Escola de Engenharia

Universidade do Minho

Relatório Exercício Prático 1

Programação em Lógica Estendida e Conhecimento Imperfeito

Grupo de Trabalho 17
Adriana Miranda, A95126
Beatriz Rodrigues, A95678
Bruna Matos, A95827
Bruno Machado, A97374

Braga, 2 de dezembro de 2022

Conteúdo

Introdução

Neste trabalho prático 1 irá ser utilizada a Programação em Lógica usando a linguagem de programação PROLOG: uma linguagem declarativa que usa fragmentos da lógica proposicional de 1.ª ordem (Cláusulas de Horn) para representar o conhecimento sobre um dado problema e recorre a processos de inferência lógica para concluir o valor de verdade duma proposição em função da sua "relação" com os factos.

No âmbito da representação de Conhecimento imperfeito, recorre-se à utilização de valores nulos e da criação de mecanismos de raciocínio, onde o contradomínio é $(\mathbb{V}, \mathbb{F}, \mathbb{D})$, correspondendo a "Verdadeiro", "Falso" e "Desconhecido", respetivamente.

Neste trabalho, pretende-se que seja desenvolvido um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da análise e exames de marcadores de saúde. As entidades que serão objeto de estudo são: **Utente**, **Ato** e **Marcador**.

Objetivo

A realização deste projeto tem como objetivo particular a familiarização com o conceito de programação em lógica, colocando em prática os conhecimentos lecionados nesta UC. Com este projeto será possível explorar e adquirir novos conhecimentos sobre a utilização da linguagem Prolog, mas também consolidar alguns conceitos.

Pretendemos que o sistema de representação de conhecimento e raciocínio tenha uma perspetiva simplificada da realidade, no qual possamos qualificar e caracterizar uma Base de Conhecimento consistente e capaz de responder a problemas relacionados com a área da prestação de cuidados de saúde. O ser humano não limita, por norma, o seu conhecimento e o seu raciocínio aos Pressupostos do Mundo Fechado e Domínio Fechado, isto é, habitualmente aquilo que não se conhece toma como valor de verdade "Desconhecido" e não "Falso". Somos capazes de colocar questões para as quais não temos uma resposta válida no momento. Além disso, é útil ao ser humano representar conhecimento negativo, isto é, afirmar explicitamente que algo é falso e querer tomar essa informação como conhecimento.

Pretendemos que a Base de Conhecimento a desenvolver permita lidar com questões cujas respostas não sejam, necessariamente, "Verdadeiro" ou "Falso", lidando de forma válida, coerente e lógica com o Conhecimento Imperfeito. Resumidamente, pode-se dizer que se pretende atingir os seguintes objetivos:

• 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;

• 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 análises e exames por utente/ato.

1. Representação de Conhecimento e Raciocínio

Este trabalho irá incidir numa análise à representação do conhecimento e do raciocínio sobre duas perspetivas: o conhecimento perfeito e o conhecimento imperfeito. Estas possuem representações similares, onde a principal diferença está associada aos pressupostos utilizados e, consequentemente, aos mecanismos de ilação e de prova que são utilizados para dar repostas às perguntas efetuadas.

1.1 Conhecimento Perfeito

Na representação do conhecimento perfeito, as cláusulas e os predicados possuem exclusivamente como contradomínio dois valores de prova: "Verdadeiro" e "Falso". Desta forma, são considerados os seguintes pressupostos [1]:

- **Pressuposto do Mundo Fechado** - toda a informação que não se encontra mencionada na Base de Conhecimento é considerada falsa;
- **Pressuposto dos Nomes Únicos** - duas constantes diferentes designam, necessariamente, duas entidades diferentes na Base de Conhecimento;
- **Pressuposto do Domínio Fechado** - não há mais objetos no universo de discurso para além dos definidos na Base de Conhecimento.

1.2 Conhecimento Imperfeito

O Conhecimento Imperfeito deixa de assumir que a informação representada é a única válida e que as entidades representadas sejam as únicas existentes no mundo exterior. Deste modo, os pressupostos aceites podem ser diferentes daqueles que são a base da representação de conhecimento perfeito.

Desta forma, são considerados os seguintes pressupostos [1]:

- **Pressuposto do Mundo Aberto** - podem existir outros factos ou conclusões verdadeiras para além daqueles representados na Base de Conhecimento;
- **Pressuposto do Domínio Aberto** - podem existir mais objetos do universo de discurso para além daqueles designados pelas constantes da Base de Conhecimento;
- **Pressuposto dos Nomes Únicos**.

1.3 Representação de conhecimento

A Extensão da Programação em Lógica implementa mecanismos que funcionam adequadamente quando confrontados com informação incompleta ou em mudança. Um dos seus objetivos é permitir representar, explicitamente, informação falsa. Para tal, há a necessidade de considerar dois tipos de negação [1]:

- **Negação por falha na prova** - negação utilizada nos programas em Lógica Tradicional, representada pelo termo: "**não**";
- **Negação forte** (ou *clássica*) - negação utilizada como forma de identificar informação negativa, ou falsa, representada pela conetiva "$\neg$".

Estes surgem da necessidade de distinguir dois tipos de situações que permitem concluir sobre o valor lógico de uma cláusula p:

- ($n\tilde{a}o(p)$) se aceitarmos que é falso por falta de prova;
- ($\neg p$) se é possível provar que é efetivamente falso pela negação forte.

Assim, o conjunto de respostas válidas às questões (q) sobre o programa, está definido para os valores de verdade “Verdadeiro”, “Falso” e “Desconhecido”, tal que:

- **Verdadeiro** ($\mathbb{V}$): se $\exists x : q(x)$, então consegue-se provar a resposta através de factos positivos;
- **Falso** ($\mathbb{F}$): se $\exists x : \neg q(x)$, então consegue-se provar a resposta através de factos negativos;
- **Desconhecido** ($\mathbb{D}$): se $\exists x : n\tilde{a}o(q(x)) \wedge n\tilde{a}o(\neg q(x))$, então não é possível provar a resposta através dos factos e dos mecanismos de inferência.

2. Base de Conhecimento

2.1 Entidades da Base de Conhecimento

Nesta secção o principal objetivo é explicar o funcionamento da nossa Base de Conhecimento. A sua representação pode ser caracterizada por diferentes entidades:

- **Utente**: #IdUtente, Nome do utente, Data de Nascimento, Sexo;
- **Ato**: #IdAto, Data, #IdUtente, Idade, Colesterol, Pulsação, Pressão;
- **Marcador**: #IdMarcador, Análise, Idade, Sexo, Mínimo, Máximo.

2.1.1 Utente

Na Base de Conhecimento utilizada, o utente é caracterizado pelo Id do utente e o seu respetivo nome, data de nascimento e sexo. Cada utente é identificado por um Id único de forma a evitar inconformidades e ambiguidade de informação. Desta forma, todos os utentes existentes na nossa Base de Conhecimento são do tipo:

```
utente( 1,diogomoura,data( 25,3,2002 ),masculino ).
utente( 2,brunamatos,data( 19,7,2002 ),feminino ).
utente( 3,joaolavinas,data( 28,8,2000 ),masculino ).
utente( 4,josepereira,data( 17,1,2000 ),masculino ).
utente( 5,carinamatos,data( 14,5,1999 ),feminino ).
```

(...)

2.1.2 Ato

Um ato médico é caracterizado pelo seu identificador (#IdAto), pela sua data de realização, pelo identificador do utente (#IdUtente) a quem se refere, bem como a sua idade, nível de colesterol, valores da pulsação e pressão sanguínea.

Para a implementação das datas dos atos médicos, por uma questão de conseguir satisfazer os requisitos do enunciado, foi considerado um predicado auxiliar data que segue a seguinte nomenclatura:

```
dia( data( D,M,A ),D ).
mes( data( D,M,A ),M ).
ano( data( D,M,A ),A ).
```

Os factos que representam um ato médico foram escritos da seguinte forma:

```
ato(#IdAto,Data,#IdUtente,Idade,Colestrol,Pulsação, Pressão ).
```

```
ato( 1,data( 1,2,2010 ),2,20,150,60, pressao( 70,110 ) ).
ato( 2,data( 15,7,2015 ),4,22,155,65, pressao( 80,120 ) ).
ato( 3,data( 26,10,2020 ),3,22,160,60, pressao( 75,115 ) ).
ato( 4,data( 8,12,2021 ),9,21,150,60, pressao( 80,120 ) ).
ato( 5,data( 13,3,2008 ),1,20,160,70, pressao( 80,120 ) ).
```

(...)

2.1.3 Marcador

O marcador é caracterizado por um identificador (#IdMarcador) que vai servir para o distinguir dos restantes, pelo tipo de análise feita, pela idade do utente analisado, pelo seu género e pelos valores padrão que serão comparados com os medidos.

```
marcador(7,colesterol,idade( 71,150 ),masculino,vnormal( 0,190 ) ).
marcador(8,colesterol,idade( 71,150 ),feminino,vnormal( 0,180 ) ).
marcador(15,pulsacao,idade( 71,150 ),masculino,vnormal( 70,80 ) ).
marcador(16,pulsacao,idade( 71,150 ),feminino,vnormal( 70,80 ) ).
marcador(17,pressao,idade( 18,30 ),masculino, vnormal(pmin( 60,80 ),
         pmax( 110,130 )) ).
marcador(18,pressao,idade( 18,30 ),feminino,vnormal(pmin( 60,80 ),
         pmax( 110,130 )) ).
```

(...)

2.2 Integridade da Base de Conhecimento

De modo a garantir a integridade da Base de Conhecimento foram criados alguns predicados. Assim sendo, ao longo do desenvolvimento deste trabalho fomos dando uso ao conceito de invariantes. Estes permitem-nos controlar em especifico a **inserção** e a **remoção** de informações na nossa Base de Conhecimento.
Os invariantes, em Prolog, são representados da seguinte forma:

- **+Termo**:: Premissa(s) usada(s) quando se pretende adicionar algo à base de conhecimento, que tem de ser verificada(s) após o momento da inserção;
- **-Termo**:: Premissa(s) usada(s) quando se pretende adicionar algo à base de conhecimento, que tem de ser verificada(s) após o momento da remoção;

Existem já definidos, em Prolog, predicados que nos permitem adicionar e remover factos da Base de Conhecimento.Estes são o `assert` e o `retract`, respetivamente. No entanto, a utilização destes predicados, exclusivamente, não garante consistência, uma vez que os invariantes definidos não são testados por si só e, assim sendo, é importante a criação de predicados auxiliares que nos tragam garantias que a inserção e a remoção de conhecimento não alterem a consistência da Base de Conhecimento. Estes predicados auxiliares serão abordados em secções posteriores deste relatório.

2.2.1 Invariantes de Remoção

Os invariantes que estão associados ao processo de remoção da informação na base de conhecimento são os seguintes:

Utente

```
% O Utente a remover não pode ter atos associados

-utente( IDU,N,DN,S ) :: ( solucoes( IDU,ato( _,_,IDU,_,_,_,_),S ),
                           comprimento( S,L ),
                           L == 0 ).
```

2.2.2 Invariantes de Inserção

Os invariantes que estão associados ao processo de remoção da informação na base de conhecimento são os seguintes:

Ato

```
% Não permitir inserir atos com IdUt não registados

+ato( IDA,D,IDU,I,C,PL,P ) :: ( solucoes( IDU,utente( IDU,_,_,_ ),S ),
                                comprimento( S,L ),
                                L == 1 ).
```

2.3 Valores nulos

É possível, por vezes, encontrar informação incompleta, mas dentro desta ter-se diferentes tipos de desconhecimento. Esta identificação de valores é feita recorrendo ao conceito de valores nulos, que surgem como uma estratégia para fazer a distinção entre respostas a questões que devem ser concretizadas como “conhecidas” (“verdadeiras” ou “falsas’) ou respostas concretizadas como “desconhecidas” [1].

As respostas podem ser desconhecidas por três razões: por serem relativas a uma questão de onde não se conhece efetivamente **nenhum valor** ; por se saber que é válida dentro de um **conjunto determinado de valores** (valores desconhecidos, mas de um conjunto finito de valores); por fim, por serem relativas a uma questão à qual não se permite haver resposta, ou seja, **a resposta é interdita** (valores não permitidos) [1].

Para conseguir a representação destes valores, apresentados de seguida, foram criados os seguintes predicados:

- **excecao(Termo)** - para representar um caso de exceção;
- **nulo(Termo)**- para representar como nulo qualquer unificação com Termo.

Dado o contexto em que este problema está a ser desenvolvido, é necessário especificar em que medida se pode considerar um dado Termo como sendo efetivamente “falso”, quando implementado em Prolog. A seguinte formalização permite

identificar, para um dado predicado p (que pode representar qualquer uma das entidades da base de conhecimento adotada), em que situação se pode afirmar que é "falso" p(x).

$$\neg p(x) \quad \leftarrow \quad não\Big(p(x)\Big) \ \wedge \ não\Big(excecao\Big(p(x)\Big)\Big) \tag{2.1}$$

A adoção desta regra permite fazer uma extensão ao Pressuposto do Mundo Fechado, na medida em que se abre a possibilidade de existirem casos de exceções, os quais são valorados como desconhecido. Tudo o resto, para além do que é verdadeiro, é considerado falso. De seguida são apresentados os três tipos de valores nulos já identificados, bem como a forma como podem ser formulados.

2.3.1 Tipo Desconhecido/Incerto

Os valores nulos do tipo desconhecido permitem representar valores desconhecidos sem que haja a especificação de um conjunto de valores. A identificação de valores deste tipo é feita através da introdução de uma situação de exceção correspondente a uma condição anómala.

Se p é um predicado, x um argumento de p e v um valor nulo, tem-se a seguinte representação de valor nulo do tipo desconhecido:

$$excecao\Big(p(x)\Big) \quad \leftarrow \quad p(v) \tag{2.2}$$

2.3.2 Tipo Desconhecido de um dado conjunto de valores/Impreciso

A diferença entre o valor nulo do tipo desconhecido apresentado anteriormente e o valor nulo que se pretende apresentar agora, desconhecido mas de um conjunto finito e determinado de valores, está precisamente no facto de este último valor nulo representar um (ou mais) valores de um conjunto finito de valores bem determinados: só não é conhecido, especificamente, qual dos valores concretizará a questão [1].

Seja p um predicado, x um argumento de p e v1 e v2 valores nulos. Pode-se representar a possibilidade de p não ser "falso" para $\mathtt{x} \in \{v1, v2\}$.

Para tal recorremos à sequência de exceções:

$$excecao\Big(p(v1)\Big) \tag{2.3}$$

$$excecao\Big(p(v2)\Big) \tag{2.4}$$

2.3.3 Tipo não permitido/Interdito

Este terceiro tipo de valores nulos caracterizam aqueles que não se pretende que surjam na base de conhecimento, o valor que representam não é permitido especificar ou conhecer (é interdito) [1].

Se p é um predicado, x um argumento de p e v um valor nulo, tem-se a seguinte representação de valor nulo do tipo interdito, com uma exceção que representa a situação anómala a representar:

$$excecao\Big(p(x)\Big) \quad \leftarrow \quad p(v) \tag{2.5}$$

$$nulo(v) \tag{2.6}$$

Deve ainda ser implementado um mecanismo que impeça a posterior inclusão de valores que violem a condição imposta pelo valor nulo não permitido. Neste exercício prático, este mecanismo foi implementado como um invariante, que se pode verificar em secções posteriores deste relatório.

2.4 Conhecimento Perfeito

O conhecimento perfeito, é aquele sobre o qual existe informação completa.

De seguida são apresentados todos os factos (positivos e negativos) que perfazem o conhecimento perfeito da Base de Conhecimento.

2.4.1 Conhecimento Perfeito Positivo

No caso da representação do conhecimento perfeito positivo foi definida a informação da seguinte forma:

Utente

A informação relativa aos utentes é considerada tendo por base factos, tais como:

```
utente( 6,manuelcosta,data( 13,2,1970 ),masculino ).
utente( 7,mariasantos,data( 16,2,1984 ),feminino ).
utente( 8,ricardofazeres,data( 5,2,1979 ),masculino ).
utente( 9,marianaamaro,data( 17,5,2001 ),feminino ).
utente( 10,joaomaria,data( 24,5,2003 ),masculino ).
```

Ato

Recorreu-se ao predicado auxiliar data,apresentado na secção Predicados Auxiliares, para representar a data em que ocorreu um ato e ao predicado auxiliar relativo à pressão. A representação da informação relativa aos atos foi feita através de factos, tais como:

```
ato( 1,data( 1,2,2010 ),2,20,150,60, pressao( 70,110 ) ).
ato( 2,data( 15,7,2015 ),4,22,155,65, pressao( 80,120 ) ).
ato( 3,data( 26,10,2020 ),3,22,160,60, pressao( 75,115 ) ).
ato( 4,data( 8,12,2021 ),9,21,150,60, pressao( 80,120 ) ).
ato( 5,data( 13,3,2008 ),1,20,160,70, pressao( 80,120 ) ).
```

Marcador

Para os marcadores temos a seguinte representação que ultiliza os predicados auxiliares relativos à idade, ao valor normal e à pressão máxima/mínima:

```
marcador( 1,colesterol,idade( 18,30 ),masculino,vnormal( 0,160 ) ).
marcador( 2,colesterol,idade( 18,30 ),feminino,vnormal( 0,150 ) ).
marcador( 11,pulsacao,idade( 31,45 ),masculino,vnormal( 60,70 ) ).
marcador( 12,pulsacao,idade( 31,45 ),feminino,vnormal( 60,70 ) ).
marcador( 17,pressao,idade( 18,30 ),masculino,vnormal( pmin( 60,80 )
                                                    ,pmax( 110,130 )) ).
marcador( 18,pressao,idade( 18,30 ),feminino,vnormal( pmin( 60,80 )
                                                    ,pmax( 110,130 )) ).
```

2.4.2 Conhecimento Perfeito Negativo

Para representar o conhecimento negativo de forma explícita, foram definidos predicados auxiliares para cada entidade da Base de Conhecimento caracterizados pelo `-functor`, representando a negação forte do predicado.

Utente

A informação negativa relativa aos utentes pode ser apresentada da seguinte forma:

"O utente Manuel Costa com #IdUtente11, com data de nascimento 25/11/2002, nega ser do sexo masculino."

```
-utente( 11,manuelcosta, data( 25,11,2002 ),masculino ).
```

"O utente com o #IdUtente12, chamada Anastacia Gonçalves nega ter nascido no dia 12/10/1995."

```
-utente( 12,anastaciagoncalves,data( 12,10,1995 ),feminino ).
```

Ato

A representação da informação negativa relativa aos atos médicos foi feita através da negação de factos, tais que podemos representar da seguinte forma:

"Sabe-se que o ato com #IdAto6 ocorrido no dia 31/08/2000, realizado pelo utente com #IdUtente8, não teve um valor de colesterol de 150."

```
-ato( 6,data( 31,8,2000 ),8,43,150,70, pressao( 80,125 ) ).
```

Marcador

Para representarmos conhecimento negativo em relação ao marcador temos a seguinte representação:

"Não existe um marcador de colesterol #IdMarcador25 para uma idade compreendida entre os 18 e os 30 com valores normais entre 200 e 300 para o sexo masculino."

```
-marcador( 25,colesterol,idade( 0,17 ),masculino,vnormal( 200,300 ) ).
```

2.5 Conhecimento Imperfeito

O conhecimento imperfeito pode ser de três tipos:

- **Incerto**
- **Impreciso**
- **Interdito**

2.5.1 Conhecimento Imperfeito Incerto

Este tipo de conhecimento será representado utilizando os valores nulos do tipo "desconhecido", ou seja, valores dos quais desconhecemos a sua concretização e não possuímos informação sobre o conjunto de valores.

Utente

"Desconhece-se a idade do utente com o #IdUtente13, de nome Maria João e do sexo feminino."

```
utente( 13 ,mariajoao,xpto001,feminino ).
excecao( utente( IDU,N,DN,S ) ) :-
    utente( IDU,N,xpto001,S ).
```

Ato

"Desconhece-se a data em que foi prestado o ato medico com #IdAto7, realizado pelo utente com o #IdUtente10, cujos valores de colesterol, pulsação e pressão são 155,60, (80/120), respetivamente."

```
ato( 7,xpto003,10,19,155,60, pressao( 80,120 ) ).
excecao( ato( IDA,D,IDU,I,C,PL,P ) ) :-
    ato( IDA,xpto003,IDU,I,C,PL,P ).
```

"Desconhece-se o valor de colesterol no ato ocorrido em 03/06/2007, com #IdUtente6, identificado pelo #IDAto8."

```
ato( 7, data( 3,6,2007 ),6,52,xpto004,70, pressao( 80,130 ) ).
excecao( ato( IDA,D,IDU,I,C,PL,P ) ) :-
    ato( IDA,D,IDU,I,xpto004,PL,P ).
```

"Desconhece-se o valor de mínimo da pressão no ato ocorrido em 24/10/2014, com valor máximo 120, com #IdUtente5, identificado pelo #IDAto8."

```
ato( 8, data( 24/10/2014 ),5,23,160,60, pressao( xpto005,120 ) ).
excecao( ato( IDA,D,IDU,I,C,PL,P ) ) :-
    ato( IDA,D,IDU,I,C,PL,pressao( xpto005,120 ) ).
```

Marcador

"Desconhece-se o valor normal de colesterol para um individuo do sexo masculino com idade compreendida entre os 0 e os 17 anos."

```
marcador( 26,colesterol,idade( 0,17 ),masculino,xpto007 ).
excecao( marcador( IdM,M,I,S,VN ) ) :-
    marcador( IdM,M,I,S,xpto007 ).
```

2.5.2 Conhecimento Imperfeito Impreciso

Para representar o conhecimento imperfeito impreciso, serão utilizados valores nulos do tipo desconhecido, tendo por base um conjunto possíveis de valores.

Utente

"Não se sabe se a Joana Matos, utente com #IdUtente14 nasceu no dia 17/03/2008 ou no dia 16/03/2008."

```
excecao( utente( 14,joanamatos,data( 17,03,2008 ),feminino ) ).
excecao( utente( 14,joanamatos,data( 16,03,2008 ),feminino ) ).
```

"Não se sabe ao certo o dia de nascimento do Maurício, com o #IdUtente15 e do sexo masculino. Apenas se sabe que nasceu entre o dia 5 e o dia 10 ."

```
excecao( utente( 15,mauriciorodrigues,data( D,05,1994 ),masculino ) ) :-
    D >= 5, D =< 10.
```

Ato

"O ato médico com #IdAto9 ocorrido em 03/06/2008, com o #IdUtente6, teve uma medição do valor do colesterol entre 140 e 150."

```
excecao( 9,data( 3,6,2008 ),6,52,C,70, pressao( 80,130 ) ) :-
    C >= 140, C =< 150.
```

"O ato médico com #IdAto10 ocorreu ou a 04/09/2017 ou a 05/09/2017 com o #IdUtente7."

```
excecao( ato( 10,data(4,9,2017 ),7,38,150,65, pressao( 70,120) ) ).
excecao( ato( 10,data(5,9,2017 ),7,38,150,65, pressao( 70,120) ) ).
```

Marcador

"Sabe-se que o valor da pulsação minima normal para um individuo do sexo feminino com idade compreendida entre os 0 e os 17 anos está entre 55 a 65."

```
marcador( 27,pulsacao,idade( 0,17 ),feminino,pulsacao( MIN,70 ) ).
    MIN >= 55, MIN =< 65.
```

2.5.3 Conhecimento Imperfeito Interdito

Conhecimento imperfeito interdito recorre a valores nulos do tipo não permitido, ou seja, valores dos quais desconhecemos a sua concretização e que se pretende que não surjam na base de conhecimento.

Utente

"O utente com o #IdUtente17, de nome Donald Trump, nascido dia 14/06/1946, exige que não se saiba o seu sexo."

```
utente( 17,donaldtrump,data( 14,06,1946 ),xpto002 ).
excecao( utente( IDU,N,DN,S ) ) :-
    utente( IDU,N,DN,xpto002 ).
nulo( xpto002 ).
+utente( IDU,N,I,M ) :: ( solucoes( X,( utente( 17,_,_,X ),nao( nulo( X ) ) ),
                          comprimento( S,L ),
                          L == 0 ).
```

Ato

"O utente com o #IdUtente1, realizou um ato medico #IdAto11 no dia 28/7/2015, contudo não quer que se saiba o valor de pulsação medido"

```
ato( 11,data( 28,7,2015 ),1,20,155,xpto006, pressao( 80,120) ).
excecao( ato( IDA,D,IDU,I,C,PL,P ) ) :-
    ato( IDA,D,IDU,I,C,xpto006,P ).
nulo( xpto006 ).

+ato( IDA,D,IDU,I,C,PL,P ) :: ( solucoes( PL,( ato( 11,_,_,_,_,PL,_ ),nao
                                    ( nulo( PL ) ) ),S ),comprimento( S,N ),
                              N == 0 ).
```

Marcador

"Para um individuo do sexo feminino com idade compreendida entre os 0 e os 17 anos, não é possível saber o valor normal da pressão arterial."

```
marcador( 28,pressao,idade( 0,17 ),feminino,xpto009 ).
exececao( marcador( IdM,M,I,S,VN ) ) :-
    marcador( IdM,M,I,S,xpto009 ).
nulo( xpto009 ).
```

3. Predicados Auxiliares

Nesta secção são apresentados alguns predicados que foram úteis para a implementação das funcionalidades do sistema. Teremos assim uma breve apresentação dos predicados usados, bem como, uma breve explicação do seu funcionamento.

Predicado "Data"

Este predicado surge de forma a facilitar a utilizacação e manipulção da data de um ato. Torna-se mais fácil pois conseguimos variar as variáveis em separado e não torna-las completamente dependentes umas das outras.

```
   Extensao dos predicados dia, mes e ano: Data,Dia/Mes/Ano -> {V,F}

dia( data( D,M,A ),D ).
mes( data( D,M,A ),M ).
ano( data( D,M,A ),A ).
```

Predicado "Pressão/Valores Normais"

Tal como referido anteriormente, estes predicados surgem também para facilitar o acesso aos diferentes valores, máximos ou mínimos da pressão e dos valores Normais("valores tabelados") usados no predicado marcador.

Estes predicados ajudam a definir o valor máximo e o mínimo nas diferentes situações.

```
   Extensão dos predicados pressão mínima e máxima:
Pressao, Min/Max -> {V,F}

min( pressao( MIN,MAX ),MIN ).
max( pressao( MIN,MAX ),MAX ).

   Extensao dos predicados valores normais mínimos e máximos:
VNormais, Min/Max -> {V,F}

min( vnormal( MIN,MAX ),MIN ).
max( vnormal( MIN,MAX ),MAX ).

   Extensao dos predicados valores de pressão mínimos:
,Pressão Mínima Min/Max -> {V,F}

min( pmin( MIN,MAX ),MIN ).
max( pmin( MIN,MAX ),MAX ).

   Extensao dos predicados valores de pressão máximo:
, Pressão Máxima Min/Max -> {V,F}

min( pmax( MIN,MAX ),MIN ).
max( pmax( MIN,MAX ),MAX ).
```

```
   Extensao dos predicados idade mínima e máxima:
Idade, Min/Max -> {V,F}

min( idade( MIN,MAX ),MIN ).
max( idade( MIN,MAX ),MAX ).
```

Predicado "Soluções"

O predicado solucões trata-se simplesmente de uma implementação que tem a mesma funcionalidade que o predicado `findall`, já existente no Prolog.Este é usado para obter todos os resultados que satisfazem uma dada condição. Em `z` está a lista de resultados de `X` que satisfazem as condições em `Y`.

```
   Extensão do predicado soluções

solucoes( X,Y,Z ) :-
    findall( X,Y,Z ).
```

Predicado "Testa"

Tem-se também o predicado testa que recebe como argumento uma lista de invariantes e percorre toda essa lista testando cada um dos seus elementos, conjugando-os. Se algum falhar, o predicado testa falha no seu todo.

```
   Extensão do predicado que testa uma lista de termos

testa( [] ).
testa( [H|T] ) :-
    H,
    testa(T).
```

Predicado "Exceção"

```
   Extensao do predicado excecao: Termo -> {V,F}

+excecao( T ) :: ( solucoes( T,excecao( T ),S ),
                   comprimento( S,L ),
                   L == 1 ).
```

Predicado "Nulo"

```
   Extensao do predicado nulo: Termo -> {V,F}

+nulo( T ) :: ( solucoes( T,nulo( T ),S ),
                comprimento( S,L ),
                L == 1 ).

-nulo( T ) :: ( solucoes( T,nulo( T ),S ),
                comprimento( S,L ),
                L == 1 ).
```

Predicado "Não"

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

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

Inserção de Conhecimento

O predicado evolução utiliza, então, os predicados `soluções`, `inserção` e `testa`.

```
Extensão do predicado que permite a evolução de conhecimento
```

```
evolucao( Termo ) :-
    solucoes( INV,+Termo::INV,LINV ),
    insercao( Termo ),
    testa( LINV ).
```

Este realiza a **inserção**, por via do `assert`, do termo que se pretende adicionar. No caso de ocorrer falha do teste dos invariantes no predicado evolução, o mecanismo do Prolog leva a que se remova o termo adicionado através do `retract`. Assim se consegue manter a consistência da Base de Conhecimento.

```
Extensão do predicado que permite a inserção de conhecimento
```

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

Remoção de Conhecimeto

Para a remoção de conhecimento, deve ser utilizado o predicado designado de `involução`. Este predicado tem em conta todos os invariantes associados ao termo que se pretende remover, determinando, posteriormente, se a sua remoção provoca a quebra desses invariantes. Além disso, e antes da "tentativa" de remoção, é verificado se de facto o termo a remover existe ou não na Base de Conhecimento. Se o termo não se verificar, o processo quebra sem sequer haver o teste dos invariantes. Caso contrário, o processo segue a metodologia já descrita. Para a correta implementação deste predicado evolução, são novamente utilizados os predicados `solucoes, insercao e testa`.

```
Extensão do predicado que permite a involução do conhecimento
```

```
involucao( Termo ) :-
    solucoes( INV,-Termo::INV,LINV ),
    Termo,
    remocao( Termo ),
    testa( LINV ).
```

Tal como no predicado relativo à inserção, também o predicado da remoção tem que voltar a inserir o termo caso os invariantes sejam quebrados. Vem então a seguinte definição:

```
   Extensão do predicado que permite a remoção de conhecimento

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

Evolução de uma lista de termos

```
   Extensão do predicado que permite a evolução de uma
   lista de Termos

insertAll( [] ).
insertAll( [H|T] ) :-
    assert( H ),
    insertAll( T ).
```

Involução de uma lista de termos

```
   Extensão do predicado que permite a involução de uma
   lista de Termos

removeAll( [] ).
removeAll( [H|T] ) :-
    retract( H ),
    removeAll( T ).
```

Predicado "Comprimento"

Este predicado vai nos ser útil em alguns dos predicados tais como `inserirInterdito` e `evoluirConhecimento`.

```
   Predicado «comprimento» que calcula o número de elementos
   existentes numa lista

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

Predicado "Concatenação"

O predicado `concat` tem como objetivo a concatenação de duas listas numa lista resultado, conforme o exemplo.

```
   Lista 1, Lista 2, Lista resultado → {V, F}

   Extensao do predicado concat: Lista1, Lista2, R -> {V,F}

concat( [],L,L ).
concat( [X|Xs],L2,[X|L] ) :-
    concat( Xs,L2,L ).
```

Predicado "Pertence"

O predicado pertence tem como objetivo determinar a existência de um dado elemento numa lista, conforme os exemplos.

```
   Extensao do predicado pertence: Elemento, Lista -> {V,F}
```

```
pertence( X,[X|L]).
pertence( X,[Y|L] ) :-
    X \= Y,
    pertence( X,L).
```

Predicado "Remove Utente"

```
   Extensao do predicado removeUtente: IDU -> {V,F}
```

```
removeUtente( ID ) :-
    solucoes( utente( IDU,NU,DU,SU ),utente( ID,NU,DU,SU ),LUT ),
    comprimento( LUT,L ),
    L > 0,
    removeAll( LUT ).
```

Predicado "Remove Ato"

```
   Extensao do predicado removeAto: IDA -> {V,F}
```

```
removeAto( ID ) :- solucoes( ato( IDA,DA,IDU,IA,CA,PLA,PA),

                ato( ID,DA,IDU,IA,CA,PLA,PA),LA ),removeAto( ID ):
                comprimento( LA,L ), L > 0,
                removeAll( LA ).
```

"Predicados Não Existe"

Tal como se pode querer saber se um determinhado utente ou ato foi realizado pode-se ver da perpetiva contrária e ver se este não existe. Este predicado daria os resultados opostos aos obtidos no predicado `existe`

```
   Extensao do predicado naoExiste: ID -> {V,F}
```

```
naoExiste(utente(ID,N,D,S)) :-
        solucoes( ID, utente(ID,NU,DU,SU), L),
    comprimento( L,S ),
            S == 0.

naoExiste(ato(IDA,D,IDU,I,C,PL,P)):-
        solucoes( IDA, ato(IDA,DA,IDUA,IA,CA,PLA,PA), L),
    comprimento( L,S ),
            S == 0.
```

Predicado "Increment"

No código utilizado são utilizados 3 predicados `Increment`, um para o ato, um para o utente e outro para XPTO. Ao estudar por exemplo o caso do predicado relacionado com o Utente temos primeiramente estabelecido que os 17 primeiros Ids já foram usados. Este predicado vai retirar este valor da base de conhecimento atravês do `retract` e vai criar uma nova variável com mais uma unidade que vai ser inserida novamente na base e diz sempre que requesitado o valor de Id disponível para inserção através do predicado `getIncIDU`.

```
counter_idu(17).
increment( idu ) :-
    retract( counter_idu( C ) ),
    C1 is C + 1,
    assert( counter_idu( C1 ) ) .
```

Predicado "GetInc"

```
getIncIDU( IDU ) :-
    increment( idu ),
    counter_idu( IDU ).
```

4. Inserção e Evolução do conhecimento

Perante o contexto da programação em lógica estendida, existe agora a possibilidade de coexistir simultaneamente conhecimento perfeito e imperfeito na Base de Conhecimento do sistema. Desta forma, passa a ser possível inserir conhecimento imperfeito na Base de Conhecimento, continuando a garantir a integridade da mesma. Para estas duas novas necessidades, foram criados novos predicados de evolução: o predicado `evoluirConhecimento` quando se quiser evoluir de conhecimento imperfeito para perfeito, os predicados `inserirIncerto`, `inserirImpreciso` e `inserirInterdito` para inserir novo conhecimento imperfeito e `inserirPositivo` e `inserirNegativo` quando a intenção for a de inserir conhecimento perfeito.

Estes mecanismos de transição entre os dois tipos de conhecimento foram implementados predicados representativos para as entidades do sistema: utente e ato.

4.1 Evolução de conhecimento perfeito a partir de conhecimento imperfeito

De seguida vamos apresentar as formas através das quais conseguimos a evolução de conhecimento perfeito partindo de conhecimento imperfeito já existente na Base de Conhecimento.

Tanto o conhecimento **incerto** como o **impreciso** podem estar associados a um dos diferentes argumentos de um predicado do sistema de saúde, existe uma declaração do predicado `evoluirConhecimento` para cada caso específico. Por exemplo, no caso de se pretender evoluir conhecimento imperfeito sobre um utente que já exista na Base de Cconhecimento, os dados desconhecidos poderão estar associados ao valor do nome, idade ou sexo do respetivo utente.

Começando por um exemplo em que não se sabe o sexo de um certo utente. A dada altura, pode ser possível, por diversos motivos, conhecer este valor e querer atualizar os dados incompletos existentes na Base de Conhecimento. Para implementar esta funcionalidade foi criado o predicado `evoluirConhecimento`, que no seu único argumento recebe o predicado com conhecimento perfeito que se pretende adicionar à Base de Conhecimento.

```
%partir de conhecimento imperfeito incerto
evoluirConhecimento( utente( IDU,N,data( D,M,A),S ) ) :-
        demo(utente( IDU,N,data( D,M,A),S ),desconhecido),
    solucoes( (excecao( utente( IDU,N,data( D,M,A),S ) ) :-
    utente( IDU,N,data( D,M,A),X )),(utente( ID,N,data( D,M,A ),X ),
    nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeUtente( IDU ),
    evolucao( utente( IDU,N,data( D,M,A),S ) ).
```

Por outro lado, pode-se ter situações em que se parte de conhecimento imperfeito impreciso, ou seja, onde se pode chegar a uma solução apenas com os intervalos de soluções possíveis.

```
%partir de conhecimento imperfeito impreciso
evoluirConhecimento( utente( IDU,N,data( D,M,A),S )) :-
        demo(utente(IDU,N,data( D,M,A ),S),desconhecido),
    solucoes( ((excecao( utente( IDU,N,data(D,M,X ),S) ) :-
    X >= LI, X =< LS)),(excecao( utente( IDU,N,data( D,M,A ),S) )),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    evolucao( utente( IDU,N,data( D,M,A ),S ) ).
```

O predicado começa por testar se existe efetivamente conhecimento imperfeito sobre o utente, isto é, se a informação que se pretende evoluir na Base de Conhecimento é para o sistema de inferência uma informação desconhecida. De seguida, é agrupado numa lista o conjunto de exceções associados ao termo a evoluir . Para tal é utilizando predicado `soluções`, em que a condição é suficientemente capaz de unificar com as exceções pretendidas.

Tem de se ter em atenção o facto de as exceções apanhadas pelo predicado soluções não poderem ser relativas a conhecimento interdito. Para isso é testado se o valor xpto é **não-nulo**, condição que se verifica nos casos de conhecimento interdito. Todos os elementos da lista de exceções são removidos. Para tal é usado um predicado auxiliar, `removeAll`. Depois de removidas as exceções basta remover, no caso do conhecimento imperfeito do tipo incerto, o predicado anterior e inserir o novo. No caso do conhecimento impreciso tal não é necessário, pois apenas existem exceções a remover (uma para cada valor possível). De notar que a remoção de todas as exceções da lista e do antigo termo (a existir) e a inserção do novo só poderá ser feita se o tamanho dessa lista for efetivamente maior do que zero. Caso contrário, não houve unificação com nenhuma exceção e, portanto, a cláusula do predicado `evoluirConhecimento` não poderá inserir o novo termo.

4.2 Inserção de Conhecimento Imperfeito

Como já referido, no contexto da programação em lógica estendida, passa a ser possível a inserção de conhecimento do tipo imperfeito. Por se estar a lidar com valores nulos do tipo desconhecido, cláusulas com informações incompletas ou parâmetros indeterminados que queiram ser adicionadas à Base de Conhecimento, não podem ser diretamente inseridas. Para realizar este processo de inserção de

conhecimento imperfeito com as respetivas limitações e exceções associadas, foram implementados predicados associados a cada tipo de valor nulo. Estes novos predicados foram implementados para cada um dos predicados desenvolvidos para o sistema de saúde. Para a implementação destes predicados, para a inserção de novo conhecimento, foram adicionadas 4 funcionalidades ao sistema que permitem a geração automática de identificadores (#IdUtente, #IdAto e #IdMarcador) e de valores nulos (xpto). Os predicados `getIncXPTO`, `getIncIDU`, `getIncIDA` e `getIncIDM` permitem então obter e incrementar, de forma sequencial, um novo valor.

Tal permite garantir maior consistência na Base de Conhecimento e também maior controlo das entidades nela presentes. É, contudo, importante referir que para adicionar um determinado termo, terá que ser o utilizador a determinar qual o próximo identificador disponível, e passá-lo ao predicado de inserção.

4.2.1 Evolução de Conhecimento Incerto

Para a adição de conhecimento imperfeito do tipo incerto, foi criado o predicado `inserirIncerto` que recebe no primeiro argumento um termo com unicamente os parâmetros da qual se conhece o valor e no segundo argumento uma opção que indica qual é o campo do termo a inserir que se desconhece.

O processo de inserção de conhecimento incerto começa por determinar o valor que representará o parâmetro desconhecido (o XPTO). É de seguida usado o predicado evolução para **inserir** o termo na Base de Conhecimento, sendo que o valor de XPTO será utilizado em lugar do parâmetro que desconhecemos, tendo este sido indicado na opção do segundo argumento. Por se tratar de conhecimento desconhecido do tipo incerto, é ainda inserida na base de conhecimento uma exceção, associada ao termo que se acabou de inserir, com o valor XPTO gerado .

```
  Exemplo de inserção Conhecimento Incerto

inserirIncerto( utente( IDU, data( D,M,A ),S ),nome ) :-
   getIncXPTO( XPTO ),
   evolucao( utente( IDU,XPTO, data( D,M,A ),S ) ),
   evolucao( (excecao( utente( IDUT,NU, data( DU,MU,AU ),SU ) ):-
   utente( IDUT,XPTO, data( DU,MU,AU ),SU ) )).
```

4.2.2 Evolução de Conhecimento Impreciso

Para inserir conhecimento imperfeito do tipo impreciso, foi implementado o predicado `inserirImpreciso`. Recordando que um conhecimento desconhecido impreciso é caracterizado por estar definido num conjunto ou intervalo de valores, do qual se desconhece qual o valor que efetivamente toma, o predicado `inserirImpreciso` recebe como único argumento o termo a inserir na base de conhecimento, onde um dos parâmetros deste termo será uma lista, no caso de pretender especificar um conjunto de valores, ou na forma [LI - LS], caso se pretenda especificar um intervalo de valores onde LI é o limite inferior do intervalo e LS o limite superior. Em qualquer um dos casos, este parâmetro representa o valor desconhecido do termo a inserir.

O seguinte exemplo procura demonstrar a situação em que, na inserção de um utente sobre o qual desconhecemos a sua pressão máxima, sabemos quais os possiveis valores em que este valor se encontra. No local do parâmetro Pressão Máxima é passada a lista com as designações dos valores que o utente pode ter.

Como o conhecimento imperfeito do tipo impreciso é representado por um conjunto de exceções associadas ao conjunto de valores que o valor desconhecido pode tomar, o predicado `inserirImpreciso` insere de forma recursiva essas mesmas exceções.

```
inserirImpreciso(ato( IDA,data( D,M,A ),ID,I,C,PL,pressao(MIN,[MAX|MAXS]))):-
  evolucao( excecao( ato( IDA,data( D,M,A ),ID,I,C,PL,pressao( MIN,MAX )) )),
  inserirImpreciso( ato( IDA,data( D,M,A ),ID,I,C,PL,pressao( MIN,MAXS ) )).
```

4.2.3 Evolução de conhecimento interdito

A adição de conhecimento do tipo interdito, é feita recorrendo ao predicado `inserirInterdito`, que recebe como argumentos o termo a inserir com as informações completas, e uma opção que indica qual o parâmetro do termo que se pretende manter interdito. A forma como se adiciona conhecimento interdito é semelhante ao modo como se insere conhecimento incerto, com o acréscimo de ser marcado como nulo o valor interdito na Base de Conhecimento e ainda o facto de ser inserido um invariante que não permita a inserção futura de conhecimento que contrarie a interdição pretendida. Tome-se como exemplo um caso presente na Base de Conhecimento, onde a interdição se refere ao nome do utente:

```
inserirInterdito( utente( ID,D,S ),nome ) :-
    getIncXPTO( XPTO ),
    evolucao( utente( ID,XPTO,D,S ) ),
    evolucao( (excecao( utente( IDU,NU,DU,SU ) ) :- utente( IDU,XPTO,DU,SU ))
    evolucao( nulo( XPTO ) ),
    evolucao( (+utente( IDU,NU,DU,SU ) :: (solucoes( X,(utente( ID,X,_,_ )
                                ,nao( nulo( X ) )),S ),comprimento( S,L ),
                                          L == 0 )) ).
```

4.3 Inserção de conhecimento perfeito

A inserção de conhecimento perfeito faz-se de forma mais simples do que aquela com que é inserido o restante conhecimento. Para inserir conhecimento perfeito positivo e negativo foram criados dois predicados: `inserirPositivo` e `inserirNegativo`. Ambos recebem um termo geral a inserir e testam se é possível a inserção antes de fazer evolução do mesmo.

Os predicados testam se o termo ainda não existe na Base de Conhecimento, verificando a resposta do predicado demo (a apresentar de seguida) e se não existem já exceções. Além disso, é verificado se a inserção não leva a contradição de informação, isto é, se não existe já algum termo na Base de Conhecimento que contradiga a nova informação.

```
inserirPositivo( Termo ) :-
        demo(Termo,falso),
        nao(existeE(Termo)),
        naoExisteNT(-Termo),
           naoExiste(Termo),
    evolucao( Termo ).
```

```
inserirNegativo( -Termo ) :-
        demo(Termo,falso),
        nao(existeE(Termo)),
        naoExiste(Termo),
    naoExisteNT(-Termo),
    evolucao(-Termo).
```

5. Interpretação de Valores Nulos

Após terem sido abordados os diferentes tipos de valores nulos e o respetivo conhecimento associado a cada um, procede-se à implementação do mecanismo de inferência destes valores.

Desta forma, criou-se um novo predicado: `demo` habilitado a representar conhecimento no contexto de Programação em Lógica Estendida. Este é capaz de interpretar questões e responder com Verdadeiro ($\mathbb{V}$), Falso ($\mathbb{F}$) ou Desconhecido ($\mathbb{D}$).

```
Extensao do predicado demo: Questao, Resposta -> {V,F}
```

Tal como se encontra representado acima, este predicado recebe dois parâmetros como argumento: o primeiro é referente à questão a colocar ao sistema de inferência e o segundo representa a resposta a essa questão. O contradomínio do predicado `demo` terá como valores ($\mathbb{V}$) ou ($\mathbb{F}$) e terá como resposta os valores já apresentados: ($\mathbb{V}$), ($\mathbb{F}$) ou ($\mathbb{D}$).

Assim, a extensão do predicado `demo` na linguagem Prolog é a seguinte:

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

A primeira cláusula indica que a resposta de uma determinada questão terá o valor de **'Verdadeiro'** se existirem provas na Base de Conhecimento que afirmam que a questão (Q) é verdadeira.

A segunda cláusula, pelo mesmo método, afirma que uma questão (Q) é falsa se existir informação explícita na Base de Conhecimento que mostre que a questão é falsa, ou quando não existir nenhuma exceção que prove que se trata de conhecimento imperfeito. Assim, admite-se que uma questão tomará valor de **'Falso'** se for possível encontrar uma prova de -Q.

Por fim, se não existir provas que a questão Q é verdadeira ou falsa, então a resposta terá o valor de **'Desconhecido'**.

Assim, o predicado `demo` pode ser utilizado para dar resposta às questões colocadas ao sistema de inferência, tendo em consideração modificações da Base de Conhecimento, relacionadas com a existência de conhecimento imperfeito.

5.1 Adaptação do interpretador aos operadores lógicos

Dado ser necessário relacionar dois predicados diferentes, criou-se um novo programa denominado `demo2`:

```
Extensao do predicado demo2: (Questao1 Questao2), Resposta -> {V,F}
```

Este recebe no primeiro argumento um tuplo de questões (Questão 1, Questão 2) separadas pelo operador lógico que lhes é aplicado, sendo este: conjunção (`and`), disjunção (`or`), disjunção exclusiva (`xor`) ,implicação ($\Rightarrow$)ou equivalência(<=>).

Em seguida é apresentada a tabela de valores lógicos que relaciona o par de questões efetuadas que o interpretador deve receber e a resposta que deve ser obtida.

5.1.1 Conjunção de predicados

$Q1$	$Q2$	$Q1 \wedge Q2$
$\mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{V}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{V}$	$\mathbb{D}$	$\mathbb{D}$
$\mathbb{F}$	$\mathbb{V}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{D}$	$\mathbb{F}$
$\mathbb{D}$	$\mathbb{V}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{D}$	$\mathbb{D}$	$\mathbb{D}$

5.1.2 Disjunção de predicados

$Q1$	$Q2$	$Q1 \vee Q2$
$\mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{V}$	$\mathbb{F}$	$\mathbb{V}$
$\mathbb{V}$	$\mathbb{D}$	$\mathbb{V}$
$\mathbb{F}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{D}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{D}$	$\mathbb{F}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{D}$	$\mathbb{D}$

5.1.3 Disjunção exclusiva de predicados

$Q1$	$Q2$	$Q1 \overline{\vee} Q2$
$\mathbb{V}$	$\mathbb{V}$	$\mathbb{F}$
$\mathbb{V}$	$\mathbb{F}$	$\mathbb{V}$
$\mathbb{V}$	$\mathbb{D}$	$\mathbb{D}$
$\mathbb{F}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{D}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{V}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{F}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{D}$	$\mathbb{D}$

5.1.4 Implicação de predicados

$Q1$	$Q2$	$Q1 \rightarrow Q2$
$\mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{V}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{V}$	$\mathbb{D}$	$\mathbb{D}$
$\mathbb{F}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{V}$
$\mathbb{F}$	$\mathbb{D}$	$\mathbb{V}$
$\mathbb{D}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{D}$	$\mathbb{F}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{D}$	$\mathbb{D}$

5.1.5 Equivalência de predicados

$Q1$	$Q2$	$Q1 \Leftrightarrow Q2$
$\mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{V}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{V}$	$\mathbb{D}$	$\mathbb{D}$
$\mathbb{F}$	$\mathbb{V}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{V}$
$\mathbb{F}$	$\mathbb{D}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{V}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{F}$	$\mathbb{D}$
$\mathbb{D}$	$\mathbb{D}$	$\mathbb{D}$

5.2 Adaptação do interpretador a uma lista de questões

De modo a responder em simultâneo a um conjunto de questões, desenvolveu-se um programa que recebe como parâmetros duas listas: a lista de questões a testar e a lista onde serão gurdadas as respostas às questões colocadas. Para tal, criou-se um predicado designado `demoLista` que, recorrendo ao predicado `demo`, testa cada uma das questões que constituem a primeira lista e coloca a respetiva resposta na lista criada para esse efeito. A resposta a cada questão será, como já visto, um dos seguintes valores: ($\mathbb{V}$), ($\mathbb{F}$) ou ($\mathbb{D}$).

```
Extensao do predicado demoLista: Lista, Lista resultado -> {V,F}
```

```
demoLista( [],[] ).
demoLista( [Q|Qs],[R|Rs] ) :-
    demo( Q,R ),
    demoLista( Qs,Rs ).
```

5.3 Adaptação do interpretador à conjunção de uma lista de questões

De forma a determinar o valor de verdade de uma lista de questões foi implementado o predicado `demoListaConj`. Este testa o valor de verdade de cada questão, recorrendo ao predicado `demo` e conjuga as respostas utilizando o predicado `demo2` com o operador da conjunção (`and`). A conjução das respostas terá como resposta um valor contido no conjunto ($\mathbb{V}$), ($\mathbb{F}$) ou ($\mathbb{D}$).

```
Extensao do predicado demoListaConj: Lista, Resultado -> {V,F}
```

```
demoListaConj( [Q],R ) :- demo( Q,R ).
demoListaConj( [Q|Qs],R ) :-
    demo( Q,R1 ),
    demoListaConj( Qs,R2 ),
    demo2((R1 and R2),R).
```

6. Exemplos e Análise de Resultados

Nesta secção são apresentados exemplos práticos que demonstram o funcionamento de alguns dos predicados criados, nomeadamente, do predicado `demo`, `demo2`, `inserirIncerto`, `inserirImpreciso`, `inserirInterdito`, `inserirPositivo` e `inserirNegativo` e `evoluirConhecimento`.

6.1 Testes ao interpretador Demo

I. **Objetivo:** Testar uma única cláusula para cada um dos predicados existentes na Base de Conhecimento.

i. O utente com *#IdUtente* 1 de nome Diogo Moura nasceu no dia no dia 25/3/2002?

Resposta: *verdadeiro*

Justificação: Trata-se de um facto que está inserido na Base de Conhecimento.

```
?-demo(utente(1,diogomoura,data( 25,3,2002 ),masculino), R).
R = verdadeiro ?
yes
```

ii. O ato com *#IdAto* 6 ocorrido no dia 31/08/2000 teve um valor de colesterol de 150?

Resposta: *falso*

Justificação: Trata-se de conhecimento perfeito negativo presente na Base de Conhecimento.

```
?-demo(ato( 6,data( 31,8,2000 ),8,43,150,70, pressao( 80,125
↪  )), R).
R = falso ?
yes
```

iii. O utente com o *#IdUtente* 13, de nome Maria João e do sexo feminino nasceu no dia 24/06/2005?

Resposta: *desconhecido*

Justificação: Trata-se de conhecimento imperfeito. Na Base de Conhecimento está explicitamente declarado tal facto.

```
?- demo(utente(13, mariajoao,data( 24,6,2005 ), feminino), R
↪  ).
R = desconhecido ?
yes
```

6.2 Testes ao interpretador demo2

I. **Objetivo:** Testar um par de predicados, que faz a junção dos respetivos valores de verdade através da conjunção lógica.

   i. Testar o utente com *#IdUtente* 2 chamado Bruna Matos nascida no dia 19/07/2002, e do sexo feminino, e, em simultâneo, testar se o *#IdAto* 6 realizado no dia 31/08/2002 com o *#IdUtente* 8 teve um valor de colesterol de 150.

   Resposta: *falso*

   Justificação: Resultou da conjunção lógica de uma proposição verdadeira com uma falsa, resultando numa proposição falsa.

```
?- demo2((utente(2,brunamatos,data(19,7,2002),feminino) and
 ↪  ato(6,data( 31,8,2000 ),8,43,150,70, pressao( 80,125
 ↪  ))),R).
R = falso ?
yes
```

   ii. Testar se o ato médico com *#IdAto* 1 ocorrido no dia 01/02/2010, pelo utente com *#IdUtente* 2, teve valores de colesterol, pulsação e pressão de 150, 60, (70/110), respetivamente e, em simultâneo, testar se o utente com o *#IdUtente* 13, Maria João, do sexo feminino e com a data de nascimento 24/06/2005 existe.

   Resposta: *desconhecido*

   Justificação: Resultou da conjugação de uma proposição lógica verdadeira com outra sobre a qual não se pode concluir nada.

```
?- demo2((ato(1,data(1,2,2010),2,20,150,60, pressao( 70,110
 ↪  )) and utente(13,mariajoao,data( 24,6,2005
 ↪  ),feminino)),R).
R = desconhecido ?
yes
```

II. **Objetivo:** Testar um par de predicados, que faz a junção dos respetivos valores de verdade através da disjunção lógica.

   i. Testar se o utente *#IdUtente* 12 chamada Anastacia Goncalves nasceu no dia 12/10/1995, ou, se o ato com o *#IdAto* 6, realizado no dia 31/08/2002 com o *#IdUtente* 8 teve um valor de colesterol de 150.

   Resposta: *falso*

   Justificação: A disjunção lógica de duas preposições falsas, resulta numa proposição falsa.

```
?- demo2((utente(12,anastaciagoncalves, data( 12,10,1995
 ↪  ),feminino) or ato(6,data( 31,8,2000 ),8,43,150,70,
 ↪  pressao( 80,125 ))),R).
R = falso ?
yes
```

ii. Testar se o ato médico com *#IdAto* 1 ocorrido no dia 01/02/2010, pelo utente com *#IdUtente* 2, teve valores de colesterol, pulsação e pressão de 150, 60, (70/110), respetivamente ou,se o utente com o *#IdUtente* 13, Maria João, do sexo feminino e com a data de nascimento 24/06/2005 existe.

Resposta: *verdadeiro*

Justificação A disjunção lógica de uma proposição verdadeira com uma desconhecida resulta numa proposição com o valor de verdade verdadeiro.

```
?- demo2((ato(1,data(1,2,2010),2,20,150,60, pressao( 70,110
 ↪  )) or utente(13, mariajoao, data(
 ↪  24,6,2005,feminino))),R).
R = verdadeiro ?
yes
```

III. **Objetivo:** Testar um par de predicados, que faz a junção dos respetivos valores de verdade com o operador lógico disjunção exclusiva.

i. Testar se o ato médico com *#IdAto* 2 ocorrido no dia 15/07/2005, pelo utente com *#IdUtente* 4, teve valores de colesterol, pulsação e pressão de 155, 65, (80/120), respetivamente, ou, se o utente com *#IdUtente* 1, chamado Diogo Moura nasceu no dia 25/03/2002.

Resposta: *falso*

Justificação: A disjunção exclusiva lógica de duas proposições verdadeiras é falsa.

```
?- demo2((ato(2,data(15,7,2015),4,22,155,65, pressao( 80,120
 ↪  )) xor utente(1,diogomoura, data(
 ↪  25,3,2002),masculino)),R).
R = falso ?
yes
```

IV. **Objetivo:** Testar um par de predicados fazendo a junção dos valores de verdade com o operador lógico implicação.

i. Testar se o ato médico com *#IdAto* 1 ocorrido no dia 01/02/2010, pelo utente com *#IdUtente* 2, teve valores de colesterol, pulsação e pressão de 150, 60, (70/110), respetivamente, implica que o utente *#IdUtente* 11 de nome Manuel Costa, nascido em 25/11/2002 é do sexo masculino.

Resposta: *falso*

Justificação: A primeira proposição tem o valor de verdade *verdadeiro* e a segunda o valor de verdade *falso*, logo valor de verdade da implicação é *falso*.

```
?- demo2((ato(1,data(1,2,2010),2,20,150,60, pressao( 70,110
 ↪  )) => utente(11, manuelcosta, data(
 ↪  25,11,2002),masculino)),R).
R = falso ?
yes
```

V. **Objetivo:** Testar um par de predicados fazendo a junção dos valores de verdade com o operador lógico equivalência.

   i. O ato médico com *#IdAto* 2 ocorrido no dia 15/07/2005, pelo utente com *#IdUtente* 4, teve valores de colesterol, pulsação e pressão de 155, 65, (80/120), respetivamente, se e só se, se o utente *#IdUtente* 12 chamada Anastacia Goncalves nasceu no dia 12/10/1995.

   Resposta: *falso*

   Justificação: A equivalência lógica de duas proposições com valores de verdade distintos tem o valor de verdade *falso*.

```
?- demo2((ato(2,data(15,7,2015),4,22,155,65, pressao( 80,120
 ↪  ))<=> (utente(12,anastaciagoncalves, data( 12,10,1995
 ↪  ),feminino))),R).
X = analises,
R = falso ?
yes
```

6.3 Inserção Conhecimento Imperfeito

Conforme descrito na secção deste relatório, foram criados os predicados `inserirIncerto`, `inserirImpreciso` e `inserirInterdito` que permitem a inserção de Conhecimento Imperfeito na Base de Conhecimento.

I. **Objetivo:** Testar a inserção de conhecimento incerto.

Não se sabe a data de nascimento do utente António Silva do género masculino.

Resultado:

```
?- getIncIDU(X).
X = 18 ?
yes

?- inserirIncerto(utente(18,antoniosilva,masculino),data).
yes

?- listing(utente).
utente( 1,diogomoura,data( 25,3,2002 ),masculino ).
utente( 2,brunamatos,data( 19,7,2002 ),feminino ).
utente( 3,joaolavinas,data( 28,8,2000 ),masculino ).
utente( 4,josepereira,data( 17,1,2000 ),masculino ).
utente( 5,carinamatos,data( 14,5,1999 ),feminino ).
utente( 6,manuelcosta,data( 13,2,1970 ),masculino ).
utente( 7,mariasantos,data( 16,2,1984 ),feminino ).
utente( 8,ricardofazeres,data( 5,2,1979 ),masculino ).
utente( 9,marianaamaro,data( 17,5,2001 ),feminino ).
utente( 10,joaomaria,data( 24,5,2003 ),masculino ).
utente( 13 ,mariajoao,xpto001,feminino ).
utente( 17,donaldtrump,data( 14,6,1946 ),xpto002 ).
utente( 18,antoniosilva,xpto1,masculino).
yes

?- listing(excecao).
excecao(utente(IDU, N, DN, S)) :-
    utente(IDU, N, xpto001, S).
excecao(utente(14, joanamatos, data(17, 3, 2008), feminino)).
excecao(utente(14, joanamatos, data(16, 3, 2008), feminino)).
excecao(utente(15, mauriciorodrigues, data(D, 5, 1994),
 ↪  masculino)) :-
    D>=5,
    D=<10.
excecao(utente(16, adrianamiranda, data(11, M, 1994),
 ↪  feminino)) :-
    M>=2,
    M=<10.
excecao(utente(17, antoniovariacoes, data(11, 3, A),
 ↪  masculino)) :-
    A>=1940,
    A=<1950.
excecao(utente(A, B, _, C)) :-
    utente(A, B, xpto1, C).
yes
```

II. **Objetivo:** Testar a inserção de conhecimento impreciso.

Não se sabe o dia de nascimento do utente Diogo Maria, com *#IdUt* 19 e do sexo masculino. Apenas se sabe que nasceu entre os dias 10 e 20 de Agosto de 1999.

Resultado:

```
?- getIncIDU(X).
X = 19 ?
yes

?- inserirImpreciso(utente(19,diogomaria,data([10-20],8,1999),
 ↪  masculino)).
yes


?- listing(excecao).

excecao(utente(IDU, N, DN, S)) :-
    utente(IDU, N, xpto001, S).
excecao(utente(14, joanamatos, data(17, 3, 2008), feminino)).
excecao(utente(14, joanamatos, data(16, 3, 2008), feminino)).
excecao(utente(15, mauriciorodrigues, data(D, 5, 1994),
 ↪  masculino)) :-
    D>=5,
    D=<10.
excecao(utente(16, adrianamiranda, data(11, M, 1994),
 ↪  feminino)) :-
    M>=2,
    M=<10.
excecao(utente(17, antoniovariacoes, data(11, 3, A),
 ↪  masculino)) :-
    A>=1940,
    A=<1950.
excecao(utente(A, B, _, C)) :-
    utente(A, B, xpto1, C).
excecao(utente(19, diogomaria, data(A, 8, 1999), masculino))
 ↪  :-
    -A>=10,
    A=<20
yes
```

III. **Objetivo:** Testar a inserção de conhecimento interdito.

Não se pode saber o nome do utente com *#IdUt* 20, do sexo masculino e nascido em 17/12/2000.

Resultado:

```
?- getIncIDU(X).
X = 20 ?
yes

?-
 ↪  inserirInterdito(utente(20,data(17,12,2000),masculino),nome).
yes

?- listing(excecao).

excecao(utente(IDU, N, DN, S)) :-
    utente(IDU, N, xpto001, S).
excecao(utente(14, joanamatos, data(17, 3, 2008), feminino)).
excecao(utente(14, joanamatos, data(16, 3, 2008), feminino)).
excecao(utente(15, mauriciorodrigues, data(D, 5, 1994),
 ↪  masculino)) :-
    D>=5,
    D=<10.
excecao(utente(16, adrianamiranda, data(11, M, 1994),
 ↪  feminino)) :-
    M>=2,
    M=<10.
excecao(utente(17, antoniovariacoes, data(11, 3, A),
 ↪  masculino)) :-
    A>=1940,
    A=<1950.
excecao(utente(A, B, _, C)) :-
    utente(A, B, xpto1, C).
excecao(utente(19, diogomaria, data(A, 8, 1999), masculino))
 ↪  :-
    -A>=10,
    A=<20
excecao(utente(A, _, B, C)) :-
    utente(A, xpto2, B, C).
```

7. Conclusão

Como já referido anteriormente, o objetivo deste trabalho prático consistiu no desenvolvimento de uma Base de Conhecimento, para que esta pudesse representar, não só o conhecimento perfeito, mas também o conhecimento imperfeito, podendo este ser impreciso, incerto ou interdito.

Ao longo da realização deste exercício, conseguiu-se compreender e organizar toda a informação presente na Base de Conhecimento, assim como gerar todos os predicados de forma natural.

Foram criados dois importantes predicados: o `demo` e o `demo2` que associa dois predicados, retornando um valor de verdade. Para tal, os operadores lógicos de junção de predicados que se utilizaram foram a conjunção, a disjunção, a disjunção exclusiva, a implicação e a equivalência.

Para inserir novos dados e evoluir a Base de Conhecimento, implementou-se alguns predicados como o `evolucao` e o `evoluirConhecimento`.

Desta forma, supõe-se que o objetivo do exercício foi cumprido, visto que foi desenvolvido um sistema de representação de conhecimento e raciocínio capaz de demonstrar as funcionalidades subjacentes à programação em lógica estendida e ao tratamento de conhecimento imperfeito.

Bibliografia

[1] CÉSAR ANALIDE, J. N. Antropopatia em entidades virtuais. *WORKSHOP OF THESIS AND DISSERTATIONS IN ARTIFICIAL INTELLIGENCE : proceedings, 1, Porto de Galinhas, Recife, Brazil* (2002).