Universidade do Minho
Escola de Engenharia

# UNIVERSIDADE DO MINHO

## TRABALHO PRÁTICO II

### SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

(2º SEMESTRE 20/21)

A89533 - Ana Luísa Carneiro

A89612 - Ana Rita Peixoto

A89506 - Luís Miguel Pinto

A89574 - Pedro Almeida Fernandes

Braga

Maio - 2021

# RESUMO

O seguinte relatório visa abordar as temáticas lecionadas no âmbito da disciplina de Sistemas de Representação de Conhecimento e Raciocínio. Em particular, nesta segunda fase, aprofundamos as temáticas da evolução e involução de conhecimento para o ramo da vacinação global da população portuguesa no contexto COVID, bem como estendemos os tipos de conhecimento, recorrendo agora a representação de conhecimento imperfeito para caracterizar o nosso universo. De forma a atingir os objetivos propostos, foi fundamental o uso de mecânicas auxiliares como invariantes, no caso da inserção e remoção de conhecimento, e de valores nulos e exceções no caso do tratamento de conhecimento imperfeito.

Tal como na fase anterior, foi-nos proposto a utilização da linguagem de programação em logica PROLOG que nos permite realizar todas as tarefas propostas. Para além disso, em concordância com o supramencionado, a da base de conhecimento sofreu pequenos acréscimos de forma a representar algum conhecimento imperfeito.

De modo a demonstrar os resultados obtidos foi elaborada a secção análise de resultados de modo a enriquecer o trabalho e demonstrar o funcionamento do trabalho realizado.

# ÍNDICE

# ÍNDICE FIGURAS

# INTRODUÇÃO

O trabalho prático II tem como objetivo construir um sistema de representação de conhecimento e raciocínio capaz de completar o programa implementado na fase 1 acerca da vacinação no contexto da pandemia do COVID-19.

Nesta fase foi-nos proposto a extensão do sistema de conhecimento de modo a albergar conhecimento positivo e negativo, conhecimento imperfeito, manipulação de invariantes relativos à inserção e remoção de conhecimento e ainda mecanismos para lidar com a evolução do conhecimento. O programa desenvolvido em *prolog* é capaz de implementar os mecanismos de raciocínio inerentes a este sistema.

Deste modo, o presente relatório vem ilustrar em detalhe, com o auxílio de imagens de código ou ilustrativas, o trabalho efetuado e expor a abordagem adotada para cumprir com os requisitos anteriormente mencionados, incluindo as diversas componentes do conhecimento imperfeito: impreciso, interdito e incerto.

# PRELIMINARES

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

Comecemos pelo princípio, numa vertente mais técnica o trabalho realizado está assente em 6 princípios fundamentais:

- **Pressuposto do Mundo Aberto** – podem existir outros factos ou conclusões verdadeiras para além daqueles representados na base de conhecimento;

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

- **Pressuposto do Domínio Aberto** – podem existir mais objetos do universo de discurso para além daqueles designados pelas constantes da base de conhecimento.

- **Valores lógicos** – torna-se possível distinguir três tipos de conclusões para uma questão: esta pode ser **verdadeira** quando se verifica a sua existência na base de conhecimento, **falsa** se se verificar a existência de negação fraca ou de negação forte ou, quando existe conhecimento imperfeito, a resposta à questão será **desconhecida**.

- **Negação forte/fraca –** dois tipos de negação: a negação por falha ou fraca, característica dos programas em lógica tradicionais, representada pelo termo **não**, e a negação forte ou clássica, como forma de identificar informação negativa, ou falsa, representada pela conectiva **'¬'**. A introdução da negação explícita, juntamente com a negação por falha, permite uma maior expressividade da linguagem.

- **Conhecimento imperfeito –** Serão três os valores nulos aqui tratados: o primeiro, **incerto**, permitirá representar valores desconhecidos e não necessariamente de um conjunto determinado de valores; o segundo, **impreciso**, representará valores desconhecidos, mas de um conjunto determinado de valores; por fim, o terceiro género de valores nulos será de um tipo ligeiramente diferente e representará valores não permitidos, **interdito**, considerados, nomeadamente, na assimilação de informação na base de conhecimento.

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

# DESCRIÇÃO DO TRABALHO

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

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

Nesta secção será feita uma abordagem em detalhe de cada um dos ficheiros que foi alterado nesta fase, de modo a reportar as alterações feitas e regras adicionadas.

# BASE DE CONHECIMENTO

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

Para tal, foi necessário adicionar conhecimento referente aos utentes, centros de saúde, *staff,* à vacinação propriamente dita, às fases de vacinação e também registos de cancelamento de vacinas.

Na primeira fase do projeto foi implementada um conjunto de conhecimentos para os vários termos. Nesta fase foi necessário acrescentar predicados de forma a tornar a base de conhecimento criada na fase 1 compatível com esta fase. Assim criaram-se os respetivos predicados:

- Perfeito – representa conhecimento perfeito (positivo / negativo);
- Incerto – representa conhecimento imperfeito incerto;
- Impreciso – representa conhecimento imperfeito impreciso;
- Interdito – representa conhecimento imperfeito interdito;

## PERFEITO

### POSITIVO

O conhecimento perfeito positivo é um conhecimento verdadeiro onde temos acesso ao seu valor na sua totalidade. O modo de implementação do predicado perfeito é composto exclusivamente pelo termo em questão (*utente*, *staff*, *centro_saude*, etc.) contendo como argumento o identificador que determina qual o termo a utilizar. Na imagem seguinte encontra-se representado o predicado perfeito implementado para o termo utente. Para os termos *centro_saude*, *cancela_vacina, staff* a implementação do predicado é análogo.

```
perfeito(utente(1)).
perfeito(utente(2)).
perfeito(utente(3)).
perfeito(utente(4)).
perfeito(utente(5)).
perfeito(utente(6)).
perfeito(utente(7)).
perfeito(utente(8)).
perfeito(utente(9)).
perfeito(utente(10)).
perfeito(utente(11)).
perfeito(utente(12)).
perfeito(utente(13)).
perfeito(utente(14)).
perfeito(utente(15)).
perfeito(utente(16)).
perfeito(utente(17)).
```

**Figura 1:** Predicado Perfeito Positivo

Para o termo vacinacao_Covid o procedimento para registar conhecimento perfeito foi análogo. Neste caso o predicado é composto pelo termo *vacinacao_Covid* contendo como argumentos o identificador que determina qual o utente que foi vacinado e a toma da vacina. Na imagem seguinte está representado implementação do predicado perfeito para o termo *vacinacao_Covid.*

```
% id e toma
perfeito(vacinacao_Covid(8,1)).
perfeito(vacinacao_Covid(13,2)).
perfeito(vacinacao_Covid(3,1)).
perfeito(vacinacao_Covid(16,1)).
perfeito(vacinacao_Covid(7,1)).
perfeito(vacinacao_Covid(3,1)).
perfeito(vacinacao_Covid(14,1)).
perfeito(vacinacao_Covid(15,1)).
```

**Figura 2:** Predicado Perfeito Negativo

NEGATIVO

O conhecimento perfeito negativo é um conhecimento falso onde temos acesso ao seu valor na sua totalidade. Analogamente, para conhecimento negativo também foi implementamos o predicado perfeito para os termos implementados na base de conhecimento. Contudo, foi acrescentado um predicado que permite identificar conhecimento negativo forte (-Termo). Assim, um termo era negativo se não fosse positivo e não fosse desconhecido. Na figura seguinte encontra-se representado o predicado referido.

```
-Termo :- nao(Termo),
          nao(excecao(Termo)).
```

**Figura 3:** Conhecimento Negativo

## INCERTO

O conhecimento incerto diz respeito a conhecimento que é totalmente desconhecido para o utilizador. De forma a implementar conhecimento incerto é necessário identificar qual o parâmetro que temos incertezas quanto ao seu valor. De seguida temos de implementar um predicado que reconhece uma exceção caso o termo contiver o campo incerto identificado. Na figura abaixo esta representado para o termo *centro_saude* o modo de implementação de um conhecimento incerto com o nome incerto. Para os restantes termos a implementação é análoga.

```
%--- Conhecimento incerto ---
centro_saude(6,centro_Incerto, 'R Casal Borga 50 2200-097 ABRANTES', 273499959, 'centro6Saude@gmail.com').
excecao(centro_saude(ID,_,Morada,Telefone,Email)) :- centro_saude(ID,centro_Incerto,Morada,Telefone,Email).
incerto(centro_saude(6)).
```

**Figura 4:** Conhecimento Incerto - Centro de Saúde

Este conhecimento pode ser inserido e/ou removido da BC, contudo não conseguimos alterar a incerteza de outros campos do termo, uma vez que isso implicaria a redução de conhecimento.

## IMPRECISO

O conhecimento impreciso é conhecimento onde não temos certezas acerca do seu valor, contudo temos acesso a um conjunto de hipóteses para esse valor. De forma a definir o conhecimento impreciso implementamos um conjunto de exceções que podem ser vistas como as hipóteses para o campo que desconhecemos. Na figura abaixo está implementado um conhecimento impreciso no campo email para o termo *centro_saude.* Para os restantes termos a implementação é análoga.

```
%--- Conhecimento impreciso ---
excecao(centro_saude(5,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com')).
excecao(centro_saude(5,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro5Saude@gmail.com')).
impreciso(centro_saude(5)).
```

**Figura 5:** Conhecimento Impreciso - Centro de Saúde

Para este conhecimento conseguimos inserir novas hipóteses ao conhecimento impreciso já implementado. Este conhecimento pode ser inserido e/ou removido da BC, contudo não conseguimos alterar a imprecisão de outros campos do termo, uma vez que isso implicaria a redução de conhecimento.

## INTERDITO

O conhecimento interdito é conhecimento onde para além de desconhecermos o seu valor não temos nem conseguimos ter acesso a ele. De forma a incluir o conhecimento interdito é necessário identificar qual o parâmetro que é interdito. De seguida temos de implementar um predicado que reconhece uma exceção caso o termo contiver o campo interdito identificado. Finalmente é necessário definir o campo como interdito através do predicado nulo. Com este predicado conseguimos, através do uso de invariantes, identificar qual o termo que contém campos interditos impossibilitando assim a adição de valores nesse campo. Na figura abaixo esta representado para o termo *centro_saude* o modo de implementação de um conhecimento interdito referido acima. Para os restantes termos a implementação é análoga.

```
%--- Conhecimento interdito ---
centro_saude(3,centroSaude3, 'Quinta Beloura 21 2714-521 PORTELA', 32425256, email_Interdito).
excecao(centro_saude(ID,Nome,Morada,Telefone,_)) :- centro_saude(ID,Nome,Morada,Telefone,email_Interdito).
nulo(email_Interdito).
interdito(centro_saude(3)).
```

**Figura 6:** Conhecimento Interdito - Centro de Saúde

Para este conhecimento conseguimos não conseguimos alterar o campo interdito para os termos já adicionados na BC como interditos. Este conhecimento pode ser inserido e/ou removido da BC.

## REGRAS AUXILIARES

Na primeira fase do projeto criou-se o documento "regrasAuxiliares.pl" com o propósito de agrupar as diversas regras que serviram de auxílio às funcionalidades do sistema. Nesta 2ª fase este documento foi atualizado, contendo agora mais duas regras que foram necessárias para suportar os requisitos do sistema.

A primeira regra a abordar é a regra *idInsertTermo* que tem como objetivo inserir na base de conhecimento um registo de que um termo se trata de conhecimento perfeito, seja por parte de um utente, centro de saude, cancelamento de vacina ou vacinação. Esta regra baseou-se numa implementação simples dado apenas pratica uma operação de inserção. De seguida apresenta-se esta mesma regra em *prolog:*

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Insere o predicado perfeito na base de conhecimento

idInsertTermo(utente(Id,_,_,_,_,_,_,_,_,_)) :- insercao(perfeito(utente(Id))).
idInsertTermo(centro_saude(Id,_,_,_,_)) :- insercao(perfeito(centro_saude(Id))).
idInsertTermo(staff(Id,_,_,_)) :- insercao(perfeito(staff(Id))).
idInsertTermo(cancela_vacina(Id,_)) :- insercao(perfeito(cancela_vacina(Id))).
idInsertTermo(vacinacao_Covid(_,Id,_,_,Toma)) :- insercao(perfeito(vacinacao_Covid(Id,Toma))).
```

**Figura 7:** Regra Auxiliar *idInsertTermo*

A segunda regra adicionada é a regra *idRemoveTermo* que tem como propósito remover um registo de que um termo é conhecimento perfeito da BC. Esta regra procede de modo contrário à regra anterior, sendo que para cada termo recebido como argumento, efetua a operação de remoção da BC do registo de que é perfeito.

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Remove o predicado perfeito na base de conhecimento

idRemoveTermo(utente(Id,_,_,_,_,_,_,_,_,_)) :- remocao(perfeito(utente(Id))).
idRemoveTermo(centro_saude(Id,_,_,_,_)) :- remocao(perfeito(centro_saude(Id))).
idRemoveTermo(staff(Id,_,_,_)) :- remocao(perfeito(staff(Id))).
idRemoveTermo(cancela_vacina(Id,_)) :- remocao(perfeito(cancela_vacina(Id))).
idRemoveTermo(vacinacao_Covid(_,Id,_,_,Toma)) :- remocao(perfeito(vacinacao_Covid(Id,Toma))).
```

**Figura 8:** Regra Auxilar *idRemoveTermo*

## INVARIANTES

Para a implementação dos vários tipos de conhecimento imperfeito, foi necessário definir diferentes operadores. Assim, o operador :: refere-se aos invariantes do conhecimento perfeito, o operador :~: refere-se ao conhecimento imperfeito interdito/incerto e, por fim, o operador :-: refere-se ao conhecimento impreciso. Na seguinte figura é possível observar a definição destes operadores em *prolog:*

```
:- op( 900,xfy,'::' ).  %- criação de um novo operador para o prolog

%conhecimento interdito
:- op(900,xfy,:~:).

%conhecimento impreciso
:- op(900,xfy,:-:).
```

**Figura 9:** Operadores dos Invariantes

Além disso, de forma e permitir a evolução e involução do conhecimento, também foram inseridos os seguintes predicados no início do ficheiro invariantes.pl:

```
:- dynamic utente/10.
:- dynamic staff/4.
:- dynamic centro_saude/5.
:- dynamic vacinacao_Covid/5.
:- dynamic cancela_vacina/2.
:- dynamic fase/7.
:- dynamic excecao/1.
:- dynamic nulo/1.
:- dynamic interdito/1.
:- dynamic incerto/1.
:- dynamic impreciso/1.
:- dynamic perfeito/1.
:- dynamic (-)/1.
```

**Figura 10:** Dinamismo dos Predicados

## INVARIANTES ESTRUTURAIS GLOBAIS

Consideramos relevante implementar um conjunto de invariantes globais que são aplicados a todos os tipos de conhecimento. Estes invariantes impedem a inserção de conhecimento negativo caso esse mesmo conhecimento já exista e seja positivo, e vice-versa. Estes invariantes são válidos para todos os termos inseridos.

```
%insercao de um termo positivo não pode ser negativo
+Termo :: nao(-Termo).

%insercao de um termo negativo não pode ser positivo
+(-Termo) :: nao(Termo).
```

**Figura 11:** Invariantes de Inserção Globais

## INVARIANTES UTENTE

No caso do utente, consideramos conveniente permitir a adição e remoção de conhecimento perfeito positivo. A remoção deste tipo de conhecimento oferece flexibilidade para corrigir os dados da base de conhecimento, caso os valores inseridos estejam errados. Além disso, no espetro do conhecimento imperfeito, foram implementados mecanismos de adição de conhecimento impreciso, interdito ou incerto. Nas seguintes secções os invariantes referentes a cada caso serão explorados com mais detalhe.

### ADIÇÃO CONHECIMENTO PERFEITO POSITIVO

Para a **adição de conhecimento perfeito positivo** foram implementados 2 invariantes. No momento de inserção é necessário verificar diversos detalhes acerca dos parâmetros do predicado, isto é, as variáveis que representam o ID Utente, número de segurança social, ID Centro de saúde e telefone devem corresponder ao tipo de dados *integer.* Para as variáveis que denotam o nome de utente, email, profissão e morada, deverá ser considerado o tipo de dados *atom.* No caso da data de nascimento de utente, foi necessário recorrer à regra *validaData* que trata de verificar se o valor inserido é correto. Por fim, para a inserção da lista de doenças do utente, utilizou-se a regra *validaLista* de forma a verificar a validade da mesma.

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

```
% Invariante que permite adicionar conhecimento positivo
+utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro) ::(
            integer(Id),
            integer(SS),
            atom(Nome),
            validaData(Data),
            atom(Email),
            integer(Telefone),
            atom(Morada),
            atom(Profissao),
            validaLista(Doenca),
            integer(Centro)).
```

**Figura 12:** Adição de Conhecimento Positivo Utente (1)

Além disso, também foi relevante acrescentar o invariante que impede a inserção de conhecimento positivo repetido. Para isso este invariante percorre todas ao termos utente iguais ao termo inserido e caso o comprimento da lista devolvida seja 1 então o termo é único.

```
% Não permite adicionar conhecimento positivo repetido
+utente(Id,_,_,_,_,_,_,_,_,_) :: (solucoes(Id,utente(Id,_,_,_,_,_,_,_,_,_),S),
                                  comprimento( S,1 )).
```

**Figura 13:** Adição de Conhecimento Positivo Utente (2)

## ADIÇÃO CONHECIMENTO PERFEITO NEGATIVO

Analogamente à **adição de conhecimento perfeito positivo**, também para a adição de conhecimento perfeito negativo foram implementados 2 invariantes.

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

```
% Invariante que permite adicionar conhecimento negativo
+(-utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro)) :: (
                integer(Id),
                integer(SS),
                atom(Nome),
                validaData(Data),
                atom(Email),
                integer(Telefone),
                atom(Morada),
                atom(Profissao),
                validaLista(Doenca),
                integer(Centro)).
```

**Figura 14:** Adição de Conhecimento Negativo Utente (1)

Além disso, também foi relevante acrescentar o invariante que impede a inserção de conhecimento negativo repetido. O procedimento é análogo ao utilizado na inserção do conhecimento positivo. Contudo, para este tipo de conhecimento, no final da inserção além de

estar o conhecimento negativo que inserimos vai estar presente um predicado que representa conhecimento negativo (figura 15). Este raciocino está implementado na imagem abaixo.

```
% Não permite adicionar conhecimento negativo repetido
+(-utente(Id,_,_,_,_,_,_,_,_,_)) :: (solucoes(Id,-utente(Id,_,_,_,_,_,_,_,_,_),S),
                                     comprimento( S,N ), N == 2 ).
```

**Figura 15:** Adição de Conhecimento Negativo Utente (2)

## REMOÇÃO CONHECIMENTO PERFEITO POSITIVO

No caso do **conhecimento perfeito positivo**, foi considerada a sua **remoção**. Para isso, foram criados dois invariantes. O primeiro verifica os tipos de dados a ser removidos, procedendo de forma análoga à inserção. Na seguinte figura é possível observar o invariante que valida os dados da remoção:

```
% Invariante que permite remover conhecimento positivo
-utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro) ::(
                integer(Id),
                integer(SS),
                atom(Nome),
                validaData(Data),
                atom(Email),
                integer(Telefone),
                atom(Morada),
                atom(Profissao),
                validaLista(Doenca),
                integer(Centro)).
```

**Figura 16:** Remoção de Conhecimento Positivo Utente (1)

Além disso, consideramos conveniente limitar a remoção de conhecimento positivo, ou seja, apenas é possível remover um utente caso não existam registos de vacinação ou cancelamento de vacina associados ao utente a remover, de modo a não comprometer a consistência da base de conhecimento. Este invariante está visível na figura 17.

```
% Invariante que permite limitar a remocao de conhecimento positivo
-utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro) :: (
    solucoes(Id,(vacinacao_Covid(_,Id,_,_,_),cancela_vacina(Id,_)),S ),comprimento(S,N), N==0,
    solucoes(Id,utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro),S ),comprimento(S,N), N==0).
```

**Figura 17:** Remoção de Conhecimento Positivo Utente (2)

## REMOÇÃO CONHECIMENTO PERFEITO NEGATIVO

Para o **conhecimento perfeito negativo** foi considerada também a sua **remoção**. Para isso, foram criados dois invariantes. Primeiramente, é efetuada a verificação e validação dos parâmetros do termo a remover, analogamente ao procedimento efetuado na adição. Este invariante está implementado da seguinte forma:

```
% Invariante que permite remover conhecimento negativo
-(-utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro)) ::(
                integer(Id),
                integer(SS),
                atom(Nome),
                validaData(Data),
                atom(Email),
                integer(Telefone),
                atom(Morada),
                atom(Profissao),
                validaLista(Doenca),
                integer(Centro)).
```

**Figura 18:** Remoção de Conhecimento Negativo Utente (1)

Além foi necessário limitar a remoção de conhecimento negativo evitando assim inconsistências na base de conhecimento. Para isso percorremos todo o conhecimento perfeito negativo do termo utente e caso só haja um conhecimento negativo (o predicado que permite identificar o conhecimento negativo – figura 3) então o termo utente pode ser removido. Abaixo esta foi implementado o respetivo invariante.

```
% Invariante que permite limitar a remocao de conhecimento negativo
-(-utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro)) :: (
    solucoes(I,-utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro),S ),
    comprimento(S,N), N==1).
```

**Figura 19:** Remoção de Conhecimento Negativo Utente (2)

## ADIÇÃO CONHECIMENTO IMPERFEITO IMPRECISO

No âmbito do **conhecimento imperfeito**, consideramos conveniente considerar a **adição** de **conhecimento impreciso**. De modo a concretizar este objetivo, foram implementados dois invariantes.

O primeiro permite limitar a inserção do conhecimento impreciso, ou seja, é possível adicionar um registo impreciso de um utente caso esse registo não seja conhecimento perfeito nem incerto. Na figura seguinte é possível observar a implementação em *prolog* deste invariante. É de notar que o operador utilizado foi o :-: para denotar o conhecimento impreciso, tal como referido anteriormente.

```
% Inserção de conhecimento imperciso
+utente(Id,_,_,_,_,_,_,_,_,_) :-: (
    nao(perfeito(utente(Id))),
    nao(incerto(utente(Id)))
).
```

**Figura 20:** Adição de Conhecimento Impreciso Utente (1)

O segundo invariante referente à adição de conhecimento impreciso do utente tem como objetivo impedir a inserção de conhecimento repetido. O procedimento utilizado verifica se o conhecimento inserido na BC é único, ou seja, se o comprimento da lista de termos iguais ao inserido é 1. Caso seja então o conhecimento pode manter-se inserido na BC:

```
% A inserção ocorre caso não haja conhecimento impreciso repetido
+utente(Id,SS,Nome,Data,Email,Telefone,Morada,Profissao,Doenca,Centro) :-: (
    solucoes((ID,M), excecao(utente(ID,_,_,_,_,_,Morada,_,_,_)), L),
    comprimento(L,1)).
```

**Figura 21:** Adição de Conhecimento Impreciso Utente (2)

## ADIÇÃO CONHECIMENTO IMPERFEITO INTERDITO/INCERTO

Além da adição de conhecimento imperfeito impreciso, consideramos conveniente também adicionar **conhecimento imperfeito interdito/incerto** relativo a um utente. Para isso, foram implementados dois invariantes.

Um dos invariantes apenas permite a inserção se se verificar a veracidade de 3 predicados. Concretamente, para adicionar conhecimento interdito ou inserto é imperativo que não exista conhecimento perfeito, impreciso ou incerto na base de conhecimento referente a esse utente.

```
% Inserção de conhecimento interdito ou incerto
+utente(Id,_,_,_,_,_,_,_,_,_) :~: (
    nao(perfeito(utente(Id))),
    nao(impreciso(utente(Id))),
    nao(incerto(utente(Id)))
).
```

**Figura 22:** Adição de Conhecimento Interdito/Incerto Utente (1)

De forma a evitar a inserção de conhecimento interdito foi implementado o seguinte invariante que insere o conhecimento caso o termo inserido não tenha o campo interdito, neste caso seria o campo email. É de notar a utilização do operador :~: para denotar o conhecimento interdito/incerto. Este raciocínio traduziu-se em *prolog* da seguinte forma.

```
% A inserção ocorre caso email não seja interdito
+utente(Id,_,_,_,_,_,_,_,_,_):~:(solucoes(Em,(utente(Id,_,_,_,Em,_,_,_,_,_),nulo(Em)),L),
                                 comprimento(L,N),
                                 N==0).
```

**Figura 23:** Adição de Conhecimento Interdito/Incerto Utente (2)

## INVARIANTES CENTRO SAÚDE

Para o centro de saúde, consideramos conveniente permitir a adição e remoção de conhecimento perfeito positivo. A remoção deste tipo de conhecimento oferece flexibilidade para corrigir os dados da base de conhecimento, caso os valores inseridos estejam errados. Além disso, no âmbito do conhecimento imperfeito, foram implementados mecanismos de adição de conhecimento impreciso, interdito ou incerto. Nas seguintes secções os invariantes referentes a cada caso serão explorados com mais detalhe.

### ADIÇÃO CONHECIMENTO PERFEITO POSITIVO

Primeiramente, foi criado um invariante que verifica os tipos de dados do centro de saúde a ser inseridos e a sua validade. Assim, os valores ID Centro e Telefone deverão ser representados por um *integer* e os valores nome, morada e email do centro de saúde deverão ser representados pelo tipo de dados *atom.* Na seguinte figura está representada esta implementação:

```
% Invariante que permite adicionar conhecimento positivo
+centro_saude(ID,Nome,Morada,Telefone,Email) :: (
    integer(ID),
    atom(Nome),
    atom(Morada),
    integer(Telefone),
    atom(Email)).
```

**Figura 24:** Adição de Conhecimento Positivo Centro de Saúde (1)

De forma a permitir a inserção de conhecimento único e impedir repetições, foi criado um invariante da figura abaixo. Para isso este invariante percorre todos os termos *centro_saude* iguais ao termo inserido e caso o comprimento da lista devolvida seja 1 então o termo é único.

```
% Não permite adicionar conhecimento positivo repetido
+centro_saude(Id,_,_,_,_) :: (solucoes(Id,centro_saude(Id,_,_,_,_),S),
                              comprimento( S,N ), N == 1 ).
```

**Figura 25:** Adição de Conhecimento Positivo Centro de Saúde (2)

## ADIÇÃO CONHECIMENTO PERFEITO NEGATIVO

No espetro do **conhecimento perfeito negativo**, foram implementados dois invariantes. O primeiro tem como objetivo validar os dados a inserir, de acordo com o seu tipo de dados. Este invariante procedeu de modo análogo ao conhecimento perfeito positivo e está ilustrado na seguinte figura:

```
% Invariante que permite adicionar conhecimento negativo
+(-centro_saude(ID,Nome,Morada,Telefone,Email)) :: (
    integer(ID),
    atom(Nome),
    atom(Morada),
    integer(Telefone),
    atom(Email)).
```

**Figura 26:** Adição de Conhecimento Negativo Centro de Saúde (1)

Além disso, consideramos conveniente implementar um mecanismo para impedir a inserção de conhecimento negativo repetido. O procedimento é análogo ao utilizado na inserção do conhecimento positivo. Contudo, para este tipo de conhecimento, no final da inserção além de estar o conhecimento negativo que inserimos vai estar presente um predicado que representa conhecimento negativo (figura 3). Este raciocínio está implementado na imagem abaixo:

```
% Não permite adicionar conhecimento negativo repetido
+(-centro_saude(Id,_,_,_,_)) :: (solucoes(Id,-centro_saude(Id,_,_,_,_),S),
                                 comprimento( S,N ), N == 2 ).
```

**Figura 27:** Adição de Conhecimento Negativo Centro de Saúde (2)

## REMOÇÃO CONHECIMENTO PERFEITO POSITIVO

Além da adição de conhecimento perfeito positivo, consideramos conveniente também permitir a sua remoção. Assim, foram considerados 2 invariantes.

O primeiro invariante tem como objetivo verificar e validar os dados inseridos, tal como está representado na seguinte figura:

```
% Invariante que permite remover conhecimento positivo
-centro_saude(ID,Nome,Morada,Telefone,Email) :: (
    integer(ID),
    atom(Nome),
    atom(Morada),
    integer(Telefone),
    atom(Email)).
```

**Figura 28:** Remoção Conhecimento Positivo Centro de Saúde (1)

O segundo invariante implementado tem como objetivo limitar a remoção de conhecimento positivo, ou seja, apenas é possível remover um centro de saúde caso não haja registos associados ao utente ou ao *staff e* se o *centro_saude* ao fim da remoção não se encontra na BC. Este invariante está presente na seguinte figura:

```
% Invariante que permite limitar a remocao de conhecimento positivo
-centro_Saude(IdCentro,Nome,Morada,Telefone,Email) :: (solucoes(IdCentro,(utente(_,_,_,_,_,_,_,_,_,IdCentro),staff(_,IdCentro,_,_)),S ),
                                                       comprimento(S,N), N==0,
                                                       solucoes(Id,centroSaude(IdCentro,Nome,Morada,Telefone,Email),S ),
                                                       comprimento(S,N), N==0).
```

**Figura 29:** Remoção Conhecimento Positivo Centro de Saúde (2)

## REMOÇÃO CONHECIMENTO PERFEITO NEGATIVO

De forma a remover registos de **conhecimento perfeito negativo**, foram implementados dois invariantes. O primeiro visa verificar se os parâmetros do centro de saúde a remover são válidos, ou seja, se possuem os tipos de dados corretos. Este invariante está presente na seguinte figura:

```
% Invariante que permite remover conhecimento negativo
-(-centro_saude(ID,Nome,Morada,Telefone,Email)) :: (
    integer(ID),
    atom(Nome),
    atom(Morada),
    integer(Telefone),
    atom(Email)).
```

**Figura 30:** Remoção Conhecimento Negativo Centro de Saúde (1)

Além disso, foi implementado um outro invariante que limita a remoção de conhecimento negativo. Para isso percorremos todo o conhecimento perfeito negativo do termo *centro_saude* e caso só haja um conhecimento negativo (o predicado que permite identificar o conhecimento negativo – figura 3) então o termo *centro_saude* pode ser removido.:

```
% Invariante que permite limitar a remocao de conhecimento negativo
-(-centro_Saude(IdCentro,Nome,Morada,Telefone,Email)) :: (
    solucoes(Id,-centroSaude(IdCentro,Nome,Morada,Telefone,Email),S ),
    comprimento(S,N), N==1).
```

**Figura 31:** Remoção Conhecimento Negativo Centro de Saúde (2)

## ADIÇÃO CONHECIMENTO IMPERFEITO IMPRECISO

No que toca ao conhecimento imperfeito, consideramos conveniente implementar conhecimento impreciso através de 2 invariantes. O primeiro invariante, visível na figura 32, tem como objetivo verificar se já existe conhecimento perfeito ou incerto relativos ao centro de saúde a inserir. Para a inserção de conhecimento impreciso ser realizada, é necessário que não existam outros tipos de conhecimento na BC.

```
% Inserção de conhecimento imperciso
+centro_saude(Id,_,_,_,_) :-: (
    nao(perfeito(centro_saude(Id))),
    nao(incerto(centro_saude(Id)))
).
```

**Figura 32:** Adição Conhecimento Impreciso Centro de Saúde (1)

O segundo invariante referente à adição de conhecimento impreciso do utente tem como objetivo impedir a inserção de conhecimento repetido. O procedimento verifica se o conhecimento inserido na BC é único, ou seja, se o comprimento da lista de termos iguais ao inserido é 1. Caso seja então o conhecimento pode manter-se inserido na BC

```
% A inserção ocorre caso o conhecimento a inserido nao seja repetido
+centro_saude(ID,Nome,Morada,Telefone,Email) :-: (
    solucoes((ID,E), excecao(centro_saude(ID,_,_,_,Email)), L),
    comprimento(L,1)).
```

**Figura 33:** Adição Conhecimento Impreciso Centro de Saúde (2)

### ADIÇÃO CONHECIMENTO IMPERFEITO INTERDITO/INCERTO

Para o conhecimento imperfeito também consideramos a extensão do conhecimento incerto/interdito. Para isso, consideramos 2 invariantes.

Um dos invariantes tem como objetivo limitar a inserção do conhecimento, ou seja, só é inserido caso não haja previamente na BC um registo de conhecimento imperfeito, impreciso ou incerto associado a esse termo. Este invariante foi implementado da seguinte forma:

```
% Insercão de conhecimento incerto ou interdito
+centro_saude(Id,_,_,_,_) :~: (
    nao(perfeito(centro_saude(Id))),
    nao(impreciso(centro_saude(Id))),
    nao(incerto(centro_saude(Id)))
).
```

**Figura 34:** Adição Conhecimento Interdito/Incerto Centro de Saúde (1)

De forma a evitar a inserção de conhecimento interdito foi implementado o seguinte invariante que insere o conhecimento caso o termo inserido não tenha o campo interdito, neste caso seria o campo email. Este raciocínio traduziu-se em *prolog* da seguinte forma:

```
% A inserção ocorre caso numero de telefone não seja interdito
+centro_saude(ID,Nome,Morada,Telefone,Email) :~: (
    solucoes(Num,(centro_saude(ID,_,_,Num,_),nulo(Num)),L),
    comprimento(L,N),
    N==0).
```

**Figura 35:** Adição Conhecimento Interdito/Incerto Centro de Saúde (2)

## INVARIANTES STAFF

Para o staff, consideramos conveniente permitir a adição e remoção de conhecimento perfeito positivo. A remoção deste tipo de conhecimento oferece flexibilidade para corrigir os dados da base de conhecimento, caso os valores inseridos estejam errados. Além disso, no âmbito do conhecimento imperfeito, foram implementados mecanismos de adição de conhecimento impreciso, interdito ou incerto. Nas seguintes secções os invariantes referentes a cada caso serão explorados com mais detalhe.

## ADIÇÃO CONHECIMENTO PERFEITO POSITIVO

De modo a verificar se os dados introduzidos na base de conhecimento respeitam o tipo de dados do termo staff foi implementado o invariante abaixo. Assim, os valores ID staff e ID centro deverão ser representados por um *integer* e os valores nome e email do staff deverão ser representados pelo tipo de dados *atom.* Na seguinte figura está representada esta implementação:

```
% Invariante que permite adicionar conhecimento positivo
+staff(Idstaff, Idcentro, Nome, Email) :: (
    integer(Idstaff),
    integer(Idcentro),
    atom(Nome),
    atom(Email)).
```

**Figura 36:** Adição Conhecimento Positivo Staff (1)

Além disso, também foi relevante acrescentar o invariante que impede a inserção de conhecimento positivo repetido, verificando se o termo inserido é único na BC. Para isso este invariante percorre todos os termos staff iguais ao termo inserido e caso o comprimento da lista devolvida seja 1 então o termo é único.

```
% Não permite adicionar conhecimento positivo repetido
+staff(Id,_,_,_) :: (solucoes(Id,staff(Id,_,_,_),S),
                     comprimento( S,N ), N == 1 ).
```

**Figura 37:** Adição Conhecimento Positivo Staff (2)

Desta forma, cumprindo as restrições anteriormente referidas, conseguimos garantir que o conhecimento a ser inserido na BC contém valores válidos e adequados ao contexto do staff.

## ADIÇÃO CONHECIMENTO PERFEITO NEGATIVO

Analogamente para o conhecimento negativo, vamos verificar se todos os valores introduzidos respeitam o tipo de dados do termo staff de forma a manter a base de conhecimento consistente. A verificação é realizada de modo semelhante para o conhecimento negativo.

```
% Invariante que permite adicionar conhecimento negativo
+(-staff(Idstaff, Idcentro, Nome, Email)) :: (
    integer(Idstaff),
    integer(Idcentro),
    atom(Nome),
    atom(Email)).
```

**Figura 38:** Adição Conhecimento Negativo Staff (1)

Além disso, também é necessário acrescentar um invariante que impede a inserção de conhecimento negativo repetido. O procedimento é análogo ao utilizado na inserção do conhecimento positivo. Contudo, para este tipo de conhecimento, no final da inserção além de estar o conhecimento negativo que inserimos vai estar presente um predicado que representa conhecimento negativo (figura x). Este raciocino está implementado na imagem abaixo.

```
% Não permite adicionar conhecimento negativo repetido
+(-staff(Id,_,_,_)) :: (solucoes(Id,-staff(Id,_,_,_),S),
                        comprimento( S,N ), N == 2 ).
```

**Figura 39:** Adição Conhecimento Negativo Staff (2)

## REMOÇÃO CONHECIMENTO PERFEITO POSITIVO

Para a remoção de conhecimento perfeito positivo é necessário verificar que o staff a remover cumpre com o requisito do tipo de dados do staff. Para isso implementou-se o seguinte invariante.

```
% Invariante que permite remover conhecimento positivo
-staff(Idstaff, Idcentro, Nome, Email) :: (
    integer(Idstaff),
    integer(Idcentro),
    atom(Nome),
    atom(Email)).
```

**Figura 40:** Remoção Conhecimento Positivo Staff (1)

Além disso, também foi necessário verificar que o conhecimento a remover não esta a ser utilizado no termo *vacinacao_Covid* de modo a não permitir a existência de "buracos" na BC. Desta forma, verifica-se se o staff não vacinou nenhum utente e se este ao fim da remoção não se encontra na BC.

```
% Invariante que permite limitar a remocao de conhecimento positivo
-staff(Id, Idcentro, Nome, Email) :: (
    solucoes(Id,vacinacao_Covid(Id,_,_,_,_),S ),
    comprimento(S,N), N==0,
    solucoes(Id,staff(Id, Idcentro, Nome, Email),S ),
    comprimento(S,N), N==0).
```

**Figura 41:** Remoção Conhecimento Positivo Staff (2)

## REMOÇÃO CONHECIMENTO PERFEITO NEGATIVO

Para a remoção de conhecimento perfeito negativo é necessário verificar que o staff a remover cumpre com o requisito do tipo de dados do staff. Para isso implementou-se o seguinte invariante.

```
% Invariante que permite remover conhecimento negativo
-(-staff(Idstaff, Idcentro, Nome, Email)) :: (
    integer(Idstaff),
    integer(Idcentro),
    atom(Nome),
    atom(Email)).
```

**Figura 42:** Remoção Conhecimento Negativo Staff (1)

Além disso, foi necessário verificar se o staff que pretendemos remover está presente na BC. Para isso percorremos todo o conhecimento perfeito negativo do termo staff e caso só haja um conhecimento negativo (o predicado que permite identificar o conhecimento negativo – figura 3) então o termo staff pode ser removido.

```
% Invariante que permite limitar a remocao de conhecimento positivo
-(-staff(Idstaff, Idcentro, Nome, Email)) :: (
    solucoes(Id,-staff(Idstaff, Idcentro, Nome, Email),S ),
    comprimento(S,N), N==1).
```

**Figura 43:** Remoção Conhecimento Negativo Staff (2)

## ADIÇÃO CONHECIMENTO IMPERFEITO IMPRECISO

Também consideramos conveniente implementar a **adição** de **conhecimento impreciso**. De modo a concretizar este objetivo, foram implementados dois invariantes.

O primeiro permite limitar a inserção do conhecimento impreciso, ou seja, é possível adicionar um registo impreciso de um staff caso esse registo não seja conhecimento perfeito nem incerto. Na figura seguinte é possível observar a implementação em *prolog* deste invariante. É de notar que o operador utilizado foi o :-: para denotar o conhecimento impreciso, tal como referido anteriormente.

```
% Inserção de conhecimento impreciso
+staff(Ids,_,_,_) :-: (
    nao(perfeito(staff(Ids))),
    nao(incerto(staff(Ids)))
).
```

**Figura 44:** Adição Conhecimento Impreciso Staff (1)

O segundo invariante referente à adição de conhecimento impreciso do utente tem como objetivo impedir a inserção de conhecimento repetido. O procedimento verifica se o conhecimento inserido na BC é único, ou seja, se o comprimento da lista de termos iguais ao inserido é 1. Caso seja então o conhecimento pode manter-se inserido na BC.

```
% A inserção ocorre caso o conhecimento a inserido nao seja repetido
+staff(ID,Idcentro,Nome,Email) :-: (
    solucoes((ID,E), excecao(staff(ID,_,_,Email)), L),
    comprimento(L,1)).
```

**Figura 45:** Adição Conhecimento Impreciso Staff (2)

## ADIÇÃO CONHECIMENTO IMPERFEITO INTERDITO/INCERTO

Para o conhecimento imperfeito também consideramos a extensão do conhecimento incerto/interdito. Para isso, consideramos 2 invariantes.

Um dos invariantes tem como objetivo limitar a inserção do conhecimento, ou seja, só é inserido caso não haja previamente na BC um registo de conhecimento imperfeito, impreciso ou incerto associado a esse termo. Este invariante foi implementado da seguinte forma:

```
% Inserção de conhecimento incerto ou interdito
+staff(Ids,_,_,_) :~: (
    nao(perfeito(staff(Ids))),
    nao(impreciso(staff(Ids))),
    nao(incerto(staff(Ids)))
).
```

**Figura 46:** Adição Conhecimento Interdito/Incerto Staff (1)

De forma a evitar a inserção de conhecimento interdito foi implementado o seguinte invariante que insere o conhecimento caso o termo inserido não tenha o campo interdito, neste caso seria o campo email. Este raciocínio traduziu-se em *prolog* da seguinte forma:

```
% A inserção ocorre caso email não seja interdito
+staff(ID,Idcentro,Nome,Email) :~: (
    solucoes(Em,(staff(ID,_,_,Em),nulo(Em)),L),
    comprimento(L,N),
    N==0).
```

**Figura 47:** Adição Conhecimento Interdito/Incerto Staff (2)

## INVARIANTES VACINAÇÃO COVID

No caso da vacinação covid apenas foi considerado o conhecimento perfeito, uma vez que achamos por bem impedir que os registos da vacinação sejam incorretos ou incompletos evitando assim inconsistências na BC. Assim, é possível inserir e remover conhecimento positivo e negativo. Os invariantes que permitem concretizar este objetivo são apresentados em seguida com mais detalhe.

### ADIÇÃO CONHECIMENTO PERFEITO POSITIVO

Para adicionar conhecimento perfeito positivo relativo à vacinação, foi necessário implementar dois invariantes.

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

```
% Invariante que permite adicionar conhecimento positivo
+vacinacao_Covid(IDStaff,IDUtente,Data,Vacina,Toma) :: (
    integer(IDStaff),
    integer(IDUtente),
    validaData(Data),
    atom(Vacina),
    integer(Toma)).
```

**Figura 48:** Adição Conhecimento Positivo Vacinação (1)

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

```
% Limitar a insercao de novos conhecimentos positivos
+vacinacao_Covid(Staff,Id,_,Vacina,1) :: (
    solucoes(Id,utente(Id,_,_,_,_,_,_,_,_,_,_),S),
    comprimento( S,N ), N == 1, %ID tem de ser de um utente
    solucoes(Id,vacinacao_Covid(_,Id,_,_,1),S),
    comprimento( S,N ), N == 1, %Não ter tomado a primeira toma
    verificaCentro(Id,Staff)). % tem de ser vacinado no centroSaude respetivo

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

**Figura 49:** Adição Conhecimento Positivo Vacinação (2)

## ADIÇÃO CONHECIMENTO PERFEITO NEGATIVO

Para a adição de **conhecimento perfeito negativo** foi necessário verificar a validade e tipo dos dados do termo a adicionar. Este procedimento foi efetuado de forma análoga ao conhecimento positivo, traduzindo-se na figura seguinte:

```
% Invariante que permite adicionar conhecimento negativo
+(-vacinacao_Covid(IDStaff,IDUtente,Data,Vacina,Toma)) :: (
    integer(IDStaff),
    integer(IDUtente),
    validaData(Data),
    atom(Vacina),
    integer(Toma)).
```

**Figura 50:** Adição Conhecimento Negativo Vacinação (1)

Além disso, foi necessário também efetuar um invariante de modo a limitar a inserção de novos conhecimentos negativos, ou seja, para inserir um registo negativo de vacinação é necessário que o utente exista, que esse registo de vacinação não exista previamente na BC e é também necessário que o *Staff* pertença ao mesmo centro onde o utente foi vacinado. Em *prolog,* este invariante traduz se do seguinte modo:

```
% Limitar a insercao de novos conhecimentos negativos
+(-vacinacao_Covid(Staff,Id,Data,Vacina,Toma)) :: (
    solucoes(Id,-vacinacao_Covid(_,Id,_,_,_),S1),
    comprimento( S1,N1 ), N1 == 2, % Nao pode ja estar na BC
    solucoes(Id,utente(Id,_,_,_,_,_,_,_,_,_),S),
    comprimento( S,N ), N == 1, %ID tem de ser de um utente,
    verificaCentro(Id,Staff)). % tem de ser vacinado no centroSaude respetivo
```

**Figura 51:** Adição Conhecimento Negativo Vacinação (2)

## REMOÇÃO CONHECIMENTO PERFEITO POSITIVO

Além da inserção de conhecimento positivo, consideramos relevante também permitir a remoção do mesmo. De modo a concretizar este objetivo, foi necessário implementar 2 invariantes.

O primeiro invariante verifica os tipos de dados inseridos, ou seja, se um determinado parâmetro deverá ser um *integer* ou um *atom*, por exemplo. Este invariante foi implementado da seguinte forma:

```
% Invariante que permite remocao conhecimento positivo
-vacinacao_Covid(IDStaff,IDUtente,Data,Vacina,Toma) :: (
    integer(IDStaff),
    integer(IDUtente),
    validaData(Data),
    atom(Vacina),
    integer(Toma)).
```

**Figura 52:** Remoção Conhecimento Positivo Vacinação (1)

Além disso, também foi implementado um invariante que permite limitar a remoção de conhecimento positivo, ou seja, para remover um registo de vacina da 1ª fase é necessário que não exista um registo para esse mesmo utente na 2ª fase. Os registos da 2ª fase podem ser removidos livremente. Este invariante foi implementado em *prolog*, como está ilustrado de seguida:

```
% Invariante que permite limitar a remocao de conhecimento positivo
-vacinacao_Covid(IDStaff,Id,Data,Vacina,1) :: (
    solucoes(Id,vacinacao_Covid(_,Id,_,_,1),S ),
    comprimento(S,N), N==0,
    solucoes(Id,vacinacao_Covid(_,Id,_,_,2),S1 ),
    comprimento(S1,N1), N1==0).

-vacinacao_Covid(IDStaff,Id,Data,Vacina,2) :: (
    solucoes(Id,vacinacao_Covid(_,Id,_,_,2),S),
    comprimento(S,N), N==0).
```

**Figura 53:** Remoção Conhecimento Positivo Vacinação (2)

## REMOÇÃO CONHECIMENTO PERFEITO NEGATIVO

No caso da remoção de conhecimento perfeito negativo, foi necessário considerar 2 invariantes. Um deles tem a função de verificar os tipos de dados a remover, que terão que ser válidos, e o outro visa limitar a remoção de conhecimento negativo.

```
% Invariante que permite remocao conhecimento negativo
-(-vacinacao_Covid(IDStaff,IDUtente,Data,Vacina,Toma)) :: (
    integer(IDStaff),
    integer(IDUtente),
    validaData(Data),
    atom(Vacina),
    integer(Toma)).
```

**Figura 54:** Remoção Conhecimento Negativo Vacinação (1)

Para o segundo caso verificamos se o conhecimento removido não se encontra na BC, contudo no final da remoção só pode haver um conhecimento negativo (o predicado que permite identificar o conhecimento negativo – figura x) e por isso verificamos se a lista de soluções encontradas tem comprimento igual a 1.

```
% Invariante que permite limitar a remocao de conhecimento negativo
-(-vacinacao_Covid(IDStaff,IDUtente,Data,Vacina,Toma)) :: (
    solucoes(Id,-vacinacao_Covid(IDStaff,IDUtente,Data,Vacina,Toma),S ),
    comprimento(S,N), N==1).
```

**Figura 55:** Remoção Conhecimento Negativo Vacinação (2)

## INVARIANTES CANCELA VACINA

No caso do cancelamento de uma vacina, consideramos pertinente implementar apenas adição e remoção de conhecimento perfeito, uma vez que achamos por bem impedir que os registos de cancelamento sejam incorretos ou incompletos evitando assim inconsistências na BC. Nas seguintes secções cada operação de remoção ou adição será explorada em maior detalhe.

### ADIÇÃO CONHECIMENTO PERFEITO POSITIVO

Começou-se por implementar um invariante que verifica o tipo de dados inserido. Os parâmetros fornecidos ao predicado *cancelaVacina* são os parâmetros ID de Utente e Data de cancelamento. Deste modo, prodeceu-se à verificação do valor ID que deverá corresponder a um *integer* e à validação da data de modo a constituir uma data válida, recorrendo à função *validaData.* O seguinte invariante traduz este pressuposto:

```
% Invariante que permite adicionar conhecimento positivo
+cancela_vacina(Id,Data) :: (integer(Id),
                             validaData(Data)).
```

**Figura 56:** Adição Conhecimento Positivo Cancela Vacina (1)

Para adicionar um registo de cancelamento de vacina propriamente dito, foi necessário efetuar a verificação de diversas condições: verificar se existe um único utente com o ID fornecido, verificar que apenas existe um registo de cancelamento de vacina e que ainda não foram administradas a 1 e 2 doses da vacina. A figura 57 representa o invariante de adição de conhecimento.

```
% Limitar a insercao de conhecimento positivo
+cancela_vacina(Id,Data) :: (solucoes(Id,utente(Id,_,_,_,_,_,_,_,_,_),S),
                             comprimento( S,N ), N == 1,
                             solucoes(Id,cancela_vacina(Id,_),S0),
                             comprimento( S0,N0 ), N0 == 1,
                             solucoes(Id,vacinacao_Covid(_,Id,_,Vacina,1),S1),
                             comprimento( S1,N1 ), N1 == 0,
                             solucoes(Id,vacinacao_Covid(_,Id,_,Vacina,2),S2),
                             comprimento( S2,N2 ), N2 == 0, !).
```

**Figura 57:** Adição Conhecimento Positivo Cancela Vacina (2)

## ADIÇÃO CONHECIMENTO PERFEITO NEGATIVO

Para adicionar conhecimento perfeito negativo relativo ào *cancela_vacina*, começou-se por criar um invariante que verifica os parâmetros fornecidos ao predicado. Estes parâmetros são o ID de utilizador que pretende cancelar a vacina e a data de cancelamento. Assim, o ID deverá ser um *integer* e a data deverá ser válida, facto verificado pela regra *validaData.* O invariante que garante a validade dos parâmetros é o seguinte:

```
% Invariante que permite adicionar conhecimento positivo
+(-cancela_vacina(Id,Data)) :: (integer(Id),
                                validaData(Data)).
```

**Figura 58:** Adição Conhecimento Negativo Cancela Vacina (1)

Além disso, é também necessário limitar a inserção de conhecimento negativo, sendo que o utente deverá existir na BC e não deverá permitir adicionar conhecimento repetido. Isto é garantido através do seguinte invariante:

```
% Limitar a insercao de conhecimento negativo
+(-cancela_vacina(Id,Data)) :: (solucoes(Id,utente(Id,_,_,_,_,_,_,_,_,_),S),
                                comprimento( S,N ), N == 1, % Utente existe
                                solucoes(Id,-cancela_vacina(Id,_),S0),
                                comprimento( S0,N0 ), N0 == 2). % Nao pode adicionar conhecimento repetido
```

**Figura 59:** Adição Conhecimento Negativo Cancela Vacina (2)

## REMOÇÃO CONHECIMENTO PERFEITO POSITIVO

Primeiramente é necessário verificar os dados inseridos e o seu tipo, através do primeiro invariante implementado:

```
% Invariante que permite remocao conhecimento positivo
-cancela_vacina(Id,Data) :: (
    integer(Id),
    validaData(Data)).
```

**Figura 60:** Remoção Conhecimento Positivo Cancela Vacina (1)

Para remover um registo de cancelamento foi necessário implementar um invariante estrutural que garante a existência do utente em questão e a inexistência daquele mesmo registo na base de conhecimento. A seguinte figura traduz este invariante:

```
% Invariante que permite limitar a remocao de conhecimento positivo
-cancela_vacina(Id,Data) :: (
    solucoes(Id,utente(Id,_,_,_,_,_,_,_,_,_),S),
    comprimento( S,N ), N == 1,
    solucoes(Id,cancela_vacina(Id,Data),S1),
    comprimento( S1,N1 ), N1 == 0).
```

**Figura 61:** Remoção Conhecimento Positivo Cancela Vacina (2)

## REMOÇÃO CONHECIMENTO PERFEITO NEGATIVO

De forma a remover conhecimento perfeito negativo relativo ao cancelamento de uma vacina, é necessário verificar primeiramente os tipos de dados do predicado a remover, tendo em conta o invariante seguinte:

```
% Invariante que permite remover conhecimento negativo
-(-cancela_vacina(Id,Data)) :: (integer(Id),
                                validaData(Data)).
```

**Figura 62:** Remoção Conhecimento Negativo Cancela Vacina (1)

Além disso, é também conveniente limitar esta remoção de forma a que apenas seja possível remover caso o utente exista, tal como visível na imagem seguinte:

```
% Invariante que permite limitar a remocao de conhecimento negativo
-(-cancela_vacina(IdUtente,Data)) :: (
    solucoes(Id,utente(IdUtente,_,_,_,_,_,_,_,_,_),S),
    comprimento( S,N ), N == 1,
    solucoes(Id,-cancela_vacina(IdUtente,Data),S1),
    comprimento( S1,N1 ), N1 == 1).
```

**Figura 63:** Remoção Conhecimento Negativo Cancela Vacina (2)

## EVOLUÇÃO

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

O conteúdo relativo a este tópico, tal como foi referido anteriormente, encontra-se atribuído a um ficheiro individual "EvolucaoInvolucao.pl". Este ficheiro contém inicialmente um conjunto de definições iniciais com o propósito de permitir a definição de invariantes para os diferentes tipos de conhecimento. Na seguinte figura é possível observar as definições iniciais deste documento:

```
:- op( 900,xfy,'::' ).  %- criação de um novo operador para o prolog

%conhecimento interdito
:- op(900,xfy,:~:).

%conhecimento impreciso
:- op(900,xfy,:-:).
```

**Figura 64:** Criação dos Operadores

## EXTENSÃO DO PREDICADO EVOLUÇÃO

Os predicados implementados têm como objetivo permitir evolução de conhecimento perfeito positivo, perfeito negativo, imperfeito impreciso e incerto.

Implementou-se a regra *evolucao(Termo, positivo)* com o propósito de adicionar novo conhecimento positivo. Para a sua implementação, o procedimento passou por efetuar uma procura por invariantes correspondentes a um dado *Termo*, agrupando os resultados numa *Lista*, seguida da inserção desse mesmo termo e da invocação da regra *teste* à *Lista*. A procura é feita com recurso à regra *solucoes* (secção Regras Auxiliares). A inserção recorre à regra *insercao* que efetua um *assert*. A regra *teste* é abordada na secção "Regras Auxiliares". Finalmente, invocamos a regra *idInsertTermo* (que é descrita na secção "Regras Auxiliares") com o objetivo de adicionar à BC um registo do tipo do conhecimento perfeito inserido. A regra que permite concretizar este objetivo é o seguinte:

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

**Figura 65:** Predicado Evolução Positivo

Em adição, criamos a regra *evolucao(Termo, negativo)* que procede de modo análogo à anterior com a exceção do tipo de conhecimento a adicionar, que é negativo e traduz-se pelo seguinte regra:

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

**Figura 66:** Predicado Evolução Negativo

Além disso, criamos também uma regra *evolucao(Termo, impreciso)* para permitir a inserção de conhecimento imperfeito do tipo impreciso. Esta regra começa por verificar a existência de invariantes referentes ao conhecimento impreciso, de seguida efetua a inserção desse termo como uma exceção (conhecimento desconhecido), e por fim, procede ao teste da lista retornada pelo predicado *solucoes.* É de notar que a procura por invariantes deste tipo utiliza o operador :-: que trata de conhecimento impreciso. Esta regra foi implementada da seguinte forma:

```
evolucao(Termo, impreciso) :- solucoes(Invariante,+Termo:-:Invariante,ListaImpreciso),
                              insercao(excecao(Termo)),
                              teste(ListaImpreciso).
```

**Figura 67:** Predicado Evolução Impreciso

Por fim, procedemos à criação da regra *evolucao(Termo, incerto)* para a inserção de conhecimento interdito e incerto. Esta regra começa por verificar a existência de invariantes para o conhecimento perfeito como para o incerto/interdito. Assim, verifica-se as regras de inserção de conhecimento incerto/interdito (exceções que determinam conhecimento desconhecido) e as regras de inserção de conhecimento perfeito de modo a inserir corretamente o termo na BC. De seguida o termo é inserido, e por fim, é efetuado o teste às listas anteriores. Esta implementação traduz-se da seguinte forma:

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

**Figura 68:** Predicado Evolução Incerto

## EVOLUÇÃO CONHECIMENTO INCERTO

Para o conhecimento incerto, consideramos relevante implementar mecanismos de evolução de conhecimento deste tipo para o utente, centro de saúde e *staff*. No caso do utente, considerou-se a evolução do conhecimento incerto para os campos nome, morada e *email.* Para o centro de saúde, considerou-se a evolução para o conhecimento incerto relativo ao nome, morada e *email*. Finalmente, para o *staff* consideramos conveniente permitir a evolução de conhecimento incerto relativo ao nome e ao *email*.

Para cada caso, implementou-se a regra evolução que recebe como argumentos o utente em questão, um átomo "incerto" e um átomo do campo respetivo (nome, morada, email, etc.). O procedimento implementado passa por invocar a regra *evolucao(Termo, incerto) que* permite inserir o conhecimento incerto na BC. De seguida, é efetuada a invocação da regra *inserção* para inserir na base de conhecimento uma exceção, tornando-o deste modo um conhecimento desconhecido. Por fim, é efetuada a adição do predicado incerto que reconhece o termo inserido como incerto. Para os restantes campos (*email*, morada, nome, etc.) e termos (*centro_saude e utente*) o procedimento e raciocínio é análogo tal como podemos ver nas seguintes imagens.

### EVOLUÇÃO UTENTE INCERTO

```
% Caso Email incerto
evolucao(utente(Id,SS,Nome,Data,Email_Incerto,Telefone,Morada,Profissao,Doenca,Centro), incerto, email) :-
    evolucao(utente(Id,SS,Nome,Data,Email_Incerto,Telefone,Morada,Profissao,Doenca,Centro), incerto),
    insercao((excecao(utente(I,S,N,D,E,T,M,P,LD,C)) :-
                utente(I,S,N,D,Email_Incerto,T,M,P,LD,C))),
    insercao(incerto(utente(Id))).
```

**Figura 69:** Email Incerto

```
% Caso Nome incerto
evolucao(utente(Id,SS,Nome_Incerto,Data,Email,Telefone,Morada,Profissao,Doenca,Centro), incerto, nome) :-
    evolucao(utente(Id,SS,Nome_Incerto,Data,Email,Telefone,Morada,Profissao,Doenca,Centro), incerto),
    insercao(excecao(utente(I,S,N,D,E,T,M,P,LD,C)) :-
                utente(I,S,Nome_Incerto,D,E,T,M,P,LD,C)),
    insercao(incerto(utente(Id))).
```

**Figura 69:** Nome Incerto

```
% Caso Morada incerto
evolucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Incerto,Profissao,Doenca,Centro), incerto, morada) :-
    evolucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Incerto,Profissao,Doenca,Centro), incerto),
    insercao((excecao(utente(I,S,N,D,E,T,M,P,LD,C)) :-
                utente(I,S,N,D,E,T,Morada_Incerto,P,LD,C))),
    insercao(incerto(utente(Id))).
```

**Figura 71:** Morada Incerta

EVOLUÇÃO CENTRO DE SAÚDE INCERTO

```
% Morada Incerto
evolucao(centro_saude(ID,Nome,Morada_Incerto,Telefone,Email), incerto, morada) :-
    evolucao(centro_saude(ID,Nome,Morada_Incerto,Telefone,Email), incerto),
    insercao(excecao(centro_saude(I,N,M,T,E)) :-
                centro_saude(I,N,Morada_Incerto,T,E)),
    insercao(incerto(centro_saude(ID))).
```

**Figura 72:** Morada Incerta

```
% Email incerto
evolucao(centro_saude(ID,Nome,Morada,Telefone,Email_Incerto), incerto, email) :-
    evolucao(centro_saude(ID,Nome,Morada,Telefone,Email_Incerto), incerto),
    insercao(excecao(centro_saude(I,N,M,T,E)) :-
                centro_saude(I,N,M,T,Email_Incerto)),
    insercao(incerto(centro_saude(ID))).
```

**Figura 71:** Email Incerto

```
% Nome incerto
evolucao(centro_saude(ID,Nome_Incerto,Morada,Telefone,Email), incerto, nome) :-
    evolucao(centro_saude(ID,Nome_Incerto,Morada,Telefone,Email), incerto),
    insercao(excecao(centro_saude(I,N,M,T,E)) :-
                centro_saude(I,Nome_Incerto,M,T,E)),
    insercao(incerto(centro_saude(ID))).
```

**Figura 70:** Nome Incerto

EVOLUÇÃO STAFF INCERTO

```
% Caso email incerto
evolucao(staff(Ids,Idc,Nome,Email_Incerto), incerto, email) :-
    evolucao(staff(Ids,Idc,Nome,Email_Incerto), incerto),
    insercao((excecao(staff(Is,Ic,N,E)) :-
                staff(Is,Ic,N,Email_Incerto))),
    insercao(incerto(staff(Ids))).
```

**Figura 73:** Email Incerto

```
% Caso nome incerto
evolucao(staff(Ids,Idc,Nome_Incerto,Email), incerto, nome) :-
    evolucao(staff(Ids,Idc,Nome_Incerto,Email), incerto),
    insercao((excecao(staff(Is,Ic,N,E)) :-
                staff(Is,Ic,Nome_Incerto,E))),
    insercao(incerto(staff(Ids))).
```

**Figura 74:** Nome Incerto

## EVOLUÇÃO CONHECIMENTO IMPRECISO

Para o conhecimento impreciso, consideramos relevante implementar mecanismos de evolução de conhecimento deste tipo para o utente, centro de saúde e *staff*. No caso do utente, considerou-se o elemento morada como impreciso. Para o centro de saúde, consideraram se que este poderia ser impreciso no elemento *email*. Finalmente, no caso do *staff* foi considerado conhecimento impreciso para o elemento *email*.

Para permitir a evolução deste tipo de conhecimento com este elemento impreciso, foi necessário implementar duas regras.

A regra implementada recebe como parâmetros o Termo a adicionar, um átomo "impreciso" e um átomo do respetivo campo (*email* ou morada). Na primeira vertente desta regra, é efetuada uma procura por registos de conhecimento impreciso relativos ao utente de forma a verificar se o termo a inserir já está registado na BC como impreciso. Caso não esteja então o id do termo vai ser registado na BC como novo conhecimento impreciso através do predicado impreciso. Após esta verificação, efetua-se a invocação à regra *evolucao(Termo, impreciso)* que adiciona á BC o conhecimento impreciso se este cumprir com os invariantes. A outra vertente desta regra, foi criada caso o conhecimento a inserir já esteja registado na BC como impreciso. Para os termos (*centro_saude, utente, staff*) e respetivo campo o procedimento e raciocínio é análogo tal como podemos ver nas imagens.

EVOLUÇÃO UTENTE IMPRECISO

```
% Insercao de conhecimento imperciso - morada
evolucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Impreciso,Profissao,Doenca,Centro), impreciso, morada) :-
    solucoes(Id, impreciso(utente(Id)), S),
    nao(membro(Id, S)),
    evolucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Impreciso,Profissao,Doenca,Centro),impreciso),
    insercao(impreciso(utente(Id))).

evolucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Impreciso,Profissao,Doenca,Centro), impreciso, morada) :-
    solucoes(Id, impreciso(utente(Id)), S),
    membro(Id, S),
    evolucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Impreciso,Profissao,Doenca,Centro),impreciso).
```

**Figura 75:** Morada Imprecisa

EVOLUÇÃO CENTRO SAÚDE IMPRECISO

```
% Insercao de conhecimento imperciso - email
evolucao(centro_saude(ID,Nome,Morada,Telefone,Email_Impreciso), impreciso, email) :-
    solucoes(ID,impreciso(centro_saude(ID)),S),
    nao(membro(ID,S)),
    evolucao(centro_saude(ID,Nome,Morada,Telefone,Email_Impreciso),impreciso),
    insercao(impreciso(centro_saude(ID))).

evolucao(centro_saude(ID,Nome,Morada,Telefone,Email_Impreciso), impreciso, email) :-
    solucoes(ID,impreciso(centro_saude(ID)),S),
    membro(ID,S),
    evolucao(centro_saude(ID,Nome,Morada,Telefone,Email_Impreciso),impreciso).
```

**Figura 76:** Email Impreciso

EVOLUÇÃO STAFF IMPRECISO

```
% Insercao de conhecimento imperciso - email
evolucao(staff(ID,Idc,Nome,Email_Impreciso), impreciso, email) :-
    solucoes(ID,impreciso(staff(ID)),S),
    nao(membro(ID,S)),
    evolucao((staff(ID,Idc,Nome,Email_Impreciso)),impreciso),
    insercao(impreciso(staff(Id))).

evolucao(staff(ID,Idc,Nome,Email_Impreciso), impreciso, email) :-
    solucoes(ID,impreciso(staff(ID)),S),
    membro(ID,S),
    evolucao((staff(ID,Idc,Nome,Email_Impreciso)),impreciso).
```

**Figura 77:** Email Impreciso

## EVOLUÇÃO CONHECIMENTO INTERDITO

Para o conhecimento interdito, consideramos relevante implementar mecanismos de evolução de conhecimento deste tipo para o utente, *staff* e centro de saúde. Para o utente consideramos que o elemento que o tornaria interdito seria o *email*. Consideramos que o elemento que o tornaria o *staff* interdito seria o *email*. Finalmente, no caso do centro de saúde consideramos que o elemento que o tornaria interdito seria o *email*

Para cumprir coma evolução do conhecimento interdito foi implementada uma regra que recebe como parâmetros o termo em questão e respetivo campo assim como o átomo "interdito". Começa-se por efetuar a invocação da regra *evolucao(Termo,incerto)* que tanto insere um conhecimento incerto como interdito. De seguida, insere-se uma exceção que permite identificar esse termo como incerto e desconhecido. Após essa inserção são efetuadas a inserção do predicado nulo referente ao valor do campo interdito e a inserção de um registo que identifica o termo inserido como interdito. Para os termos *centro_saude, utente e staff* e respetivo campo o procedimento e raciocínio é análogo tal como podemos ver nas imagens abaixo.

EVOLUÇÃO UTENTE INTERDITO

```
% Caso email interdito
evolucao(utente(Id,SS,Nome,Data,Email_Interdito,Telefone,Morada,Profissao,Doenca,Centro), interdito, email) :-
    evolucao(utente(Id,SS,Nome,Data,Email_Interdito,Telefone,Morada,Profissao,Doenca,Centro), incerto),
    insercao(excecao(utente(I,S,N,D,E,T,M,P,LD,C)) :-
                        utente(I,S,N,D,Email_Interdito,T,M,P,LD,C)),
    insercao(nulo(Email_Interdito)),
    insercao(interdito(utente(Id))).
```

**Figura 78:** Email Interdito

EVOLUÇÃO CENTRO DE SAÚDE INTERDITO

```
% Caso email interdito
evolucao(centro_saude(ID,Nome,Morada,Telefone,Email_Interdito), interdito, email) :-
    evolucao(centro_saude(ID,Nome,Morada,Telefone,Email_Interdito), incerto),
    insercao(excecao(centro_saude(I,N,M,T,E)) :-
                    centro_saude(I,N,M,T,Email_Interdito)),
    insercao(nulo(Email_Interdito)),
    insercao(interdito(centro_saude(ID))).
```

**Figura 79:** Email Interdito

EVOLUÇÃO STAFF INTERDITO

```
% Caso email interdito
evolucao(staff(Ids,Idc,Nome,Email_Interdito), interdito, email) :-
    evolucao(staff(Ids,Idc,Nome,Email_Interdito), incerto),
    insercao((excecao(staff(Is,Ic,N,E)) :-
                    staff(Is,Ic,N,Email_Interdito))),
    insercao((nulo(Email_Interdito))),
    insercao(interdito(staff(Ids))).
```

**Figura 80:** Email Interdito

# INVOLUÇÃO

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

O conteúdo relativo a este tópico, tal como a evolução, encontra-se atribuído a um ficheiro individual “EvolucaoInvolucao.pl”.

## EXTENSÃO DO PREDICADO INVOLUÇÃO

Os predicados implementados têm como objetivo permitir involução de conhecimento perfeito positivo, perfeito negativo, imperfeito impreciso e incerto.

Implementou-se a regra *involucao(Termo, positivo)* com o propósito de remover conhecimento positivo. Para a sua implementação, o procedimento passou por efetuar uma procura por invariantes correspondentes a um dado *Termo*, agrupando os resultados numa *Lista*, seguida da remoção desse mesmo termo e da invocação da regra *teste* à *Lista*. A procura é feita com recurso à regra *solucoes* (secção Regras Auxiliares). A inserção recorre à regra *insercao* que efetua um *assert*. Por fim é invocada a regra *idRemoveTermo* (que é descrita na secção “Regras Auxiliares”) com o objetivo de remover da BC um registo do predicado perfeito associado ao termo. A regra que permite concretizar este objetivo é o seguinte:

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

**Figura 81:** Predicado Involução Positivo

Em adição, foi criada a regra *involucao(Termo, negativo)* que procede de modo análogo à anterior com a exceção de que o tipo de conhecimento é negativo e traduz-se pelo seguinte regra:

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

**Figura 82:** Predicado Involução Negativo

Além disso, criamos também uma regra *involucao(Termo, impreciso)* para permitir a remoção de conhecimento imperfeito do tipo impreciso. Esta regra começa por verificar a existência de invariantes referentes ao termo em questão. Os invariantes utilizados neste predicado permitem a remoção de conhecimento segundo as regras utilizadas para o conhecimento perfeito evitando assim uma BC inconsistente. De seguida efetua a remoção DA exceção desse termo visto que se trata de conhecimento imperfeito, caso o termo cumpra com as regras dos invariantes. Esta regra foi implementada da seguinte forma:

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

**Figura 83:** Predicado Involução Impreciso

Por fim, procedemos à criação da regra *involucao(Termo, incerto)* para a remoção de conhecimento interdito e incerto. Esta regra começa por verificar a existência de invariantes com o operador ::, pelas mesmas razões explicadas na involução de conhecimento impreciso. De seguida o termo é removido caso este cumpra com as regras dos invariantes. Esta implementação traduz-se da seguinte forma:

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

**Figura 84:** Predicado Involução Incerto

## INVOLUÇÃO CONHECIMENTO INCERTO

Para o conhecimento impreciso, consideramos relevante implementar mecanismos de involução de conhecimento deste tipo para o utente, centro de saúde e *staff*. Para o utente, foi conveniente remover conhecimento no caso do nome, *email* e morada. Para o centro de saúde, considerou-se a involução para o conhecimento incerto relativo ao nome, morada e *email*. Finalmente, para o *staff* consideramos conveniente permitir a involução de conhecimento incerto para o nome e *email*.

Para cada caso, implementou-se a regra involução que recebe como argumentos o termo em questão, um átomo "incerto" e um átomo do campo respetivo (nome, morada, email, ect). O procedimento implementado passa por invocar a regra *involucao(Termo, incerto) que* permite remover o conhecimento incerto na BC. De seguida, é efetuada a invocação da regra *remoção* para remover da base a exceção que o tornava um conhecimento desconhecido. Por fim, é efetuada a remoção do predicado incerto que reconhecia como um o termo incerto. Caso o termo não se encontre na BC então não será possível remover o termo e por isso a involução devolve *false*. Para os restantes campos (*email*, morada, nome, ect) e termos *centro_saude, utente e staff* o procedimento e raciocínio é análogo tal como podemos ver nas seguintes imagens.

INVOLUÇÃO UTENTE INCERTO

```
% Caso numero Nome incerto
involucao(utente(Id,SS,Nome_Incerto,Data,Email,Telefone,Morada,Profissao,Doenca,Centro), incerto, nome) :-
    involucao(utente(Id,SS,Nome_Incerto,Data,Email,Telefone,Morada,Profissao,Doenca,Centro), incerto),
    remocao((excecao(utente(I,S,N,D,E,T,M,P,LD,C)) :-
                utente(I,S,Nome_Incerto,D,E,T,Morada,P,LD,C))),
    remocao(incerto(utente(Id))).
```

**Figura 85:** Nome Incerto

```
% Caso numero Email incerto
involucao(utente(Id,SS,Nome,Data,Email_Incerto,Telefone,Morada,Profissao,Doenca,Centro), incerto, email) :-
    involucao(utente(Id,SS,Nome,Data,Email_Incerto,Telefone,Morada,Profissao,Doenca,Centro), incerto),
    remocao((excecao(utente(I,S,N,D,E,T,M,P,LD,C)) :-
                utente(I,S,Nome,D,Email_Incerto,T,Morada,P,LD,C))),
    remocao(incerto(utente(Id))).
```

**Figura 86:** Email Incerto

```
% Caso numero Morada incerto
involucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Incerto,Profissao,Doenca,Centro), incerto, morada) :-
    involucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Incerto,Profissao,Doenca,Centro), incerto),
    remocao((excecao(utente(I,S,N,D,E,T,M,P,LD,C)) :-
                utente(I,S,Nome,D,E,T,Morada_Incerto,P,LD,C))),
    remocao(incerto(utente(Id))).
```

**Figura 87:** Morada Incerta

## INVOLUÇÃO CENTRO DE SAÚDE INCERTO

```
% Nome incerto
involucao(centro_saude(ID,Nome_Incerto,Morada,Telefone,Email), incerto, nome) :-
    involucao(centro_saude(ID,Nome_Incerto,Morada,Telefone,Email), incerto),
    remocao((excecao(centro_saude(I,N,M,T,E)) :-
                centro_saude(I,Nome_Incerto,Morada,T,E))),
    remocao(incerto(centro_saude(ID))).
```

**Figura 88:** Nome Incerto

```
% Morada Incerto
involucao(centro_saude(ID,Nome,Morada_Incerto,Telefone,Email), incerto, morada) :-
    involucao(centro_saude(ID,Nome,Morada_Incerto,Telefone,Email), incerto),
    remocao((excecao(centro_saude(I,N,M,T,E)) :-
                centro_saude(I,Nome,Morada_Incerto,T,E))),
    remocao(incerto(centro_saude(ID))).
```

**Figura 89:** Morada Incerta

```
% Email incerto
involucao(centro_saude(ID,Nome,Morada,Telefone,Email_Incerto), incerto, email) :-
    involucao(centro_saude(ID,Nome,Morada,Telefone,Email_Incerto), incerto),
    remocao(excecao(centro_saude(I,N,M,T,E)) :-
                centro_saude(I,Nome,M,T,Email_Incerto)),
    remocao(incerto(centro_saude(ID))).
```

**Figura 90:** Email Incerto

## INVOLUÇÃO STAFF INCERTO

```
% Caso nome incerto
involucao(staff(Ids,Idc,Nome_Incerto,Email), incerto, nome) :-
    involucao(staff(Ids,Idc,Nome_Incerto,Email), incerto),
    remocao((excecao(staff(Is,Ic,N,E)) :-
                staff(Is,Ic,Nome_Incerto,Email))),
    remocao(incerto(staff(Ids))).
```

**Figura 91:** Nome Incerto

```
% Caso email incerto
involucao(staff(Ids,Idc,Nome,Email_Incerto), incerto, email) :-
    involucao(staff(Ids,Idc,Nome,Email_Incerto), incerto),
    remocao((excecao(staff(Is,Ic,N,E)) :-
                staff(Is,Ic,Nome,Email_Incerto))),
    remocao(incerto(staff(Ids))).
```

**Figura 92:** Email Incerto

## INVOLUÇÃO CONHECIMENTO IMPRECISO

Para o conhecimento impreciso, consideramos relevante implementar mecanismos de involução de conhecimento deste tipo para o utente, centro de saúde e *staff*. No caso do utente, considerou-se o elemento morada como impreciso. Para o centro de saúde, consideraram se que este poderia ser impreciso no elemento *email*. Finalmente, no caso do *staff* foi considerado conhecimento impreciso para o elemento *email*.

A regra implementada recebe como parâmetros o termo a adicionar, um átomo "impreciso" e um átomo do respetivo campo (*email* ou morada). Em primeiro, é efetuada uma procura por registos de conhecimento impreciso relativos ao id do termo em questão de forma a verificar se o termo a remover será o último a ser removido. Caso seja então o predicado incerto do termo é removido. Caso o termo não seja o último, isto é, se ainda houver hipóteses de incerteza na BC então o predicado não é removido. Após esta verificação, efetua-se a invocação à regra in*volucao(Termo, impreciso)* que adiciona à BC o conhecimento impreciso se este cumprir com os invariantes. Para os termos *centro_saude, utente e staff* e respetivo campo o procedimento e raciocínio é análogo tal como podemos ver nas imagens.

### INVOLUÇÃO UTENTE IMPRECISO

```
% Remocao de conhecimento impreciso - morada
involucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Impreciso,Profissao,Doenca,Centro), impreciso, morada) :-
    solucoes((Id,M), excecao(utente(Id,_,_,_,_,_,_,_,_,_)), L),
    comprimento(L,N),
    N == 1,
    involucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Impreciso,Profissao,Doenca,Centro),impreciso),
    remocao(impreciso(utente(Id))).
involucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Impreciso,Profissao,Doenca,Centro), impreciso, morada) :-
    solucoes((Id,M), excecao(utente(Id,_,_,_,_,_,_,_,_,_)), L),
    comprimento(L,N),
    N > 1,
    involucao(utente(Id,SS,Nome,Data,Email,Telefone,Morada_Impreciso,Profissao,Doenca,Centro),impreciso).
```

**Figura 93:** Morada Imprecisa

### INVOLUÇÃO CENTRO DE SAÚDE IMPRECISO

```
involucao(centro_saude(ID,Nome,Morada,Telefone,Email_Impreciso), impreciso, email) :-
    solucoes((ID,E), excecao(centro_saude(ID,_,_,_,_)), L),
    comprimento(L,N),
    N > 1,
    involucao(centro_saude(ID,Nome,Morada,Telefone,Email_Impreciso), impreciso).
involucao(centro_saude(ID,Nome,Morada,Telefone,Email_Impreciso), impreciso, email) :-
    solucoes((ID,E), excecao(centro_saude(ID,_,_,_,_)), L),
    comprimento(L,N),
    N == 1,
    involucao(centro_saude(ID,Nome,Morada,Telefone,Email_Impreciso), impreciso),
    remocao(impreciso(centro_saude(ID))).
```

**Figura 94:** Email Impreciso

INVOLUÇÃO STAFF IMPRECISO

```
% Remocao de conhecimento impreciso - email
involucao(staff(Ids,Idc,Nome,Email_Impreciso), impreciso, email) :-
    solucoes((Ids,E), excecao(staff(Ids,_,_,_)), L),
    comprimento(L,N),
    N > 1,
    involucao((staff(Ids,Idc,Nome,Email_Impreciso)),impreciso).

involucao(staff(Ids,Idc,Nome,Email_Impreciso), impreciso, email) :-
    solucoes((Ids,E), excecao(staff(Ids,_,_,_)), L),
    comprimento(L,N),
    N == 1,
    involucao((staff(Ids,Idc,Nome,Email_Impreciso)),impreciso),
    remocao(impreciso(staff(Ids))).
```

**Figura 95:** Email Impreciso

## INVOLUÇÃO CONHECIMENTO INTERDITO

Para o conhecimento interdito, consideramos relevante implementar mecanismos de involução de conhecimento deste tipo para o utente, *staff* e centro de saúde. Para o utente consideramos que o elemento que o tornaria interdito seria o *email*. Consideramos que o elemento que o tornaria o *staff* interdito seria o *email*. Finalmente, no caso do centro de saúde consideramos que o elemento que o tornaria interdito seria o *email*

Para cumprir com a involução do conhecimento interdito foi implementada uma regra que recebe como parâmetros o termo em questão e respetivo campo assim como o átomo "interdito". Começa-se por efetuar a invocação da regra *involucao(Termo,incerto)* que tanto remove um conhecimento incerto como interdito. De seguida, remove-se a exceção que permite identificar esse termo como incerto e desconhecido. Após essa remoção são removidas o predicado nulo referente ao valor do campo interdito e a o registo que identifica o termo inserido como interdito. Para os termos *centro_saude, utente e staff* e respetivo campo o procedimento e raciocínio é análogo tal como podemos ver nas imagens abaixo.

INVOLUÇÃO UTENTE INTERDITO

```
% Caso email interdito
involucao(utente(Id,SS,Nome,Data,Email_interdito,Telefone,Morada,Profissao,Doenca,Centro), interdito, email) :-
    involucao(utente(Id,SS,Nome,Data,Email_interdito,Telefone,Morada,Profissao,Doenca,Centro), incerto),
    remocao(excecao(utente(IdUt,S,N,D,E,T,M,P,LD,C)) :-
                utente(IdUt,S,Nome,D,Email_interdito,T,M,P,LD,C)),
    remocao((nulo(Email_interdito))),
    remocao(interdito(utente(Id))).
```

**Figura 96:** Email Interdito

INVOLUÇÃO CENTRO DE SAÚDE INTERDITO

```
% Caso email interdito
involucao(centro_saude(ID,Nome,Morada,Telefone,Email_Interdito), interdito, email) :-
    involucao(centro_saude(ID,Nome,Morada,Telefone,Email_Interdito), incerto),
    remocao(excecao(centro_saude(I,N,M,T,E)) :-
                centro_saude(I,Nome,M,T,Email_Interdito)),
    remocao(nulo(Email_Interdito)),
    remocao(interdito(centro_saude(ID))).
```

**Figura 97:** Email Incerto

INVOLUÇÃO CENTRO DE SAÚDE INTERDITO

```
% Caso email interdito
involucao(staff(Ids,Idc,Nome,Email_Interdito), interdito, email) :-
    involucao(staff(Ids,Idc,Nome,Email_Interdito), incerto),
    remocao(excecao(staff(Is,Ic,N,E)) :-
                staff(Is,Ic,Nome,Email_Interdito)),
    remocao((nulo(Email_Interdito))),
    remocao(interdito(staff(Ids))).
```

**Figura 98:** Email Interdito

# ANÁLISE DE RESULTADOS

Nesta secção é apresentado o resultado de testes efetuados no sistema de conhecimento implementado.

## EVOLUÇÃO

De modo a testar a regra *evolucao* implementada, procedemos à inserção na base de conhecimento de registos do centro de saúde como conhecimento positivo. Dado que este registo ainda não existia na BC, a sua inserção foi bem-sucedida. Posteriormente, ao invocar o sistema de inferência (*si)* com esse mesmo termo, o resultado obtido é "verdadeiro" caso o conhecimento seja perfeito positivo, "falso" caso o conhecimento seja negativo e "desconhecido" caso seja imperfeito, o que vai de acordo com o pretendido. Por outro lado, foi efetuado também uma inserção de conhecimento negativo relativo ao centro de saúde e a invocação do sistema de inferência, de modo a verificar se o conhecimento foi de facto adicionado. Este procedimento repetiu-se para os diferentes tipos de conhecimento no ramo do imperfeito: incerto, interdito e impreciso. Na seguinte figura é possível observar os testes efetuados e o respetivo resultado:

```
|    evolucao(centro_saude(100,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),positivo).
true .

?- si(centro_saude(100,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),D).
D = verdadeiro .

?- evolucao(centro_saude(200,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),negativo).
true .

?- si(centro_saude(200,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),D).
D = falso .

?- evolucao(centro_saude(300,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, incerto),incerto,email).
true .

?- si(centro_saude(300,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),D).
D = desconhecido.

?- evolucao(centro_saude(400,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, interdito),interdito,email).
true .

?- si(centro_saude(400,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),D).
D = desconhecido.

?- evolucao(centro_saude(5,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'outro@gmail.com'),impreciso,email).
true .

?- si(centro_saude(5,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'outro@gmail.com'),D).
D = desconhecido.
```

**Figura 99:** Teste Evolução Centro de Saúde

O procedimento efetuado para o termo *utente* é semelhante ao efetuado para o *centro_saude*. De seguida podemos observar os testes efetuados para o *utente* e respetivos resultados:

```
?- evolucao(utente(100,1234,'Joana Sousa',data(1968,2,22), 'js@gmail.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),positivo).
true .

?- si(utente(100,1234,'Joana Sousa',data(1968,2,22), 'js@gmail.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),D).
D = verdadeiro .

?- evolucao(utente(200,1234,'Antonia',data(1968,2,22), 'js@gmail.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),negativo).
true .

?- si(utente(200,1234,'Antonia',data(1968,2,22), 'js@gmail.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),D).
D = falso .

?- evolucao(utente(300,1234,'Antonia',data(1968,2,22), incerto,216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),incerto, email).
true .

?- si(utente(300,1234,'Antonia',data(1968,2,22), 'email@email.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),D).
D = desconhecido.

?- evolucao(utente(400,1234,'Antonia',data(1968,2,22), interdito,216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),interdito, email).
true .

?- si(utente(400,1234,'Antonia',data(1968,2,22), 'email@gmail.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),D).
D = desconhecido.

?- evolucao(utente(19,1234,'Gabriela Costa Pinto',data(1991,2,5), 'PretaDoCabeloCacheado@armyspy.com',220154873, 'LISBOA', 'assistente de bordo',[gravidez], 2),impreciso, morada).
true .

?- si(utente(19,1234,'Gabriela Costa Pinto',data(1991,2,5), 'PretaDoCabeloCacheado@armyspy.com',220154873, 'LISBOA', 'assistente de bordo',[gravidez], 2),D)
|
D = desconhecido.
```

**Figura 100:** Teste Evolução Utente

O procedimento efetuado para o termo *staff* é análogo ao efetuado para o *centro_saude*. De seguida podemos observar os testes efetuados para o *staff* e respetivos resultados:

```
?- evolucao(staff(100,2,'Joana Sousa','js@gmail.com'),positivo).
true .

?- si(staff(100,2,'Joana Sousa','js@gmail.com'),D).
D = verdadeiro .

?- evolucao(staff(200,2,'Joana Sousa','js@gmail.com'),negativo).
true .

?- si(staff(200,2,'Joana Sousa','js@gmail.com'),D).
D = falso .

?- evolucao(staff(300,2,'Joana Sousa',incerto),incerto,email).
true .

?- si(staff(300,2,'Joana Sousa','js@gmail.com'),D).
D = desconhecido.

?- evolucao(staff(400,2,'Joana Sousa',interdito),interdito,email).
true .

?- si(staff(400,2,'Joana Sousa','js@gmail.com'),D).
D = desconhecido.

?- evolucao(staff(7,6, 'Luana Barros Oliveira','outro@gmail.com'),impreciso,email).
true .

?- si(staff(7,6, 'Luana Barros Oliveira','outro@gmail.com'),D).
D = desconhecido.
```

**Figura 101:** Teste Evolução Staff

O procedimento anteriormente referido foi também análogo para o *cancela_vacina*. De seguida podemos observar os testes efetuados para o *cancela_vacina* e respetivos resultados:

```
?- evolucao(cancela_vacina(21,data(2021,2,10)),positivo).
true .

?- si(cancela_vacina(21,data(2021,2,10)),D).
D = verdadeiro .

?- evolucao(cancela_vacina(2,data(2021,2,10)),negativo).
true .

?- si(cancela_vacina(2,data(2021,2,10)),D).
D = falso .
```

**Figura 102:** Teste Evolução Cancela Vacina

Por último para a *vacinacao_Covid* também foram efetuados alguns testes, de modo a testar o comportamento do programa. Os testes efetuados e respetivos resultados estão apresentados de seguida:

```
?- evolucao(vacinacao_Covid(1,2,data(2021,1,10),pfizer,1),positivo).
true .

?- si(vacinacao_Covid(1,2,data(2021,1,10),pfizer,1),D).
D = verdadeiro .

?- evolucao(vacinacao_Covid(1,4,data(2021,1,10),pfizer,1),negativo).
true .

?- si(vacinacao_Covid(1,4,data(2021,1,10),pfizer,1),D).
D = falso .
```

**Figura 103:** Teste Evolução Vacinação Covid

## INVOLUÇÃO

Para testar a involução procedeu-se de modo análogo, mas desta vez invocando a regra *involucao.* Na figura 104 é possível observar testes de remoção dos diferentes tipos conhecimento, tanto perfeito como imperfeito, da BC e os respetivos resultados. É possível observar que após a remoção de conhecimento, a invocação ao sistema de inferência fornece o resultado "falso", uma vez que como o conhecimento foi removido então o sistema de inferência como não encontra o termo devolve falso.

```
?- involucao(centro_saude(100,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),positivo).
true .

?- si(centro_saude(100,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),D).
D = falso .

?- involucao(centro_saude(200,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),negativo).
true .

?- si(centro_saude(200,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),D).
D = falso .

?- involucao(centro_saude(300,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, incerto),incerto,email).
true .

?- si(centro_saude(300,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'centro@gmail.com'),D).
D = falso .

?- involucao(centro_saude(400,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, interdito),interdito,email).
true .

?- si(centro_saude(400,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, interdito),D).
D = falso .

?- involucao(centro_saude(5,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'outro@gmail.com'),impreciso,email).
true .

?- si(centro_saude(5,centroSaude5, 'R Nossa Senhora Fátima 89 3330-107 RELVA DA MÓ', 273452452, 'outro@gmail.com'),D).
D = falso .

?-
```

**Figura 104:** Teste Involução Centro de Saúde

No caso do termo *utente* também foram efetuados alguns testes de modo a verificar e validar o comportamento na remoção de conhecimento. Após efetuar a remoção de um termo, quer seja de conhecimento perfeito ou imperfeito, é efetuada a chamada ao sistema de inferência e verifica-se se o resultado obtido é o pretendido. Os testes efetuados ao *utente* e respetivos resultados apresentam-se na figura abaixo.

```
?- involucao(utente(100,1234,'Joana Sousa',data(1968,2,22), 'js@gmail.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),positivo).
true .

?- si(utente(100,1234,'Joana Sousa',data(1968,2,22), 'js@gmail.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),D).
D = falso .

?- involucao(utente(200,1234,'Antonia',data(1968,2,22), 'js@gmail.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),negativo).
true .

?- si(utente(200,1234,'Antonia',data(1968,2,22), 'js@gmail.com',216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),D).
D = falso .

?- involucao(utente(300,1234,'Antonia',data(1968,2,22), incerto,216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),incerto, email).
true .

?- si(utente(300,1234,'Antonia',data(1968,2,22), incerto,216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),D).
D = falso .

?- involucao(utente(400,1234,'Antonia',data(1968,2,22), interdito,216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),interdito, email).
true .

?- si(utente(400,1234,'Antonia',data(1968,2,22), interdito,216950825, 'Rua Longuinha 65 2620-418 RAMADA', staff,[], 2),D).
D = falso .

?- involucao(utente(19,1234,'Gabriela Costa Pinto',data(1991,2,5), 'PretaDoCabeloCacheado@armyspy.com',220154873, 'LISBOA', 'assistente de bordo',[gravidez], 2),impreciso, morada).
true .

?- si(utente(19,1234,'Gabriela Costa Pinto',data(1991,2,5), 'PretaDoCabeloCacheado@armyspy.com',220154873, 'LISBOA', 'assistente de bordo',[gravidez], 2),D)
|
|
D = falso
```

**Figura 105:** Teste Involução Utente

No caso do termo *staff* também foram efetuados alguns testes de modo a verificar o comportamento do programa. Após efetuar a remoção de um termo, quer seja de conhecimento perfeito ou imperfeito, é efetuada a chamada ao sistema de inferência e verifica-se se o resultado obtido é o pretendido. Os testes efetuados ao *staff* e respetivos resultados apresentam-se na figura abaixo.

```
?- involucao(staff(100,2,'Joana Sousa','js@gmail.com'),positivo).
true .

?- si(staff(100,2,'Joana Sousa','js@gmail.com'),D).
D = falso .

?- involucao(staff(200,2,'Joana Sousa','js@gmail.com'),negativo).
true .

?- si(staff(200,2,'Joana Sousa','js@gmail.com'),D).
D = falso .

?- involucao(staff(300,2,'Joana Sousa',incerto),incerto,email).
true .

?- si(staff(300,2,'Joana Sousa',incerto),D).
D = falso .

?- involucao(staff(400,2,'Joana Sousa',interdito),interdito,email).
true .

?- si(staff(400,2,'Joana Sousa',interdito),D).
D = falso .

?- involucao(staff(7,6, 'Luana Barros Oliveira','outro@gmail.com'),impreciso,email).
true .

?- si(staff(7,6, 'Luana Barros Oliveira','outro@gmail.com'),D).
D = falso .
```

**Figura 106:** Teste Involução Staff

Para os termos *vacinacao_Covid* e *cancela_vacina* também foram efetuados testes. Estes testes foram análogos aos anteriormente referidos. Primeiro é efetuada a remoção dos termos da base de conhecimento, tendo em conta os conhecimentos implementados para cada termo, e de seguida efetua-se a chamada ao sistema de inferência. Nas figuras seguintes é possível observar os testes efetuados e os respetivos resultados, que comprovam o bom funcionamento do programa:

```
?- involucao(vacinacao_Covid(1,2,data(2021,1,10),pfizer,1),positivo).
true .

?- si(vacinacao_Covid(1,2,data(2021,1,10),pfizer,1),D).
D = falso .

?- involucao(vacinacao_Covid(1,4,data(2021,1,10),pfizer,1),negativo).
true .

?- si(vacinacao_Covid(1,4,data(2021,1,10),pfizer,1),D).
D = falso .
```

**Figura 107:** Teste Involução Vacinação Covid

```
?- involucao(cancela_vacina(21,data(2021,2,10)),positivo).
true .

?- si(cancela_vacina(2,data(2021,2,10)),D).
D = falso .

?- involucao(cancela_vacina(2,data(2021,2,10)),negativo).
true .

?- si(cancela_vacina(2,data(2021,2,10)),D).
D = falso .

?- si(cancela_vacina(21,data(2021,2,10)),D).
D = falso .
```

**Figura 108:** Teste Involução Cancela Vacina

# CONCLUSÃO E SUGESTÕES

Dada por concluída a 2ª fase do projeto, consideramos relevante efetuar uma análise crítica do trabalho realizado, apontando aspetos positivos e negativos, apresentando os resultados finais e dando ênfase a possíveis evoluções futuras do trabalho.

Notamos que existiram aspetos positivos a realçar, entre eles a boa organização e documentação do código entregue e a existência de um programa funcional e completo.

No que toca às dificuldades sentidas, consideramos que a maior foi a deteção de alguns erros semânticos e sintáticos no código que, por sua vez, levantaram incongruências difíceis de detetar devido à extensão do código implementado.

Consideramos ainda que o resultado final se revelou bastante satisfatório dado que alberga todas as funcionalidades propostas bem como retrata, claramente, todos tipos de conhecimento. Caso existisse uma fase futura, poderíamos talvez tentar implementar mais alguns extras de forma a ter um projeto ainda mais informado, completo e robusto.

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

# REFERÊNCIA

## REFERÊNCIAS BIBLIOGRÁFICAS

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

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

## REFERÊNCIAS ELETRÓNICAS

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

# LISTA DE SIGLAS E ACRÓNIMOS

**BC** Base de Conhecimento

**SI** Sistema de Inferência

**PMF** Pressuposto Mundo Fechado