Universidade do Minho
Escola de Engenharia

Licenciatura em Engenharia Biomédica

Trabalho Prático – 1ª Parte

Inteligência Artificial em Engenharia Biomédica

1º Semestre – Ano Letivo 2023/2024

Grupo 14:

A100466, Beatriz Cunha

A101394, Carlota Brito

A101467, Diana Fiuza

Docentes:

César Analide Rodrigues

Inês Alves

Braga, 9 de novembro de 2023

Resumo

A realização deste exercício prático teve como principal objetivo a utilização da extensão à programação em lógica, no âmbito da representação de conhecimento imperfeito, permitindo assim aprimorar a competência sobre esta área. Desta forma, recorreu-se à utilização de valores nulos e da criação de mecanismos de raciocínios adequados. Como linguagem de programação em lógica foi usado o *Prolog*.

Assim, aplicou-se um sistema com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde.

Posto isto, em primeiro lugar procedeu-se à devida descrição de todas as metodologias aplicadas de modo a ser possível implementar a base de conhecimento em causa. Posteriormente, foram descritos os requisitos enunciados para a execução deste exercício prático, dos quais se destacam a representação de conhecimento positivo e negativo, assim como conhecimento imperfeito, o desenvolvimento de um sistema de inferência e a adequada atualização da base de conhecimento.

Índice

1 Introdução **1**

2 Preliminares **2**

2.1 Pressupostos da Programação Lógica 2

2.2 Extensão à Programação em Lógica 2

2.3 Conhecimento Imperfeito 3

3 Descrição e análise do trabalho **4**

3.1 Construção da Base de Conhecimento 4

3.1.1 Declarações Iniciais 4

3.1.2 Definições Iniciais 5

4 Sistema de Inferência **6**

4.1 Meta-predicado *si* 6

4.2 Meta-predicado *conj* e *disj* 6

5 Representação de Conhecimento Perfeito **11**

5.1 Conhecimento Perfeito Positivo 11

5.2 Conhecimento Perfeito Negativo 12

6 Representação de Conhecimento Imperfeito **14**

6.1 Conhecimento Imperfeito Incerto 14

6.2 Conhecimento Imperfeito Impreciso 15

6.3 Conhecimento Imperfeito Interdito 17

7 Invariantes **20**

7.1 Invariante para não permitir a inserção de um IdUtente repetido 20

7.2 Invariante para não permitir a inserção de um IdRegisto repetido 20

7.3 Invariantes para restringir a idade 21

7.4 Invariantes para restringir a altura 22

7.5 Invariantes para restringir o peso 24

7.6 Invariante para o sexo 25

7.7 Invariante para não inserir conhecimento negativo de um utente quando há conhecimento perfeito positivo para o mesmo 26
7.8 Invariante para não inserir um utente quando há conhecimento perfeito negativo para esse utente 26

8 Evolução e Involução **28**

9 Evolução do conhecimento **30**
9.1 Predicado de passagem de conhecimento positivo para negativo 30
9.2 Predicado de passagem de conhecimento negativo para positivo 30
9.3 Predicado de passagem de conhecimento incerto para positivo 31

10 Cálculo do IMC **33**
10.1 Classificção do IMC 33
10.2 Cálculo e representação do IMC 33

11 Alterações extra **37**
11.1 Alteração do *Sexo* 37
11.2 Alteração da *Idade* 37
11.3 Alteração da *Profissão* 38
11.4 Alteração da *Morada* e consequentemente *Hospital* 39
11.5 Alteração da *Altura* 39
11.6 Alteração do *Peso* 40

12 Predicados Auxiliares **41**
12.1 Predicado *nao* 41
12.2 Predicado *comprimento* 41
12.3 Predicado ***si*** 42
12.4 Predicado *pertence* 42
12.5 Predicado *validar* 43
12.6 Predicado *insercao e remocao* 43

13 Conclusão **45**

Anexos **47**

Lista de Figuras

1 Declarações iniciais. 4
2 Definições iniciais. 5
3 Extensão do meta-predicado *si*. 6
4 Extensão do meta-predicado *conj*. 7
5 Extensão do meta-predicado *disj*. 8
6 Testagem de *conj* e *disj*. 9
7 Extensão do predicado *utente*. 11
8 Extensão do predicado *registo*. 11
9 Extensão dos predicados *-utente*, *-registo* e *-imc*. 12
10 Exemplo de aplicação do predicado *-utente*. 12
11 Resultado obtido recorrendo ao SI. 13
12 Exemplos de conhecimento imperfeito incerto. 14
13 Resultado obtido recorrendo ao SI do conhecimento imperfeito incerto. . 15
14 Exemplos de conhecimento imperfeito impreciso. 16
15 Resultado obtido recorrendo ao si do conhecimento imperfeito do tipo impreciso. 17
16 Exemplos de conhecimento imperfeito interdito. 18
17 Resultado obtido recorrendo ao SI e tentativa de evolução do conhecimento imperfeito interdito. 19
18 Invariante que impede a repetição do *IdUtente*. 20
19 Tentativa de adicionar um utente com um *IdUtente* já existente. 20
20 Invariante que impede a repetição do *IdRegisto*. 21
21 Tentativa de adicionar um utente com um *IdRegisto* já existente. 21
22 Invariante que impede a inserção de um utente com mais de 65 anos. . . 21
23 Invariante que impede a inserção de um utente com menos de 18 anos. . 22
24 Tentativa de adicionar um utente com mais de 65 anos e outro com menos de 18. 22
25 Invariante que impede a inserção de um utente com mais de 2.70 metros de altura. 23

26 Invariante que impede a inserção de um utente com menos de 0.60 metros de altura. . . . 23
27 Tentativa de adicionar um utente com mais de 2.70 metros e outro com menos de 0.60 metros. . . . 23
28 Invariante que impede a inserção de um utente com mais de 600 kg de peso. . . . 24
29 Invariante que impede a inserção de um utente com menos de 10 kg de peso. . . . 24
30 Tentativa de adicionar um utente com mais de 600 kg e outro com menos de 10 kg. . . . 25
31 Invariante que impede a inserção de um utente com um sexo diferente de 1 ou 2. . . . 25
32 Tentativa de adicionar um utente com o sexo diferente de 1 ou 2. . . . 25
33 Invariante que impede a inserção de conhecimento negativo de um utente se houver conhecimento perfeito positivo. . . . 26
34 Tentativa de adição de conhecimento negativo 26
35 Invariante que impede a inserção de de um utente se houver conhecimento perfeito negativo para esse utente. . . . 27
36 Tentativa de adição de conhecimento negativo 27
37 Extensão do predicado *evolucao*. . . . 28
38 Extensão do predicado *involucao*. . . . 29
39 Extensão do predicado do *conhecimento positivo para negativo*. . . . 30
40 Extensão do predicado do *conhecimento negativo para positivo*. . . . 31
41 Um exemplo da extensão do predicado do *conhecimento incerto para positivo*. . . . 32
42 Definição da classificção do IMC. . . . 33
43 Cálculo do IMC. . . . 34
44 Cálculo do IMC através do *IdUtente* do respetivo utente. . . . 34
45 Cálculo do IMC através do *Nome* do respetivo utente. . . . 34
46 Tentativa de adição de conhecimento negativo 35
47 Predicado para remover o registo do indivíduo com 'Peso Normal'. . . . 35
48 Remoção do registo do indivíduo. . . . 36

49 Alteração do *Sexo* através do *IdRegisto*. 37
50 Alteração do *Sexo* através do *Nome*. 37
51 Alteração da *Idade* através do *IdRegisto*. 37
52 Alteração da *Idade* através do *Nome*. 38
53 Alteração da *Profissao* através do *IdRegisto*. 38
54 Alteração da *Profissao* através do *Nome*. 38
55 Alteração da *Morada* e *Hospital* através do *Nome*. 39
56 Alteração da *Altura* através do *IdRegisto*. 39
57 Alteração da *Altura* através do *Nome* 40
58 Alteração do *Peso* através do *IdRegisto*. 40
59 Alteração do *Peso* através do *Nome*. 40
60 Extensão do predicado *nao*. 41
61 Extensão do predicado *comprimento*. 41
62 Extensão do predicado *si*. 42
63 Extensão do predicado *pertence*. 42
64 Extensão do predicado *validar*. 43
65 Extensão do predicado *insercao e remocao*. 44

1 Introdução

O trabalho realizado teve como principal objetivo a utilização da extensão à programação em lógica e a utilização da linguagem *Prolog* no âmbito da representação de conhecimento imperfeito, recorrendo ao uso de valores nulos e à criação de mecanismos de raciocínio, para a construção de sistemas inteligentes suportados por técnicas de inteligência artificial simbólica [1]. O caso prático foi criado de modo a demonstrar as seguintes capacidades:

- Representar conhecimento positivo e negativo;
- Representar casos de conhecimento imperfeito, pela utilização de valores nulos de todos os tipos estudados;
- Representar invariantes que designem restrições à inserção e à remoção de conhecimento do sistema;
- Lidar com a problemática da evolução do conhecimento, criando os procedimentos adequados;
- Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas;
- Relatar o cálculo do IMC.

2 Preliminares

Para a correta realização do trabalho proposto, foi necessário ter em consideração novos conceitos ao nível da representação de conhecimento imperfeito através da extensão à programação em lógica. Estes novos conceitos são apresentados de seguida.

2.1 Pressupostos da Programação Lógica

Qualquer base de dados computacional necessita de ter a capacidade de armazenar e manipular informação. É neste aspecto que as linguagens de manipulação de informação, tal como o *Prolog*, ou outras linguagens de programação em lógica são importantes. Estas linguagens de manipulação de informação precisam de ter alicerces em diferentes pressupostos, sendo um destes o **Pressuposto Mundo Fechado** em que toda a informação que não existe mencionada na base de dados é considerada falsa.

No âmbito deste trabalho, com o objetivo de possibilitar a representação de conhecimento imperfeito, que engloba conhecimento incerto, impreciso e interdito, foi adotado o pressuposto do mundo aberto. Isto leva a que as questões realizadas à base de conhecimento devam obedecer a respostas de **verdadeiro**, **falso**, **desconhecido**.

2.2 Extensão à Programação em Lógica

A extensão à programação em lógica tem como objetivo permitir a representação explícita do conhecimento negativo. Este conhecimento pode ser realizado através de dois tipos de negação, a **negação por falha na prova** e a **negação forte**, onde se utiliza a conectiva "-". É importante salientar que a negação forte é simulada pela negação por falha na prova, representada pelo predicado *nao*. Ou seja, na negação por falha o *Prolog* executava o código caso não houvesse informação sobre o dado que estávamos a analisar, no entanto, há situações em que isto se torna inaceitável, bastando que a informação não estivesse contemplada. Concluímos, portanto, a necessidade de ter a possibilidade de representar conhecimento negativo e tomar decisões, sobre a existência do mesmo.

2.3 Conhecimento Imperfeito

Na área da prestação de cuidados de saúde, analisada no âmbito deste trabalho, existe a necessidade de representação de conhecimento imperfeito. Para além da existência de valores **verdadeiro** e **falso**, é também essencial a introdução de um novo valor, o **desconhecido**.

Esta representação de conhecimento imperfeito assentam numa situação na qual a informação está incompleta. Aqui, existem, então, valores denominados **valores nulos**, que permitem a distinção entre situações nas quais as respostas às questões deverão ser conhecidas (verdadeiro ou falso) ou desconhecidas.

Os tipos de valores nulos encontram-se enunciados e esclarecidos de seguida:

- **Incerto (valor nulo I):** os valores têm infinitas possibilidades, que podem ser definidas posteriormente;
- **Impreciso (valor nulo II):** o valor encontra-se dentro de um determinado conjunto de hipóteses, tendo possibilidades finitas;
- **Interdito (valor nulo III):** o valor não pode ser definido nem conhecido, tendo infinitas possibilidades.

3 Descrição e análise do trabalho

3.1 Construção da Base de Conhecimento

De forma a construir uma base de conhecimento que permita ao utilizador manipulá-la com maior facilidade, são implementadas funções de ajuste do ambiente *Prolog*, tais como as declarações iniciais e as definições iniciais.

3.1.1 Declarações Iniciais

No que diz respeito às declarações iniciais, estas denominam-se por *flags* e, têm como principal objetivo eliminar avisos que são emitidos pelo programa. Estas podem ser definidas por um determinado valor que varia de acordo com as necessidades do utilizador.

No nosso trabalho foram aplicadas as seguintes declarações iniciais: (Figura 1):

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

Figura 1: Declarações iniciais.

A declaração inicial "set_prolog_flag(discontinuous_warnings, off)" tem como objetivo desativar avisos relacionados a predicados descontínuos. Em Prolog, um predicado descontínuo ocorre quando a definição de um predicado é interrompida por outra declaração. A seguinte declaração "set_prolog_flag(single_var_warning, off)" consiste em desativar avisos relacionados a variáveis únicas. Assim, ao introduzir esta função, é possível evitar avisos onde variáveis únicas são intencionais. Por fim, a declaração inicial "set_prolog_flag(unknown, fail)" configura o comportamento quando o Prolog encontra um predicado desconhecido.

Nas duas primeiras declarações atribuiu-se o valor *off* refletindo-se na inibição de aparecimento de avisos quando as cláusulas de um certo predicado não estão juntas e nos avisos sobre variáveis únicas, respetivamente. Na última declaração é atribuído o valor *fail*, dando origem a uma falha quando são chamados predicados que não se encontram definidos

3.1.2 Definições Iniciais

Em relação às definições iniciais, estas são utilizadas para garantir que os predicados usados ao longo do trabalho sejam dinâmicos, ou seja, fornecem a estrutura básica para a definição de operadores, predicados e sua manipulação dinâmica no contexto de um programa em *Prolog*.

Na Figura 2, são apresentadas as definições iniciais inseridas no trabalho.

```
:- op(900,xfy,'::').
:- op(300,xfy, ou).
:- op(300,xfy, e).
:- dynamic(('-')/1).
:- dynamic (si/2).
:- dynamic (utente/4).
:- dynamic (registo/8).
:- dynamic (imc/4).
:- dynamic (interdito/1).
:- dynamic (excecao/1).
```

Figura 2: Definições iniciais.

Relativamente à primeira definição (::), verifica-se que esta corresponde à criação de um operador que identifica os invariantes. Os operadores *"e"* e *"ou"* mencionados na segunda e terceira definição, respetivamente, têm como principal função permitir a construção da conjunção e disjunção. Na seguinte definição, o símbolo *"-"* foi definido como dinâmico para permitir a elaboração do conhecimento negativo. Paralelamente, o predicado *"excecao"*, também tornado dinâmico, permite a elaboração do conhecimento imperfeito. Nas restantes definições, são fornecidos os predicados e os respetivos argumentos, de forma a torná-los dinâmicos.

Para concluir, é correto afirmar que, tanto as declarações como as definições, oferecem uma forma de adaptação do ambiente *Prolog* às preferências dos utilizadores, permitindo-lhes ajustar o programa de acordo com as suas necessidades específicas.

4 Sistema de Inferência

4.1 Meta-predicado *si*

O Sistema de Inferência (SI) é um meta-predicado que pode dar três respostas possíveis a um problema, Verdadeiro, Falso ou Desconhecido, consoante a questão colocada. Como tal, este *si* apresenta a seguinte forma (Figura 3):

```
% Extensão do meta-predicado si: Questao, Resposta -> {V, F}
si(Questao, verdadeiro):-Questao.
si(Questao,falso):- -Questao.
si(Questao,desconhecido):- nao(Questao),
                           nao(-Questao).
```

Figura 3: Extensão do meta-predicado *si*.

A interpretação do *Prolog* a este meta-predicado é a seguinte:

- Devolução de verdadeiro caso haja evidências que de a questão é verdadeira;
- Devolução de falso se houver evidências explícitas que a questão é falsa;
- Devolução de desconhecido para o caso de não haver evidência concreta de que a questão é verdadeira ou falsa.

4.2 Meta-predicado *conj* e *disj*

A elaboração destes sistemas de inferências permitiu que estes meta-predicados fossem capazes de responder a uma conjugação de duas questões relacionadas entre si através de uma conjunção ou de uma disjunção. Para desenvolver estes meta-predicados definiu-se que ambos usam como base o predicado *si* e executam recursivamente disjunções e conjunções até que 'Q1' e 'Q2' sejam valores únicos, e, aí, se execute o predicado *si*. Além disso, as possibilidades de resposta (verdadeiro, falso ou desconhecido) foram definidas para cada caso possível.

Nas Figuras 4 e 5 é possível observar a extensão do meta-predicado *conj* e *disj* respetivamente.

```
conj(Q1 e Q2, verdadeiro) :-
        si(Q1, verdadeiro),
        si(Q2, verdadeiro).
conj(Q1 e Q2, falso) :-
        si(Q1, verdadeiro),
        si(Q2, falso).
conj(Q1 e Q2, desconhecido) :-
        si(Q1, verdadeiro),
        si(Q2, desconhecido).
conj(Q1 e Q2, falso) :-
        si(Q1, falso),
        si(Q2, verdadeiro).
conj(Q1 e Q2, falso) :-
        si(Q1, falso),
        si(Q2, falso).
conj(Q1 e Q2, falso) :-
        si(Q1, falso),
        si(Q2, desconhecido).
conj( Q1 e Q2, desconhecido) :-
        si(Q1, desconhecido),
        si(Q2, verdadeiro).
conj(Q1 e Q2, falso) :-
        si(Q1, desconhecido),
        si(Q2, falso).
conj(Q1 e Q2, desconhecido) :-
        si(Q1, desconhecido),
        si(Q2, desconhecido).
```

Figura 4: Extensão do meta-predicado *conj*.

```
disj(Q1 ou Q2, verdadeiro) :-
        si(Q1, verdadeiro),
        si(Q2, verdadeiro).
disj(Q1 ou Q2, verdadeiro) :-
        si(Q1, verdadeiro),
        si(Q2, falso).
disj(Q1 ou Q2, verdadeiro) :-
        si(Q1, verdadeiro),
    si(Q2, desconhecido).
disj(Q1 ou Q2, verdadeiro) :-
        si(Q1, falso),
        si(Q2, verdadeiro).
disj(Q1 ou Q2, falso) :-
        si(Q1, falso),
        si(Q2, falso).
disj(Q1 ou Q2, desconhecido) :-
        si(Q1, falso),
        si(Q2, desconhecido).
disj(Q1 ou Q2, verdadeiro) :-
        si(Q1, desconhecido),
        si(Q2, verdadeiro).
disj(Q1 ou Q2, desconhecido) :-
        si(Q1, desconhecido),
        si(Q2, falso).
disj(Q1 ou Q2, desconhecido) :-
        si(Q1, desconhecido),
        si(Q2, desconhecido).
```

Figura 5: Extensão do meta-predicado *disj*.

Na Figura 6 podemos ver os resultados obtidos a partir da testagem de diversas conjunções de disjunções.

```
?- conj(utente(123456780, 'Antonio', guimaraes, barbeiro) e
↪  utente(987654321, 'Beatriz', braga, cozinheira),R).
R = verdadeiro .

?- disj(utente(123456780, 'Antonio', guimaraes, pintor) ou
↪  utente(987654321, 'Beatriz', braga, cozinheira),R).
R = verdadeiro .

?- conj(utente(146151888, jessica, barcelos, atleta) e
↪  utente(987654321, 'Beatriz', braga, cozinheira),R).
R = desconhecido .

?- disj(utente(146151888, jessica, barcelos, atleta) ou
↪  utente(987654321, 'Beatriz', braga, cozinheira),R).
R = verdadeiro .

?- conj(utente(146151888, jessica, barcelos, atleta) e
↪  utente(987654321, 'Beatriz', braga, cabeleireira),R).
R = falso .

?- disj(utente(146151888, jessica, barcelos, atleta) ou
↪  utente(987654321, 'Beatriz', braga, cabeleireira),R).
R = desconhecido .

?- disj(utente(123456780, 'Antonio', guimaraes, pintor) ou
↪  utente(987654321, 'Beatriz', braga, cabeleireira),R).
R = falso .

?- conj(utente(123456780, 'Antonio', guimaraes, barbeiro) e
↪  disj(utente(987654321, 'Beatriz', braga, cabeleireira) ou
↪  utente(146151888, jessica, barcelos, atleta)),R).
R = desconhecido .

?- disj(utente(123456780, 'Antonio', guimaraes, barbeiro) ou
↪  conj(utente(987654321, 'Beatriz', braga, cabeleireira) e
↪  utente(146151888, jessica, barcelos, atleta)),R).
R = verdadeiro .
```

Figura 6: Testagem de *conj* e *disj*.

No primeiro caso, testou-se a conjunção de um valor 'verdadeiro' e outro 'verdadeiro', obtendo-se 'verdadeiro na resposta.

Para a segunda situação, foi testada a disjunção de um valor 'falso' com um 'verdadeiro', resultando numa resposta verdadeira.

Na terceira testagem, foi analisada a conjunção de um valor 'desconhecido' com um 'verdadeiro', obtendo-se no final 'desconhecido'.

Em quarto lugar, avaliou-se a disjunção de um valor 'desconhecido' com um 'verdadeiro', resultando num final 'verdadeiro'.

No quinto caso, testou-se a conjunção de um valor 'desconhecido' com um 'falso', sendo a resposta 'falso'.

De seguida, foi testada a disjunção de um valor 'desconhecido' com um 'falso', resutando num 'desconhecido' como resposta.

Depois disso, analisou-se a disjunção de um valor 'falso' com outro 'falso', sendo a resposta 'falso'.

Além disso, foi tamb'em testada a conjunção de um valor 'verdadeiro' com a disjunção de um valor 'falso' e um 'desconhecido', resultando num 'desconhecido' como resposta.

Por último, foi também testada a disjunção de um valor 'verdadeiro' com a conjunção de um 'falso' e um 'desconhecido', cuja resposta foi 'verdadeiro'.

5 Representação de Conhecimento Perfeito

Com o intuito de povoar a base de conhecimento, realizaram-se várias inserções de conhecimento perfeito, sendo que este conhecimento pode ser definido como positivo ou negativo.

5.1 Conhecimento Perfeito Positivo

Apresentam-se alguns exemplos dos predicados utilizados para *utente*, *registo* e *imc*. A extensão completa dos predicados utilizados para povoar a base de conhecimento encontra-se no Anexo 1.

Os utentes foram definidos através do predicado *utente*, o qual tem como critérios o IdUtente, Nome, Cidade e Profissão. Um exemplo de aplicação deste predicado apresenta-se na Figura 7.

```
% Extensão do predicado utente: IdUtente, Nome, Morada, Profissao
    -> {V, F}
utente(123456780, 'Antonio', guimaraes, barbeiro).
```

Figura 7: Extensão do predicado *utente*.

Do mesmo modo, definaram-se os registos através do predicado *registo*, tendo por base o respetivo IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso e Hospital. A Figura 8 demonstra um exemplo da aplicação deste predicado.

```
% Extensão do predicado registo: IdRegisto, Data, IdUtente, Sexo,
    Idade, Altura, Peso, Hospital -> {V,F}
registo(gmr123, (30/09/2023), 123456780, 1, 21, 1.80, 75.7,
    hospital_da_luz).

```

Figura 8: Extensão do predicado *registo*.

5.2 Conhecimento Perfeito Negativo

Para a representação de conhecimento perfeito negativo relativamente ao utente, registo e imc, recorreu-se aos predicados *-utente*, *-registo* e *-imc*. Estes predicados garantem que o conhecimento definido é negativo. As suas respetivas extensões encontram-se na Figura 9.

```
-utente(IdUtente, Nome, Cidade, Profissao):-
nao(utente(IdUtente, Nome, Cidade, Profissao)),
nao(excecao(utente(IdUtente, Nome, Cidade, Profissao))).

-registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
↪  Hospital):-
nao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
↪  Hospital)),
nao(excecao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
↪  Peso, Hospital))).

-imc(IdIMC, Classificacao, LimiteInferior, LimiteSuperior):-
↪  nao(imc(IdIMC, Classificacao, LimiteInferior, LimiteSuperior)),
nao(excecao(imc(IdIMC, Classificacao, LimiteInferior,
↪  LimiteSuperior))).
```

Figura 9: Extensão dos predicados *-utente*, *-registo* e *-imc*.

Estes predicados permitem definir que qualquer facto que não esteja explicitamente presente na base de conhecimento sob forma de conhecimento positivo ou imperfeito, é considerado negativo. A título de exemplo, definiu-se o seguinte conhecimento perfeito negativo (Figura 10).

```
1  -utente(124578963, 'Carolina', guimaraes, cuf).
```

Figura 10: Exemplo de aplicação do predicado *-utente*.

Quando verificado pelo Sistema de Inferência (SI), o exemplo apresentado é considerado falso (Figura 11):

```
| ?- si(utente(124578963, 'Carolina', guimaraes, cuf),R).
R = falso
```

Figura 11: Resultado obtido recorrendo ao SI.

A partir da utilização de conhecimento negativo, é possível enriquecer a base de conhecimento com conhecimento mais realístico e versátil.

6 Representação de Conhecimento Imperfeito

O conhecimento imperfeito é um dos vários tipos de conhecimento existentes e é utilizado para representar situações onde o conhecimento não se encontra totalmente definido. Dito por outras palavras, é conhecimento que não é nem verdadeiro nem falso obrigando a introduzir um nova valor, o desconhecido. Este tipo de conhecimento pode ser dividido em:

- Incerto
- Impreciso
- Interdito

6.1 Conhecimento Imperfeito Incerto

O conhecimento imperfeito incerto representa uma informação que não é conhecida no momento atual mas pode se vir a conhecer no futuro. Ao longo da realização deste conhecimento no trabalho, foram definidas exceções com o objetivo de representar os parâmetros desconhecidos.

```
1 registo(gmr696, (29/09/2023), 147258369, 1, 20, incognita,
  ↪ 80, cuf).
2 excecao(registo(IdRegisto, Data, IdUtente, Sexo, Idade,
  ↪ Altura, Peso, Hospital )):- registo(IdRegisto, Data,
  ↪ IdUtente, Sexo, Idade, incognita, Peso, Hospital).
3
4 registo(gmr555, (29/09/2023), 963245987, 2, incognita,
  ↪ 1.71, 65.3, lusiadas).
5 excecao(registo(IdRegisto, Data, IdUtente, Sexo, Idade,
  ↪ Altura, Peso, Hospital )):- registo(IdRegisto, Data,
  ↪ IdUtente, Sexo, incognita, Altura, Peso, Hospital).
```

Figura 12: Exemplos de conhecimento imperfeito incerto.

Para este primeiro conhecimento, foram criados dois exemplos, como mostra na Figura 12. No primeiro exemplo, quando o utente foi a uma consulta não foi possível medir a altura dele, uma vez que o aparelho para o fazer não se encontrava disponível. Deste modo, qualquer altura introduzida na consola, vai tomar o valor desconhecido. Isto só é possível, pois foi acrescentado um novo utente sobre o qual o parâmetro altura é desconhecido, e definida uma exceção capaz de inserir as informações deste novo utente na base de dados, e implementar o conhecimento incerto. No exemplo seguinte, a ordem de raciocínio é a mesma uma vez que o utente não forneceu a sua idade e, portanto, qualquer valor inserido neste parâmetro vai dar o valor desconhecido. No entanto, ainda é possível que o utente forneça a sua idade. Na Figura 13 são testados os exemplos referidos.

```
1 | ?- si(registo(gmr696, (29/09/2023), 147258369, 1, 20,
  ↪  1.90, 80, cuf),R).
2 R = desconhecido.
3 | ?- si(registo(gmr555, (29/09/2023), 963245987, 2, 50,
  ↪  1.71, 65.3, lusiadas),R).
4 R = desconhecido.
```

Figura 13: Resultado obtido recorrendo ao SI do conhecimento imperfeito incerto.

6.2 Conhecimento Imperfeito Impreciso

O conhecimento imperfeito impreciso é usado quando o valor é desconhecido, mas há um intervalo de valores possíveis, ou seja, é um valor que se encontra dentro de um determinado intervalo conhecido, no entanto, não se sabe qual é o valor ao certo. Assim, este conhecimento é definido com exceções, de modo que quando for introduzido um valor fora do intervalo estipulado a resposta seja falso, caso contrário a resposta retornada é desconhecido (Figura 14):

```
excecao(registo(gmr001, (30/09/2023), 156325478, 2, 52,
↪  163, Peso, hospital_santa_maria)):- Peso>=54, Peso=<60.

excecao(registo(gmr523, (30/09/2023), 148878856, 1, 20,
↪  158, 62, lusiadas)).

excecao(registo(gmr523, (30/09/2023), 148878856, 1, 25,
↪  158, 62, lusiadas)).
```

Figura 14: Exemplos de conhecimento imperfeito impreciso.

Como é possível observar na Figura 14, para representar este conhecimento foram criadas duas situações. Na primeira, o utente não sabe ao certo o seu peso, mas sabe que pesa mais que a sua irmã, que pesa 54 Kg, mas menos que o seu irmão, que pesa 60 Kg. Isto é, o peso do utente está entre os 54 Kg e os 60 Kg, o que significa que qualquer valor introduzido que esteja dentro deste intervalo vai retornar desconhecido. O mesmo acontece para a segunda situação, onde o utente sabe que tem ou 20 anos ou 25 anos, fazendo com que qualquer valor introduzido que seja diferente de um destes vai retornar Falso.

```
| ?- si(registo(gmr001, (30/09/2023), 156325478, 2, 52,
↪  163, 56, hospital_santa_maria),R).
R = desconhecido.

| ?- si(registo(gmr001, (30/09/2023), 156325478, 2, 52,
↪  163, 63, hospital_santa_maria),R).
R = falso.

| ?- si(registo(gmr523, (30/09/2023), 148878856, 1, 20,
↪  158, 62, lusiadas),R).
R = desconhecido.

| ?- si(registo(gmr523, (30/09/2023), 148878856, 1, 19,
↪  158, 62, lusiadas),R).
R = falso.

```

Figura 15: Resultado obtido recorrendo ao si do conhecimento imperfeito do tipo impreciso.

6.3 Conhecimento Imperfeito Interdito

A principal diferença entre o conhecimento imperfeito interdito e os restantes é que este é utilizado quando não se sabe informação sobre um determinado parâmetro, desconhecido, e nunca vai ser possível saber pois não é permitido adicionar informação sobre este parâmetro. Para representar este conhecimento, é crucial definir o parâmetro específico com um valor constante, 'interdito', e estabelecer uma exceção para esse evento. Assim, sempre que houver consultas na base de dados, qualquer resposta relacionada a esse conhecimento será desconhecida. Além disso, é essencial criar uma invariante que impeça a adição de informações a esse parâmetro, associando o valor 'interdito' a um predicado interdito.

```
1  utente(146151888, jessica, barcelos, profissao_indefinida).
2  excecao(utente(IdUtente, Nome, Morada, Profissao)):-
   ↪  utente(IdUtente, Nome, Morada, profissao_indefinida).
3  interdito(profissao_indefinida).
4  +utente(IdUtente, Nome, Morada,
   ↪  Profissao)::(findall(Profissao,(utente(146151888,
   ↪  jessica, barcelos, Profissao),
   ↪  nao(interdito(Profissao))),S), comprimento(S,N),N==0).
5
6  registo(gmr999, 30/09/2023, 155266399, 1, 17, 174, 61,
   ↪  hospital_indefinido).
7  excecao(registo(IdRegisto, Data, IdUtente, Sexo, Idade,
   ↪  Altura, Peso, Hospital ):- registo(IdRegisto, Data,
   ↪  IdUtente, Sexo, Idade, Altura, Peso,
   ↪  hospital_indefinido).
8  interdito(hospital_indefinido).
9  +registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
   ↪  Peso,Hospital )::(findall(Hospital, (registo(gmr999,
   ↪  30/09/2023, 155266399, 1, 17, 174, 61, Hospi-
   ↪  tal),nao(interdito(Hospital))),S),comprimento(S,N),N==0).
```

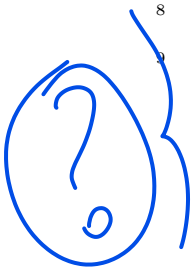


Figura 16: Exemplos de conhecimento imperfeito interdito.

Os exemplos em cima apresentados representam duas das várias aplicações que este conhecimento pode ter. Analisando o primeiro caso, verificamos que o utente não revela a sua profissão e não pretende fazê-lo no futuro. Para satisfazer este requisito, estipulou-se uma exceção que considera a profissão do utente em causa como interdita, e implementou-se uma invariante que não permite que este dado seja adicionado futuramente.

```
| ?- si(utente(146151888, jessica, barcelos,
↪  massagista),R).
R = desconhecido.

| ?-evolucao(utente(146151888, jessica, barcelos,
↪  massagista)).
false.

| ?- si(registo(gmr999, 30/09/2023, 155266399, 1, 17, 174,
↪  61, cuf),R).
R = desconhecido.

| ?-evolucao(registo(gmr999, 30/09/2023, 155266399, 1, 17,
↪  174, 61, cuf)).
false.
```

Figura 17: Resultado obtido recorrendo ao SI e tentativa de evolução do conhecimento imperfeito interdito.

7 Invariantes

De forma a impedir que fossem adicionadas informações incoerentes à base de conhecimento, definiram-se invariantes associados à evolução e involução do conhecimento que garantem a integridade da base de conhecimento.

7.1 Invariante para não permitir a inserção de um IdUtente repetido

Na Figura 18 apresenta-se a invariante que não permite ao utilizador adicionar um utente com um número de utente repetido.

```
1 +utente(IdUtente, Nome , Morada,
  ↪ Profissao)::(findall(IdUtente, (utente(IdUtente, _ , _,
  ↪ _)), S),
2                     comprimento( S,N ),
3                     N==0).
```

Figura 18: Invariante que impede a repetição do *IdUtente*.

Para testar esta invariante foi inserido na base de conhecimento informação relativa a um utente novo que apresentava um *IdUtente* já existente. Na Figura 19 verifica-se que a inserção desta informação não foi permitida, uma vez que o *IdUtente* utilizado já se encontrava na base de conhecimento.

```
1 ?- evolucao(utente(123456780,'Filipe',22,pescador)).
2 false.
```

Figura 19: Tentativa de adicionar um utente com um *IdUtente* já existente.

7.2 Invariante para não permitir a inserção de um IdRegisto repetido

Na Figura 20 apresenta-se a invariante que não permite ao utilizador adicionar um utente com um número de registo repetido.

```
+registo(IdRegisto, _, _, _, _, _, _,
    _)::(findall(IdRegisto, (registo(IdRegisto, _, _, _, _,
    _, _, _)), S),
                    comprimento(S,N),
                    N==1).
```

Figura 20: Invariante que impede a repetição do *IdRegisto*.

Para testar o funcionamento desta invariante inseriu-se na base de conhecimento informação relativa a um utente novo que apresentava um *IdRegisto* já existente. Na Figura 21 verifica-se que a inserção desta informação não foi permitida, uma vez que o *IdRegisto* utilizado já se encontrava na base de conhecimento.

```
?- evolucao(registo(gmr123, (30/9/2023), 111222345, 1, 39,
    1.78, 77.7, cuf)).
false.
```

Figura 21: Tentativa de adicionar um utente com um *IdRegisto* já existente.

ver se é relevante para a TA (penso que não)

7.3 Invariantes para restringir a idade

Nas Figuras 22 e 23 são apresentadas as invariantes que não permitem ao utilizador adicionar um utente com mais de 65 anos nem com menos de 18 anos, uma vez que os valores de IMC são calculados para pessoas que se encontram neste intervalo de idades.

```
+registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
    Peso, Hospital
    )::(findall(Idade,(registo(_,_,_,_,Idade,_,_,_),
    Idade>65),S),
                    comprimento(S,N),
                    N==0).
```

Figura 22: Invariante que impede a inserção de um utente com mais de 65 anos.

```
+registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
    Peso, Hospi-
    tal)::(findall(Idade,(registo(_,_,_,_,Idade,_,_,_),
    Idade<18),S),
                        comprimento(S,N),
                        N==0).
```

Figura 23: Invariante que impede a inserção de um utente com menos de 18 anos.

Para testar o funcionamento destas invariantes inseriu-se na base de conhecimento informação relativa a um utente novo que tinha uma idade superior à idade máxima permitida (65 anos) e outro com uma idade inferior à mínima permitida (18 anos). Na Figura 24 verifica-se que a inserção desta informação não foi permitida.

```
?- evolucao(registo(gmr302, (30/9/2023), 101292345, 1, 70,
    1.8, 70.7, lusiadas)).
false.

?- evolucao(registo(gmr332, (30/9/2023), 100200305, 2, 10,
    1.90, 72.7, lusiadas)).
false.

```

Figura 24: Tentativa de adicionar um utente com mais de 65 anos e outro com menos de 18.

7.4 Invariantes para restringir a altura

Para limitar os valores de altura, foram criadas duas invariantes que não permitem ao utilizador introduzir um utente cuja altura seja superior a 2,70 metros ou inferior a 0,60 metros, porque foi considerado que para as faixas etárias consideradas ninguém ultrapassa estes limites de altura. Nas Figuras 25 e 26 são apresentadas estas invariantes.

```
+registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
    Peso, Hospital
    )::(findall(Altura,(registo(_,_,_,_,_,Altura,_,_),
    Altura>2.70),S),
                          comprimento(S,N),
                          N==0).
```

Figura 25: Invariante que impede a inserção de um utente com mais de 2.70 metros de altura.

```
+registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
    Peso, Hospital)::(findall (Altura,
    (registo(_,_,_,_,_,Altura,_,_), Altura<0.60),S),
                          comprimento(S,N),
                          N==0).
```

Figura 26: Invariante que impede a inserção de um utente com menos de 0.60 metros de altura.

Inseriu-se, então, na base de conhecimento informação relativa a um utente novo que tinha uma altura superior à máxima permitida e outro com uma altura inferior à mínima permitida. Na Figura 27 verifica-se que a inserção desta informação foi negada.

```
?- evolucao(registo(gmr707, (30/9/2023), 987123478, 2, 40,
    3.0, 82.7, cuf)).
false.

?- evolucao(registo(gmr697, (30/9/2023), 988735632, 1, 23,
    0.50, 89.7, trofa_saude)).
false.

```

Figura 27: Tentativa de adicionar um utente com mais de 2.70 metros e outro com menos de 0.60 metros.

7.5 Invariantes para restringir o peso

Para evitar que houvesse valores de peso que não fossem reais e possíveis criaram-se duas invariantes que não permitem ao utilizador introduzir um utente cujo peso seja superior a 600 quilogramas inferior e a 10 quilogramas, porque se considerou que não era possível existirem valores de peso fora deste intervalo. Nas Figuras 28 e 29 é possível ver estas invariantes.

```
1  +registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
   ↪  Peso, Hospital
   ↪  )::(findall(Peso,(registo(_,_,_,_,_,_,Peso,_),
   ↪  Peso>600),S),
2                          comprimento(S,N),
3                          N==0).
```

Figura 28: Invariante que impede a inserção de um utente com mais de 600 kg de peso.

```
1  +registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
   ↪  Peso,
   ↪  Hospital)::(findall(Peso,(registo(_,_,_,_,_,_,Peso,_),
   ↪  Peso<10),S),
2                          comprimento(S,N),
3                          N==0).
```

Figura 29: Invariante que impede a inserção de um utente com menos de 10 kg de peso.

Para testar o funcionamento destas, inseriu-se na base de conhecimento informação relativa a um utente novo que tinha um peso superior ao máximo permitido e outro com um peso inferior ao mínimo permitido. Na Figura 30 verifica-se que a inserção desta informação foi negada.

```
?- evolucao(registo(gmr605, (30/9/2023), 907543174, 1, 30,
↪  1.70, 610, trofa_saude)).
false.

?- evolucao(registo(gmr521, (30/9/2023), 078634562, 2, 27,
↪  1.60, 7, cuf)).
false.
```

Figura 30: Tentativa de adicionar um utente com mais de 600 kg e outro com menos de 10 kg.

aqui podemos usar [M,F] para distinguir masculino de feminino, em vez de 1 e 2

7.6 Invariante para o sexo

Assumindo que o número 1 corresponde ao sexo masculino e o 2 ao feminino, criaram-se duas invariantes que não permitem ao utilizador introduzir um utente cujo sexo não corresponda ao número 1 ou 2. Na Figura 31 é apresentada esta invariante.

```
+registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
↪  Peso, Hospital)::(pertence(Sexo,[1,2])).
```

Figura 31: Invariante que impede a inserção de um utente com um sexo diferente de 1 ou 2.

Testou-se o funcionamento desta invariante com a tentativa de inserir na base de conhecimento informação relativa a um utente novo que tinha o sexo diferente de 1 ou 2. Na Figura 32 verifica-se que a inserção desta informação foi negada.

```
evolucao(registo(gmr411, (30/9/2023), 372901635,3, 24, 1.87,
↪  74.2, lusiadas)).
false.
```

Figura 32: Tentativa de adicionar um utente com o sexo diferente de 1 ou 2.

7.7 Invariante para não inserir conhecimento negativo de um utente quando há conhecimento perfeito positivo para o mesmo

Na Figura 33 é apresentada uma invariante que não permite que um dado utente seja referenciado como conhecimento negativo quando este se encontra na base de conhecimento como conhecimento positivo.

```
1 +(-utente(IdUtente, Nome , Morada,
  ↪  Profissao))::(findall(IdUtente,(utente(IdUtente, Nome ,
  ↪  Morada, Profissao)),S),
2                     comprimento(S,N), N==0).
```

Figura 33: Invariante que impede a inserção de conhecimento negativo de um utente se houver conhecimento perfeito positivo.

Testou-se o funcionamento desta invariante com a tentativa de inserir na base de conhecimento informação negativo de um utente já existente na base de conhecimento como conhecimento positivo. Na Figura 34 verifica-se que a inserção desta informação foi negada.

```
1 ?- evolucao(-utente(135246978, 'Carlos', vilaverde,
  ↪  carteiro)).
2 false.
```

Figura 34: Tentativa de adição de conhecimento negativo

7.8 Invariante para não inserir um utente quando há conhecimento perfeito negativo para esse utente

Na Figura 35, à semelhança do caso anterior, é apresentada uma invariante que não permite que um dado utente seja adicionado como conhecimento positivo quando este se encontra na base de conhecimento como conhecimento negativo.

```
+(-utente(IdUtente, Nome , Morada,
    ↪ Profissao))::(findall(IdUtente,(utente(IdUtente, Nome ,
    ↪ Morada, Profissao)),S),
                        comprimento(S,N), N==0).
```

Figura 35: Invariante que impede a inserção de de um utente se houver conhecimento perfeito negativo para esse utente.

Testou-se o funcionamento desta invariante com a tentativa de inserir na base de conhecimento informação positiva quando esta mesma já existia na base de conhecimento como informação negativa. Na Figura 36 verifica-se que a inserção desta informação foi negada.

```
?- evolucao(utente(124578963, 'Carolina', guimaraes, cuf)).
false.
```

Figura 36: Tentativa de adição de conhecimento negativo

8 Evolução e Involução

O predicado ***evolucao*** permite a evolução da base de conhecimento, ou seja, a inserção de novos factos. Este predicado tem como argumento um termo que diz respeito ao conhecimento que se pretende acrescentar. Na Figura 37 encontra-se descrito o predicado ***evolucao***, assim como os predicados ***insercao*** e ***validar***, necessários à sua construção referidos nos capítulos 12.6 e 12.5, respetivamente.

```
1 % Extensão do predicado que permite a evolução do conhecimento
2 evolucao(Termo) :-
3         findall(Invariante, +Termo::Invariante,Lista),
4         insercao(Termo),
5         validar(Lista).
6
```

Figura 37: Extensão do predicado *evolucao*.

Com base no predicado *findall* (predicado já definido pelo interpretador de *Prolog*), são encontrados todos os invariantes que se relacionam com o termo introduzido e guardados numa lista. De seguida, utiliza-se o predicado ***insercao*** que, recorrendo à funcionalidade *assert*, possibilita a inserção do termo à base de conhecimento. Depois, aplica-se o predicado ***validar*** que é responsável por verificar se cada um dos invariantes presentes na lista criada anteriormente se continua a manter válido. Caso algum invariante já não seja válido, então o teste falha e retrocede-se à segunda cláusula do predicado ***insercao***, levando à remoção do termo através da funcionalidade *retract*.

Assim sendo, o predicado ***evolucao*** tem a capacidade de garantir manter a validade da base de conhecimento conforme os invariantes que foram construídos. Se apenas fossem utilizadas as funcionalidades *assert* e *retract* não seria possível manter a consistência da base.

O predicado ***involucao*** permite a involução da base de conhecimento, ou seja, possibilita a remoção de factos. Este predicado tem como argumento um termo que corresponde ao conhecimento que se pretende remover. Na Figura 38 encontram-se representados os predicados ***involucao*** e ***remocao***, predicados mencionados também nos capítulos 12.6 e 12.5, respetivamente.

```
% Extensão do predicado que permite a involução do conhecimento
involucao(Termo):-
        findall(Invariante,-Termo::Invariante,Lista),
        validar(Lista),
        remocao(Termo).

```

Figura 38: Extensão do predicado *involucao*.

De facto, este predicado é muito semelhante ao predicado ***evolucao***, sendo que a única diferença se prende com o facto de nesta situação se recorrer ao predicado ***remocao***. Logo, se algum invariante não se manter válido após a remoção de conhecimento, então o teste falha e retrocede-se, pelo que não é removido o conhecimento da base. Desta forma, tal como anteriormente, é garantida a validade da base de conhecimento consoante os invariantes criados.

9 Evolução do conhecimento

9.1 Predicado de passagem de conhecimento positivo para negativo

O predicado que permite a passagem do conhecimento positivo para negativo recebe dois argumentos que correspondem ao número de identificação do registo (IdRegisto) e ao número de identificação do utente (IdUtente). Para que isto aconteça é necessário que esse utente já esteja inserido na base de conhecimento. Deste modo, o predicado vai atuar transformando um utente em não utente. Esta transformação ocorre recorrendo ao predicado involucao, que retira o utente da base e, de seguida, ao predicado evolucao, que insere o insere como negativo.

```
1 %Transformacao de um utente para um nao utente
2 p_n(IdRegisto,IdUtente):- involucao(registo(IdRegisto, Data, IdUtente,
  ↪  Sexo, Idade, Altura, Peso, Hospital)),
3 involucao(utente(IdUtente,Nome,Morada, Profissao)),
4 evolucao(-registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
  ↪  Hospital )),
5 evolucao(-utente(IdUtente,Nome,Morada, Profissao)).
```

Figura 39: Extensão do predicado do *conhecimento positivo para negativo*

.

9.2 Predicado de passagem de conhecimento negativo para positivo

Este predicado tem como função fazer exatamente o contrário do predicado anterior. Desta forma, são passados os mesmos argumentos, transformando um não utente em utente.

```
% Transformacao de um nao utente para um utente
n_p(IdRegisto,IdUtente):- involucao(-registo(IdRegisto, Data, IdUtente,
    Sexo, Idade, Altura, Peso, Hospital )),
involucao(-utente(IdUtente,Nome,Morada, Profissao)),
evolucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
    Hospital )),
evolucao(utente(IdUtente,Nome,Morada, Profissao)).

```

Figura 40: Extensão do predicado do *conhecimento negativo para positivo*

.

9.3 Predicado de passagem de conhecimento incerto para positivo

Como já foi referido anteriormente, foram construídas diversas funções, referidas no capítulo 11, que permitiam a modificação de determinados parâmetros referentes ao utente. Cada função destas, poderia receber como argumento o número de identificação do registo ou então o próprio nome do utente e o parâmetro que pretendiam alterar. Contudo, estes predicados definidos anteriormente, também representam uma forma de passagem de conhecimento incerto para conhecimento perfeito. Este fenómeno é possível, uma vez que numa primeira fase é verificado se o utente está na base de conhecimento e se tal se verificar é removida a sua informação com o antigo parâmetro que é pretendido remover e, posteriormente, é inserido um novo registo com o parâmetro atualizado.

não percebi porque é que é de conhecimento incerto para positivo

```
1  alterar_idade_registo(IdRegisto, NovaIdade):- involucao(registo(IdRegisto,
   ↪  Data, IdUtente, Sexo, Idade, Altura, Peso, Hospital )),
   ↪  evolucao(registo(IdRegisto, Data, IdUtente, Sexo, NovaIdade, Altura,
   ↪  Peso, Hospital )).
2
3  alterar_idade_nome(Nome,NovaIdade):- utente(IdUtente, Nome,_,_),
4  registo( IdRegisto,Data, IdUtente, Sexo,Idade,Altura,Peso,Hospital),
5  involucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
   ↪  Hospital)),
6  evolucao(registo(IdRegisto, Data, IdUtente, Sexo, NovaIdade, Altura, Peso,
   ↪  Hospital)).
7
```

Figura 41: Um exemplo da extensão do predicado do *conhecimento incerto para positivo*

.

10 Cálculo do IMC

10.1 Classificção do IMC

De acordo com os documentos auxiliares, foi possível estabelecer os limites para as diferentes classificações de Índice de Massa Corporal [2].

Assumindo que a fórmula usada para o cálculo do IMC foi a seguinte:

$$IMC = \frac{Peso}{Altura^2} \tag{1}$$

Através da equação referida anteriormente, é possível distinguir 6 categorias definidas por um limite superior e inferior de valores de IMC. Estes valores dependem essencialmente do peso e da altura de cada indivíduo.

```
1 classificar_imc(IMC, 'Magreza') :-  IMC =< 10, IMC < 18.5.
2 classificar_imc(IMC, 'Peso Normal') :- IMC >= 18.5, IMC < 25.
3 classificar_imc(IMC, 'Sobrepeso') :- IMC >= 25, IMC < 30.
4 classificar_imc(IMC, 'Obesidade Grau I') :- IMC >= 30, IMC < 35.
5 classificar_imc(IMC, 'Obesidade Grau II') :- IMC >= 35, IMC < 40.
6 classificar_imc(IMC, 'Obesidade Grau III') :- IMC >= 40.
```

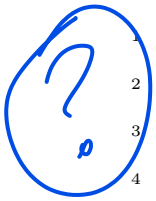
Figura 42: Definição da classificção do IMC.

.

10.2 Cálculo e representação do IMC

Para além da possibilidade de calcular o valor do IMC de um indivíduo através da fórmula base, foram também desenvolvidas outras funções para o que mesmo fosse possível através do fornecimento de algum dado do utente em questão, em particular através do *IdUtente* ou *Nome*.

No primeiro caso, acede diretamente ao registo através do *Idutente* para a recolha dos dados necessários para a determinação da categoria pretendida.



```
calcular_e_mostrar_imc(IdUtente) :-
    registo(IdRegisto, _, IdUtente, _, _, Altura, Peso, _),
    IMC is Peso / (Altura * Altura),
    classificar_imc(IMC, Classificacao),
    format("O utente ~w tem um IMC de ~2f, o que pode ser classificado
    ↪  como ~w.", [Nome, IMC, Classificacao]).
```

Figura 43: Cálculo do IMC.

.

Testou-se o funcionamento desta invariante com o fornecimento do *IdUtente*, para que o sistema calculasse o valor do IMC. Na Figura 44 verifica-se que o mesmo foi possível.

```
?- calcular_e_mostrar_imc(123456780).
O utente Antonio tem um IMC de 23.36, o que pode ser
   ↪  classificado como Peso Normal.
true.
```

Figura 44: Cálculo do IMC através do *IdUtente* do respetivo utente.

Na segunda situação, o sistema acede ao registo, através do *Nome*, e em seguida, pelo *IdUtente*, consegue aceder aos dados do registo do respetivo número de utente, e aí retirar os dados necessários para o cálculo e classificação do IMC.

```
calcular_e_mostrar_imc1(Nome):-
    utente(IdUtente, Nome,_,_),
    registo(_, _, IdUtente,_, _ , Altura,Peso,_),
    IMC is Peso / (Altura * Altura),
    classificar_imc(IMC, Classificacao),
    format("O utente ~w tem um IMC de ~2f, o que pode ser classificado
    ↪  como ~w.", [Nome, IMC, Classificacao]).
```

Figura 45: Cálculo do IMC através do *Nome* do respetivo utente.

.

Testou-se o funcionamento desta invariante com o fornecimento do *Nome*, para que o sistema calculasse o valor do IMC. Na Figura 46 verifica-se que o mesmo foi possível.

```
?- calcular_e_mostrar_imc1('Antonio').
O utente Antonio tem um IMC de 23.36, o que pode ser
    ↪  classificado como Peso Normal.
true.
```

Figura 46: Tentativa de adição de conhecimento negativo

Por fim, se com o cálculo deste parâmetro for concluído que a classificação do nível de IMC de um determinado indivíduo é *Peso Normal*, o mesmo pode ser removido da base de conhecimento, com o auxilio do predicado *involucao*, por falta de necessidade de acompanhamento médico.

```
% Regra para remover paciente da base de conhecimento se o IMC for 'Peso
    ↪  Normal'.

verificar_e_remover_paciente(IdRegisto) :-
    registo(IdRegisto, _, _, _, _, Altura, Peso, _),
    IMC is Peso / (Altura * Altura),
    classificar_imc(IMC, 'Peso Normal'),
    involucao(registo(IdRegisto, _, _, _, _, Altura, Peso, _)),
    write('Registo do utente removido da base de conhecimento.').

```

Figura 47: Predicado para remover o registo do indivíduo com 'Peso Normal'.

.

Verificou-se assim que , numa fase inicial, foi calculado o IMC e, posteriormente, de acordo com a sua classificação, e através do predicado definido, o registo do utente em questão foi removido da base de conhecimento Na Figura 48 verifica-se isso mesmo.

```
?-verificar_e_remover_paciente(gmr123).
Paciente removido da base de conhecimento.
true .
```

Figura 48: Remoção do registo do indivíduo.

.

11 Alterações extra

Seguem-se alguns predicados que permitem a alteração de parâmetros quer do *registo* quer do *utente*. Estes predicados assentam maioritariamente no acesso à base de conhecimento através dos parâmetros *IdRegisto* e *Nome*.

11.1 Alteração do *Sexo*

Predicado que permite a alteração do *Sexo* do indivíduo através do *IdRegisto*.

```
1 alterar_sexo_registo(IdRegisto, NovoSexo):- involucao(registo(IdRegisto,
  ↪ Data, IdUtente, Sexo, Idade, Altura, Peso, Hospital )),
2 evolucao(registo(IdRegisto, Data, IdUtente, NovoSexo, Idade, Altura, Peso,
  ↪ Hospital )).
```

Figura 49: Alteração do *Sexo* através do *IdRegisto*.

Predicado que permite a alteração do *Sexo* do indivíduo através do *Nome*.

```
1 alterar_sexo_nome(Nome,NovoSexo):- utente(IdUtente, Nome,_,_),
2 registo( IdRegisto,Data, IdUtente ,Sexo,Idade,Altura,Peso,Hospital),
3 involucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
  ↪ Hospital)),
4 evolucao(registo(IdRegisto, Data, IdUtente, NovoSexo, Idade, Altura, Peso,
  ↪ Hospital)).
```

Figura 50: Alteração do *Sexo* através do *Nome*.

11.2 Alteração da *Idade*

Predicado que permite a alteração da *Idade* do indivíduo através do *IdRegisto*.

```
1 alterar_idade_registo(IdRegisto, NovaIdade):- involucao(registo(IdRegisto,
  ↪ Data, IdUtente, Sexo, Idade, Altura, Peso, Hospital )),
2 evolucao(registo(IdRegisto, Data, IdUtente, Sexo, NovaIdade, Altura, Peso,
  ↪ Hospital )).
```

Figura 51: Alteração da *Idade* através do *IdRegisto*.

Predicado que permite a alteração da *Idade* do indivíduo através do *Nome*.

```
alterar_idade_nome(Nome,NovaIdade):- utente(IdUtente, Nome,_,_),
registo( IdRegisto,Data, IdUtente ,Sexo,Idade,Altura,Peso,Hospital),
involucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
↪  Hospital)),
evolucao(registo(IdRegisto, Data, IdUtente, Sexo, NovaIdade, Altura, Peso,
↪  Hospital)).
```

Figura 52: Alteração da *Idade* através do *Nome*.

11.3 Alteração da *Profissão*

Predicado que permite a alteração do *Profissao* do indivíduo através do *IdRegisto*.

```
alterar_profissao_registo(IdRegisto, NovaProf):-
↪  registo(IdRegisto,_,IdUtente,_,_,_,_,_),
utente(IdUtente,Nome,Morada,Profissao),
involucao(utente(IdUtente, Nome, Morada, Profissao)),
evolucao(utente(IdUtente, Nome, Morada, NovaProf)).
```

Figura 53: Alteração da *Profissao* através do *IdRegisto*.

Predicado que permite a alteração da *Profissao* do indivíduo através do *Nome*.

```
alterar_profissao_nome(Nome,NovaProf):- involucao(utente(IdUtente, Nome,
↪  Morada, Profissao)),
evolucao(utente(IdUtente, Nome, Morada, NovaProf)).
```

Figura 54: Alteração da *Profissao* através do *Nome*.

11.4 Alteração da *Morada* e consequentemente *Hospital*

Predicado que permite a alteração da *Morada* e *Hospital* do indivíduo através do *Nome*.

```
alterar_mh_nome(Nome,NovaMorada,NovoHospital):- utente(IdUtente,
↪ Nome,_,_),
registo( IdRegisto,Data, IdUtente,Sexo,Idade,Altura,Peso,Hospital),
involucao(utente(IdUtente, Nome, Morada, Profissao)),
involucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
↪ Hospital)),
evolucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
↪ NovoHospital)),
evolucao(utente(IdUtente, Nome, NovaMorada, Profissao)).
```

Figura 55: Alteração da *Morada* e *Hospital* através do *Nome*.

11.5 Alteração da *Altura*

Predicado que permite a alteração da *Altura* do indivíduo através do *IdRegisto*.

```
alterar_altura_registo(IdRegisto, NovaAltura):-
↪ involucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura,
↪ Peso, Hospital )),
evolucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, NovaAltura, Peso,
↪ Hospital )).
```

Figura 56: Alteração da *Altura* através do *IdRegisto*.

Predicado que permite a alteração da *Altura* do indivíduo através do *Nome*.

```
alterar_altura_nome(Nome,NovaAltura):- utente(IdUtente, Nome,_,_),
registo( IdRegisto,Data, IdUtente ,Sexo,Idade,Altura,Peso,Hospital),
involucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
↪ Hospital)),
evolucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, NovaAltura, Peso,
↪ Hospital)).
```

Figura 57: Alteração da *Altura* através do *Nome*

.

11.6 Alteração do *Peso*

Predicado que permite a alteração do *Peso* do indivíduo através do *IdRegisto*.

```
alterar_peso_registo(IdRegisto, NovoPeso):- involucao(registo(IdRegisto,
↪ Data, IdUtente, Sexo, Idade, Altura, Peso, Hospital )),
evolucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, NovoPeso,
↪ Hospital )).
```

Figura 58: Alteração do *Peso* através do *IdRegisto*.

Predicado que permite a alteração do *Peso* do indivíduo através do *Nome*.

```
alterar_peso_nome(Nome,NovoPeso):- utente(IdUtente, Nome,_,_),
registo( IdRegisto,Data, IdUtente ,Sexo,Idade,Altura,Peso,Hospital),
involucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, Peso,
↪ Hospital)),
evolucao(registo(IdRegisto, Data, IdUtente, Sexo, Idade, Altura, NovoPeso,
↪ Hospital)).
```

Figura 59: Alteração do *Peso* através do *Nome*.

.

12 Predicados Auxiliares

Com o objetivo das funções implementadas ao longo do trabalho cumprirem correta o seu objetivo, é necessário inserir determinados predicados auxiliares que sustentam a formação de outros predicados.

12.1 Predicado *nao*

O predicado ***nao*** tem como principal função verificar se a questão passada como argumento está ou não definida na base de conhecimento. Este predicado é fundamental para a construção do conhecimento negativo.

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

Figura 60: Extensão do predicado *nao*.

12.2 Predicado *comprimento*

O predicado ***comprimento***, tem como objetivo determinar o comprimento de uma lista, passada como argumento, e esta lista pode ser vazia ou não. Se for fazia, a lista não apresenta qualquer elemento e, portanto, o seu comprimento vai ser zero. Em contrapartida, se não for vazia são contados os elementos nela presente, devolvendo o número total de elementos(N1). Este predicado faz a contagem destes elementos recorrendo à recursividade e só para quando atingir o critério de paragem, lista vazia.

supostamente não é obrigatório ter isto porque vem incuído no prolog

```
% Extensão do predicado comprimento: Lista, Resultado --> {V, F}
comprimento([],0).
comprimento([X|L],N):- comprimento(L,N1),  N is N1+1.
```

Figura 61: Extensão do predicado *comprimento*.

12.3 Predicado *si*

O sistema de inferência ***si***, permite que as questões sejam interpretadas e avaliadas como "Verdadeiro", "Falso" ou "Desconhecido".

```
% Extensão do predicado si: questao, resposta,estado -> {V, F}
si(Questao, verdadeiro):-Questao.
si(Questao,falso):- -Questao.
si(Questao,desconhecido):-nao(Questao),nao(-Questao).
```

Figura 62: Extensão do predicado *si*.

12.4 Predicado *pertence*

O predicado *pertence* é utilizado para verificar se um elemento X pertence a uma lista dada. Este opera recursivamente sobre a lista, comparando o primeiro elemento com X e, se não for igual, chama ,recursivamente, o mesmo predicado para o restante da lista. Este predicado foi utilizado na realização das invariantes, com maior destaque na invariante que não permite a inserção de outro número exceto o 1 e 2 que representam o sexo masculino e feminino, respetivamente.

```
1 % Extensao do predicado pertence:
2 pertence(X, [X|_]).
3 pertence(N, [_|Tail]) :-
4     pertence(N,Tail).
```

Figura 63: Extensão do predicado *pertence*.

12.5 Predicado *validar*

O predicado *validar* valida uma lista de restrições, onde cada elemento da lista (R) representa uma condição a ser satisfeita. A lista é considerada globalmente válida se todas as condições individuais forem atendidas. Este predicado foi enunciado em funções como o evolução e involução.

```
1 % Extensao do predicado validar:
2 validar([]).
3 validar([R|LR]):- R, validar(LR).
```

Figura 64: Extensão do predicado *validar*.

12.6 Predicado *insercao e remocao*

O predicado *insercao* serve para inserir um termo na base de conhecimento usando assert e para remover um termo usando retract. O comportamento deste predicado baseia-se em se o termo já existe na base de conhecimento então o assert adiciona uma nova ocorrência do mesmo termo, mas se o termo não existe na base de conhecimento, o assert adiciona o termo à base de conhecimento. Em contrapartida, o retract remove o termo da base de conhecimento se ele estiver presente. Se retract for bem-sucedido (ou seja, o termo existe e é removido), a cláusula falha usando fail. O mesmo acontece para o predicado *remocao* mas o raciocino é feito ao contrário. O predicado insercao é utilizado na evolução, enquanto que o remocao é usado na involução.

```
1 % Extensao do predicado insercao:
2 insercao(Termo) :-assert(Termo).
3 insercao(Termo)
  ↪  :-retract(Termo),!,fail.
4
5 % Extensao do predicado remocao:
6 remocao(Termo):-retract(Termo).
7 remocao(Termo):-assert(Termo),!,fail.
```

Figura 65: Extensão do predicado *insercao* e *remocao*.

13 Conclusão

A elaboração deste trabalho prático permitiu consolidar os conhecimentos teóricos e práticos adquiridos até agora nesta Unidade Curricular, nomeadamente as noções de sistemas de representação de conhecimento e raciocínio em *Prolog*, bem como expandir o nosso conhecimento na área da programação lógica.

O trabalho cumpre com os requisitos que foram pedidos para que fosse possível caracterizar um universo de discurso na área da saúde para o cálculo e análise do índice de massa corporal (IMC), tendo sido desenvolvido o sistema proposto com as capacidades que eram necessárias, e algumas capacidades extra.

Referências

[1] César Analide e Inês Alves. Programação em lógica. Universidade do Minho.

[2] Maple Tech. International LLC. Bmi calculator. https://www.calculator.net/bmi-calculator.html.

Anexos

Anexo 1 - Povoamento da base de conhecimento

```
% Extensão do predicado utente: IdUutente, Nome, Morada, Profissao -> {V,
↪  F}
utente(123456780, 'Antonio', guimaraes, barbeiro).
utente(987654321, 'Beatriz', braga, cozinheira).
utente(135246978, 'Carlos', vilaverde, carteiro).
utente(963245987, 'Carlota', porto, bailarina).


% Extensão do predicado registo: IdRegisto, data, IdUtente, sexo, idade,
↪  altura, peso, hospital -> {V, F}
registo(gmr123, (30/09/2023), 123456780, 1, 21, 1.80, 75.7,
↪  hospital_da_luz).
registo(gmr234, (30/09/2023), 987654321, 2, 35, 1.61, 55.3, trofa_saude).
registo(gmr567, (30/09/2023), 135246978, 1, 43, 1.76, 93.5, cuf).
```