



UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO DE ENGENHARIA  
INFORMÁTICA

SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO  
(2º SEMESTRE - 2020/21)

Relatório SRCR - TP2  
Grupo Nº 31

A89561 Gustavo Lourenço

A77667 Luís Pereira

A81366 João Neves

A89501 Martim Almeida

A84241 Luís Maia

3 de maio de 2021

# Resumo

O presente relatório é constituído por uma descrição e demonstração da primeira fase e segunda fase do trabalho prático proposto no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

O objetivo desta 2<sup>a</sup> Parte do Trabalho é a utilização da extensão à programação em lógica, usando a linguagem de programação em lógica *PROLOG*, seguindo a representação de conhecimento imperfeito, construindo um sistema de representação de conhecimento e raciocínio capaz de representar um universo de discurso na área da vacinação da população portuguesa no contexto COVID.

Com isto, será apresentado ao longo deste relatório toda a base de conhecimento construída, tanto positiva como negativa, será também explicado todo o processo e conteúdo referente ao conhecimento imperfeito. Serão também apresentados todos os invariantes estruturais e referenciais que permitem garantir uma correta evolução e involução da base de conhecimento.

Para além do já referido, será ainda apresentada a temática de evolução e involução, assim como o sistema de inferência, capaz de implementar os mecanismos de raciocínio presentes no sistema.

Por fim, será ainda apresentada uma secção de funcionalidades, pedidas pela equipa docente na primeira parte do Trabalho, assim como algumas funcionalidades extra que se revelaram úteis para testes e *debug* ao longo do desenvolvimento do mesmo.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>5</b>
<b>2</b>	<b>Preliminares</b>	<b>6</b>
<b>3</b>	<b>Descrição do Trabalho e Análise de Resultados</b>	<b>8</b>
3.1	Base de Conhecimento . . . . .	9
3.1.1	Conhecimento Perfeito . . . . .	9
3.1.2	Conhecimento Imperfeito . . . . .	10
3.2	Invariantes sobre conhecimento . . . . .	12
3.2.1	Invariantes sobre Utentes . . . . .	12
3.2.2	Invariantes sobre Centro de Saúde . . . . .	14
3.2.3	Invariantes sobre Staff . . . . .	15
3.2.4	Invariante sobre Vacinação COVID . . . . .	16
3.3	Evolução e involução de conhecimento . . . . .	18
3.3.1	Introdução . . . . .	18
3.3.2	Predicados relevantes de evolução . . . . .	18
3.3.3	Predicados relevantes de involução . . . . .	19
3.4	Sistema de Inferência . . . . .	20
<b>4</b>	<b>Funcionalidades do Programa</b>	<b>21</b>
4.1	Funcionalidades Principais . . . . .	21
4.1.1	Definição de Fases de Vacinação . . . . .	21
4.1.2	Identificar pessoas não vacinadas . . . . .	22
4.1.3	Identificar pessoas vacinadas . . . . .	22
4.1.4	Identificar as pessoas vacinadas indevidamente . . . . .	23
4.1.5	Identificar pessoas candidatas a vacinação . . . . .	24
4.1.6	Identificar pessoas a quem falta a segunda toma da vacina . . . . .	24
4.2	Funcionalidades Extras . . . . .	25
<b>5</b>	<b>Conclusão</b>	<b>27</b>
<b>6</b>	<b>Referências</b>	<b>28</b>
6.1	Referências Bibliográficas . . . . .	28
6.2	Referências Eletrónicas . . . . .	29
<b>A</b>	<b>Base de Conhecimento</b>	<b>30</b>

# Lista de Figuras

2.1	Declarações Iniciais . . . . .	7
3.1	Definições Iniciais . . . . .	9
3.2	Conhecimento Perfeito Negativo . . . . .	9
3.3	Conhecimento Imperfeito Incerto . . . . .	10
3.4	Conhecimento Imperfeito Impreciso . . . . .	11
3.5	Conhecimento Imperfeito Interdito . . . . .	11
3.6	Invariantes sobre Utente . . . . .	12
3.7	Invariantes sobre Utente . . . . .	12
3.8	Invariantes sobre Utente . . . . .	12
3.9	Invariantes sobre Utente . . . . .	12
3.10	Invariantes sobre Utente . . . . .	12
3.11	Invariantes sobre Utente . . . . .	13
3.12	Invariantes sobre Centro de Saúde . . . . .	14
3.13	Invariantes sobre Centro de Saúde . . . . .	14
3.14	Invariantes sobre Centro de Saúde . . . . .	14
3.15	Invariantes sobre Centro de Saúde . . . . .	14
3.16	Invariantes sobre Centro de Saúde . . . . .	14
3.17	Invariantes sobre Centro de Saúde . . . . .	14
3.18	Invariantes sobre Staff . . . . .	15
3.19	Invariantes sobre Staff . . . . .	15
3.20	Invariantes sobre Staff . . . . .	15
3.21	Invariantes sobre Staff . . . . .	15
3.22	Invariantes sobre Staff . . . . .	15
3.23	Invariantes sobre Staff . . . . .	15
3.24	Invariante Vacinação Covid . . . . .	16
3.25	Invariante Vacinação Covid . . . . .	16
3.26	Invariante Vacinação Covid . . . . .	16
3.27	Invariante Vacinação Covid . . . . .	16
3.28	Invariante Vacinação Covid . . . . .	16
3.29	Invariante Vacinação Covid . . . . .	16
3.30	Invariante Vacinação Covid . . . . .	16
3.31	Invariante Vacinação Covid . . . . .	16
3.32	Invariante Vacinação Covid . . . . .	17
3.33	Invariante Vacinação Covid . . . . .	17
3.34	Invariante Vacinação Covid . . . . .	17
3.35	Predicados de Evolução . . . . .	18
3.36	Predicados de Involução . . . . .	19
3.37	Sistema de Inferência . . . . .	20
4.1	Funcionalidade de Definição de Fases de Vacinação . . . . .	22
4.2	Funcionalidade de Identificação de Pessoas não vacinadas . . . . .	22
4.3	Funcionalidade de Identificação de Pessoas vacinadas . . . . .	23
4.4	Funcionalidade de Identificação de Pessoas vacinadas indevidamente . . . . .	23
4.5	Funcionalidade de Identificação de Pessoas candidatas a vacinação . . . . .	24
4.6	Funcionalidade de Identificação de Pessoas a quem falta a segunda toma da vacina . . . . .	24

4.7	Funcionalidades Extras . . . . .	26
A.1	Base de Conhecimento Positivo . . . . .	30
A.2	Base de Conhecimento Negativo . . . . .	31

# Capítulo 1

## Introdução

Uma das limitações da lógica clássica é a incapacidade de expressar situações de senso comum, para além disso, também a programação em lógica tradicional não permite a representação de informação incompleta. Por isto, é necessário estender a lógica de forma a permitir o reconhecimento de conhecimento imperfeito. Com isto, foi proposto pela equipa docente, utilizar a base de conhecimento da fase anterior e actualizá-la de forma a que seja possível representar conhecimento imperfeito.

Com isto, poderá ser possível, ao longo deste relatório, conhecer todas as funcionalidades que culminaram numa representação de um universo de discurso na área da vacinação da COVID, sendo este universo capaz de representar conhecimento perfeito (positivo e negativo), terá a capacidade de representar conhecimento imperfeito, assim como a manipulação de invariantes de forma a controlar a inserção e remoção de conhecimento do sistema.

Por fim, teremos ainda a explicação de como lidamos com a evolução e involução, assim como o sistema de inferência que implementa os mecanismos de raciocínio necessários.

## Capítulo 2

# Preliminares

O presente trabalho foi desenvolvido recorrendo ao conhecimento adquirido nas aulas da presente UC, tirando partido da linguagem *Prolog*.

*Prolog* é uma linguagem declarativa de lógica de primeira-ordem, sendo generalizada em duas frases:

*"Providing axioms that indicate some facts about the world."*;

*"Providing rules that allow to infer other facts about the world"*.

A sintaxe de Prolog é bastante semelhante a outras linguagens de programação populares, contudo existem algumas diferenças, que podem ser sistematizadas nas seguintes:

### 1. Objetos

- (a) Facto - Constata algo que se reconhece e se sabe verdadeiro;
- (b) Predicado - Implementa uma relação;
- (c) Regra - Utilizada para definir um novo predicado;
- (d) Invariante - Regra inviolável ao longo da execução do programa.

### 2. Pontuação:

- (a) . Utilizado para terminar uma declaração;
- (b) , Utilizado para representar a conjunção lógica;
- (c) :- Utilizado para representar a implicação conversiva lógica ( $\leftarrow$ );
- (d) ; Utilizado para representar a disjunção inclusiva lógica;
- (e) // Utilizado para representar a unificação de dois conjuntos.

### 3. Predicados:

- (a) *findall* - Constrói a lista com os termos que respeitam condições argumento.

Neste trabalho foi adotada a programação em lógica associada à existência de conhecimento imperfeito. Isto é, esta programação utilizará três diferentes valores lógicos:

- Verdadeiro;
- Falso;
- Desconhecido.

De forma a dar início ao desenvolvimento do projeto, foram usadas três *flags* específicas da linguagem *Prolog*, que evitam *warnings* ou erros de carregamento/consulta do ficheiro usado para desenvolver o projeto, podendo ser assim executado corretamente e de acordo com os parâmetros definidos para o conhecimento.

```
:- set_prolog_flag( disjoint_warnings,off ).  
:- set_prolog_flag( single_var_warnings,off ).  
:- set_prolog_flag( unknown,fail ).
```

Figura 2.1: Declarações Iniciais

A fase anterior estava assente numa programação em lógica não estendida, onde todo o conhecimento era representado por conhecimento perfeito positivo, baseando-se em três pressupostos definidos:

1. Pressuposto do Mundo Fechado → Toda a informação que não existe na base de conhecimento é considerada falsa;
2. Pressuposto dos Nomes Únicos → Dois campos diferentes designam, necessariamente, duas entidades diferentes do universo de discurso;
3. Pressuposto do Domínio Fechado → Não existem mais entidades para além das representadas.

Estas ideologias, que sustentaram a fase anterior do trabalho, neste momento terão de ser substituídas, visto que aplicando os conceitos acima referenciados num contexto real, teríamos vários problemas. Por exemplo, no caso da inscrição de utente na base de conhecimento, segundo o Pressuposto do Mundo Fechado, caso não seja possível saber todos os parâmetros do utente, este não poderia fazer parte da nossa base de conhecimento. Desta forma, foi necessário estender a nossa base de conhecimento de forma a implementar, por exemplo, o pressuposto do Domínio Aberto (PDA), que garante a existência de mais objectos para além dos referenciados explicitamente na nossa base de conhecimento.

Com isto, a Extensão à Programação em Lógica estendida, oferece-nos a possibilidade de representar informação incompleta. Por isso, existem agora três tipos de conclusões para uma questão: Verdadeira, Falsa e o Desconhecido, que representa a informação à qual não há forma que permita saber se a informação é falsa ou verdadeira.

Concluindo, é ainda possível especificar o conhecimento imperfeito, dividindo-o em 3 tipos:

- **Incerto:** utilizado quando não sabemos algum dos campos, trata-se de um valor indefinido dentro de um conjunto ilimitado de hipóteses;
- **Impreciso:** acontece quando não temos a certeza qual das informações para um determinado campo é o correto, ou seja, existe um número limitado de hipóteses para um determinado campo;
- **Interdito:** utilizado quando não é de todo possível saber o valor de um campo específico.



## Capítulo 3

# Descrição do Trabalho e Análise de Resultados

O Presente trabalho encontra-se dividido em 5 partes que em conjunto constroem uma base de conhecimento manipulável e fiável.

O primeiro será a base de conhecimento, onde será guardado tudo referente às entidades, *Utente*, *Centro de Saúde*, *Staff*, *Vacinação Covid*, que serão posteriormente atualizadas através de mecanismos de evolução e involução de conhecimento.

A componente dos invariantes é a 2<sup>a</sup> parte deste trabalho e tem como objectivo compor uma lista de regras que irão ajudar a manter a integridade do nosso projeto.

De seguida temos a componente da evolução e involução, que no nosso caso so será relativo ao conhecimento perfeito, tendo como trabalho controlar a inserção e remoção de conhecimento.

Após isto, temos a componente do sistema de inferência, que é responsável por responder às inquisições do utilizador sobre informação existente na nossa base de conhecimento.

Por fim temos as Funcionalidades do Sistema, onde irá ser abordado de uma forma mais séria todas as funcionalidades propostas e a sua resolução, assim como, algumas funcionalidades extras.

### 3.1 Base de Conhecimento

Tendo em conta que o sistema foi criado de raiz pelo grupo, torna-se necessário criar uma base de conhecimento de forma a podermos testar todas as funcionalidades sobre a mesma.

Com isto, serão apresentadas de seguida as bases de conhecimento desenvolvidas.

1. utente: #idutente, Nº Seguranca\_Social, Nome, Data\_Nasc, Email, Telefone, Morada, Profisso, [Doencas\_Cronicas], #CentroSade  $\rightarrow \{V, F, D\}$
2. centro\_saude: #idcentro, Nome, Morada, Telefone, Email  $\rightarrow \{V, F, D\}$
3. staff: #Idstaff, #Idcentro, Nome, email  $\rightarrow \{V, F, D\}$
4. vacinaco\_Covid: #Staff, #utente, Data, Vacina, Toma  $\rightarrow \{V, F\}$

```
:- op( 900,xfy,'::' ).  
:- dynamic utente/10.  
:- dynamic centro_saude/5.  
:- dynamic staff/4.  
:- dynamic vacinacao_covid/5.
```

Figura 3.1: Definies Iniciais

Tendo em conta os requisitos da atual fase do trabalho, foi necessrio atualizar o domnio de solues de forma a suportar a representao de informao incompleta. Por isso, os resultados das questes podem ser Verdadeira, Falsa ou Desconhecida.

Aps a anlise do trabalho proposto, decidimos que todos os valores que no esto presentes na nossa base de conhecimento nem tenham uma exceo associada so considerados falsos.

```
-utente(Id,Seg,Nome,Data,Email,Nr,Mor,Prof,Doencas,Cs) :-  
    nao(utente(Id,Seg,Nome,Data,Email,Nr,Mor,Prof,Doencas,Cs)),  
    nao(excecao(utente(Id,Seg,Nome,Data,Email,Nr,Mor,Prof,Doencas,Cs))).  
  
-centro_saude(Id,Nome, Morada, Telefone, Email) :-  
    nao(centro_saude(Id,Nome, Morada, Telefone, Email)),  
    nao(excecao(centro_saude(Id,Nome, Morada, Telefone, Email))).  
  
-staff(Id,Centro,Nome, Email) :-  
    nao(staff(Id,Centro,Nome, Email)),  
    nao(excecao(staff(Id,Centro,Nome, Email))).  
  
-vacinacao_covid(Staff, Utente, Data, Vacina, Toma) :-  
    nao(vacinacao_covid(Staff, Utente, Data, Vacina, Toma)),  
    nao(excecao(vacinacao_covid(Staff, Utente, Data, Vacina, Toma))).
```

Figura 3.2: Conhecimento Perfeito Negativo

#### 3.1.1 Conhecimento Perfeito

O conhecimento perfeito pode ser caracterizado em dois pontos distintos:

- Conhecimento Positivo;
- Conhecimento Negativo.

Este tipo de conhecimento permite-nos saber tudo em relação a uma certa entidade, sendo cada entidade representada por um predicado na forma normal, distinguindo as entidades negativas utilizando o símbolo de negação antes do predicado em questão.

De forma a completar a nossa base de conhecimento decidimos adicionar conhecimento positivo e negativo para todas as fontes de conhecimento existentes, isto é, utente, centro de saúde, staff e vacinação\_covid.

De forma a demonstrar o trabalho realizado, segue em anexo o código correspondente à inserção deste tipo de conhecimento.

### 3.1.2 Conhecimento Imperfeito

Como já foi mencionado anteriormente, ao contrário da fase anterior, temos a possibilidade de representar respostas desconhecidas na nossa base de conhecimento. Por isso, todas as respostas verdadeiras e falsas serão consideradas conhecimento perfeito, sendo que de seguida abordaremos as respostas desconhecidas que representam o conhecimento imperfeito, tendo em conta que existem três tipos de conhecimento desconhecido:

- Conhecimento Incerto;
- Conhecimento Impreciso;
- Conhecimento Interdito.

Com isto, decidimos povoar a nossa base desconhecimento de forma a cobrir todos os tipos de conhecimento imperfeito.

#### Conhecimento Imperfeito Incerto

O conhecimento imperfeito incerto traduz-se na incerteza do valor de um campo de predicado sendo que existe um conjunto ilimitado de hipóteses para esse mesmo campo. Por exemplo, na figura 3.3 temos o caso de um utente Xuan ao qual não se sabe qual é a sua morada.

De forma a implementar este tipo de conhecimento, é necessário criar um utente em que o campo da morada é um valor incerto (neste caso, representado por *unknown01*) e uma exceção de forma a que quando a base de conhecimento for questionada sobre a morada do utente Xuan, esta responde desconhecido. O mesmo processo aplica-se para as restantes bases de conhecimento.

```
%-----
% Nao se sabe a morada do senhor Xuan
utente(13,3412356,"Xuan Sintra",date(1955,10,05),"Xuan@gmail.com",965170412,unknown01,"Profissional de saude",[],1).
excecao(utente(Id,_,_,_,_,Morada,_,_)) :-
    utente(Id,_,_,_,unknown01,_,_).
%-----
% Centro de Saude -> Conhecimento Imperfeito Incerto
% Nao se sabe a morada do centro de saude Hospital de Amarante
centro_saude(4,"Hospital de Amarante",ukmorada1,213659483,"centro4@gmail.com").
excecao(centro_saude(Id,_,_,Morada,_,_)) :-
    centro_saude(Id,_,_,ukmorada1,_,_).
%-----
% Staff -> Conhecimento Imperfeito Incerto
% Nao se sabe o email da Ana
staff(7,1,"Ana Rocha",ukmail01).
excecao(staff(Id,_,_,Email)) :-
    staff(Id,_,_,ukmail01).
```

Figura 3.3: Conhecimento Imperfeito Incerto

#### Conhecimento Imperfeito Impreciso

O conhecimento imperfeito impreciso consiste na incerteza de um determinado campo, dentro de um conjunto limitado de hipóteses.

Na figura que se segue abaixo, podemos observar mais uma vez o caso do Utente Xuan. No entanto, desta vez existem duas possibilidades para a sua morada: *unknown01* e *unknown02*. Para criar este tipo de conhecimento é necessário que o utente em questão já exista na base de conhecimento, sendo

que será então necessário criar exceções para as alternativas de morada, sendo que quando a base de conhecimento for questionada sobre a morada do Utente Xuan, esta responde *Desconhecido* caso a morada seja uma das alternativas contempladas nas exceções e falso noutro caso.

Os restantes casos são análogos ao explicado acima.

```
% Utentes -> Conhecimento Imperfeito Impreciso
%-----
% nao se sabe se o senhor Xuan tem a morada unknown01 ou unknown02
excecao( utente(14,3412356,"Xuan Sintra",date(1955,10,05),"Xuan@gmail.com",965170412,unknown01,"Profissional de saude",[],1) ).
excecao( utente(14,3412356,"Xuan Sintra",date(1955,10,05),"Xuan@gmail.com",965170412,unknown02,"Profissional de saude",[],1) ).

% Centro de Saude -> Conhecimento Imperfeito Impreciso
%-----
% nao se sabe se a morada do Hospital padre é ukmorada2 ou ukmorada3
excecao( centro_saude(5,"Hospital de Padre",ukmorada2,213659483,"centro5@email.com") ).
excecao( centro_saude(5,"Hospital de Padre",ukmorada3,213659483,"centro5@email.com") ).

% Staff -> Conhecimento Imperfeito Impreciso
%-----
% nao se sabe se o email da maria é ukmail02 ou ukmail03
excecao( staff(8,2,"Maria Carvalho",ukmail02) ).
excecao( staff(8,2,"Maria Carvalho",ukmail03) ).
```

Figura 3.4: Conhecimento Imperfeito Impreciso

### Conhecimento Imperfeito Interdito

Em relação ao conhecimento imperfeito interdito, este consiste na impossibilidade de conhecer o valor de um campo de predicado.

De forma explicativa, consideremos o exemplo do utente, que se segue na figura 3.5.

De forma a criarmos conhecimento interdito, temos de primeiro criar o utente, sendo que o campo da morada é um valor nulo (neste caso, representado por *nulnumber01*). Para além disto é necessário criar uma exceção, para que este tipo de conhecimento não se insira com o resto do conhecimento (positivo,negativo), assim como é necessário criar uma instância de *nulo()* de forma que, ao construirmos o invariante que não permite a evolução de conhecimento errado, este tipo de conhecimento seja considerado como informação impossível de conhecer e consequentemente não adicionada.

```
% Utentes -> Conhecimento Imperfeito Interdito
%-----
% e impossivel saber a morada do senhor Xuan
utente(15,3412356,"Xuan Sintra",date(1955,10,05),"Xuan@gmail.com",nulnumber01,"Rua top","Profissional de saude",[],1).
excecao(utente(Id,_,_,Nr,_,_,_)) :- utente(Id,_,_,nulnumber01,_,_,_).
nulo(nulnumber01).

% Centro de Saude -> Conhecimento Imperfeito Interdito
%-----
% e impossivel saber a morada do Hospital de Guimaraes
centro_saude(6,"Hospital de Guimaraes",nuladd01,213659483,"centro6@email.com").
excecao(centro_saude(Id,_,Add,_,_)) :- staff(Id,_,nuladd01,_,_).
nulo(nuladd01).

% Staff -> Conhecimento Imperfeito Interdito
%-----
% é impossivel saber o email da Ines
staff(9,2,"Ines Cardoso",nulmail01).
excecao(staff(Id,_,_,Email)) :- staff(Id,_,_,nulmail01).
nulo(nulmail01).
```

Figura 3.5: Conhecimento Imperfeito Interdito

## 3.2 Invariantes sobre conhecimento

De forma a manter a integridade da base de conhecimento e também a sua fiabilidade, recorreremos à aplicação de invariantes que ditem as regras de inserção e remoção de conhecimento.

A motivação da existência dos invariantes será por razões lógicas, isto é, respeitando as regras de programação em lógica, mas também respeitar as regras do sistema de vacinação.

### 3.2.1 Invariantes sobre Utentes

O 1º Invariante não permite a inserção de conhecimento repetido, evitando a existência de vários Utentes com o mesmo ID.

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

Figura 3.6: Invariantes sobre Utente

O 2º Invariante não permite a inserção de um Utente num Centro de Saúde não existente.

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

Figura 3.7: Invariantes sobre Utente

O 3º Invariante não permite a remoção de um Utente caso ainda exista Registo de Vacinação.

```
-utente(Id,_,_,_,_,_,_,_) :: (solucoes( Id,(vacinacao_covid(_,Id,_,_,_)),S ),
                                comprimento(S, 0)
                                ).
```

Figura 3.8: Invariantes sobre Utente

O 4º Invariante não permite a inserção de conhecimento contraditório.

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

Figura 3.9: Invariantes sobre Utente

O 5º Invariante não permite a inserção de uma negação que já exista.

```
+(-utente(Id,Seg,Nome,Data,Email,Nr,Mor,Prof,Doencas,Cs)) :: (solucoes( Id,(-utente(Id,Seg,Nome,Data,Email,Nr,Mor,Prof,Doencas,Cs)),S ),
                                comprimento( S,N ),
                                N < 1).
```

Figura 3.10: Invariantes sobre Utente

O 6º Invariante não permite a alteração de conhecimento interdito.

```
+utente(Id,_,_,_,Email,Nr,Mor,_,_,_) :: (solucoes(Email,((utente(Id,_,_,_,Email,Nr,Mor,_,_,_)), nulo(Email)),S1 ),  
    comprimento( S1,0),  
    solucoes(Nr,(utente(Id,_,_,_,Email,Nr,Mor,_,_,_), nulo(Nr)),S2 ),  
    comprimento( S2,0),  
    solucoes(Mor,(utente(Id,_,_,_,Email,Nr,Mor,_,_,_), nulo(Mor)),S3),  
    comprimento( S3,0)).
```

Figura 3.11: Invariantes sobre Utente

### 3.2.2 Invariantes sobre Centro de Saúde

O 1º Invariante não permite a inserção de conhecimento repetido, evitando a existência de vários Centros de Saúde com o mesmo ID.

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

Figura 3.12: Invariantes sobre Centro de Saúde

O 2º Invariante não permite a remoção do Centro de Saúde caso ainda exista algum Staff empregue no Centro de Saúde.

```
-centro_saude(Id,_,_,_) :: (solucoes( U,(staff(U,Id,_,_)),S ),
                             comprimento(S, 0)
                             ).
```

Figura 3.13: Invariantes sobre Centro de Saúde

O 3º Invariante não permite a remoção do Centro de Saúde caso ainda exista algum Utente que frequente o Centro de Saúde.

```
-centro_saude(Id,_,_,_) :: (solucoes( U,(utente(U,_,_,_,_,_,_,Id)),S ),
                             comprimento(S, 0)
                             ).
```

Figura 3.14: Invariantes sobre Centro de Saúde

O 4º Invariante não permite a inserção de conhecimento contraditório.

```
+(-centro_saude(Id,_,_,_)) :: (solucoes( Id,(centro_saude(Id,_,_,_)),S ),
                             comprimento( S,N ),
                             N == 0).
```

Figura 3.15: Invariantes sobre Centro de Saúde

O 5º Invariante não permite a inserção de uma negação que já exista.

```
+(-centro_saude(Id,Nome, Morada, Telefone, Email)) :: (solucoes( Id,(-centro_saude(Id,Nome, Morada, Telefone, Email)),S ),
                             comprimento( S,N ),
                             N < 1).
```

Figura 3.16: Invariantes sobre Centro de Saúde

O 6º Invariante não permite a alteração de conhecimento interdito.

```
+centro_saude(Id,_, Morada, Telefone, Email) :: (solucoes(Email,((centro_saude(Id,_, Morada, Telefone, Email)), nulo(Email)),S1 ),
                             comprimento( S1,0),
                             solucoes(Nr,(centro_saude(Id,_, Morada, Telefone, Email), nulo(Nr)),S2 ),
                             comprimento( S2,0),
                             solucoes(Mor,(centro_saude(Id,_, Morada, Telefone, Email), nulo(Mor)),S3),
                             comprimento( S3,0)).
```

Figura 3.17: Invariantes sobre Centro de Saúde

### 3.2.3 Invariantes sobre Staff

O 1º Invariante não permite a inserção de conhecimento repetido, evitando a existência de vários Staff com o mesmo ID.

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

Figura 3.18: Invariantes sobre Staff

O 2º Invariante não permite a inserção de um Staff num Centro de Saúde não existente.

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

Figura 3.19: Invariantes sobre Staff

O 3º Invariante não permite a remoção de um Staff caso ainda exista Registo de Vacinação efetuado por ele.

```
-staff(Id,_,_,_) :: (solucoes( Id,(vacinacao_covid(Id,_,_,_)),S ),  
                    comprimento(S, 0)  
                    ).
```

Figura 3.20: Invariantes sobre Staff

O 4º Invariante não permite a inserção de conhecimento contraditório.

```
+(-staff(Id,_,_,_)) :: (solucoes( Id,(staff(Id,_,_,_)),S),  
                        comprimento( S,N ),  
                        N == 0).
```

Figura 3.21: Invariantes sobre Staff

O 5º Invariante não permite a inserção de uma negação que já exista.

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

Figura 3.22: Invariantes sobre Staff

O 6º Invariante não permite a alteração de conhecimento interdito.

```
+staff(Id,_,_,Email) :: (solucoes(Email,((staff(Id,_,_,Email))), nulo(Email)),S1 ),  
                        comprimento( S1,0)).
```

Figura 3.23: Invariantes sobre Staff



### 3.2.4 Invariante sobre Vacinação COVID

O 1º Invariante não permite a inserção de Vacinação caso o Staff não exista.

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

Figura 3.24: Invariante Vacinação Covid

O 2º Invariante não permite a inserção de Vacinação caso o Utente não exista.

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

Figura 3.25: Invariante Vacinação Covid

O 3º Invariante não permite a inserção de Vacinação caso o Utente seja excecional.

```
+vacinacao_covid(_,U,_,_) :: (solucoes( U,(excecao(utente(U,_,_,_,Nr,_,_,_))),S ), comprimento( S,N ), N == 0).
```

Figura 3.26: Invariante Vacinação Covid

O 4º Invariante evita a inserção de números de Tomas incorretas.

```
+vacinacao_covid(_,_,_,_,T) :: (T > 0, T =< 2).
```

Figura 3.27: Invariante Vacinação Covid

O 5º Invariante evita a inserção de Vacinação quando o Staff e o Utente encontram-se em Centros de Saúde diferentes.

```
+vacinacao_covid(S,Id,_,_) :: (utente(Id,_,_,_,_,_,_,Centro), staff(S,Centro,_,_)).
```

Figura 3.28: Invariante Vacinação Covid

O 6º Invariante apenas permite a inserção da Segunda Toma da Vacina por um Utente quando existe registo da Primeira Toma.

```
+vacinacao_covid(_,Id,_,_,2) :: vaccinacao_covid(_,Id,_,_,1).
```

Figura 3.29: Invariante Vacinação Covid

O 7º Invariante apenas permite a inserção da Primeira Toma por um Utente só uma vez.

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

Figura 3.30: Invariante Vacinação Covid

O 8º Invariante apenas permite a inserção de dois registos de Vacinação por Utente, ou seja, a inserção das duas Tomas.

```
+vacinacao_covid(_,Id,_,_) :: (solucoes( T,(vacinacao_covid(_,Id,_,_,T)),S ),comprimento( S,N ), N =< 2).
```

Figura 3.31: Invariante Vacinação Covid

O 9º Invariante não permite a remoção da Primeira Toma quando temos conhecimento da Segunda Toma.

```
-vacinacao_covid(_,Id,_,_,1) :: nao(vacinacao_covid(_,Id,_,_,2)).
```

Figura 3.32: Invariante Vacinação Covid

O 10º Invariante não permite a inserção de Conhecimento Contraditório.

```
+(-vacinacao_covid(Staff, Utente, Data, Vacina, Toma)) :: (solucoes( Id,(vacinacao_covid(Staff, Utente, Data, Vacina, Toma)),S ),  
    comprimento( S,N ),  
    N == 0).
```

Figura 3.33: Invariante Vacinação Covid

O 11º Invariante não permite a inserção de uma negação que já exista.

```
+(-vacinacao_covid(Staff, Utente, Data, Vacina, Toma)) :: (solucoes( Id,(-vacinacao_covid(Staff, Utente, Data, Vacina, Toma)),S ),  
    comprimento( S,N ),  
    N <= 1).
```

Figura 3.34: Invariante Vacinação Covid

## 3.3 Evolução e involução de conhecimento

### 3.3.1 Introdução

A introdução/remoção de conhecimento na nossa base de dados é controlada por diversos predicados de evolução que têm como objetivo regular o seu modo de inserção e efeito na base de conhecimento realizada por nós.

### 3.3.2 Predicados relevantes de evolução

- *evolucao* É utilizado para adicionar conhecimento à nossa base de conhecimento;
- *solucoes* É utilizado como alternativa ao predicado *findall*.
- *teste* É utilizado para testar se todos os termos de uma lista têm o valor *Verdadeiro*;
- *insercao* É utilizado para adicionar um Termo na nossa base de conhecimentos ou, no caso da inserção ser inválida, não permitir a sua adição.

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

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

teste([]).
teste([R|Lr]) :- R, teste(Lr).

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

Figura 3.35: Predicados de Evolução

### 3.3.3 Predicados relevantes de involução

- *involucao* É utilizado para remover conhecimento da nossa base de conhecimento;
- *remove* É utilizado para remover um Termo da nossa base de conhecimento ou, no caso da remoção ser inválida, não permitir a sua remoção.

```
involucao(Termo) :- Termo,  
                    solucoes(Invariante, -Termo::Invariante, Lista),  
                    remove(Termo),  
                    teste(Lista).  
  
remove(Termo) :- retract(Termo).  
remove(Termo) :- assert(Termo), !, fail.  
  
solucoes(X,P,S):- findall(X,P,S).  
  
teste([]).  
teste([R|Lr]) :- R, teste(Lr).
```

Figura 3.36: Predicados de Involução

### 3.4 Sistema de Inferência

Devido à programação em lógica estendida, foi necessário criar um sistema de inferência capaz de lidar com os três valores lógicos do sistema:

- Verdadeiro, caso o conhecimento exista explicitamente na nossa base de conhecimento;
- Falso, caso a negação do conhecimento exista através do predicado de negação (*-Questao*);
- Desconhecido, caso exista conhecimento imperfeito acerca da Questão a ser testada.

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

Figura 3.37: Sistema de Inferência

## Capítulo 4

# Funcionalidades do Programa

### 4.1 Funcionalidades Principais

#### 4.1.1 Definição de Fases de Vacinação

A seguinte Funcionalidade é uma das mais fundamentais do trabalho proposto, visto que organiza a vacinação por fases e faz uma distribuição dos utentes pelos seguintes critérios:

- **Fase 1:** Destina-se a pessoas com mais de 50 anos e duas ou mais patologias associadas ou pessoas com Empregos Excepcionais (Médico, Enfermeiro, Profissional de Saúde, Profissional de Lar, Profissional das Forças de Segurança, Profissional das Forças Armadas);
- **Fase 2:** Destina-se a pessoas com idade igual ou superior a 65 anos ou pessoas com idade igual ou superior a 50 anos com patologias associadas;
- **Fase 3:** Destina-se às restantes Pessoas.

O predicado *excepcoesEmprego* é utilizado para adicionarmos à Base de Conhecimento, os Empregos Excepcionais.

O predicado *aceiteNaFase* é utilizado para verificar se o utente foi aceite numa determinada Fase de Vacinação (verificando os critérios acima apresentados).

O predicado *faseVacinacao* é utilizado para reduzir a lista de Fases que o utente é aceite para a Fase respetiva.

Por fim, o predicado *personasNaFase* é utilizado para obter uma lista com os IDs dos utentes que irão tomar a Vacina numa determinada Fase.

```

excepcoesEmprego("Médico").
excepcoesEmprego("Enfermeiro").
excepcoesEmprego("Profissional de saúde").
excepcoesEmprego("Profissional de lar").
excepcoesEmprego("Profissional das forças de segurança").
excepcoesEmprego("Profissional das forças armadas").

aceiteNaFase(Id,1) :- utente(Id,_,Data,_,_,Doencas,_),
    idade(Data,Age), Age >= 50,
    comprimento(Doencas,Comp), Comp >= 2.

aceiteNaFase(Id,1) :- utente(Id,_,_,_,_,F,_), excepcoesEmprego(F).

aceiteNaFase(Id,2) :- utente(Id,_,Data,_,_,Doencas,_), idade(Data,Age), Age >= 65,
    nao(aceiteNaFase(Id,1)).

aceiteNaFase(Id,2) :- utente(Id,_,Data,_,_,Doencas,_), idade(Data,Age), Age >= 50,
    comprimento(Doencas,Comp), Comp > 0,
    nao(aceiteNaFase(Id,1)).

aceiteNaFase(Id,3) :- utente(Id,_,_,_,_,_),
    nao(aceiteNaFase(Id,1)),
    nao(aceiteNaFase(Id,2)).

accMin([],X,X).
accMin([X|L],A,Min) :- X < A, accMin(L,X,Min).
accMin([X|L],A,Min) :- X >= A, accMin(L,A,Min).
min([X|L],A) :- accMin(L,X,A).

faseVacinao(Id,F) :- utente(Id,_,_,_,_,_), (solucoes( Fase,(aceiteNaFase(Id,Fase)),Fases)), min(Fases,F).

pessoasNaFase(F,S) :- (solucoes( Id,(faseVacinao(Id,F)),S)).

```

Figura 4.1: Funcionalidade de Definição de Fases de Vacinação

#### 4.1.2 Identificar pessoas não vacinadas

Para identificar as pessoas não vacinadas utilizamos os seguintes predicados:

- *utenteIdNames*: É utilizado para, através de uma lista de IDs de utentes, obtermos os respectivos nomes dos utentes identificados nessa lista;
- *naoVacinadosAux*: É utilizado, através de uma lista de IDs de utentes, para verificar se existem registos de vacinação com esses IDs e, caso não existam, são adicionados a uma lista, ou seja, obtém uma lista com os IDs de utentes não vacinados;
- *naoVacinados*: Através da utilização dos predicados descritos anteriormente, este predicado é utilizado para obter uma lista com os nomes dos utentes não vacinados.

```

utenteIdNames([],S,S).
utenteIdNames([H|T],Acc,S) :- utente(H,_,Nome,_,_,_), utenteIdNames(T,[Nome|Acc],S).
utenteIdNames([H|T],Acc,S) :- utenteIdNames(T,Acc,S).

naoVacinadosAux([],S,S).
naoVacinadosAux([H|T],Acc,S) :- vacinao_covid(_,H,_,_), naoVacinadosAux(T,Acc,S).
naoVacinadosAux([H|T],Acc,S) :- naoVacinadosAux(T,[H|Acc],S).

naoVacinados(S) :- solucoes(U,(utente(U,_,_,_,_,_)),Ids ), naoVacinadosAux(Ids,[],IdsNVac), utenteIdNames(IdsNVac,[],S).

```

Figura 4.2: Funcionalidade de Identificação de Pessoas não vacinadas

#### 4.1.3 Identificar pessoas vacinadas

Para identificar as pessoas vacinadas utilizamos os seguintes predicados:

- *vacinadosAux*: É utilizado, através de uma lista de IDs de utentes, para verificar se existem registos da segunda toma da vacinação com esses IDs e, caso existam, são adicionados a uma lista, ou seja, obtém uma lista com os IDs de utentes que foram totalmente vacinados;

- *vacinados*: Através da utilização dos predicados descritos anteriormente, este predicado é utilizado para obter uma lista com os nomes dos utentes totalmente vacinados.

```
vacinadosAux([],S,S).
vacinadosAux([H|T],Acc,S) :- vacinacao_covid(_,H,_,2), vacinadosAux(T,[H|Acc],S).
vacinadosAux([H|T],Acc,S) :- vacinadosAux(T,Acc,S).

vacinados(S) :- solucoes(U,(utente(U,_,_,_,_,_,_,_)),Ids ), vacinadosAux(Ides,[],IdesNVac), utenteIdNames(IdesNVac,[],S).
```

Figura 4.3: Funcionalidade de Identificação de Pessoas vacinadas

#### 4.1.4 Identificar as pessoas vacinadas indevidamente

Para identificar as pessoas vacinadas indevidamente utilizamos os seguintes predicados:

- *todosVacinadosAux*: É utilizado através de uma Lista de IDs de utentes, para verificar se todos os utentes da lista possuem registo da segunda toma da vacina, ou seja, é utilizado, dado uma lista, verificar se todos os utentes estão vacinados;
- *todosVacinados*: Através de predicados descritos anteriormente e de um determinado número de Fase, verifica se todos os utentes dessa fase estão vacinados;
- *faseAtual*: É utilizado para obter a fase de vacinação atual, verificando se os utentes das fases anteriores já estão vacinados;
- *vacinadoIndevido*: Utilizando predicados descritos anteriormente, dado o ID de um utente, verifica se este já foi vacinado, e compara a sua fase de vacinação com a fase de vacinação atual, obtendo assim o conhecimento que indica se o utente foi vacinado indevidamente. Isto é, foi vacinado antes da sua respetiva fase;
- *vacinadosIndevidosIds*: É utilizado, através de uma lista de IDs de utentes e predicados descritos anteriormente, para verificar os IDs dos utentes vacinados indevidamente;
- *vacinadosIndevidos*: Através da utilização dos predicados descritos anteriormente, este predicado é utilizado para obter uma lista com os nomes dos utentes indevidamente vacinados.

```
todosVacinadosAux([]).
todosVacinadosAux([H|T]) :- vacinacao_covid(_,H,_,2), todosVacinadosAux(T).

todosVacinados(F) :- pessoasNaFase(F,Ids), todosVacinadosAux(Ides).

faseAtual(F) :- todosVacinados(1), todosVacinados(2), F is 3 .
faseAtual(F) :- todosVacinados(1), F is 2 .
faseAtual(F) :- nao(todosVacinados(1)), F is 1 .

vacinadoIndevido(Id) :- faseAtual(F), vacinacao_covid(_,Id,_,_), faseVacinacao(Id,Fvac), Fvac > F.

vacinadosIndevidosIds(S) :- solucoes(Id, vacinadoIndevido(Id),S).

vacinadosIndevidos(S) :- vacinadosIndevidosIds(Ides), utenteIdNames(Ides,[],S).
```

Figura 4.4: Funcionalidade de Identificação de Pessoas vacinadas indevidamente



#### 4.1.5 Identificar pessoas candidatas a vacinação

Para identificar as pessoas candidatas a vacinação utilizamos os seguintes predicados:

- *vacinadoCandidato*: Utilizando predicados descritos anteriormente, dado o ID de um utente, verifica se a sua fase de vacinação é igual ou inferior à fase de vacinação atual e, caso ainda não tenha tomado a segunda toma da vacina, classifica-o como candidato. Isto é, caso um utente tenha fase inferior ou igual à fase atual e ainda não tomou todas as doses da vacina, este é considerado um utente candidato à vacinação;
- *vacinadosCandidatosIds*: É utilizado, através de uma lista de IDs de utentes e predicados descritos anteriormente, para verificar os IDs dos utentes candidatos à vacinação;
- *vacinadosCandidatos*: Através da utilização dos predicados descritos anteriormente, este predicado é utilizado para obter uma lista com os nomes dos utentes candidatos à vacinação.

```
vacinadoCandidato(Id) :- faseAtual(F), faseVacinacao(Id,Fvac), F >= Fvac, nao(vacinacao_covid(_,Id,_,2)).  
  
vacinadosCandidatosIds(S) :- (solucoes(Id, vacinadoCandidato(Id),S)).  
vacinadosCandidatos(S) :- vacinadosCandidatosIds(Ids), utenteIdNames(Ids,[],S).
```

Figura 4.5: Funcionalidade de Identificação de Pessoas candidatas a vacinação

#### 4.1.6 Identificar pessoas a quem falta a segunda toma da vacina

Para identificar as pessoas a quem falta a segunda toma da vacinação utilizamos os seguintes predicados:

- *vacinadoIncompleteAux*: Utilizando predicados descritos anteriormente, dado uma lista com IDs de utentes, verifica quais destes utentes que só possuem registo da primeira toma da vacina, adicionando a uma lista, obtendo assim, uma lista com os IDs de utentes que só têm a primeira toma da vacina;
- *vacinadosIncompleteIds*: É utilizado, através de predicados descritos anteriormente, para verificar os IDs dos utentes a quem falta a segunda toma da vacina;
- *vacinadosIncomplete*: Através da utilização dos predicados descritos anteriormente, este predicado é utilizado para obter uma lista com os nomes dos utentes a quem falta a segunda toma da vacina.

```
vacinadosIncompleteAux([],S,S).  
vacinadosIncompleteAux([H|T],Acc,S) :- vacinacao_covid(_,H,_,1), nao(vacinacao_covid(_,H,_,2)), vacinadosIncompleteAux(T,[H|Acc],S).  
vacinadosIncompleteAux([H|T],Acc,S) :- vacinadosIncompleteAux(T,Acc,S).  
  
vacinadosIncompleteIds(S) :- solucoes(U,(utente(U,_,_,_,_,_,_)),Ids), vacinadosIncompleteAux(Ids,[],S).  
vacinadosIncomplete(S) :- vacinadosIncompleteIds(Ids), utenteIdNames(Ids,[],S).
```

Figura 4.6: Funcionalidade de Identificação de Pessoas a quem falta a segunda toma da vacina

## 4.2 Funcionalidades Extras

Foram utilizados os seguintes predicados auxiliares ao trabalho:

- *listaCentros*: É utilizado para obtermos uma lista com os nomes dos Centros de Saúde;
- *listaStaff*: É utilizado para obtermos uma lista com os nomes dos Staff de determinado Centro de Saúde;
- *utentesCentro*: É utilizado para obtermos uma lista com os nomes dos utentes que frequentam um determinado Centro de Saúde;
- *nVacinasStaff*: É utilizado, dado o ID de um Staff, para obter o número de vacinações realizadas por ele mesmo;
- *listaUtentes*: É utilizado para obtermos uma lista com os nomes dos utentes;
- *nVacinasCentroAux*: É utilizado, dado uma lista de IDs de Staff, e através de predicados descritos anteriormente, para obter o número total de vacinações efetuadas pelo Staff de um Centro de Saúde;
- *nVacinasCentro*: É utilizado, através de predicados descritos anteriormente, para obter o número total de vacinações efetuadas num determinado Centro de Saúde;
- *year*: É utilizado para obter o ano de uma data;
- *month*: É utilizado para obter o mês de uma data;
- *day*: É utilizado para obter o dia de uma data;
- *yearsBetweenDates*: É utilizado para calcular a diferenças de anos entre duas datas;
- *bigDate2Small*: É utilizado para converter uma data para que esta só contenha os valores de ano, mês e dia;
- *actualDate*: É utilizado para obter a data atual do sistema através de predicados descritos anteriormente;
- *idade*: É utilizado para obter a idade de uma pessoa, dado a sua data nascimento e através de predicados descritos anteriormente.

```

% Centros de saude disponiveis
listaCentros(S) :- solucoes(C ,centro_saude(_,C,_,_), S).

% Staff disponivel em certo centro de saude por Id
listaStaff(Id,S) :- solucoes(Nome ,staff(_,Id,Nome,_), S).

% Utentes de cada Centro
utentesCentro(Id,Centros) :- solucoes(Nome ,utente(_,_,Nome,_,_,_,_,Id), Centros).

% # Vacinas administradas por cada membro staff
nVacinasStaff(Staff,N) :- solucoes(Staff ,vacinacao_covid(Staff,_,_,_), S),comprimento( S,N ).

% Lista de Utentes
listaUtentes(S) :- solucoes(Nome ,utente(_,_,Nome,_,_,_,_,_), S).

% # de Vacinas administradas em cada centro de saude
nVacinasCentro(Id,S) :- solucoes(Staff ,staff(Staff,Id,_,_), N),nVacinasCentroAux(N,0,S).

nVacinasCentroAux([],Acc,S) :- S is Acc.
nVacinasCentroAux([H|T],Acc,S) :- solucoes(U ,vacinacao_covid(H,U,_,_), M),comprimento(M,X),Z is Acc + X,nVacinasCentroAux(T,Z,S).

% Predicados Auxiliares.

year(date(Y,M,D),Y).
month(date(Y,M,D),M).
day(date(Y,M,D),D).
yearsBetweenDates(D1,D2,S):- year(D1,Y1), year(D2,Y2),
                                month(D2,M2), month(D1,M1),
                                M2 > M1, S is Y2 - Y1.
yearsBetweenDates(D1,D2,S):- year(D1,Y1), year(D2,Y2),
                                month(D2,M2), month(D1,M1),
                                day(D2,Da2), day(D1,Da1),
                                M2 == M1, Da2 >= Da1, S is Y2 - Y1.
yearsBetweenDates(D1,D2,S):- year(D1,Y1), year(D2,Y2), S is Y2 - Y1-1.

bigDate2Small(date(Y,M,D,_,_,_,_,_),date(Y,M,D)).

actualDate(Date) :-
    get_time(Stamp),
    stamp_date_time(Stamp, DateTime, local),
    bigDate2Small(DateTime,Date).

% Cálculo de Idade de um Utente.
idade(Data_Nasc, Idade) :- actualDate(Date), yearsBetweenDates(Data_Nasc,Date,Idade).

```

Figura 4.7: Funcionalidades Extras

## Capítulo 5

# Conclusão

Tendo em conta tudo o que foi pedido, acreditamos que conseguimos com êxito criar um universo que resultasse numa correta representação do problema pretendido. Foram desenvolvidas várias entidades referentes ao processo de vacinação, contendo cada uma destas informações que permitiram uma representação completa e transparente do problema. Para além do referido, acreditamos que conseguimos com sucesso implementar um sistema de inferência, um sistema de evolução e involução, que permite a manutenção da congruência da base de conhecimento, assim como um sistema de invariantes. Por concluir, acreditamos que realizamos com sucesso a conversão de todo o conhecimento imperfeito presente na base de conhecimento.

## Capítulo 6

# Referências

### 6.1 Referências Bibliográficas

- [Analide, 2011] ANALIDE, Cesar, Novais, Paulo, Neves, José,  
“Sugestões para a Elaboração de Relatórios”,  
Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.
- [Brakto, 2000] BRATKO, Ivan,  
Programming for Artificial Intelligence, 3rd Edition Portugal, 2000.

## 6.2 Referências Eletrónicas

Plano de Vacinação COVID-19. Disponível em:

<https://www.sns.gov.pt/noticias/2020/12/04/plano-de-vacinacao-contr-a-covid-19>

# Apêndice A

## Base de Conhecimento

```
% utente: #Idutente,Nº Segurança_Social,Nome, Data_Nasc, Email, Telefone, Morada,Profissão, [Doenças_Crônicas],#CentroSaúde
utente(1,2317968,"Luciano Barreira Cardoso",date(1976,07,15),"Luciano@gmail.com",911354828,"Rua 1, casa 1","Profissional de lar",[Diabetes],1).
utente(2,5069310,"Dinis Fazendeiro Dorneles",date(2012,01,25),"Dinis@gmail.com",939988659,"Rua 1, casa 2","Estudante",[],2).
utente(3,8846099,"Ruan Monsanto Sintra",date(1955,09,05),"Ruan@gmail.com",968170412,"Rua 2, casa 1","Profissional de saúde",[],3).
utente(4,3630128,"Marisa Cerveira Arouca",date(1978,01,25),"Marisa@gmail.com",910737835,"Rua 2, casa 2","Profissional das forças de segurança",[Asma,Cancro,Obesidade],1).
utente(5,8773696,"Izabel Lobato Gomes",date(1974,08,11),"Izabel@gmail.com",931478407,"Rua 1, casa 3","Veterinaria",[Asma],2).
utente(6,2132578,"Ion Estrada Guterres",date(2001,12,09),"Ion@gmail.com",919971087,"Rua 3, casa 1","Profissional das forças armadas",[],3).
utente(7,2132578,"Jonas Branco Grilo",date(1962,06,07),"Jonas@gmail.com",9128392716,"Rua 4, casa 1","Profissional de saúde",[],1).
utente(8,2132578,"Reinaldo Matos Barbalho",date(1960,03,17),"Reinaldo@gmail.com",9327381928,"Rua 5, casa 1","Profissional de saúde",[Obesidade,Asma],1).
utente(9,2132578,"Lidiana Soares Regalado",date(1974,04,03),"Lidiana@gmail.com",967382918,"Rua 4, casa 2","Profissional de saúde",[],2).
utente(10,2132578,"Adriana Anes Silveira",date(1954,06,22),"Adriana@gmail.com",9194728396,"Rua 3, casa 2","Profissional de saúde",[Asma],2).
utente(11,2132578,"Lola Monsanto Bicalho",date(1949,02,14),"Lola@gmail.com",9274829680,"Rua 4, casa 3","Profissional de saúde",[],3).
utente(12,332578,"Monique Calho",date(1952,03,24),"Monique@gmail.com",928492019,"Rua 5, casa 2","Cabeleireira",[],3).

% centro_saude:#Idcentro, Nome, Morada, Telefone, Email
centro_saude(1,"Hospital de Braga","Morada1","213659483","centro1@email.com").
centro_saude(2,"Hospital Geral de Santo António","Morada2","262856728","centro2@email.com").
centro_saude(3,"Hospital de Santa Maria","Morada3","253759368","centro3@email.com").

% staff: #Idstaff,#Idcentro,Nome, email
staff(1,1,"Jonas Branco Grilo","Jonas@email.com").
staff(2,1,"Reinaldo Matos Barbalho","Reinaldo@email.com").
staff(3,2,"Lidiana Soares Regalado","Lidiana@email.com").
staff(4,2,"Adriana Anes Silveira","Adriana@email.com").
staff(5,3,"Ruan Monsanto Sintra","Ruan@email.com").
staff(6,3,"Lola Monsanto Bicalho","Lola@email.com").

% vacinação_Covid: #Staff, #utente, Data, Vacina, Toma
vacinacao_covid(1,1,date(2021,01,20),"Pfizer", 1).
vacinacao_covid(2,1,date(2021,02,14),"Pfizer", 2).
vacinacao_covid(3,2,date(2021,02,25),"Moderna", 1).
vacinacao_covid(6,3,date(2021,01,07),"AstraZeneca", 1).
vacinacao_covid(6,3,date(2021,01,23),"AstraZeneca", 2).
vacinacao_covid(2,4,date(2021,03,23),"Pfizer", 1).
vacinacao_covid(5,6,date(2021,02,27),"Moderna", 1).
vacinacao_covid(5,6,date(2021,03,17),"Moderna", 2).
vacinacao_covid(4,12,date(2021,01,19),"Moderna", 1).
```

Figura A.1: Base de Conhecimento Positivo

```

% Utentes -> Conhecimento Negativo
-utente(7,2132578,"Jonas Branco Grilo",date(1962,06,07),"Jonas@gmail.com",9128392716,"Rua 4, casa 1","Profissional de saude",[],1).
-utente(8,2132578,"Reinaldo Matos Barbalho",date(1960,03,17),"Reinaldo@gmail.com",9327381928,"Rua 5, casa 1","Profissional de saude",[Obesidade,Asma],1).
-utente(9,2132578,"Lidiana Soares Regalado",date(1974,04,03),"Lidiana@gmail.com",967382918,"Rua 4, casa 2","Profissional de saude",[],2).
-utente(10,2132578,"Adriana Anes Silveira",date(1954,06,22),"Adriana@gmail.com",9194728396,"Rua 3, casa 2","Profissional de saude",[Asma],2).
-utente(11,2132578,"Lola Monsanto Bicalho",date(1949,02,14),"Lola@gmail.com",9274829680,"Rua 4, casa 3","Profissional de saude",[],3).
-utente(12,332578,"Monique Calho",date(1952,03,24),"Monique@gmail.com",928492019,"Rua 5, casa 2","Cabeleireira",[],3).

% Centro de saude -> Conhecimento Negativo
-centro_saude(3,"Hospital de Santa Maria","Morada3","253759368","centro3@email.com").

% Staff -> Conhecimento Negativo
-staff(4,2,"Adriana Anes Silveira","Adriana@email.com").
-staff(5,3,"Ruan Monsanto Sintra","Ruan@gmail.com").
-staff(6,3,"Lola Monsanto Bicalho","Lola@email.com").

% Vacinação -> Conhecimento Negativo
-vacinacao_covid(2,4,date(2021,03,23),"Pfizer", 1).
-vacinacao_covid(5,6,date(2021,02,27),"Moderna", 1).
-vacinacao_covid(5,6,date(2021,03,17),"Moderna", 2).
-vacinacao_covid(4,12,date(2021,01,19),"Moderna", 1).

```

Figura A.2: Base de Conhecimento Negativo