# UNIVERSIDADE DO MINHO

# Programação em Lógica

Licenciatura em Engenharia Biomédica

Inteligência Artificial em engenharia Biomédica

1ºsemestre/2022-2023

A96353 Inês Margarida Mendes

A95637 Luís Manuel Gonçalves

A95524 Miguel Borges Vale

Braga, 31 de novembro de 2022

# Resumo

O estudo da Lógica contribui constantemente para o desenvolvimento sucessivo da Inteligência artificial e suas polivalências. Este projeto tem então como principal objetivo o desenvolvimento de conhecimentos nestas áreas, através da construção de diferentes predicados e estruturas, permitindo assim a manipulação e dados, fornecendo uma maior quantidade de respostas que sejam possíveis responder.

Assim para estes efeitos foi utilizado o programa Prolog. Prolog é uma linguagem de programação que segue o paradigma da programação lógica matemática. É uma linguagem polivalente que é particularmente importante para a inteligência artificial e linguística computacional. Este programa, diante uma base de conhecimento anteriormente instaurada, permite analisar os exames de um dado grupo de utentes. Para efeito foram construídos diversos predicados, invariantes e sistemas de inferência, que em conjunto definem regras para controlar a evolução do conhecimento.

# ÍNDICE



....

# 1. Introdução

No âmbito da disciplina de Inteligência Artificial em Engenharia Biomédica, foi-nos proposto desenvolver um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo na área da análise e exames de marcadores de saúde.

O nosso trabalho consistiu na criação de um conjunto de regras a partir de variáveis adequadas, sobre a base de conhecimento e os predicados implementados, os utentes, os atos e os marcadores, para produzir uma base de conhecimento acessível a qualquer indivíduo que se pretenda informar, adicionar, remover ou alterar a informação criada.

De modo a ter um sistema útil e funcional, este possui diversas funcionalidades entre as quais:

- Registar utentes, atos e marcadores
- Remover utentes, atos e marcadores
- Identificação da existência de utentes, atos e marcadores com características específicas
- Avaliação da normalidade de resultados de análises

# 2. Descrição do Trabalho e Analise

## 2.1. Declarações e definições iniciais

As declarações iniciais têm a função de designar um procedimento que deverá ser executado pelo programa para tal programa de modo a auxiliar o programador. Estas bandeiras ou flags são uma propriedade que permite ajudar na portabilidade, sendo estas apenas de leituras.

Assim neste caso foram definidas três, a primeira que quando encontra um procedimento que não definido, faz com que este falhe, a segunda que permite desativar mensagens de aviso, e a

terceira que desabilita as mensagens de aviso de quando uma variável que contenha "_" +e consultada ou compilada.

De modo a ser possível adicionar, remover ou verificar conhecimento da base de dados os predicados usados devem ser dinâmicos, isto é, temos de defini-los como tais através das definições iniciais, tendo em conta o número de argumentos em cada procedimento.

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


:- op( 900,xfy,'::' ).

:- dynamic '-'/1.
:- dynamic avaliar/3.
:- dynamic comparar/4.
:- dynamic siL/2.
:- dynamic utente/6.
:- dynamic evolucao/1.
:- dynamic ato/8.
:- dynamic excecao/1.
:- dynamic marcador/7.
:- dynamic pertence/2.
```

*Figura 1- Declarações e definições iniciais*

## 2.2. Meta predicados

### 2.2.1. Predicado Não

```
% Extensao do meta-predicado nao: Questao -> {V,F}

nao( Questao ) :-
    Questao, !, fail.
nao( Questao ).
```

*Figura 2- Extensão do meta-predicado não*

### 2.2.2. Sistema de inferência

O sistema de inferência desenvolvido foi dividido em 4 parte, *si, siL, siC e siD*, que podemos ver nas figuras, 3, 4, 5, 6 respetivamente. Dentro destes predicados o de maior importância é o *si,* uma vez que este servirá de base para os outros deste sistema.

Na sua base o predicado *si* recebe 2 variáveis, uma questão e uma resposta para essa mesma questão. Caso na nossa base de conhecimento tivermos um facto que comprove o que foi inferido na nossa variável, a resposta dada será verdadeira. Se tivermos um facto negativo que que negue a pergunta teremos então a resposta falso. Mas por fim se não tivermos um facto que negue ou comprove a pergunta teremos assim a resposta desconhecido.

```
% Extensao do meta-predicado si: Questao,Resposta -> {Verdadeiro,Falso,Desconhecido}

si( Questao,verdadeiro ) :-
    Questao.
si( Questao,falso ) :-
    -Questao.

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

*Figura 3- Extensão do meta-predicado si*

No predicado siL teremos agora uma lista com diferentes questões a fazer a este sistema e uma lista de respostas a essas mesmas questões. Como dito anteriormente este predicado tem como base o predicado si usando-o para obter as respostas ás perguntas uma por uma por uma certa ordem.

```
% Extensao do meta-predicado siL: [Questao],[Resposta] -> {Verdadeiro, Falso, Desconhecido}
siL([X|T1],[Y|T2]):-
    si(X,Y),
    siL(T1,T2).
siL([],[]).
```

*Figura 4- Extensão do meta-predicado siL*

Assim nos predicados siC e siD temos como como variáveis uma lista de perguntas, mas com apenas uma resposta. Esta resposta é obtida através do predicado si e é feita a conjunção e a disjunção respetivamente, das respostas que são recebidas ao longo da lista de perguntas.

```
% Extensao do meta-predicado siC: [Q | T], Resposta -> {Verdadeiro, Falso, Desconhecido}

siC([], verdadeiro).

siC([Q|T], verdadeiro) :-
    si(Q, verdadeiro),
    siC(T, verdadeiro).

siC([Q|T], desconhecido) :-
    si(Q, desconhecido),
    nao( siC(T, falso)).

siC([Q|T], desconhecido) :-
    nao( si(Q, falso)),
    siC(T, desconhecido).

siC([Q|T], falso) :-
    si(Q, falso).

siC([Q|T], falso) :-
    siC(T, falso).
```

*Figura 5- Extensão do meta-predicado siC*

```
% Extensao do meta-predicado siD: [Q | T], Resposta -> {verdadeiro, falso, desconhecido}

siD([], falso).

siD([Q|T], verdadeiro) :-
    si(Q, verdadeiro).

siD([Q|T], verdadeiro) :-
    siD(T, verdadeiro).

siD([Q|T], desconhecido) :-
    si(Q, desconhecido),
    nao( siD(T, verdadeiro)).

siD([Q|T], desconhecido) :-
    nao( si(Q, verdadeiro)),
    siD(T, desconhecido).

siD([Q|T], falso) :-
    si(Q, falso),
    siD(T, falso).
```

*Figura 6- Extensão do meta-predicado siD*

## 2.3. Formalização do Pressuposto Mundo Fechado

No pressuposto Mundo Fechado ou PMF faz com que toda a informação que foi mencionada na nossa base de conhecimento é tipo como falsa. Assim a sua formalização para os 3 principais predicados encontram-se na figura 7.

```
%Formalização do PMF _____________________________

-utente(I,N,D,M,A,G):-
    nao(utente(I,N,D,M,A,G)),
    nao(excecao(utente(I,N,D,M,A,G))).
-ato(Ida,D,Idu,I,C,Pu,Prd,Prs):-
    nao(ato(Ida,D,Idu,I,C,Pu,Prd,Prs)),
    nao(excecao(ato(Ida,D,Idu,I,C,Pu,Prd,Prs))).
-marcador(Idm,A,Imin,Imax,G,Vmin,Vmax):-
    nao(marcador(Idm,A,Imin,Imax,G,Vmin,Vmax)),
    nao(excecao(marcador(Idm,A,Imin,Imax,G,Vmin,Vmax))).
```

*Figura 7- Formalização do PMF*

## 2.4. Base de conhecimento

Para formar um sistema de representação de conhecimento com a capacidade de descrever uma área de prestação de cuidados de saúde, foi criada uma base de conhecimento, constituída por diversos factos a partir do uso dos predicados utente, ato e marcador.

O predicado *utente* contém quatro tipos de conhecimento, sendo eles o ID do Utente, o nome do utente, o dia, o mês e o ano de nascimento e o seu género, onde serão guardadas as informações do utente.

```
%Extensão do predicado utente: idUtente, Nome, Dia de nascimento, Mês de nascimento,
Ano de nascimento, Género -> {V,F,D}

utente(123 456 780,antonio,30,06,1990,masculino).
utente(987 654 321,beatriz,30,07,1985,feminino).
utente(135 246 978,carlos,30,08,1987,masculino).
utente(598 792 742,abel,29,02,1969,masculino).
utente(100 101 110,doofenshmirtz,04,08,1975,masculino).
utente(123 456 789,josefina,11,11,2002,feminino).
utente(valor_desconhecido01,mateus,17,10,2001,masculino).
utente(178 900 100,valor_desconhecido02,12,03,1995,feminino).
utente(145 457 910,carolina,06,05,2003,valor_interdito).
-utente(156 345 781,manuela,09,09,1999,masculino).
utente(123 678 902,ines,12,11,2004,feminino).
```

*Figura 8- Extensão do predicado utente*

O predicado *ato* contém 8 variáveis referentes ao ato ou ao utente, o id do ato, a data do ato, o id do utente, a sua idade, o seu colesterol, a sua pulsação, a pressão diastólica e a pressão sistólica, onde serão guardadas algumas informações do utente e os resultados dos testes.

```
%Extensão do predicado ato: IdAto, Data do ato, IdUtente, Idade, Colesterol,
Pulsação, Pressão diastólica, Pressão sistólica -> {V,F,D}


ato(gmr5148,16/05/2019,100 101 110,47,140,70,65,133).
ato(gmr02067,30/06/2020,123 456 780,32,140,70,60,123).
ato(gmr2183,30/07/2021,987 654 321,37,190,60,70,142).
ato(gmr2297,30/06/2022,123 456 780,35,230,90,65,151).
ato(gmr3457,17/11/2022,123 456 789,20,150,70,70,111).
ato(gmr4200,14/12/2020,598 792 742,53,250,120,50,90).
ato(gmr6100,18/02/2021,987 654 321,36,valor_interdito,123,66,145).
-ato(gmr7207,09/12/2020,850 759 592,86,157,99,78,154).
ato(valor_desconhecido03,11/11/2022,123 678 902,18,135,80,68,134).
```

*Figura 9- Extensão do predicado ato*

O predicado *marcador* contém 7 variáveis referentes ao Id do marcador, o tipo de teste que foi realizado, as idades a que os valores comuns correspondem, o género do utente e os valores comuns do teste, onde serão guardadas informações necessárias à conclusão retirada para os utentes.

Neste predicado nos casos de idades onde só se encontram adaptadas com idades mínimas, isto é, “A partir dos 65 anos” por exemplo, ao ser adaptado ao nosso problema tivemos de definir

uma idade máxima sendo esta definida como 150 anos uma idade inatingível atualmente para um ser humano.

Como poderemos ver futuramente não haverá nenhum conhecimento imperfeito, uma vez que como grupo concordamos que não faria muito sentido ter este tipo de conhecimento numa base de dados clínica sobre valores ideais.

```
%Extensão do predicado marcador: Idmarcador, Analise, Idade Mínima,
Idade Máxima, Género, Valor Normal Mínimo, Valor Nomral Máximo -> {V,F,D}


marcador(ctm01,colesterol,18,30,masculino,0,170).
marcador(ctf02,colesterol,18,30,feminino,0,160).
marcador(ctm03,colesterol,31,45,masculino,0,190).
marcador(ctf04,colesterol,31,45,feminino,0,180).
marcador(ctm05,colesterol,46,150,masculino,0,200).
marcador(ctf06,colesterol,46,150,feminino,0,195).
marcador(psm07,pulsacao,18,25,masculino,56,73).
marcador(psf08,pulsacao,18,25,feminino,61,78).
marcador(psm09,pulsacao,26,35,masculino,55,74).
marcador(psf10,pulsacao,26,35,feminino,60,76).
marcador(psm11,pulsacao,36,45,masculino,57,75).
marcador(psf12,pulsacao,36,45,feminino,60,78).
marcador(psm13,pulsacao,46,55,masculino,58,76).
marcador(psf14,pulsacao,46,55,feminino,61,77).
marcador(psm15,pulsacao,56,65,masculino,57,75).
marcador(psf16,pulsacao,56,65,feminino,60,77).
marcador(psm17,pulsacao,65,150,masculino,56,73).
marcador(psf18,pulsacao,65,150,feminino,60,76).
marcador(pam19,pressaoarterial,18,39,masculino,70,119).
marcador(paf20,pressaoarterial,18,39,feminino,68,110).
marcador(pam21,pressaoarterial,40,59,masculino,77,124).
marcador(paf22,pressaoarterial,40,59,feminino,74,122).
marcador(pam23,pressaoarterial,60,150,masculino,69,133).
marcador(paf24,pressaoarterial,60,150,feminino,69,139).
```

*Figura 10- Extensão do predicado marcador*

As seguintes exceções compreendem o conjunto de situações em que poderemos encontrar informação incompleta na base de conhecimento, tal como conhecimento incerto, impreciso ou interdito, de modo a aumentar as possibilidades de informação na base de conhecimento.

```
%Excecoes-------------------------------------------------------------------------

excecao(utente(I, N, D, M, A, G)) :- utente(valor_desconhecido01, N, D, M, A, G).

excecao(utente(I,N,D,M,A,G)) :- utente(I,valor_desconhecido02,D,M,A,G).

excecao(utente(167 888 234,jose,23,04,1993,masculino)).

excecao(utente(167 888 234,carlos,23,04,1993,masculino)).

excecao(utente(I,N,D,M,A,G)):- utente(I,N,D,M,A,valor_interdito).

excecao(ato(Id_ato,Data_ato,Id_utente,Idade,C,P,Prd,Prs)):-
    ato(Id_ato,Data_ato,Id_utente,Idade,C,P,Prd,Prs).

excecao(ato(gmr6789, 15/12/2022, 123 456 789, 20, 70, 60, 70, 142)).

excecao(ato(gmr6789, 15/12/2022,123 456 789,20,100,60,70,142)).

excecao(ato(Id_ato,Data_ato,Id_utente,Idade,C,P,Prd,Prs)):-
    ato(Id_ato,Data_ato,Id_utente,Idade,valor_interdito,P,Prd,Prs).
```

*Figura 11- Exceções relativas ao predicado utente e ato*

## 2.5. Predicados para a análise de resultados

### Predicado *avaliar*

O seguinte predicado avalia a normalidade dos valores das análises feitas por um certo utente, numa data específica, para um dos três testes possíveis, e para isso recebe como argumentos o IdUtente, a Data do ato, e a Análise que se pretende avaliar. Inicialmente é verificado se existe realmente um *utente* com o Id pretendido, um *ato* com a data e Idutente pretendidos e *marcadores* adequados. De seguida, verifica-se de a idade do utente está compreendida entre a idade mínima e a idade máxima descritas no marcador, e se isto se verificar, prossegue-se para o predicado *comparar*.

```
%Extensão do predicado avaliar: IdUtente, DataAto, Análise -> {V,F,D}

avaliar(Idutente,Data,Analise):-
    utente(Idutente,N,D,M,A,G),
    ato(T,Data,Idutente,I,C,P,Prd,Prs),
    marcador(Id,Analise,Imin,Imax,G,Vmin,Vmax),
    Imin=<I,
    I=<Imax,
    comparar(Id,Analise,Idutente,T).
```

*Figura 12- Extensão do predicado avaliar*

Por exemplo, se o utilizador pretender avaliar a normalidade do valor do Colesterol do utente com Id igual a “123 456 780”, no ato realizado na data “30/06/2022”, o processo será visualizado deste modo:

```
?- avaliar(123 456 780,30/06/2022,colesterol).
false.
```

*Figura 13– Teste da função avaliar*

E se, por exemplo, se o utilizador pretender avaliar a normalidade do valor do Colesterol do utente com Id igual a “123 456 789”, no ato realizado na data “17/11/2022”, o processo será visualizado deste modo:

```
?- avaliar(123 456 789,17/11/2022,colesterol).
true .
```

*Figura 14– Teste da função avaliar*

## Predicado *comparar*

O predicado auxiliar comparar tem como função comparar o valor obtido de uma determinada análise com os valores considerados normais. Para isso, temos três predicados diferentes, cada um referente a uma análise diferente.

Para o caso da pulsação, é verificado se o valor obtido no ato (*P*) está compreendido entre os valor mínimo (*Vmin*) e máximo (*Vmax*) de pulsação para as características do utente em questão.

Para o caso do colesterol, o processo é muito similar ao da pulsação.

Já para o caso da pressão arterial o processo é ligeiramente diferente, pois os valores normais apresentados no marcador não se tratam de valores mínimos e máximos, mas sim de valores “padrão”. Então, para testar a normalidade dos valores de pressão arterial distólica e sistólica do utente, verifica-se se os valores obtidos no ato se encontram dentro de uma margem de 5%, superiormente e inferiormente, em relação ao valor padrão respetivo.

```
%Extensão do predicado comparar: IdMarcador, Análise, IdUtente, IdAto -> {V,F,D}

comparar(Id,pulsacao,Idutente,T):-
    ato(T,Data,Idutente,I,C,P,Prd,Prs),
    marcador(Id,pulsacao,Imin,Imax,G,Vmin,Vmax),
    Vmin=<P,
    P=<Vmax.

comparar(Id,colesterol,Idutente,T):-
    ato(T,Data,Idutente,I,C,P,Prd,Prs),
    marcador(Id,colesterol,Imin,Imax,G,Vmin,Vmax),
    Vmin=<C,
    C=<Vmax.

comparar(Id,pressaoarterial,Idutente,T):-
    ato(T,Data,Idutente,I,C,P,Prd,Prs),
    marcador(Id,pressaoarterial,Imin,Imax,G,Vmin,Vmax),
    Vmin*0.95=<Prd,
    Prd=<Vmin*1.05,
    Vmax*0.95=<Prs,
    Prs=<Vmax*1.05.
```

*Figura 15- Extensão do predicado comparar*

## 2.6. Auxiliares de evolução

### Predicado *comprimento*

O predicado *comprimento* tem como função determinar o tamanho de uma lista, ou seja, o número de elementos que a compõem. Para isso recebe o argumento S, sendo ele a lista, e N que é o resultado. Para a determinação de N é usada a operação *length*, já integrada no Prolog.

```
%Extensão do predicado comprimento: Lista, Comprimento -> {V,F}

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

*Figura 16- Extensão do predicado comprimento*

### Predicado *insercao*

O predicado *insercao* tem como função inserir o Termo que lhe é dado como argumento na base de conhecimento. Para isso utiliza a operação *assert*, e no caso de ocorrência de falha, o Termo que outrora fora adicionado, é agora retirado através da operação *retract*.

```
%Extensão do predicado inserção: Termo -> {V,F}

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

*Figura 17 Extensão do predicado inserção*

### Predicado *remocao*

Este predicado permite a remoção do Termo que lhe é dado como argumento da base de conhecimento através da operação *retract*.

```
%Extensão do predicado remoção: Termo -> {V,F}

remocao(Termo):- retract(Termo).
```

*Figura 18- Extensão do predicado remoção*

### Predicado *teste*

O predicado *teste* recebe como argumento uma lista e vai "testando" cada um dos elementos de forma recursiva. Isto é, se R for *true*, é efetuado o predicado *teste* da cauda LR da lista. Sendo que, neste trabalho, este predicado serve para testar se são cumpridos os invariantes.

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

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

*Figura 19- Extensão do predicado teste*

## Predicado *removeTermos*

O predicado *removeTermos* tem como função remover da base de conhecimento uma lista de Termos, o que é efetuado de forma recursiva.

```
%Extensão do predicado removeTermos: ListaTermos -> {V,F}

removeTermos( [] ).
removeTermos( [X] ) :-
    retract(X).
removeTermos( [X|L] ) :-
    retract(X),
    removeTermos( L ).
```

*Figura 20- Extensão do predicado removeTermos*

## Predicado *interdito*

Este predicado dá a o significado de interdito ao termo geral *valor_interdito.*

```
%Extensão do predicado interdito: Valor Interdito -> {V,F}

interdito(valor_interdito).
```

*Figura 21- Extensão do predicado interdito*

## Predicado *pertence*

Como vemos na figura 20, o predicado *pertence*, recebe 2 variáveis e procura a primeira dentro de uma lista, isto é, a segunda variável, retornado verdadeiro caso pertença e falso caso contrário.

```
%Extensão do predicado pertence: Valor, Lista -> {V,F}

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

*Figura 22- Extensão do predicado pertence*

## Predicado *is_List*

Na figura 21 temos o predicado *is_list* que vê se a variável que recebe é ou não uma lista retornando um valor de verdade quando esta for.

```
%Extensão do predicado is_list: Variável -> {V,F}

is_list(X) :-
        var(X), !,
        fail.
is_list([]).
is_list([_|T]) :-
        is_list(T).
```

*Figura 23- Extensão do predicado is_list*

## 2.7. Evolução

### 2.7.1 Evolução do predicado utente

#### Predicado *evolucaoU*

O predicado *evolucaoU* tem a função de fazer a evolução do conhecimento a nível dos utentes, na medida em que permite a inserção de novos utentes na base de conhecimento e fazer a transição entre diferentes tipos de conhecimento (imperfeito e perfeito). No nosso trabalho temos três predicados diferentes que "testam" o argumento que é inserido. Os primeiros dois têm com principal função a identificação de listas no Termo inserido como argumento, devido ao facto de na nossa base de conhecimento termos conhecimento incerto e pode ser pretendido fazer a transição entre conhecimento incerto e impreciso.

No primeiro verifica-se se o Termo que se pretende inserir na base de conhecimento se trata de uma lista. Se assim for, prossegue para o predicado *verificaEvolucaoImpU*.

Caso o argumento não se trate de uma lista, prossegue-se para o segundo predicado onde se verifica se o elemento I é uma lista. Se assim for, procede-se para o predicado *verificaEvolucaoImpU*.

Caso o elemento I não se trate de uma lista, segue-se então o terceiro predicado *evolucaoU*. Neste, o predicado *findall* identifica os invariantes associados ao Termo a inserir e cria uma lista com esses invariantes. De seguida, procede-se à inserção do Termo na base de conhecimento através do predicado *insercao*. Depois disto, prossegue-se para o predicado *verificaEvolucaoU* e no fim, a lista de invariantes é verificada através do predicado *teste*, de modo a ver se todos os invariantes foram cumpridos.

```
%Extensão do predicado evolucaoU: Termo -> {V,F}

evolucaoU(Termo):-
    is_list(Termo),
    verificaEvolucaoImpU(Termo).

evolucaoU(utente(I,N,D,M,A,G)):-
    is_list(I),
    verificaEvolucaoImpU(utente(I,N,D,M,A,G)).

evolucaoU(Termo):-
    findall(Invariante,+ Termo :: Invariante, Lista),
    insercao(Termo),
    verificaEvolucaoU(Termo),
    teste(Lista).
```

*Figura 24- Extensão do predicado evoluçãoU*

Por exemplo, se o utilizador quiser introduzir na base de conhecimento um novo utente com Id igual a "156 789 123", nome igual a "gabriela", dia, mês e ano de nascimento respetivamente "23", "02", "1987", e género "feminino" o processo será visualizado deste modo:

```
?- evolucaoU(utente(156 789 123,gabriela,23,02,1987,feminino)).
true .
```

*Figura 25- Teste de introdução de conhecimento*

E se o utilizador quiser fazer a evolução de conhecimento incerto para conhecimento perfeito, por exemplo, do utente com Id igual a "178 900 100", identificando o seu nome como "rute", será visualizado deste modo:

```
?- evolucaoU(utente(178 900 100,rute,12,03,1995,feminino)).
true .
```

*Figura 26- Teste de evolução de conhecimento Incerto para Impreciso*

## Predicado *verificaEvolucaoU*

Existem quatro predicados diferentes para o *verificaEvolucaoU*, sendo que estes foram criados de modo a "identificar" o tipo de evolução de conhecimento que se pretende efetuar. Todos eles recebem como argumento o Termo que se pretende inserir na base de conhecimento.

No primeiro predicado, inicialmente efetua-se o *findall* de modo a criar uma lista de todos os Ids de utente que tenham as características de Nome, Dia, Mês, Ano e Género do *Termo* que se pretende inserir na base de conhecimento. De seguida, usa-se o predicado *comprimento* para descobrir o tamanho da lista e verifica-se se este é igual a 1. Este *findall* tem como objetivo, resumidamente, identificar se se trata de um tipo de evolução de conhecimento incerto -> perfeito.

Se o tamanho desta lista for também 1, segue-se o *findall* que vai encontrar todas as exceções associadas ao Nome, que tenham o Id, Dia, Mês, Ano e Género do Termo que se pretende inserir na base de conhecimento e coloca-las numa lista. De seguida, usa-se novamente o

predicado *comprimento* para descobrir o tamanho da lista e verifica-se se este é 0. O objetivo deste *findall* é essencialmente verificar se se trata de uma evolução de conhecimento impreciso -> perfeito.

```
%Extensão do predicado verificaEvolucaoU: utente(IdUtente,Nome,Dia,Mês,Ano,Género) -> {V,F}

verificaEvolucaoU( utente(I,N,D,M,A,G) ) :-
    findall(B,utente(B,N,D,M,A,G),F),
    comprimento(F,R),
    R==1,
    findall(excecao(utente(I,No,D,M,A,G)),excecao(utente(I,Nome,D,M,A,G)),S),
    comprimento(S,K),
    K=0.
```

*Figura 27- Extensão do predicado verifivaEvolucaoU, Parte 1*

Se houver falha no primeiro predicado, passa-se para o segundo, em que se vai fazer a verificação da evolução de conhecimento impreciso -> perfeito. Primeiro, usa-se o *findall* para criar uma lista com todos os Nomes presentes nas exceções associadas às características do Termo que se pretende inserir, sendo que, de seguida, usa-se o predicado *pertence* para verificar se o Nome presente no Termo está nessa lista. Se assim for, efetua-se o *findall* de todas as exceções associadas às características do Termo e coloca-se numa lista. Por fim, usa-se o predicado *removeTermos* que recebe como argumento a lista criada, e efetua a remoção das exceções da base de conhecimento.

```
verificaEvolucaoU(utente(I,N,D,M,A,G)) :-
    findall(Nome,excecao(utente(I,Nome,D,M,A,G)),S),
    pertence(N,S),
    findall(excecao(utente(I,No,D,M,A,G)),excecao(utente(I,Nome,D,M,A,G)),W),
    removeTermos(W).
```

*Figura 28- Extensão do predicado verifivaEvolucaoU, Parte 2*

Na ocorrência de falha no segundo predicado, segue-se o terceiro, onde primeiramente se usa o *findall* para criar uma lista com todos os Ids de utentes que tenham as características Nome, Dia, Mês, Ano e Género do Termo. De seguida, verifica-se se o comprimento da lista é igual a 2. Se assim for, procede-se à remoção do *utente(valor_desconhecido01,N,D,M,A,G)*.

```
verificaEvolucaoU( utente(I,N,D,M,A,G) ) :-
    findall(B,utente(B,N,D,M,A,G),S2),
    comprimento(S2,Size),
    Size==2,
    removeTermos([utente(valor_desconhecido01,N,D,M,A,G)]).
```

*Figura 29- Extensão do predicado verifivaEvolucaoU, Parte 3*

Se o terceiro predicado falhar, passa-se para o quarto, onde o processo é similar ao do terceiro, com a diferença de que em vez de se criar uma lista de Ids, cria-se uma lista de Nomes, e no fim, faz-se a remoção do *utente(I,valor_desconhecido02,D,M,A,G)*.

```
verificaEvolucaoU(utente(I,N,D,M,A,G)) :-
    findall(X,utente(I,X,D,M,A,G),S),
    comprimento(S,Size2),
    Size2==2,
    removeTermos([utente(I,valor_desconhecido02,D,M,A,G)]).
```

*Figura 30- Extensão do predicado verifivaEvolucaoU, Parte 4*

## Predicado *verificaEvolucaoImpU*

O predicado *verificaEvolucaoImpU*, figura 28, pode ser dividido em 2 grandes casos de evolução, um onde teremos uma lista que delimita os valores para os quais teremos um valor desconhecido quando inferirmos sobre ele, e outro em que o utilizador irá dar uma lista com predicados específicos que teremos valor desconhecido.

Assim para este primeiro grande caso ainda temos mais 3 sub casos.

O primeiro em que teríamos apenas um valor máximo e onde o primeiro termo da lista deve ser substituído por um sinal - (sinal de menos ou de negação). Neste primeiro caso encontrou-se a necessidade de mudar a localização do teste de invariantes e a remoção de factos já atualizados para este predicado em vês do *evolucaoU*.

Para o segundo teremos apena um valor limita por um mínimo, sendo representado a segunda variável com o sinal +. Analogamente ao caso anterior tivemos de mudar a posição do teste dos invariantes e da remoção de termos.

Para este último subcaso teremos realmente uma lista com dois valores um mínimo, o primeiro valor da lista e um máximo, o segundo valor da lista. Mais uma vez analogamente aos casos anteriores foi necessário trocar a posição do predicado *teste* e *removeTermos*.

Por fim, temos uma lista com diferentes predicados onde serão inseridos na base de conhecimento como exceção. Neste caso mantemos a remoção do termo desatualizado, mas continuando a mudar a posição do predicado teste.

```
%Extensão do predicado verificaEvolucaoImpU: Termo -> {V,F}

verificaEvolucaoImpU(utente([H,T],N,D,M,A,G)):-
    H =="-",
    findall(Invariante,+ utente(T,N,D,M,A,G) :: Invariante, Lista),
    findall(B,utente(B,N,D,M,A,G),S2),
    comprimento(S2,Size),
    Size==1,
    insercao(excecao(utente(X,N,D,M,A,G)):-X=<T),
    removeTermos([utente(valor_desconhecido01,N,D,M,A,G)]),
    teste(Lista),
    findall(Invariante,+ excecao(utente(T,N,D,M,A,G)) :: Invariante, Lista2),
    teste(Lista2).

verificaEvolucaoImpU(utente([H,T],N,D,M,A,G)):-
    T=="+",
    findall(Invariante,+ utente(H,N,D,M,A,G) :: Invariante, Lista),
    findall(B,utente(B,N,D,M,A,G),S2),
    comprimento(S2,Size),
    Size==1,
    removeTermos([utente(valor_desconhecido01,N,D,M,A,G)]),
    insercao(excecao(utente(X,N,D,M,A,G)):-X>=H),
    teste[Lista],
    findall(Invariante,+ excecao(utente(T,N,D,M,A,G)) :: Invariante, Lista2),
    teste(Lista2).

verificaEvolucaoImpU(utente([H,T],N,D,M,A,G)):-
    findall(Invariante,+ utente(H,N,D,M,A,G) :: Invariante, Lista),
    findall(Invariante,+ utente(T,N,D,M,A,G) :: Invariante, Lista2),
    verificaEvolucaoU(utente(H,N,D,M,A,G)),
    insercao(excecao(utente(X,N,D,M,A,G)):-(X>=H, X=<T)),
    findall(B,utente(B,N,D,M,A,G),S2),
    comprimento(S2,Size),
    Size==1,
    removeTermos([utente(valor_desconhecido01,N,D,M,A,G)]),
    teste(Lista),
    teste(Lista2),
    findall(Invariante,+ excecao(utente(H,N,D,M,A,G)) :: Invariante, Lista3),
    teste(Lista3).

verificaEvolucaoImpU([H|T]):-
    findall(Invariante,+ H :: Invariante, Lista),
    verificaEvolucaoU(H),
    insercao(excecao(H)),
    verificaEvolucaoImpU(T),
    teste(Lista),
    findall(Invariante,+ excecao(utente(H,N,D,M,A,G)) :: Invariante, Lista3),
    teste(Lista3).

verificaEvolucaoImpU([]).
```

*Figura 31- Extensão do predicado verificaEvolucaoImpU*

### 2.7.2. Evolução do predicado Ato

#### Predicado *evolucaoA*

O predicado *evolucaoA* tem a função de fazer a evolução do conhecimento a nível dos atos, na medida em que permite a inserção de novos atos na base de conhecimento e fazer a transição entre diferentes tipos de conhecimento (imperfeito e perfeito). Existem dois predicados diferentes de *evolucaoA* que recebem como argumento o Termo que se pretende inserir.

No primeiro, verifica-se se o Termo se trata de uma lista e, se isto se verificar, prossegue-se para o predicado *verificaEvolucaoImpA*. Por fim, retira-se o *ato(valor_desconhecido03,D,Idu,I,C,Pu,Prd,Prs)* da base de conhecimento. A retirada deste ato não foi efetuada no predicado *verificaEvolucaoImpA* pois a recursividade deste não o tornava possível.

Caso o Termo que se pretende inserir não se trate de uma lista, segue-se então o segundo predicado. Este funciona muito similarmente ao terceiro predicado *evolucaoU*, em que se cria inicialmente uma lista com todos os invariantes associados ao termo que se pretende inserir, sendo depois o termo inserido na base de conhecimento através do predicado *insercao*. Depois prossegue-se para o predicado *verificaEvolucaoA* e no fim, a lista de invariantes é verificada através do predicado teste.

```
%Extensão do predicado evolucaoA: Termo -> {V,F}

evolucaoA(Termo):-
    is_list(Termo),
    verificaEvolucaoImpA(Termo),
    removeTermos([ato(valor_desconhecido03,D,Idu,I,C,Pu,Prd,Prs)]).

evolucaoA(Termo):-
    findall(Invariante,+Termo :: Invariante,Lista),
    insercao(Termo),
    verificaEvolucaoA(Termo),
    teste(Lista).
```

*Figura 32- Extensão do predicado evoluçãoA*

Por exemplo, se o utilizador quiser introduzir na base de conhecimento um novo ato com Id igual a "gmr2030", data igual "27/09/2022", IdUtente igual a "123 678 902", idade igual a "18", colesterol igual a "133", pulsação igual a "87", pressão diastólica igual a "65" e pressão sistólica igual a "130", o processo será visualizado deste modo:

```
?- evolucaoA(ato(gmr2030,27/09/2022,123 678 902,18,133,87,65,130)).
true .
```

*Figura 33- Teste á introdução de conhecimento pelo predicado ato*

E se o utilizador quiser fazer a evolução de conhecimento impreciso para conhecido perfeito, por exemplo, do ato com Id "gmr6789", identificando o colesterol como "70", será visualizado deste modo:

```
?- evolucaoA(ato(gmr6789, 15/12/2022, 123 456 789, 20, 70, 60, 70, 142)).
true .
```

Figura 34-Teste á evolução de conhecimento

Porém, se invés de identificar o colesterol como "70" o utilizador quisesse identificar o colesterol como "80", o processo ia ser visualizado do seguinte modo:

```
?- evolucaoA(ato(gmr6789, 15/12/2022, 123 456 789, 20, 80, 60, 70, 142)).
false.
```

Figura 35- Teste á evolução de conhecimento

## Predicado *verificaEvolucaoA*

Existem três predicados diferentes *verificaEvolucaoA*, criados basicamente com o mesmo objetivo do *verificaEvolucaoU*. Todos eles recebem como argumento o *Termo* que se pretende inserir na base de conhecimento.

- No primeiro predicado, usa-se o *findall* para criar uma lista com todos os Ids de atos que tenham as características Data, IdUtente, Idade, Colesterol, Pulsação, Pressão Diastólica e Pressão Sistólica do Termo que se pretende inserir na base de conhecimento. Depois, usa-se o predicado comprimento para descobrir o tamanho da lista, verificando-se de seguida se esse tamanho é igual a 1. Tal como no *verificaEvolucaoU*, esta parte serve para identificar se se trata de uma evolução de conhecimento incerto -> perfeito.

Se o tamanho da lista for de facto 1, procede-se para o *findall* de todas as exceções associadas ao Colesterol que tenham as restantes características do Termo, e de seguida, verifica-se se o tamanho dessa lista é zero. Mais uma vez, similarmente ao *verificaEvolucaoU*, isto serve para identificar se se trata de uma evolução de conhecimento impreciso -> perfeito.

```
%Extensão do predicado verificaEvolucaoA: ato(IdAto,Data,IdUtente,Idade,Colesterol,Pulsação,Pressão Diastólica,Pressão Sistólica) -> {V,F}

verificaEvolucaoA(ato(Ida,D,Idu,I,C,Pu,Prd,Prs)):-
    findall(U,ato(U,D,Idu,I,C,Pu,Prd,Prs),J),
    comprimento(J,X),
    X==1,
    findall(excecao(ato(U,D,Idu,I,Co,Pu,Prd,Prs)),excecao(ato(U,D,Idu,I,Colesterol,Pu,Prd,Prs)),S),
    comprimento(S,K),
    K=0.
```

Figura 36- Extensão do Predicado verificaEvolucaoA Parte1

Na ocorrência de falha do primeiro predicado, segue-se o segundo, onde se usa o *findall* para criar uma lista com todos os valores de Colesterol presentes nas exceções associadas às características do Termo que se pretende inserir, sendo que, de seguida, usa-se o predicado *pertence* para verificar se o valor de Colesterol presente no Termo está nessa lista. Se assim for, efetua-se o *findall* de todas as exceções associadas às características do Termo e coloca-se numa lista. Por fim, usa-se o predicado *removeTermos* que recebe como argumento a lista criada, e efetua a remoção das exceções da base de conhecimento.

```
verificaEvolucaoA(ato(Ida,D,Idu,I,C,Pu,Prd,Prs)):-
    findall( B,excecao(ato(Ida,D,Idu,I,B,Pu,Prd,Prs)),S),
    pertence(C,S),
    findall(excecao(ato(Ida,D,Idu,I,B,Pu,Prd,Prs)),excecao(ato(Ida,D,Idu,I,B,Pu,Prd,Prs)),S2),
    comprimento(S2,Y),
    Y>0,
    removeTermos(S2).
```

Figura 37- Extensão do Predicado verificaEvolucaoA Parte2

Se houver falha no segundo predicado, passa-se para o terceiro, onde mais uma vez se usa o predicado *findall*. Este vai criar uma lista com todos os Ids dos atos com as características Data, IdUtente, Idade, Colesterol, Pulsação, Pressão Diastólica, Pressão Sistólica do Termo que se pretende inserir na base de conhecimento. De seguida, verifica-se se o comprimento da lista é igual a 2 e procede-se à remoção do *ato(valor_desconhecido03,D,Idu,I,C,Pu,Prd,Prs)*.

```
verificaEvolucaoA(ato(Ida,D,Idu,I,C,Pu,Prd,Prs)):-
    findall(X,ato(X,D,Idu,I,C,Pu,Prd,Prs),S),
    comprimento(S,Size2),
    Size2==2,
    removeTermos([ato(valor_desconhecido03,D,Idu,I,C,Pu,Prd,Prs)]).
```

Figura 38- Extensão do Predicado verificaEvolucaoA Parte1

### Predicado *verificaEvolucaoImpA*

No predicado *verificaEvolucaoImpA*, figura 36, analogamente ao último caso do *verificaEvolucaoImpU* são inseridos uma lista de termos onde serão inseridos na base de conhecimento como exceção, mudando a posição do predicado *teste*.

```
%Extensão do predicado verificaEvolucaoImpA: Termo -> {V,F}

verificaEvolucaoImpA([H|T]):-
    insercao(excecao(H)),
    findall(Invariante, + excecao(H) :: Invariante, Lista),
    teste(Lista),
    verificaEvolucaoImpA(T).

verificaEvolucaoImpA([]).
```

*Figura 39- Extensão do predicado verificaEvolucaoImpA*

## 2.7.3. Evolução do predicado Marcador

### Predicado *evolucaoM*

O predicado *evolucaoM* tem a função de fazer a evolução do conhecimento a nível dos marcadores, na medida em que permite a inserção de novos marcadores na base de conhecimento.

Recebendo como argumento o *Termo* que se pretende inserir na base de conhecimento, inicialmente, o predicado *findall* identifica os invariantes associados ao termo inserido e cria uma lista com esses invariantes. De seguida, procede-se à inserção do termo na base de conhecimento. Depois disto, prossegue-se para o predicado *verificaEvolucaoM* e no fim, a lista de invariantes é verificada através do predicado *teste*.

```
%Extensão do predicado evolucaoM: Termo -> {V,F}

evolucaoM(Termo):-
    findall(Invariante,+Termo :: Invariante,Lista),
    insercao(Termo),
    verificaEvolucaoM(Termo),
    teste(Lista).
```

*Figura 40- Extensão do predicado evoluçãoM*

Por exemplo, se o utilizador quiser introduzir na base de conhecimento um novo marcador com Id igual a “ctm25”, análise igual a “colesterol”, idade mínima igual a “10”, idade máxima igual a “15”, género igual a “masculino”, valor normal mínimo igual a “0” e valor normal máximo igual a “150”, o processo será visualizado do seguinte modo:

```
?- evolucaoM(marcador(ctm25,colesterol,10,15,masculino,0,150)).
true .
```

### Predicado *verificaEvolucaoM*

O predicado *verificaEvolucaoM* recebe como argumento o Termo que se pretende inserir na base de conhecimento. Inicialmente, é usado o *findall* para criar uma lista com todos os *Ids* dos marcadores com as características Análise, Idade Mínima, Idade Máxima, Género, Valor Normal Mínimo e Valor Normal Máximo do Termo e, de seguida, verifica-se se o tamanho dessa lista é igual a 0. Este predicado serve essencialmente para verificar que se insere um marcador que já exista, mas com um Id diferente.

```
%Extensão do predicado verificaEvolucaoM :
%marcador(IdMarcador,Analise,IdadeMinima, IdadeMaxima, Genero, ValorMinimo, ValorMaximo) -> {V,F}

verificaEvolucaoM( marcador(Idm,A,Imin,Imax,G,Vmin,Vmax)) :-
    findall( B,marcador(B,A,Imin,Imax,G,Vmin,Vmax),S),
    comprimento( S,X ),
    X == 1.
```

*Figura 41-Extensão do predicado verificaEvoluçãoM*

## 2.8. Involução de conhecimento

### Predicado *involucaoU*

O predicado *involucaoU* tem a função de efetuar a retirada de um determinado utente da base de conhecimento e recebe como argumento o *Termo* a retirar.

Primeiro, o predicado *findall* é usado para identificar todos os invariantes associados ao Termo a retirar e coloca-os numa lista. De seguida, faz-se a remoção do Termo e usa-se o predicado *teste*, que recebe como argumento a lista criada anteriormente, para testar se todos os invariantes foram cumpridos.

Os predicados *involucaoA* e *involucaoM* funcionam exatamente da mesma maneira.

```
%Extensão do predicado involucaoU: Termo -> {V,F}

involucaoU(Termo):-
    findall(Invariante, -Termo:: Invariante,Lista),
    remocao(Termo),
    teste(Lista).
```

*Figura 42- Extensão do predicado involuçãoU*

```
%Extensão do predicado involucaoA: Termo ->{V,F}

involucaoA(Termo):-
    findall(Invariante, - Termo::Invariante,Lista),
    remocao(Termo),
    teste(Lista).
```

*Figura 43- Extensão do predicado involuçãoA*

```
%Extensão do predicado involucaoM: Termo -> {V,F}

involucaoM(Termo):-
    findall(Invariante, - Termo::Invariante,Lista),
    remocao(Termo),
    teste(Lista).
```

*Figura 44- Extensão do predicado involuçãoM*

Por exemplo, se o utilizador quiser remover da base de conhecimento o utente(135 246 978,carlos,30,08,1987,masculino), o processo será visualizado do seguinte modo:

```
?- involucaoU(utente(135 246 978,carlos,30,08,1987,masculino)).
true.
```

*Figura 45- Teste á involução de conhecimento*

## 2.9. Invariantes

Os invariantes como testes à consistência da base de conhecimento, deverão criar limitações no uso da base de conhecimento de modo a assegurar a coerência da informação apresentada, deste modo foram criados nove invariantes para tal objetivo.

Os três seguintes invariantes têm a mesma função de limitar o uso do mesmo Id para vários utentes, atos ou análises. Estes irão adicionar as variáveis respetivas de cada predicado quando o comprimento da lista de todos os valores de Id iguais ao que se querem adicionar for zero, ou seja, o comprimento da lista de todos os Id's com esse valor específico, terá que ser igual a 1, ou porque foi adicionado não quebrando nenhumas regras dos invariantes ou porque já existia um e por tanto não foi adicionado.

```
%Invariante que verifica que não se adiciona um utente com Id repetido
+utente(I,N,D,M,A,G)::(findall(I,(utente(I,Nome,Dia,Mes,Ano,Genero)),S),comprimento(S,Z),Z==1).
```

*Figura 46- Invariante de repetição do utente*

Se o utilizador tentar inserir na base de conhecimento, por exemplo, o utente(123 456 789,mariana,11,05,2001,feminino), o processo será visualizado deste modo:

```
?- evolucaoU(utente(123 456 789,mariana,11,05,2001,feminino)).
false.
```

*Figura 47- Teste de invariante*

```
%Invariante que verifica que não se adiciona um ato com Id repetido
+ato(Ida,D,Idu,I,C,Pu,Prd,Prs)::(findall(Ida,(ato(Ida,Data,Idutente,Idade,Colesterol,Pulsacao,Pressaod,Pressaos)),S),comprimento(S,Z),Z==1).
```

*Figura 48- Invariante de repetição do ato*

```
%Invariante que verifica que não se adiciona um marcador com Id repetido
+marcador(Idm,A,Imin,Imax,G,Vmin,Vmax)::(findall(Idm,(marcador(Idm,Analise,Idademin,Idademax,Genero,Valormin,Valormax)),S),comprimento(S,Z),Z==1).
```

*Figura 49- Invariante de repetição da análise*

Os seguintes invariantes têm como função não permitir a inserção de conhecimento quando se tem conhecimento perfeito negativo oposto.

```
%Invariante que não permite que se insira conhecimento quando se tem conhecimento perfeito negativo oposto
+utente( I,N,D,M,A,G ) :: nao( -utente( I,N,D,M,A,G ) ).
+ato(Ida,D,Idu,I,C,Pu,Prd,Prs) :: nao(-ato(Ida,D,Idu,I,C,Pu,Prd,Prs)).
+marcador(Idm,Analise,Imin,Imax,G,Vmin,Vmax) :: nao(-marcador(Idm,Analise,Imin,Imax,G,Vmin,Vmax)).
```

*Figura 50- Invariante de conhecimento perfeito negativo*

Por exemplo, se o utilizador tentar inserir na base de conhecimento o utente(156 345 781,manuela,09,09,1999,masculino), o processo será visualizado deste modo:

```
?- evolucaoU(utente(156 345 781,manuela,09,09,1999,masculino)).
false.
```

*Figura 51- Teste de invariante*

Os seguintes invariantes não permitem que se adicione exceções a conhecimento perfeito positivo.

```
%Invariante que não permite adicionar excecoes a conhecimento perfeito positivo
+excecao( utente( I,N,D,M,A,G ) ) :: nao( utente( I,N,D,M,A,G ) ).
+excecao(ato(Ida,D,Idu,I,C,Pu,Prd,Prs)) :: nao(ato(Ida,D,Idu,I,C,Pu,Prd,Prs)).
+excecao(marcador(Idm,Analise,Imin,Imax,G,Vmin,Vmax)) :: nao(marcador(Idm,Analise,Idmin,Idmax,G,Vmin,Vmax)).
```

*Figura 52- Invariante de exceções sobre conhecimento perfeito positivo*

Os seguintes invariantes têm como função não permitir a evolução do conhecimento interdito, visto que o valor de elementos interditos não é possível conhecer.

```
%Invariante que não permite a evolucao do conhecimento interdito
+utente(I,N,D,M,A,G)::(findall(X,(utente(I,N,D,M,A,X),nao(interdito(X))),S),comprimento(S,L),L==1).
+ato(Ida,D,Idu,I,C,Pu,Prd,Prs)::(findall(X,(ato(Ida,D,Idu,I,X,Pu,Prd,Prs),nao(interdito(X))),S),comprimento(S,L),L==1).
```

*Figura 53- Invariante da evolução de conhecimento interdito*

Por exemplo, se o utilizador tentar inserir na base de conhecimento o ato(gmr6100,18/02/2021,987 654 321,36,135,123,66,145), o processo será visualizado deste modo:

```
?- evolucaoA(ato(gmr6100,18/02/2021,987 654 321,36,135,123,66,145)).
false.
```

*Figura 54- Teste de invariante*

Este invariante não permite a remoção de utentes da base de conhecimento que tenham atos associados.

```
%Invariante que não permite a remoção de utentes com atos associados
-utente(Idu,N,D,M,A,G)::(findall(X,ato(X,Data,Idu,I,C,Pu,Prd,Prs),J),comprimento(J,L),L==0).
```

*Figura 55- Invariante de utentes com atos*

Por exemplo, se o utilizador tentar remover da base de conhecimento o utente(987 654 321,beatriz,30,07,1985,feminino), o processo será visualizado deste modo:

```
?- involucaoU(utente(987 654 321,beatriz,30,07,1985,feminino)).
false.
```

*Figura 56- Teste de invariante*

Este invariante tem como função não permitir que se adicionem atos à base de conhecimento relativos a um utente que não exista.

```
%Invariante que não deixa inserir atos relacionados a um utente que não exita.
+ato(Ida,D,Idu,I,C,Pu,Prd,Prs)::(findall(Idu,utente(Idu,N,Dia,M,A,G),S),comprimento(S,L),L==1).
```

*Figura 57- Invariante de atos sem utentes*

Por exemplo, se o utilizador tentar inserir na base de conhecimento o ato(gmr8880,15/10/2022,222 000 333,20,150,70,70,111), o processo será visualizado deste modo:

```
?- evolucaoA(ato(gmr8880,15/10/2022,222 000 333,20,150,70,70,111)).
false.
```

Figura 58- Teste de invariante

Este invariante tem como função não permitir a inserção de exceções com IdAto repetido.

```
Invariante que não deixa inserir exceções com Idato repetido
excecao(ato(Ida,D,Idu,I,C,Pu,Prd,Prs))::(findall(Ida,(ato(Ida,Data,Idutente,Idade,Colesterol,Pulsacao,Pressaod,Pressaos)),S),comprimento(S,Z),Z==0).
```

*Figura 59- Invariante de exceções com Id repetido*

## 3. Conclusões e futuras sugestões

Neste trabalho, foram implementados diversos tipos de predicados e de invariantes, tendo sempre como objetivo dar uma resposta adequada ao problema proposto. Elaborando e implementando algo com o máximo de coesão e de simplicidade para um sistema de serviços médicos.

Este trabalho permite assim ao utilizador aceder a dados de diferentes utentes e atos, bem como fazer uma avaliação rápida de uma análise do ato feito através dos marcadores. Permite ainda a introdução e remoção de atos e de utentes.

Neste trabalho há alguns aspetos que como grupo, poderiam ser melhorados futuramente:

- A adição de uma maior quantidade de analises bem como os respetivos marcadores
- Criação de um predicado que encontra e apresenta os diferentes atos para um mesmo utente, comparando de maneira geral aos ótimos.

# 4. Bibliografia

http://capsi.apsi.pt/index.php/capsi/article/viewFile/276/265

https://elearning.uminho.pt/bbcswebdav/pid-1287115-dt-content-rid-6429685_1/courses/2223.J205N3_1/SugestoesRT.pdf