UNIVERSIDADE DO MINHO

Programação Lógica e Invariantes

MIEI

Sistemas de Representação de Conhecimento e Raciocínio

(2º Semestre - 2020/2021)

A85489 Simão Monteiro

A89557 Pedro Veloso

A89587 Carlos Preto

A93785 Ricardo Gomes

Maio 2021

Resumo

De forma a consolidar os conhecimentos até agora obtidos nesta Unidade Curricular, foi desenvolvido um pequeno exercício prático, um Sistema de Representação de Conhecimento e Raciocínio com capacidade para lidar com conhecimento afirmativo, negativo e imperfeito de um tema bastante atual que é a vacinação da população contra o vírus Covid-19.

Índice

Índice de figuras

Introdução

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

Na primeira fase do trabalho foram desenvolvidas, na linguagem de programação em lógica PROLOG, funcionalidades como a definição de fases de vacinação, tendo em conta alguns critérios, verificar se algum utente foi vacinado indevidamente, entre outras. Contudo durante a realização da primeira parte do trabalho, não se considerou a existência de conhecimento negativo ou imperfeito. Desta forma neste trabalho pretende-se aplicar mecanismos que permitam a manipulação de tal conhecimento. Assim além da implementação, de novos mecanismos que darão apoio aos novos tipos de conhecimento será necessário corrigir alguns das regras já desenvolvidas.

Preliminares

Para o desenvolvimento deste trabalho o conhecimento adquirido até ao momento nas aulas, bem como o desenvolvimento da primeira fase do trabalho, mostrou-se essencial para a sua correta realização.

Por fim, foi necessário, de modo a adquirir conhecimento sobre a área de aplicação do sistema, consultar a página da DGS e algumas notícias que pudessem transmitir conhecimento útil para o desenvolvimento do trabalho.

Descrição do Trabalho

O sistema desenvolvido, para esta fase do trabalho, carateriza o mesmo cenário da vacinação da população, proposta na fase anterior. Contudo nesta fase pretende-se definir a existência não só de conhecimento afirmativo, como também o conhecimento negativo e imperfeito.

1.1. Conhecimento positivo

Considerou-se que o conhecimento positivo, não sofreu alterações, sendo por isso dado do seguinte modo:

utente: #Idutente, Nº Seguranca Social, Nome, Data_Nasc, Email, Telefone, Morada, Profissão, [Doenças_Crónicas], #CentroSaúde $\rightsquigarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

centro_saúde: #Idcentro, Nome, Morada, Telefone, Email $\rightsquigarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

staff: #Idstaff, #Idcentro, Nome, email $\rightsquigarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

vacinação_Covid: #Staf, #utente, Data, Vacina, Toma $\rightsquigarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

profissaoPrioritaria: Profissao $\rightsquigarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

doença_risco: Doenca $\rightsquigarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

1.2. Conhecimento negativo

De modo a conhecer afirmações negativas é necessário representar o conhecimento falso adquirido, na base de conhecimento. Para a representação de conhecimento falso, utiliza-se a etiqueta '-' antes do nome do predicado, o que origina no caso do utente um conhecimento expresso da seguinte forma.

-utente: #Idutente, Nº Seguranca Social, Nome, Data_Nasc, Email, Telefone, Morada, Profissão, [Doenças_Crónicas], #CentroSaúde $\rightsquigarrow$ { $\mathbb{V}$, $\mathbb{F}$ }

Ao utilizar este tipo de negação é possível garantir a falsidade forte do termo, uma vez que é um conhecimento adquirido, ao contrário da negação pelo termo "não" usado na fase 1 do trabalho, que representava a negação de um termo por falha da prova (i.e., verificar a existência/inexistência do termo na base de conhecimento).

1.3. Conhecimento imperfeito

1.3.1. Tipo incerto (Tipo 1)

Para este tipo de conhecimento imperfeito considera-se a informação que não é conhecida a 100%, ou seja, informação onde parte da mesma é desconhecida. Para interrogações sobre este tipo de conhecimento apenas se obtém respostas com valor desconhecido.

1.3.2. Tipo impreciso (Tipo 2)

O conhecimento do tipo dois, é utilizado para definir conhecimento desconhecido, contudo limitado por um conjunto de valores. Neste tipo de conhecimento é possível ter um conhecimento impreciso cujo conjunto de hipóteses é totalmente discriminado ou que respeite uma gama de valores. Assim é possível obter tanto um valor desconhecido ou falso, ao realizar interrogações sobre este tipo de conhecimento imperfeito.

1.3.3. Tipo interdito (Tipo 3)

Este conhecimento corresponde a situações em que existe informação desconhecida e que nunca poderá vir a ser conhecida, ou seja, conhecimento deste tipo nunca poderá sofrer evolução.

1.4. Termo "Demo"

Uma vez que existe vários tipos de conhecimento (positivo, negativo e imperfeito) é necessário criar um termo que infere a veracidade do conhecimento. Assim criou-se o termo "demo" responsável por verificar se um determinado conhecimento é verdadeiro, falso ou desconhecido.

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

Figura 1: Predicado demo

Para além do termo "demo" criou-se outras variantes – "*demoLista*" e "*demoListaReduce*" – que permitem realizar questões para mais do que um termo. A variante "*demoLista*" permite questionar a veracidade de uma lista de termos, apresentando o resultado de cada termo numa outra lista, enquanto a "*demoListaReduce*" apresenta o resultado num único valor, conjugando o resultado de cada termo da seguinte forma:

Tabela 1: Relação entre valores de conhecimento

Questão1	Questão2	Resultado
Falso	Falso	Falso
Falso	Desconhecido	Falso
Falso	Verdadeiro	Falso
Desconhecido	Falso	Falso
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Desconhecido
Verdadeiro	Falso	Falso
Verdadeiro	Desconhecido	Desconhecido
Verdadeiro	Verdadeiro	Verdadeiro

1.5. Base de conhecimento desenvolvida

1.5.1. Utente

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

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

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

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

Figura 2: Conhecimento Utente

Ao observar a figura acima, percebe-se que alguns utentes apresentam conhecimento desconhecido, como por exemplo para o utente com o identificador 6 não é conhecido o seu número de segurança social. Uma vez que, o se desconhece por completo o

número, este representa um conhecimento imperfeito do tipo 1, o que implica acrescentar uma exceção, tal como se observa na figura seguinte.

```
excecao( utente(ID , _, Nome, Nasc, Mail, Tel, Morada,Prof, D, C) ) :-
  utente(ID, seguranca_social_desconhecida, Nome, Nasc, Mail, Tel, Morada,Prof, D, C).
```

Figura 3: Exemplo de uma exceção de um utente

De modo a garantir que a falsidade de interrogações com conhecimento não representado, determinou-se, pelo princípio da negação forte, que não é um utente, uma pessoa que não esteja registada como tal ou da qual não se sabe nada.

```
-utente(ID , SS, Nome, Nasc, Mail, Tel, Morada,Prof, D, C) :-
    nao(utente(ID , SS, Nome, Nasc, Mail, Tel, Morada,Prof, D, C)),
    nao(excecao(utente(ID , SS, Nome, Nasc, Mail, Tel, Morada,Prof, D, C))).
```

Figura 4: Negação forte do conhecimento utente

1.5.2. Centro de Saúde

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

```
centroSaude(1, nomeCentro_desconhecido, 'Rua General Humberto Delgado', 243592604, 'posto.almeirim@synlab.pt').
centroSaude(2, 'Posto de Analises Clinicas da Amadora' , moradaCentro_desconhecida, 914155759, 'laboratorio.amadora@synlab.pt').
centroSaude(3, 'Posto de Analises Clinicas do Barreiro', 'Avenida do Bocage', 910728308, mailCentro_desconhecido).
centroSaude(4, 'Posto de Analises Clinicas de Braga', moradaCentro_desconhecida, 935465241, 'posto.ambrosio.santos@synlab.pt').
centroSaude(5, 'Posto de Analises Clinicas de Coimbra', 'Praceta Professor Robalo Cordeiro', 239701512, 'laboratorio.coimbra@synlab.pt').
centroSaude(6, 'Posto de Analises Clinicas da Covilha', 'Av. Infante D. Henrique', telemovelCentro_desconhecido, 'posto.covilha@synlab.pt').
centroSaude(7, 'Posto de Analises Clinicas de Evora', 'Praceta Horta do Bispo', telemovelCentro_desconhecido, 'laboratorio.evora@synlab.pt').
```

Figura 5: Conhecimento Centro de Saúde

Uma vez mais é de notar a existência de conhecimento imperfeito do tipo 1, por exemplo no centro de saúde, com identificador 1, do qual não é conhecido o nome. Contudo, para o centro de saúde, também se considerou a existência de conhecimento imperfeito do tipo 2 com um conjunto de hipóteses discriminado. Desta forma adicionou-se as seguintes exceções:

```
excecao(centroSaude(8,'Posto de Analises Clinicas de Lisboa', 'Av. Horta do Bispo', 239703516, 'laboratorio.lisboa@synlab.pt')).
excecao(centroSaude(8,'Posto de Analises Clinicas de Lisboa', 'Av. Infante D. Joao', 239703516, 'laboratorio.lisboa@synlab.pt')).
```

Figura 6: Exceções aplicadas para garantir conhecimento imperfeito do tipo 2

Com a adição das exceções, o sistema passou a conhecer a existência de um centro de saúde com o identificador 8, contudo não consegue distinguir se a sua morada é 'Av. Horta do Bispo' ou 'Av. Infante D. João'.

Para a negação de um centro de saúde, determinou-se que um centro de saúde deve ser falso caso não exista nenhum conhecimento sobre ele.

```
-centroSaude(ID, Nome, Morada, Tel, Mail) :-
    nao(centroSaude(ID, Nome, Morada, Tel, Mail)),
    nao(excecao(centroSaude(ID, Nome, Morada, Tel, Mail))).
```

Figura 7: Negação forte do conhecimento centro de saúde

1.5.3. Staff

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

Os seguintes predicados representam exemplos de staff:

```
staff(1, 1, 'Tatiana', mailStaff_desconhecido).
staff(2, 2, 'Augusta', 'augusta@staff.com').
staff(3, 3, nomeStaff_desconhecido, mailStaff_desconhecido).
staff(4, 4, 'Cristina', 'cristina@staff.com').
staff(5, 5, 'Eva', mailStaff_desconhecido).
staff(6, centro_desconhecido, 'Joana', 'rui@staff.com').
staff(7, 7, 'Vitor', mailStaff_desconhecido).
```

Figura 8: Conhecimento Staff

Para os elementos do staff acima apresentados, é possível identificar conhecimento verdadeiro (staff com o identificador 2) e conhecimento imperfeito para os outros elementos do staff, devido à falta de conhecimento sobre o email, nome ou centro de saúde em que o profissional trabalha. Contudo também é possível caracterizar conhecimento negativo, representando-o da seguinte forma:

```
-satff(1, 2, 'Tatiana', mailStaff_desconhecido).

-staff(IDS, IDC, Nome, Mail) :-
    nao(staff(IDS, IDC, Nome, Mail)),
    nao(excecao(staff(IDS, IDC, Nome, Mail))).
```

Figura 9: Negação forte do conhecimento satff

1.5.4. Vacinação Covid

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

Desta forma, deve ser conhecida informação sobre o staff que vacinou o utente, manifestada sobre a forma do identificador do staff. Também deve ser conhecido o utente que tomou a vacina, a data em que a toma foi dada e o nome da vacina que o

utente tomou. Uma vez que determinadas vacinas, para realizar efeito devem ser tomadas mais que uma vez, deve ser conhecido o número de vezes que o utente já realizou a toma.

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

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

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

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

%Fase3
vacinacaoCovid(5, 2, data(20,10,2021), 'astrazeneca', 1).
vacinacaoCovid(1, 3, data(20,1,2021), vacina_desconhecida, 1).
vacinacaoCovid(7, 10, data(1,1,2021), 'astrazeneca', 1).
vacinacaoCovid(7, 10, nulo, 'astrazeneca', 2).
```

Figura 10: Conhecimento vacinação contra covid

Ao observar o conhecimento representado acima, uma vez mais, é possível identificar conhecimento imperfeito do tipo 1, contudo para a conhecimento da vacinação da segunda toma do utente 10, percebe-se a existência de um valor 'nulo'. Este valor pretende representar o conhecimento interdito (tipo 3), o que implica acrescentar os seguintes dados:

```
nulo(nulo).
+vacinacaoCovid(_,_,_,_,_)::(solucoes(A,(vacinacaoCovid(7,10,A,'astrazeneca', 2), nao(nulo(A))),S), comprimento(S,N), N==0).
```

Figura 11: Conhecimento que garante o conhecimento imperfeito do tipo 3

Assim a evolução do conhecimento fica interdita, deixando de ser possível evoluir este conhecimento no futuro.

Relativamente ao conhecimento negativo do predicado, pretende-se que qualquer informação que não seja representada no conhecimento de forma afirmativa ou imperfeita seja considerada falsa, logo adiciona-se o seguinte conhecimento negativo:

```
-vacinacaoCovid(IDS, IDU, D, V, T) :-
    nao(vacinacaoCovid(IDS, IDU, D, V, T)),
    nao(excecao(vacinacaoCovid(IDS, IDU, D, V, T))).
```

Figura 12: Negação do conhecimento vacinação covid

1.5.5. Profissão Prioritária

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

Os seguintes predicados representam exemplos dessas profissões:

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

Figura 13: Conhecimento profissão prioritária

Além do conhecimento afirmativo representado em cima, adicionou-se o seguinte conhecimento negativo:

```
-profissaoPrioritaria(P) :-
    nao(profissaoPrioritaria(P)),
    nao(excecao(profissaoPrioritaria(P))).
```

Figura 14: Conhecimento negativo sobre as profissões prioritárias

1.5.6. Doença de Risco

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

Os predicados, que constam na figura 15, representam exemplos de doenças de risco:

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

Figura 15: Conhecimento doença de risco

Já o predicado presente na próxima figura representa o conhecimento negativo sobre as doenças de risco.

```
-doenca_risco(D) :-
    nao(doenca_risco(D)),
    nao(excecao(doenca_risco(D))).
```

Figura 16: Negação do conhecimento doença de risco

1.6. Adição e remoção de conhecimento

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

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

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

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

Figura 17: Predicado - Soluções

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

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

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

Figura 18: Predicado - Comprimento

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

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

1.6.1. Evolução

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

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

Figura 19: Predicado - Evolução

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

O predicado *insere*, é responsável por inserir novo conhecimento, enquanto o predicado *teste*, realiza testes sobre todos os invariantes relativos ao "Termo". Contudo, como se observa na figura 19, o predicado insere primeiro o conhecimento e só depois testa a sua coerência. Assim, caso o predicado *teste* falhe, é necessário retirar o termo da base de conhecimento, como se pode observar na figura seguinte:

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

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

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

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

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

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

Figura 20: Predicado – Teste/Insere/Remove

1.6.2. Involução

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

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

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

Figura 21: Predicado - Involução

Neste caso o predicado *remove* elimina conhecimento. Ao colocar a variável "Termo" como condição para a remoção, pretende-se verificar a existência do conhecimento a remover. Posteriormente, realiza-se, de modo semelhante à evolução, a procura por invariantes relativos à remoção do conhecimento dado. Para identificar o invariante coloca-se o símbolo '-', a preceder o nome desse termo. Posteriormente remove-se o elemento e testa-se a coerência. Caso o conhecimento fique incoerente volta-se a inserir o conhecimento, como observado na figura 21.

1.6.3. Utentes (Invariantes)

Na adição de um novo utente deve-se ter em conta, se o conhecimento adicionado é um conhecimento afirmativo ou negativo, e consoante o seu tipo ter em conta as seguintes restrições:

Conhecimento positivo:

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

```
+utente(ID,SS,Nome,DA,Email,TS,Morada,Prof,D,PS) :: (integer(ID),
                                                     numeroSegurancaSocialValida(SS),
                                                     numTelemovelValido(TS),
                                                     dataNascimentoValida(DA),
                                                     postoSaudeValido(PS),
                                                     solucoes(ID,(utente(ID,_,_,_,_,_,_,_,_,_)),S ),
                                                     comprimento( S,C ),
                                                     C == 1,
                                                     nao(-utente(ID,SS,Nome,DA,TS,Email,Morada,Prof,D,PS))
                                                    ).
```

Figura 22: Invariante para a adição afirmativa de um utente

Conhecimento negativo:

- O identificador tem de ser um número;
- O número de segurança social tem de ter 11 dígitos;
- A data de nascimento tem de ser válida;
- O número de telefone tem de ter 9 dígitos;
- Não pode existir uma negação equivalente.

```
+(-utente(ID,SS,Nome,DA,Email,TS,Morada,Prof,D,PS)) :: (integer(ID),
                                                        numeroSegurancaSocialValida(SS),
                                                        numTelemovelValido(TS),
                                                        data(DA),
                                                        postoSaudeValido(PS),
                                                        solucoes(ID,(-utente(ID,SS,Nome,DA,Email,TS,Morada,Prof,D,PS)),S ),
                                                        comprimento( S,C ),
                                                        C > 0,
                                                        C < 3
                                                       ).
```

Figura 23: Invariante para a adição negativa de um utente

Em relação à remoção, também é necessário distinguir a remoção de conhecimento negativo e positivo, contudo como não existe restrições à remoção de conhecimento negativo, criou-se apenas o seguinte invariante para a remoção do conhecimento positivo.

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

Figura 24: Invariante remoção Utente

Segundo o invariante apenas podem ser removidos utentes não registados como vacinados. Com este invariante pretende-se garantir a coerência de conhecimento, não permitindo existir o conhecimento de ID's de utentes, sem conhecer toda a informação do utente. Para realizar a remoção de um utente que já se encontre vacinado, deve ser primeiramente removida a informação sobre a sua vacinação e só depois pode remover-se o conhecimento relativo ao utente.

1.6.4. Centro de Saúde (Invariantes)

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

- O identificador tem de ser um inteiro;
- O número de telefone tem de ter 9 dígitos;
- Não pode existir um centro de saúde com o mesmo identificador;
- Não pode existir conhecimento negativo que contradiz o conhecimento a inserir.

```
+centroSaude(ID,N,M,T,E) :: (integer(ID),
                             numTelemovelValido(T),
                             solucoes(ID,(centroSaude(ID,_,_,_,_)),S ),
                             comprimento( S,C ),
                             C == 1,
                             nao(-centroSaude(ID,N,M,T,E))
                            ).
```

Figura 25: Invariante adição afirmativa Centro de Saúde

Para adicionar conhecimento negativo existe um invariante, que impede a inserção de negações onde o identificador não é um inteiro, o número de telefone não apresenta 9 dígitos ou negações que já são conhecidas.

```
+(-centroSaude(ID,N,M,T,E)) :: (integer(ID),
                                numTelemovelValido(T),
                                solucoes(ID,(-centroSaude(ID,N,M,T,E)),S ),
                                comprimento( S,C ),
                                C > 0,
                                C < 3
                               ).
```

Figura 26: Invariante adição negativa Centro de Saúde

Como o conhecimento negativo não apresenta restrições, apenas existe um invariante que restringe a remoção de conhecimento positivo. Assim, para remover o conhecimento positivo de um centro de saúde, deve-se garantir que não existe nenhum staff associado ao centro.

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

Figura 27: Invariante remoção Centro de Saúde

1.6.5. Staff (Invariantes)

Para adicionar uma afirmação sobre o conhecimento de um elemento do staff, é necessário o conhecimento respeitar as seguintes restrições:

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

```
+staff(ID,IC,N,M) :: (integer(ID),
                      centroSaudeValido(IC),
                      solucoes(ID,(staff(ID,_,_,_)),S ),
                      comprimento( S,C ),
                      C == 1,
                      nao(-satff(ID,IC,N,M))
                     ).
```

Figura 28: Invariante para adição afirmativa de um Staff

Para adicionar uma negação do conhecimento staff, é necessário respeitar as seguintes restrições:

- O identificador do centro de saúde e do staff tem de ser um inteiro;
- Não pode existir um conhecimento equivalente.

```
+(-staff(ID,IC,N,M)) :: (integer(ID),
                         integer(IC),
                         solucoes(ID,(-staff(ID,IC,N,M)),S ),
                         comprimento( S,C ),
                         C > 0,
                         C < 3
                        ).
```

Figura 29: Invariante para adição negativa de um Staff

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

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

Figura 30: Invariante remoção Staff

O conhecimento negativo não apresenta restrições de remoção.

1.6.6. Vacinação covid (Invariantes)

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

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

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

Figura 31: Invariante adição afirmativa de uma Vacinação contra Covid

Para adicionar conhecimento negativo relativo à vacinação contra a covid, deve-se ter em conta as seguintes restrições:

- Não podem ser dados repetidos;
- O identificador do staff e do utente tem que ser inteiro;
- Data da vacinação tem de ser válida.

```
+(-vacinacaoCovid(IS,IU,D,V,T)) :: (integer(IU),
                                    integer(IS),
                                    data(D),
                                    integer(T),
                                    solucoes(IS,(-vacinacaoCovid(IS,IU,D,V,T)),S ),
                                    comprimento( S,C ),
                                    C > 0,
                                    C < 3
                                   ).
```

Figura 32: Invariante adição negativa de uma Vacinação contra Covid

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

1.6.7. Profissão prioritária (Invariantes)

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

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

+(-profissaoPrioritaria(P)) :: (solucoes(P,(-profissaoPrioritaria(P)),S ),
                                comprimento( S,C ),
                                C > 0,
                                C < 3
                               ).
```

Figura 33: Invariante adição Profissão Prioritária

1.6.8. Doença de risco (Invariantes)

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

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

+(-doenca_risco(D)) :: (solucoes(D,(-doenca_risco(D)),S ),
                        comprimento( S,C ),
                        C > 0,
                        C < 3
                       ).
```

Figura 34: Invariante adição Doença Risco

1.7. Implementação das funcionalidades com os diferentes tipos de conhecimento

Nesta secção serão apresentadas as principais alterações realizadas em alguns predicados, desenvolvidos para a fase 1, devido à utilização de conhecimento imperfeito. Sendo que para todos os predicados foi necessário garantir que em caso de verificação do valor lógico de um termo que não esteja caracterizado no conhecimento, obter-se-á o valor falso. Assim, adicionou-se, para todos os termos, a negação como no exemplo da próxima figura, que no caso de desconhecimento garante a falsidade do termo.

```
-idade(D, IDADE) :-
    nao(idade(D, IDADE)),
    nao(excecao(idade(D, IDADE))).
```

Figura 35: Exemplo de negação

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

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

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

Figura 36: Exemplo de conhecimento semelhante da Vacinação contra o Covid

1.7.1. Definição de Fases de Vacinação

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

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

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

Figura 37: Predicado – Pessoas Candidatas

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

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

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

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

Figura 38: Predicado – Fase 1

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

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

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

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

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

Figura 39: Predicado – Fase 2

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

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

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

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

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

Figura 40: Predicado – Fase 3

Contudo, para esta funcionalidade, como posteriormente pode existir mais do que três fazes de vacinação definiu-se que no caso de o sistema ser interrogado por uma fase superior às definidas acima, o resultado deve ser um valor desconhecido. Assim acrescentou-se as exceções típicas de um conhecimento imperfeito do tipo 2 com uma gama de valores.

```
excecao( pessoasCandidatas(X, _) ) :- X >= 4.
```

Figura 41: Exceção pessoas candidatas

Por fim adicionou-se, como demonstrado na figura seguinte, a negação deste predicado.

```
-pessoasCandidatas(F, X) :-
    nao(pessoasCandidatas(F, X)),
    nao(excecao(pessoasCandidatas(F, X))).
```

Figura 42: Negação do predicado pessoas candidatas

1.7.2. Término de uma fase

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

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

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

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

Figura 43: Predicado – Última Data Fase

Uma vez que, com a existência de conhecimento imperfeito é possível que certo conhecimento sobre as vacinações apresente datas desconhecidas, considera-se como última data de uma fase a data mais recente conhecida. Contudo, foi necessário alterar o predicado antes e depois – obtendo o resultado apresentado na figura seguinte – uma vez que agora podem receber valores desconhecidos.

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

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

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

antes(D1, D2) :- nao(depois(D1,D2)).

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

Figura 44: Novo predicado antes e depois

1.7.3. Verificar que staff deu vacinas, numa fase

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

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

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

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

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

Figura 45: Predicado – Staff ativo numa fase

Neste e noutros predicados, é importante referir que nos resultados podem aparecer combinações de identificadores com valores desconhecidos para o campo nome, uma vez que o conhecimento imperfeito é permitido.

1.7.4. Determina em que centro de saúde um utente foi vacinado

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

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

Figura 46: Predicado – Lugar onde foi vacinado

Uma vez que a utilização de conhecimento imperfeito permite o desconhecimento de algumas partes do conhecimento, é possível que para um utente não seja conhecido o elemento do staff que o vacinou, sendo assim impossível descobrir o centro de saúde a que o utente se deslocou. Devido a este problema, acrescentou-se a exceção seguinte, que introduzindo um conhecimento imperfeito do tipo 2, permite obter um valor desconhecido quando não se consegue inferir o centro de saúde.

```
excecao( lugarOndeVacinado(ID, _) ) :- vacinacaoCovid(id_staff_desconhecido,ID,_,_,_).
```

Figura 47: Exceção lugar onde vacinado

1.7.5. Determina se um utente foi vacinado por um elemento do staff

Coma criação desta funcionalidade pretende-se verificar a veracidade de uma interrogação sobre um utente ter sido vacinado por um dado staff. Contudo como o identificador do staff que vacinou o utente pode ser desconhecido, é necessário

acrescentar a seguinte exceção de modo a obter um valor desconhecido quando o staff não é conhecido.

```
excecao(utenteVacinadoStaff(IDStaff, NStaff, IDUtente, NUtente)) :- vacinacaoCovid(id_staff_desconhecido,NStaff,_,_,_).
```

Figura 48 Exceção utente vacinado staff

1.8. Demonstração do sistema

1.8.1. Conhecimento positivo

Considerando a base de conhecimento desenvolvida para o staff e apresentada na figura 8, percebe-se a existência de um elemento com o identificador 2 que possui conhecimento totalmente conhecido e positivo. Assim ao interrogar o sistema sobre a existência desse elemento do staff deve-se obter uma resposta afirmativa, como se demonstra em seguida.

```
|    demo(staff(2,2,'Augusta','augusta@staff.com'),V).
V = verdadeiro ,

?- demo(staff(2,2,N,'augusta@staff.com'),V).
N = 'Augusta',
V = verdadeiro ,
```

Figura 49: Exemplo de execução para conhecimento verdadeiro

1.8.2. Conhecimento negativo

Tendo em consideração o mesmo conhecimento sobre os elementos do staff, é possível verificar (figura 9) que existe um conhecimento negativo sobre o staff com o identificador 1, para além da negação de qualquer interrogação sobre elementos de staff não registados. Desta forma o sistema comporta-se da seguinte forma:

```
|    demo(staff(1,2,'Tatiana',_),V).
V = falso ,

?- demo(staff(10,_,_,_),V).
V = falso ,
```

Figura 50: Exemplo de execução para conhecimento negativo

1.8.3. Conhecimento imperfeito

Para os elementos de staff é possível verificar a existência de conhecimento imperfeito do tipo 1, logo ao interrogar o sistema, por exemplo sobre a existência de um elemento

do staff com o nome Tatiana e com o identificador 1, que trabalha no centro de saúde 1, o sistema deve responder com o valor desconhecido, independentemente do email questionado, uma vez que o email não é conhecido, e como tal pode ser qualquer um.

```
|    demo(staff(1,1,'Tatiana','tatiana@email.com'),V).
V = desconhecido.

?- demo(staff(1,1,'Tatiana','tatiana@staff.com'),V).
V = desconhecido.

?- demo(staff(1,1,'Tatiana','t@staff.com'),V).
V = desconhecido.
```

Figura 51: Exemplo de execução para conhecimento imperfeito do tipo 1

No conhecimento sobre os centros de saúde, existe conhecimento imperfeito do tipo 2 relativo ao centro de saúde identificado pelo id 8. Desta forma, uma vez que o sistema não consegue distinguir se a morada do centro de saúde é "Av. Horta do Bispo" ou "Av. Infante D. João", deve indicar este conhecimento como desconhecido quando questionado pela existência deste centro numa das suas moradas, enquanto para outras moradas deve indicar o conhecimento como falso.

```
|    demo(centroSaude(8,_,'Praca das Artes',_,_),V).
V = falso .

?- demo(centroSaude(8,_,'Av. Horta do Bispo',_,_),V).
V = desconhecido.

?- demo(centroSaude(8,_,'Av. Infante D. Joao',_,_),V).
V = desconhecido.
```

Figura 52: Exemplo1 de execução para conhecimento imperfeito do tipo 2

Uma outra variante do conhecimento imperfeito do tipo 2, está relacionado ao termo *pessoasCandidatas*, onde o sistema deve determinar, quando questionado por fases superiores a 3, como conhecimento desconhecido.

```
?- demo(pessoasCandidatas(1,X),V).
X = [(7, 'Augusta'),  (8, 'Carlota'),  (9, 'Cristina')],
V = verdadeiro .

?- demo(pessoasCandidatas(4,X),V).
V = desconhecido.
```

Figura 53: Exemplo2 de execução para conhecimento imperfeito do tipo 1

Por fim o conhecimento imperfeito do tipo 3 está ligado à segunda vacinação do utente com o identificador 10. Desta forma a evolução deste conhecimento deve ser impossível. Além desse bloqueio, ao questionar o sistema sobre a data de vacinação o sistema deve responder como conhecimento desconhecido.

```
|    demo(vacinacaoCovid(7,10,data(23,3,2020),'astrazeneca',2),V).
V = desconhecido.

?- evolucao(vacinacaoCovid(7,10,data(23,3,2020),'astrazeneca',2)).
false.
```

Figura 54: Exemplo de execução para conhecimento imperfeito do tipo 3

1.8.4. Evolução do conhecimento

Ao tentar evoluir o sistema, o conhecimento inserido deve respeitar os invariantes anteriormente apresentados. Logo considerando os invariantes para o predicado staff temos os seguintes resultados:

```
|    evolucao(staff('13',1,'Pedro','pedro@mail.com')).
false.

?- evolucao(staff(13,1,'Pedro','pedro@mail.com')).
true .

?- evolucao(staff(13,1,'Pedro','pedro@mail.com')).
false.

?- evolucao(-staff(1,2,'Tatiana','mailStaff_desconhecido')).
false.

?- evolucao(-staff(10,2,'Tatiana','mail@mail.com')).
true .
```

Figura 55: Exemplo de execução para a evolução do staff

Ao observar a figura acima, percebe-se a existência da falha de inserção de alguns conhecimentos. Relativamente à primeira falha, a mesma deu-se devido ao identificador do novo staff não ser um inteiro. A segunda falha dá-se devido ao elemento do staff a inserir já existir, uma vez que foi inserido na operação anterior. Por fim a última falha acontece porque o conhecimento negativo a inserir já se encontra caracterizado no conhecimento atual.

1.8.5. Involução do conhecimento

Tal como na evolução do conhecimento, a involução deve respeitar os invariantes de remoção de conhecimento. Assim considerando por exemplo o conhecimento adquirido para o termo staff, doença de risco e vacinação à Covid temos os seguintes resultados:

```
|
|     involucao(staff(1,1,'Augusta','augusta@staff.com')).
false.

?- involucao(doenca_risco('Asma')).
true .

?- doenca_risco('Asma').
false.

?- involucao(-vacinacaoCovid(1,7,data(1,2,2021),'astrazeneca',1)).
true .
```

Figura 56: Exemplo de execução para a involução do conhecimento

Conclusão

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

Com a utilização da linguagem de programação em lógica PROLOG, desenvolveu-se um sistema capaz de inferir conhecimento positivo, negativo e impreciso sobre os diferentes factos e regras desenvolvidas na fase 1 do trabalho.

Criou-se um predicado, denominado por “demo”, que permite aos utilizadores interrogar o sistema sobre os diferentes tipos de conhecimento, obtendo como resposta o valor verdadeiro, falso ou desconhecido. O valor verdadeiro ou falso só é obtido quando o saber está expresso diretamente na base de conhecimento, contudo de modo a garantir a falsidade do conhecimento não adquirido, desenvolveu-se uma regra capaz de negar tal conhecimento. Assim, o valor desconhecido é obtido, apenas quando se interroga o sistema sobre conhecimento imperfeito, seja ele do tipo 1, 2 ou 3 abordados neste relatório.

Por fim criou-se mecanismos que permitem ao sistema sofrer uma evolução ou uma involução, isto é, permite ao utilizador acrescentar ou remover conhecimento, sempre respeitando o conjunto de invariantes, referido anteriormente, que garante a consistência do conhecimento.

Anexos

1.1. Teste do predicado: pessoas candidatas

```
|    demo(pessoasCandidatas(1,X),V).
X = [(7, 'Augusta'), (8, 'Carlota'), (9, 'Cristina')],
V = verdadeiro .

?- demo(pessoasCandidatas(2,X),V).
X = [(6, 'Ana'), (4, 'Vasco'), (11, 'Rui')],
V = verdadeiro .

?- demo(pessoasCandidatas(3,X),V).
X = [(1, 'Simao'), (2, 'Telmo'), (3, 'Tiago'), (5, nome_desconhecido), (10, 'Eva')],
V = verdadeiro .

?- demo(pessoasCandidatas(4,X),V).
V = desconhecido.
```

1.2. Teste do predicado: pessoas vacinadas

```
?- demo(pessoasVacinadas(X),V).
X = [(1, 'Simao'), (7, 'Augusta'), (8, 'Carlota'), (9, 'Cristina'), (10, 'Eva')],
V = verdadeiro .
```

1.3. Teste do predicado: pessoas não vacinadas

```
?- demo(pessoasNaoVacinadas(X),V).
X = [(2, 'Telmo'), (3, 'Tiago'), (4, 'Vasco'), (5, nome_desconhecido), (6, 'Ana'), (11, 'Rui')],
V = verdadeiro .
```

1.4. Teste do predicado: pessoas vacinadas indevidamente

```
|    demo(pessoasVacinadasIndevidamente(X),V).
X = [(10, 'Eva'), (3, 'Tiago'), (2, 'Telmo'), (1, 'Simao')],
V = verdadeiro .
```

1.5. Teste do predicado: pessoas não vacinadas candidatas

```
?- demo(pessoasNaoVacinadasCandidatas(X),V).
X = [(4, 'Vasco'), (6, 'Ana')],
V = verdadeiro .
```

1.6. Teste do predicado: pessoas que falta a segunda toma

```
?- demo(pessoasFaltaSegundaFaseVacinacao(X),V).
X = [(2, 'Telmo'), (3, 'Tiago'), (11, 'Rui')],
V = verdadeiro .
```

1.7. Teste do predicado: última data de vacinação

```
?- demo(ultimaDataFase(1,X),V).
X = data(1, 2, 2021),
V = verdadeiro .

?- demo(ultimaDataFase(2,X),V).
V = falso .

?- demo(ultimaDataFase(3,X),V).
V = falso .
```

1.8. Teste do predicado: pessoas que tomaram x vacina

```
|    demo(pessoasQueTomaramXvacina('astrazeneca',X),V).
X = [(11, 'Rui'), (10, 'Eva'), (9, 'Cristina'), (8, 'Carlota'), (7, 'Augusta'), (2, 'Telmo')],
V = verdadeiro .
```

1.9. Teste do predicado: utente vacinado por staff

```
?- demo(utenteVacinadoStaff(8,_,7,_),V).
V = desconhecido.

?- demo(utenteVacinadoStaff(4,_,8,_),V).
V = verdadeiro .
```

1.10. Teste do predicado: lugar onde foi vacinado

```
?- demo(lugarOndeVacinado(7,ID_CentroSaude),V).
V = desconhecido.

?- demo(lugarOndeVacinado(8,ID_CentroSaude),V).
ID_CentroSaude = 4,
V = verdadeiro .
```