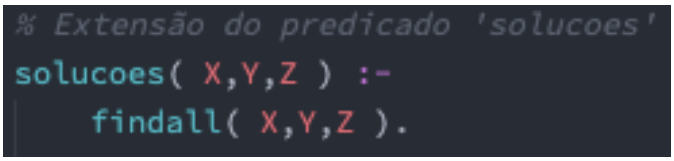

Figura 35: Extensão do predicado *solucoes*

3.7.7 Predicado *comprimento*

O predicado *comprimento* terá como função devolver o comprimento de uma determinada lista.

```
% Extensão do predicado 'comprimento'
comprimento( S,N ) :-
    length( S,N ).
```

Figura 36: Extensão do predicado *comprimento*

3.7.8 Predicado *removeRepetidos*

O predicado *removeRepetidos* efetua a remoção dos elementos repetidos de uma determinada lista.

```
% Extensão do predicado 'removeRepetidos' que remove os elementos repetidos duma lista
removeRepetidos([], []).
removeRepetidos([H|T], R) :- contains(H, T) , removeRepetidos(T, R).
removeRepetidos([H|T], [H|R]) :- nao(contains(H, T)) , removeRepetidos(T, R).
```

Figura 37: Extensão do predicado *removeRepetidos*

3.7.9 Predicado *bissexto*

O predicado *bissexto* é utilizado para verificar se um ano é bissexto.

```
% Extensão do predicado 'bissexto': Ano => {V, F}
bissexto(X) :- X mod 4 == 0.
```

Figura 38: Extensão do predicado *bissexto*

3.7.10 Predicado *data*

O predicado *data* serve para validar uma data inserida.

```
% Extensão do predicado 'data': Ano, Mes, Dia => {V, F}
data(A,M,D) :- M\=2, A>=2010, contains(M,[1,3,5,7,8,10,12]), D>=0, D=<31.
data(A,M,D) :- M\=2, A>=2010, contains(M,[4,6,9,11]), D>0, D=<30.
data(A,M,D) :- M==2 , bissexto(A), A>=2010, D>0, D=<29.
data(A,M,D) :- M==2 , nao(bissexto(A)), A>=2010, D>0, D=<28.
```

Figura 39: Extensão do predicado *data*

3.7.11 Predicado *telefone*

O predicado *telefone* serve para validar um número de telemóvel inserido.

```
% Extensão do predicado 'telefone': Tel => {V, F}
telefone(Tel) :- Tel >= 900000000, Tel =< 999999999.
```

Figura 40: Extensão do predicado *telefone*

3.7.12 Predicado *mesmoCentro*

O predicado *mesmoCentro* verifica se um membro do staff e um utente têm o mesmo centro de saúde.

```
% Extensão do predicado 'mesmoCentro': ID_Staff, ID_Utente => {V, F}
mesmoCentro(ID_Staff, ID_Utente) :- staff(ID_Staff,ID_Centro,_,_), utente(ID_Utente,_,_,_,_,_,_,_,_,_,ID_Centro).
```

Figura 41: Extensão do predicado *mesmoCentro*

4 Conclusão

Após o término deste trabalho prático de grupo da Unidade Curricular de SRCR, atribuimos um balanço positivo ao trabalho desenvolvido.

Reconhecemos a importância deste primeiro contacto com a linguagem de programação lógica de *PROLOG*, para além do trabalho realizado durante as aulas práticas, que se revelou um desafio interessante no que diz respeito à descoberta da melhor estratégia a utilizar para a realização daquilo que era pretendido. Com o desenrolar do projeto deparamo-nos com algumas dificuldades, que fomos ultrapassando, sendo que a mais significativa foi a implementação dos invariantes.

Assim, consideramos que foi alcançada uma melhor consolidação da matéria lecionada nas aulas, permitindo, ainda, a exploração de novos domínios necessários à concretização do nosso projeto.

Seguimos determinados e focados em levar este projeto a bom porto.