Universidade do Minho
Departamento de Informática
Mestrado Integrado em Engenharia Informática

Relatório do Exercício 2

Programação em Lógica Estendida e Conhecimento Imperfeito

Sistemas de Representação de Conhecimento e Raciocínio

Grupo de trabalho 20
Ana Fernandes, A74321
Diogo Machado, A75399
Miguel Miranda, A74726
Rui Leite, A75551

Braga, 9 de Abril de 2017

Conteúdo

Resumo

Este segundo trabalho, associado à unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, tem como objetivo principal desenvolver e explorar outras formas de representação de conhecimento dentro da linguagem Prolog. Desta forma, e com o aprimorar de uma Base de Conhecimento para um contexto mais próximo da realidade, vamos passar a conseguir representar além do Conhecimento Perfeito, um novo tipo de conhecimento, o Conhecimento Imperfeito.

Para a capacidade de inferência sobre esta nova forma de conhecimento foi utilizado o conceito de valores nulos, com o objetivo de representar um tipo de valor de verdade não considerado até então: o "Desconhecido".

O presente relatório procura assim descrever e justificar todas as decisões tomadas e a forma como foram ultrapassadas as eventuais dificuldades na representação e raciocínio assentes no paradigma da programação em lógica estendida.

Introdução

Neste segundo exercício prático é usada a Programação em Lógica como forma de representação de Conhecimento e de Raciocínio. O contexto do exercício mantém-se em relação ao anterior, tomando-se como ponto de partida o exercício prático 1. Deste modo, são assumidos e utilizados todos os pressupostos e inferências anteriormente expostas para representar o Conhecimento Perfeito, apenas com algumas alterações a serem apresentadas em secções específicas deste relatório.

Por outra perspetiva, este exercício prático incide na representação do Conhecimento Imperfeito, onde o contradomínio é $(\mathbb{V}, \mathbb{F}, \mathbb{D})$, correspondendo, respetivamente, a "Verdadeiro", "Falso" e "Desconhecido" [4, 3]. Deste modo, é adaptada a Programação em Lógica usada anteriormente para a Programação em Lógica Estendida, onde passa a ser possível descrever informação do tipo incerta, imprecisa e interdita.

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 prestação de cuidados de saúde pela realização de atos médicos. As entidades que serão objeto de estudo são: **Instituições de Saúde**, **Utentes**, **Cuidados prestados** e **Atos médicos**.

Motivações e Objetivos

Este exercício tem como objetivo por em prática os conhecimentos obtidos na Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, nomeadamente conhecimentos na linguagem de Programação em Lógica, recorrendo a mecanismos de inferência para representar Conhecimento Imperfeito. Desta forma, é de interesse discutir e fazer conclusões sobre conhecimento que se encontra definido num domínio aberto.

Este tipo de conhecimento torna-se importante de exprimir, uma vez que se encontra mais próximo da realidade. 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;
- Manipular 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.

1. Representação de Conhecimento e Raciocínio

Neste exercício, tal como supracitado, será feita uma análise à representação do conhecimento e do raciocínio sobre duas perspetivas duais: o conhecimento perfeito e o conhecimento imperfeito. Embora representem domínios distintos, de uma maneira geral, estas duas perspetivas possuem representações muito similares, em que a principal diferença entre elas está associada aos pressupostos utilizados e, consequentemente, aos mecanismos de ilação e de prova que são utilizados para dar repostas às perguntas.

1.1 Conhecimento Perfeito

Na representação do conhecimento perfeito, no âmbito da Programação em Lógica, as cláusulas e os predicados possuem como contradomínio apenas dois valores de prova: “verdadeiro” e “falso”. Deste modo, são considerados os seguintes pressupostos [2, 3]:

- **Pressuposto do Mundo Fechado** - toda a informação que não existe mencionada na Base de Conhecimento é considerada falsa;
- **Pressuposto dos Nomes Únicos** - duas constantes diferentes, isto é, que definam valores atómicos ou objetos, designam, necessariamente, duas entidades diferentes no universo de discurso;
- **Pressuposto do Domínio Fechado** - não existem mais objetos no universo de discurso para além daqueles definidos na Base de Conhecimento.

1.2 Conhecimento Imperfeito

O Conhecimento Imperfeito advém, no âmbito da Programação em Lógica Estendida, de deixar de assumir que a informação representada é a única que é válida e que as entidades representadas sejam as únicas existentes no mundo exterior. Deste modo, passamos a aceitar pressupostos diferentes daqueles que são a base da representação de conhecimento perfeito. Estes pressupostos levam a adaptações da linguagem de programação, tanto a nível de representação simbólica como a nível da computação da informação.

Desta forma, são considerados os seguintes pressupostos [2, 3]:

- **Pressuposto do Mundo Aberto** - podem existir outros factos ou conclusões verdadeiros 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 esquemas de raciocínio não-monótono, isto é, mecanismos que funcionam adequadamente quando confrontados com informação incompleta ou em mudança [1], justificando-se pela consideração de pressupostos temporários, pela obtenção de conclusões plausíveis e pela flexibilização da evolução do conhecimento [2].

Um dos objetivos de estender a Programação em Lógica é o de permitir representar, explicitamente, informação negativa. Para tal, há a necessidade de passar a contar com dois tipos de negação [2]:

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

A definição destes dois tipos de negação justifica-se pela necessidade de diferenciar as situações que permitem concluir sobre o valor lógico de uma cláusula p, concretamente se aceitarmos que é falso por falta de prova ($não(p)$) ou se é possível provar que é efetivamente falso pela negação forte ($\neg p$).

Em suma, é considerado que o conjunto das respostas 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 conseguimos provar a resposta através de factos negativos;
- **Desconhecido** ($\mathbb{D}$): se $\exists x : não(q(x)) \wedge não(\neg q(x))$ então não é possível provar a resposta através dos factos e dos mecanismos de inferência.

2. Base de Conhecimento

Neste capítulo são apresentadas todas as etapas de resolução das propostas do exercício prático, bem como todas as decisões efetuadas durante o processo de resolução.

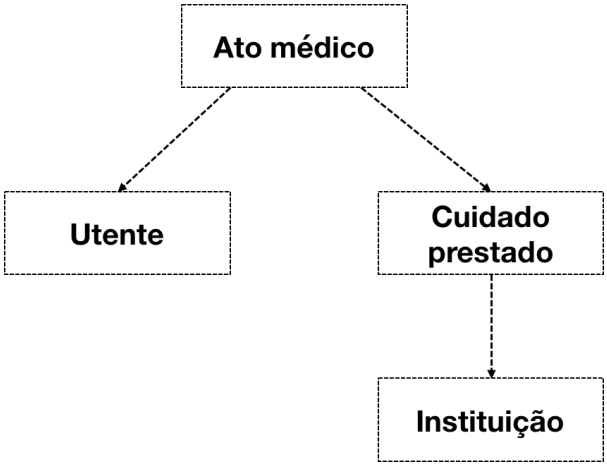

Figura 2.1: Mapa Lógico do Conhecimento

Tal como se pode ver na figura 2.1, a estrutura da base de conhecimento manteve-se relativamente ao primeiro exercício, com pequenas alterações a ser discutidas de seguida. Ela pode ser caracterizada da seguinte forma:

Utente: *#IdUt*, Nome, Idade, Morada;

Cuidado Prestado: *#IdServ*, Descrição, Instituição;

Ato Médico: *#IdA*, Data, *#IdUt*, *#IdServ*, Custo;

Instituição: Designação, Cidade.

As únicas alterações que se podem notar em relação à anterior Base de Conhecimento dizem respeito às entidades “Ato médico” e “Instituição”.

Na entidade ‘Ato médico” foi incluído um identificador (*#IdA*), como forma de reconhecer um ato médico de forma única e exclusiva. Tomou-se esta decisão pelo facto de não ser possível este reconhecimento único apenas com informação da data, do utente, do cuidado prestado e do custo do ato médico.

Anteriormente, a informação da “cidade” estava disponível na entidade “Cuidado Prestado”, porém chegou-se à conclusão que não faz sentido representar esta informação dessa forma, uma vez que é um atributo da “Instituição”.

2.1 Integridade da Base de Conhecimento

Por forma a manter a Base de Conhecimento o mais próximo da realidade, é necessário recorrer a mecanismos que permitam garantir a integridade da mesma. Para tal, foram criados predicados que controlem a inserção, remoção e consulta de informação na Base de Conhecimento. Estes predicados tiram partido do conceito de invariante, que podem ser definidos da seguinte forma:

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

Em Prolog existem predicados que permitem adicionar e remover factos na base de conhecimento, como o `assert` e o `retract`. Porém, apesar da existência destes, é necessário definir predicados auxiliares, uma vez que a simples utilização desses predicados não garante o controlo de integridade desejado na base de conhecimento.

Recorrendo a predicados auxiliares que auxiliem a inserção e remoção de conhecimento, é possível ter consistência e fiabilidade, graças ao uso de invariantes. Caso não sejam cumpridas todas as restrições impostas, a informação não é inserida/removida da base de conhecimento.

2.2 Invariantes

O controlo da inserção e da remoção de informação da Base de Conhecimento é feito recorrendo a invariantes. Deste modo, foram criados dois tipos de invariantes: estruturais, que não permitem a inserção de informação repetida, e invariantes referenciais, que não admitem que determinadas regras lógicas associadas ao domínio do conhecimento sejam quebradas.

2.2.1 Invariantes de Inserção

Os invariantes que estão associados ao processo de evolução da informação na base de conhecimento, isto é, de inserção de conhecimento, são os seguintes:

Cuidado prestado

Não é possível inserir cuidados repetidos (com a mesma descrição) para uma mesma instituição:

```
+cuidado( ID,D,I ) :: (solucoes( D,cuidado( _,D,I ),S ),
                       comprimento( S,L ),
                       L == 1).
```

A instituição mencionada no cuidado a inserir deve já existir na base de conhecimento:

```
+cuidado( ID,D,instituicao( I,L ) ) :: instituicao( I,L ).
```

Ato médico

Não é possível inserir atos com identificadores de "utente" não registados:

```
+ato( ID,D,IDU,IDS,C ) :: (solucoes( IDU,utente( IDU,_,_,_ ),S ),
                           comprimento( S,L ),
                           L == 1).
```

Não é possível inserir atos com identificadores de "cuidado" não registados:

```
+ato( ID,D,IDU,IDS,C ) :: (solucoes( IDS,cuidado( IDS,_,_ ),S ),
                           comprimento( S,L ),
                           L == 1 ).
```

2.2.2 Invariantes de Remoção

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

Utente

Não se remove um utente se estiver associado a algum ato médico:

```
-utente( ID,N,I,M ) :: (solucoes( ID,ato( _,_,ID,_,_ ),S ),
                        comprimento( S,L ),
                        L == 0).
```

Cuidado prestado

O cuidado prestado a remover não pode ter atos médicos associados e registados na base de conhecimento:

```
-cuidado( ID,D,I ) :: (solucoes( ID,ato( _,_,_,ID,_ ),S ),
                       comprimento( S,L ),
                       L == 0).
```

Instituição

A instituição a remover não pode ter cuidados associados e registados na base de conhecimento:

```
-instituicao( I,C ) :: (solucoes( I,cuidado(_,_,I,_),S ),
                        comprimento( S,L ),
                        L == 0).
```

2.3 Valores nulos

Nesta secção pretende-se fazer um estudo do tipo de valores mais comuns que podem surgir numa situação de informação incompleta. Esta identificação de valores é feita recorrendo ao conceito de valores nulos, que surgem como uma estratégia para a enumeração de casos para os quais se pretende fazer a distinção entre respostas a questões que devem ser concretizadas como "conhecidas" ("verdadeiras" ou "falsas') ou respostas concretizadas como "desconhecidas" [2].

As respostas podem ser desconhecidas por três razões: por serem relativas a uma questão de onde não se conhece efetivamente nenhum valor (valores desconhecidos e não necessariamente de um conjunto determinado de valores); 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, considerados, nomeadamente, na assimilação de informação na base de conhecimento) [2].

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`.

Como dito anteriormente, neste exercício o *Pressuposto do Mundo Fechado* foi abandonado. Porém, dado o contexto em que está a ser desenvolvido (o contexto deste problema), 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" $p(x)$.

$$\neg p(x) \quad \leftarrow \quad n\tilde{a}o\Big(p(x)\Big) \ \wedge \ n\tilde{a}o\Big(excecao\Big(p(x)\Big)\Big) \tag{2.1}$$

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

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 conjunto dado de valores

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 [2].

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 $x \in \{v1, v2\}$, através da seguinte sequência de exceções, uma para cada valor nulo do conjunto de valores possíveis:

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

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

2.3.3 Tipo não permitido

O terceiro género de valores nulos caracterizam, além de valores desconhecidos, um tipo de dados que não se pretende que surjam na base de conhecimento, isto é, o valor que representa não é permitido especificar ou conhecer (é interdito). [2].

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 não permitido, 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 atómicos 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, tal como referido na secção 1.1, será representado de forma explícita, i.e., são apresentados os factos acerca dos quais existe informação completa. Este tipo de conhecimento será tratado e implementado de forma equivalente à representação de conhecimento e raciocínio do trabalho prático anterior.

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( 1,diogo,21,braga ).
utente( 2,rui,20,braga ).
utente( 3,esm,21,prado ).
utente( 4,miguel,22,viana ).
utente( 5,joao,26,guimaraes ).
utente( 6,lisandra,25,fafe ).
utente( 7,paulo,24,braganca ).
```

Cada utente é caracterizado por um identificador único, por forma a evitar inconformidade e ambiguidade de informação.

Cuidado Prestado

Os cuidados inseridos na Base de Conhecimento são:

```
cuidado( 1,analises,instituicao( hpbraga,braga ) ).
cuidado( 2,tac,instituicao( hsjoao,porto ) ).
cuidado( 3,parto,instituicao( hpbraga,braga ) ).
cuidado( 4,ortopedia,instituicao( hviana,viana ) ).
cuidado( 5,doar-sangue,instituicao( hpbraga,braga ) ).
cuidado( 6,raioX,instituicao( hporto,porto ) ).
cuidado( 7,medicina-geral,instituicao( hporto,porto ) ).
cuidado( 8,parto,instituicao( hfaro,faro ) ).
cuidado( 9,ecografia,instituicao( hfaro,faro ) ).
cuidado( 10,quimioterapia,instituicao( hsjoao,porto ) ).
```

Instituição

A representação da informação relativa às instituições foi feita através de factos, tais como:

```
instituicao( hpbraga,braga ).
instituicao( hsjoao,porto ).
instituicao( hviana,viana ).
instituicao( hporto,porto ).
instituicao( hfaro,faro ).
```

Cada instituição é representada pelo nome da instituição e pela cidade correspondente.

Ato Médico

Recorreu-se ao predicado auxiliar `data`, já apresentado no exercício anterior, para representar a data em que ocorreu um ato. A representação da informação relativa aos atos foi feita através de factos, tais como:

```
ato( 1,data( 1,2,1996 ),3,3,10 ).
ato( 2,data( 15,3,2017 ),1,2,15 ).
ato( 3,data( 17,4,1997 ),4,4,5 ).
ato( 4,data( 15,3,2007 ),1,5,0 ).
ato( 5,data( 15,3,2007 ),2,5,0 ).
ato( 6,data( 15,3,2007 ),3,2,0 ).
```

```
ato( 7,data( 16,3,2017 ),5,6,12 ).
ato( 8,data( 16,3,2007 ),6,9,20 ).
ato( 9,data( 16,3,2007 ),3,1,40 ).
```

Decidiu-se caracterizar um ato com um identificador único, por forma a evitar inconformidade e ambiguidade de informação.

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 representados pelo `-functor`, representando a negação forte do predicado.

Utente

A representação da informação negativa relativa aos utentes foi feita através da negação de factos, tais como:

"O utente *manuel* com *#IdUt* 8, com 20 anos, nega ser de *bragança*."

```
-utente( 8,manuel,20,braganca ).
```

"O utente com o *#IdUt* 9, de nome *anastacia*, de *felgueiras*, nega ter 30 anos."

```
-utente( 9,anastacia,30,felgueiras ).
```

Cuidado Prestado

A representação da informação negativa relativa aos cuidados prestados foi feita através da negação de factos, tais como:

"A instituição *hfaro* não presta o cuidado *ortopedia* com *#IdServ* 11"

```
-cuidado( 11,ortopedia,instituicao( hfaro,faro ) ).
```

Instituição

A representação da informação negativa relativa à instituição foi feita através da negação de factos, tais como:
"Em *braga* não existe nenhuma instituição com o nome *hsjoao*."

```
-instituicao( hsjoao,braga ).
```

Ato Médico

A representação da informação negativa relativa aos atos médicos foi feita através da negação de factos, tais como:

"Sabe-se que o ato com *#IdA* 10, ocorrido no dia 31 de agosto de 2000, pelo utente *#IdUt* 3 e cuidado *#IdServ* 7, não teve um custo de 50 euros."

```
-ato( 10 ,data( 31,8,2000 ),3,7,50 ).
```

2.5 Conhecimento Imperfeito

O conhecimento imperfeito pode ser de três tipos: incerto, impreciso e interdito (ver secções 2.3.1, 2.3.2 e 2.3.3).

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 *#IdUt* 10, chamada *maria* e residente na *guarda*."

```
utente( 10,maria,xpto001,guarda ).

excecao( utente( IDU,N,I,M ) ) :-
    utente( IDU,N,xpto001,M ).
```

Cuidado Prestado

"É desconhecido o nome da instituição que presta o serviço com *#IdServ* 12 e designado por *pediatria*."

```
cuidado( 12,pediatria,xpto003 ).

excecao( cuidado( IdS,D,I ) ) :-
    cuidado( IdS,D,xpto003 ).
```

" É desconhecida a descrição do cuidado com o *#IdServ* 13, prestado no *hsjoao* no *porto*."

```
cuidado( 13,xpto004,instituicao( hsjoao,porto ) ).

excecao( cuidado( IdS,D,I ) ) :-
    cuidado( IdS,xpto004,I ).
```

Instituição

"Desconhece-se a localidade do *hsmaria*."

```
instituicao( hsmaria,xpto005 ).

excecao( instituicao( I,L ) ) :-
    instituicao( I,xpto005 ).
```

Ato Médico

"Desconhece-se a data em que foi prestado o ato médico com *#IdA* 11, realizado pelo utente com o *#IdUt* 6, relativo ao cuidado com *#IdServ* 2 e com um custo de 80 euros."

```
ato( 11,xpto006,6,2,80 ).

excecao( ato( ID,D,IDU,IdS,P ) ) :-
    ato( ID,xpto006,IDU,IdS,P ).
```

" Desconhece-se o custo do ato medico com ocorrido em 03-06-2007,com *#IdServ* 2 e *#IdUt* 6, identificado pelo *#IdA* 12."

```
ato( 12,data(3,6,2007 ),6,2,xpto007 ).

excecao( ato( ID,D,IDU,IDS,P ) ) :-
    ato( ID,D,IDU,IDS, xpto007).
```

"Desconhece-se o dia em que foi prestado o cuidado *#IdServ* 3 ao utente *#IdUt* 5, com um custo de 20 euros. Apenas se sabe que esse ato médico foi registado com *#IdA* 13 e que foi prestado em março de 2007."

```
ato( 13,data( xpto008,3,2007 ),5,3,20 ).

excecao( ato( ID,data( D,M,A ),IDU,IdS,P ) ) :-
    ato( ID,data( xpto008,M,A ),IDU,IdS,P ).
```

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*, utente com *#IdUt* 11 e com 22 anos de idade, mora em *braga* ou em *guimaraes*."

```
excecao( utente( 11,joana,22,braga ) ).
excecao( utente( 11,joana,22,guimaraes ) ).
```

"Não se sabe ao certo a idade do *mauricio*, com o *#IdUt* 12 e residente em *lisboa*. Apenas se sabe que possui entre 18 a 24 anos."

```
excecao( utente( 12,mauricio,I,lisboa ) ) :-
    I >= 18, I =< 24.
```

Cuidado Prestado

"O cuidado com o *#IdServ* 14, designado por *oftalmologia*, é prestado em uma das seguintes instituições: *hsjoao* ou *hporto*."

```
excecao(cuidado(14,oftalmologia,instituicao(hsjoao,porto))).
excecao(cuidado(14,oftalmologia,instituicao(hporto,porto))).
```

Instituição

" Não se sabe ao certo se a instituição *hsjose* se encontra localizada em *braga* ou no *porto*."

```
excecao( instituicao( hsjose,braga ) )
excecao( instituicao( hsjose,porto ) ).
```

Ato Médico

"O ato médico com *#IDA* 14 ocorrido em 03-06-2007, com o *#IdUt* 6 e *#IdServ* 9 custou entre 10 a 25 euros."

```
excecao( ato( 14, data(3,6,2007 ),6,9,P ) ) :-
    P >= 10, P =< 25.
```

" O ato médico com *#IdA* 15 ocorrido em 04-06-2007, com o *#IdUt* 7 e *#IdServ* 1 custou 20 ou 30 euros.

```
excecao( ato( 15,data(4,6,2007 ),6,9,20 ) ).
excecao( ato( 15,data(4,6,2007 ),6,9,30 ) ).
```

2.5.3 Conhecimento Imperfeito Interdito

O 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 *#IdUt* 13, de nome *trump*, com 70 anos, exige que não se saiba a sua morada."

```
utente( 13,trump,70,xpto002 ).

excecao( utente( IDU,N,I,L ) ) :-
    utente( IDU,N,I,xpto002 ).

nulo( xpto002 ).
```

Para garantir que o conhecimento é interdito, criou-se um invariante que não permite associar uma morada ao utente com identificador número 13. Tal situação pode acontecer se por exemplo existir necessidade de atualizar a informação do utente na Base de Conhecimento.

```
+utente(IDU,N,I,M)::(solucoes(X,(utente(13,_,_,X),nao(nulo(X))),S),
                     comprimento( S,N ),
                     N == 0).
```

Ato

"O utente com o *#IdUt* 13, usufruiu do cuidado com o *#IdServ* 10 em 01-04-2017, contudo pretende que não se saiba quanto pagou por este ato (identificado pelo *#IdA* 16)."

```
ato(16, data( 1,4,2017 ),13,10,xpto009 ).

excecao( ato( D,IDU,IdS,P ) ) :-
    ato( D,IDU,IdS,xpto009 ).

nulo( xpto009 ).
```

Por forma a garantir que a interdição é mantida, criou-se um invariante que não permite inserir custos ao ato médico com código de identificação 16:

```
+ato( ID,D,IDU,IdS,P ) :: (solucoes( P,( ato( 16,_,_,_,P ),
                            nao( nulo( P ) ) ),S ),
                            comprimento( S,N ),
                            N == 0).
```

3. Inserção e evolução de 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 para todos os predicados representativos das entidades do sistema: os predicados `utente`, `cuidado`, `instituicao` e `ato`.

3.1 Evolução de conhecimento perfeito a partir de conhecimento imperfeito

De seguida são apresentadas todas as decisões tomadas para conseguir implementar mecanismos que permitam a evolução de conhecimento perfeito partindo de conhecimento imperfeito já existente na base de conhecimento.

Como descrito anteriormente, o conhecimento incerto é representado por um valor nulo do tipo desconhecido, sobre o qual não se possui informações nem é possível determinar em que conjunto de valores poderá estar.

```
utente(10,maria,xpto001,guarda ).

excecao( utente( IDU,N,I,M ) ) :-
    utente( IDU,N,xpto001,M ).
```

Tomando por exemplo o caso em que para o utente com *#IdUt* 10, *maria* e residente em *guarda*, não se sabe qual é a sua idade. 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 e que se pretende adicionar à 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 conhecimento, os dados desconhecidos poderão estar associados ao valor do nome, idade ou morada do respetivo utente.

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 (sejam exceções sobre conhecimento incerto, como o caso do exemplo supracitado, ou sobre conhecimento impreciso). Para tal é utilizando predicado `solucoes` (já abordado no exercício anterior), 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 `solucoes` 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 (como já explicado anteriormente).

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.

3.2 Inserção de conhecimento imperfeito

Como já referido, no contexto da programação em lógica estendida, passa a ser possível a inserção conhecimento do tipo imperfeito. Por estarmos 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 três novos 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 (*#IdUt*, *#IdServ* e *#IdA*) e de valores nulos (*xpto*). Os predicados `getIncXPTO`, `getIncIDU`, `getIncIDS` e `getIncIDA` 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 passa-lo ao predicado de inserção.

3.2.1 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.

3.2.2 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 desconhecemos 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 cidade, sabemos quais as possíveis cidades em que este vive. No local do parâmetro *cidade* é passada a lista com as designações da cidades onde o utente pode morar.

```
insercaoImpreciso( utente( ID,N,I,[C|CS] ) )
```

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.

3.2.3 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 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 custo do ato *#IdA* 16:

```
ato( 16,data(1,4,2017),13,10,xpto009 ).

excecao( ato( ID,D,IDU,IdS,P ) ) :-
    ato( ID,D,IDU,IdS,xpto009 ).

nulo( xpto009 ).

+ato(ID,D,IDU,IdS,P)::(solucoes(P,(ato(16,_,_,_,P),nao(nulo(P))),S),
                       comprimento( S,N ),
                       N == 0).
```

3.3 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, verificado 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.

4. Interpretação de valores nulos

Apresentados os diferentes tipos de valores nulos e o respetivo conhecimento imperfeito associado a cada um, é necessário proceder à implementação de um mecanismo de inferência sobre estes valores.

Para tal, foi criado o predicado `demo` que, na prática, é um interpretador de questões capaz de representar conhecimento no contexto da Programação em Lógica Estendida, suportando as três respostas possíveis para qualquer questão: *verdadeiro* ($\mathbb{V}$), *falso* ($\mathbb{F}$) e *desconhecido* ($\mathbb{D}$).

```
demo: Questão, Resposta -> {V,F}
```

O predicado `demo` recebe como parâmetros dois argumentos: o primeiro é referente à questão a colocar ao sistema de inferência e o segundo representa a resposta a essa questão. Sendo este predicado uma extensão de um programa em lógica, o seu contradomínio terá o conjunto de valores {verdadeiro, falso} e terá como resposta um dos três valores possíveis já citados: $\mathbb{V}$, $\mathbb{F}$ ou $\mathbb{D}$.

O seguinte programa descreve assim a extensão do predicado `demo` na linguagem Prolog:

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

A primeira cláusula do interpretador de questões indica que a resposta a uma determinada questão terá o valor “verdadeiro” se existir informação explicita na Base de Conhecimento que permita provar a questão como verdadeira.

De forma semelhante, o predicado `demo` determinará que a resposta a uma questão tomará o valor de verdade “falso” quando for possível encontrar uma prova de $\neg Q$, ou seja, existir na base de conhecimento uma prova concreta que a questão é falsa ou quando não existir nenhuma exceção que prove que se trata de conhecimento imperfeito.

Se não existir prova de a que questão seja falsa e se não houver prova de que seja verdadeiro, então a resposta terá o valor de verdade “desconhecido”.

Com estas características, o predicado `demo` pode ser utilizado para dar resposta às questões feitas ao sistema tendo em consideração as alterações à base de conhecimento relacionadas com a existência de conhecimento imperfeito.

4.1 Adaptação do interpretador aos operadores lógicos

Para ser possível realizar a inferência sobre um par de questões, segundo um dos operadores lógicos de primeira ordem, foi desenvolvido um novo programa designado por `demo2`:

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

Este predicado recebe no primeiro argumento o tuplo de questões a inferir separadas pelo operador lógico que lhes será aplicado. Decidiu-se implementar este mecanismo para os operadores de conjunção `(and)`, de disjunção `(or)`, de disjunção exclusiva `(xor)`, de implicação `(=>)` e de equivalência `(<=>)`. Todos as cláusulas de suporte a este predicado podem ser consultadas no anexo deste relatório.

De seguida, para cada um dos operadores lógicos definidos, é apresentada a tabela de relação entre o valor lógico do par que questões que o interpretador deve receber e a respetiva resposta que se deve obter.

4.1.1 Conjunção de predicados

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

4.1.2 Disjunção de predicados

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

4.1.3 Disjunção exclusiva de predicados

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

4.1.4 Implicação de predicados

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

4.1.5 Equivalência de predicados

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

4.2 Adaptação do interpretador a uma lista de questões

Para responder em simultâneo a um conjunto de questões, foi desenvolvido um predicado que recebe como parâmetros a lista de questões a testar e a lista onde guardar as respostas às questões colocadas.

Foi criado o predicado `demoLista` que vai testar cada uma das questões existentes na lista do primeiro argumento e colocar a respetiva resposta na lista de respostas. A resposta a cada questão pode novamente ser um dos seguintes valores: $\mathbb{V}$, $\mathbb{F}$ ou $\mathbb{D}$.

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

Este predicado faz uso dos anteriores interpretadores para chegar a uma resposta.

4.3 Adaptação do interpretador à conjunção de uma lista de questões

Com a intenção de determinar o valor de verdade de uma lista de questões foi implementado o predicado `demoListConj`. Este predicado testa o valor de verdade de cada questão, recorrendo ao predicado `demo` apresentado anteriormente, e faz a conjugação das respostas usando o predicado `demo2` com o operador `and`, também apresentado anteriormente. A conjunção das respostas será um valor atómico e está novamente contido no conjunto $\mathbb{V}$, $\mathbb{F}$ ou $\mathbb{D}$.

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

5. 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`.

5.1 Testes ao interpretador demo

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

   i. O ato médico com *#IdA* 1 ocorrido no dia 01-02-1996, ao utente com *#IdUt* 3 e *#IdServ* 3 teve um custo de 10 euros?

   Resposta: *verdadeiro*

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

```
?- demo( ato(1, data( 1, 2, 1996), 3, 3, 10 ), R ).
R = verdadeiro ?
yes
```

   ii. É prestado o cuidado ortopedia na instituição *hfaro*, em Faro?

   Resposta: *falso*

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

```
?-demo(cuidado(11,ortopedia,instituicao(hfaro,faro)),R).
R = falso ?
yes
```

   iii. O utente com o *#IdUt* 10, de nome *maria* e residente na *guarda* possui 35 anos?

   Resposta: *desconhecido*

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

```
?- demo(utente(10, maria, 35, guarda), R ).
R = desconhecido ?
yes
```

5.2 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 ato médico com *#IdA* 1 ocorrido no dia 01-02-1996, pelo utente com *#IdUt* 3, *#IdServ* 3 e um custo de 10 euros, e, em simultâneo, testar se é efetuado o cuidado *ortopedia* na instituição *hfaro*, em *faro*.

   Resposta: *falso*

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

```
?- demo2((ato(1,data(1,2,1996),3,3,10) and
↪  cuidado(11,ortopedia,instituicao(hfaro, faro))),R).
R = falso ?
yes
```

   ii. Testar se o ato médico com *#IdA* 1 ocorrido no dia 01-02-1996, pelo utente com *#IdUt* 3, *#IdServ* 3 e um custo de 10 euros e, em simultâneo, testar se o utente com o *#IdUt* 10, com idade 35, de nome *maria* e residente na *guarda* 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(1,2,1996),3,3,10) and utente(10, maria,
↪  35, guarda)),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 é efetuado o cuidado ortopedia na instituição *hfaro*, em *faro*, ou, se o utente com o *#IdUt* 9, chamada *anastacia* de *felgueiras* tem 30 anos.

   Resposta: *falso*

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

```
?- demo2((cuidado(11, ortopedia, instituicao(hfaro,faro)) or
↪  utente(9, anastacia, 30, felgueiras)),R).
R = falso ?
yes
```

   ii. Testar se o ato médico com *#IdA* 1 ocorrido no dia 01-02-1996, pelo utente com *#IdUt* 3, *#IdServ* 3 custou 10 euros, ou, se o utente com o *#IdUt* 10, com idade 35, de nome *maria* reside na *guarda*.

   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(1,2,1996),3,3,10) or utente(10, maria,
↪  35, guarda)),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 *#IdA* 1 ocorrido no dia 01-02-1996, pelo utente com *#IdUt* 3, *#IdServ* 3 custou 10 euros, ou, se o utente com *#IdUt* 1, chamado *diogo* com 21 anos vive em *braga*, sendo que só uma delas pode acontecer.

   Resposta: *falso*

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

```
?- demo2((ato(1,data(1,2,1996),3,3,10) xor
↪  utente(1,diogo,21,braga)),R).
R = falso ?
yes
```

   ii. Testar se o ato médico *#IdA* 1 ocorrido no dia 01-02-1996, pelo utente com *#IdUt* 3, *#IdServ* 3 e um custo de 10 euros, ou, se é efetuado o cuidado com *#IdServ* 11 *ortopedia* na instituição *hfaro*, em *faro*, sendo que só uma delas pode acontecer.

   Resposta: *verdadeiro*

   Justificação: O primeiro predicado tem o valor de verdade *verdadeiro*, e o segundo o valor de verdade *falso*,logo a disjunção lógica tem o valor de verdade *verdadeiro*.

```
?- demo2((ato(1,data(1,2,1996),3,3,10) xor cuidado(11,
↪  ortopedia, instituicao(hfaro,faro))),R).
R = verdadeiro ?
yes
```

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

   i. Testar se o ato médico com *#IdA* 1 ocorrido no dia 01-02-1996, pelo utente com *#IdUt* 3, *#IdServ* 3 e um custo de 10 euros, implica que seja efetuado o cuidado ortopedia na instituição *hfaro*, em *faro*.

   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(1,2,1996),3,3,10) => cuidado(11,
↪  ortopedia, instituicao(hfaro,faro))),R).
R = falso ?
yes
```

   ii. Se for efetuado o cuidado *ortopedia* na instituição *hfaro*, em *faro*, então o utente com o *#IdUt* 10, com 35 anos, de nome *maria* e residente na guarda existe.

   Resposta: *verdadeiro*

   Justifição: O primeiro termo tem o valor de verdade *falso* e o segundo o valor de verdade *desconhecido*, tendo a implicação resultante o valor de verdade verdadeiro.

```
?- demo2((cuidado(11, ortopedia, instituicao(hfaro,faro)) =>
↪  utente(10, maria, 35, guarda)),R).
R = verdadeiro ?
yes
```

iii. Se é efetuado o cuidado *analises* na instituição *hpbraga*, em *braga*, então a instituição designada por *hpbraga*, localizada em *braga* tem que existir.

Resposta: *verdadeiro*

Justificação: Ambas as proposições tem o valor de verdade *verdadeiro*.

```
?- demo2(cuidado(1,analises,instituicao( hpbraga,braga)) =>
↪  instituicao(hpbraga,braga)),R).
R = verdadeiro ?
yes
```

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

i. É efetuado o cuidado com *#IdServ* 1 designado por *analises* na instituição *hpbraga*, em *braga*, se e só se, a instituição designada por *hpbraga*, localizada em *braga* existir.

Resposta: *verdadeiro*

Justificação: Ambas as proposições tem o valor de verdade *verdadeiro*, logo a sua equivalência lógica tem também o valor de verdade *verdadeiro*.

```
?- demo2((cuidado(1,analises,instituicao(hpbraga,braga)))<=>
↪  (instituicao(hpbraga,braga)),R).
R = verdadeiro ?
yes
```

ii. O ato médico com *#IdA* 15 ocorrido em 04-06-2007, com o *#IdUt* 7 e *#IdServ* 1 custou 20 euros, se e só se, o cuidado tiver sido prestado na instituição *hpbraga* em braga.

Resposta: *falso*

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

```
?- demo2((ato(15,data(4,6,2007),7,1,20))<=>
↪  (cuidado(1,X,instituicao(hpbraga,braga))),R).
X = analises,
R = falso ?
yes
```

5.3 Inserção Conhecimento Imperfeito

Conforme descrito na secção 3.2 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 idade do utente *manuel* de *viseu*.

Resultado:

```
?- getIncIDU(X).
X = 14 ?
yes

?- inserirIncerto(utente(14,manuel,viseu),idade).
yes

?- listing(utente).
utente(1, diogo, 21, braga).
utente(2, rui, 20, braga).
utente(3, esm, 21, prado).
utente(4, miguel, 22, viana).
utente(5, joao, 26, guimaraes).
utente(6, lisandra, 25, fafe).
utente(7, paulo, 24, braganca).
utente(10, maria, xpto001, guarda).
utente(13, trump, 70, xpto002).
utente(14, manuel, xpto1, viseu).
yes

?- listing(excecao).
excecao(utente(A,B,_,C)) :-
        utente(A, B, xpto001, C).
excecao(utente(11,joana,22,braga)).
excecao(utente(11,joana,22,guimaraes)).
excecao(utente(12,mauricio,A,lisboa)) :-
        A>=18,
        A=<24.
excecao(utente(A,B,C,_)) :-
        utente(A, B, C, xpto002).
excecao(utente(A,B,_,C)) :-
        utente(A, B, xpto1, C).
yes
```

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

O ato médico ocorrido em 08-04-2017, com *#IdUt* 1 e *#IdServ* 3 custou entre 10 a 30 euros.

Resultado:

```
?- getIncIDA(X).
X = 17 ?
yes

?- inserirImpreciso(ato(17,data(8,4,2017),1,3,[10-30])).
yes


?- listing(excecao).
excecao(ato(A,_,B,C,D)) :-
        ato(A, xpto006, B, C, D).
excecao(ato(A,B,C,D,_)) :-
        ato(A, B, C, D, xpto007).
excecao(ato(A,data(_,B,C),D,E,F)) :-
        ato(A, data(xpto008,B,C), D, E, F).
excecao(ato(14,data(3,6,2007),6,9,A)) :-
        A>=10,
        A=<25.
excecao(ato(15,data(4,6,2007),6,9,20)).
excecao(ato(15,data(4,6,2007),6,9,30)).
excecao(ato(A,B,C,D,_)) :-
        ato(A, B, C, D, xpto009).
excecao(utente(A,B,_,C)) :-
        utente(A, B, xpto1, C).
excecao(ato(17,data(8,4,2017),1,3,A)) :-
        A>=10,
        A=<30
yes
```

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

Não se pode saber em que instituição é prestado o cuidado médico descrito como *eutanásia*.

Resultado:

```
?- getIncIDS(X).
X = 15 ?
yes

?- inserirInterdito(cuidado(15,eutanasia),instituicao).
yes

?- listing(cuidado).
cuidado(1, analises, instituicao(hpbraga,braga)).
cuidado(2, tac, instituicao(hsjoao,porto)).
cuidado(3, parto, instituicao(hpbraga,braga)).
cuidado(4, ortopedia, instituicao(hviana,viana)).
cuidado(5, doar-sangue, instituicao(hpbraga,braga)).
cuidado(6, raioX, instituicao(hporto,porto)).
cuidado(7, medicina-geral, instituicao(hporto,porto)).
cuidado(8, parto, instituicao(hfaro,faro)).
cuidado(9, ecografia, instituicao(hfaro,faro)).
cuidado(10, quimioterapia, instituicao(hsjoao,porto)).
cuidado(12, pediatria, xpto003).
```

```
cuidado(13, xpto004, instituicao(hsjoao,porto)).
cuidado(15, eutanasia, xpto2).
yes

?- listing(nulo).
nulo(xpto002).
nulo(xpto009).
nulo(xpto2).
yes

?- listing(excecao).
excecao(cuidado(A,B,_)) :-
        cuidado(A, B, xpto003).
excecao(cuidado(A,_,B)) :-
        cuidado(A, xpto004, B).
excecao(cuidado(13,oftalmologia,instituicao(hsjoao,porto))).
excecao(cuidado(13,oftalmologia,instituicao(hporto,porto))).
excecao(cuidado(A,B,_)) :-
        cuidado(A, B, xpto2).
yes

?- listing.
+cuidado(_,A,B)::(solucoes(A,cuidado(_,A,B),C),
                  comprimento(C,D),D==1).
+cuidado(_,_,instituicao(A,B))::instituicao(A,B).
+cuidado(_,_,_)::(solucoes(A,(cuidado(14,_,A),nao(nulo(A))),B),
                  comprimento(B,C),C==0).
yes
```

5.4 Inserção de Conhecimento Perfeito

Conforme descrito na secção 3.3 deste relatório, foram criados os predicados `inserirPositivo` e `inserirNegativo` que permitem a inserção de Conhecimento Perfeito Positivo e Negativo na Base de Conhecimento.

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

O utente *pedro* tem 21 anos e reside em *vizela*.

Resultado:

```
?- getIncIDU(X).
X = 15 ?
yes

?- inserirPositivo(utente(15,pedro,21,vizela)).
yes

?- listing(utente).
utente(1, diogo, 21, braga).
utente(2, rui, 20, braga).
utente(3, esm, 21, prado).
utente(4, miguel, 22, viana).
utente(5, joao, 26, guimaraes).
utente(6, lisandra, 25, fafe).
utente(7, paulo, 24, braganca).
utente(10, maria, xpto001, guarda).
utente(13, trump, 70, xpto002).
utente(14, manuel, xpto1, viseu).
utente(15, pedro, 21, vizela).
```

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

O cuidado médico com a designação de *ortopedia* não é prestado na instituicao *hsjoao* no *porto*.

Resultado:

```
?- getIncIDS(X).
X = 16 ?
yes

?-
 ↪  inserirNegativo(-cuidado(16,ortopedia,instituicao(hsjoao,porto))).
yes

?- listing.
-cuidado(11,ortopedia,instituicao(hfaro,faro)).
-cuidado(16,ortopedia,instituicao(hsjoao,porto)).
```

5.5 Evolução de conhecimento

Conforme descrito em secções anteriores, foi criado o predicado `evoluirConhecimento` que permite a inserção de Conhecimento Perfeito partindo de conhecimento imperfeito presente na Base de Conhecimento.

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

O utente com *#IdUt* 1, de nome *maria* e residente na *guarda*, afirma ter 20 anos.

Resultado:

```
?- listing(utente).
utente(1, diogo, 21, braga).
utente(2, rui, 20, braga).
utente(3, esm, 21, prado).
utente(4, miguel, 22, viana).
utente(5, joao, 26, guimaraes).
utente(6, lisandra, 25, fafe).
utente(7, paulo, 24, braganca).
utente(10, maria, xpto001, guarda).
utente(13, trump, 70, xpto002).

?- listing(excecao).
excecao(utente(A,B,_,C)) :-
        utente(A, B, xpto001, C).
excecao(utente(11,joana,22,braga)).
excecao(utente(11,joana,22,guimaraes)).
excecao(utente(12,mauricio,A,lisboa)) :-
        A>=18,
        A=<24.
excecao(utente(A,B,C,_)) :-
        utente(A, B, C, xpto002)

?- evoluirConhecimento(utente(10,maria,20,guarda)).
yes

?- listing(utente).
utente(1, diogo, 21, braga).
```

```
utente(2, rui, 20, braga).
utente(3, esm, 21, prado).
utente(4, miguel, 22, viana).
utente(5, joao, 26, guimaraes).
utente(6, lisandra, 25, fafe).
utente(7, paulo, 24, braganca).
utente(13, trump, 70, xpto002).
utente(10, maria, 20, guarda).
yes

?- listing(excecao).
excecao(utente(11,joana,22,braga)).
excecao(utente(11,joana,22,guimaraes)).
excecao(utente(12,mauricio,A,lisboa)) :-
        A>=18,
        A=<24.
excecao(utente(A,B,C,_)) :-
        utente(A, B, C, xpto002).
yes
```

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

O cuidado 14, com descrição *oftalmologia* é prestado na instituição *hporto*, no *porto*.

Resultado:

```
?- evoluirConhecimento(cuidado( 14,oftalmologia,instituicao(
↪  hporto,porto ) )).
yes

?- listing(cuidado).
cuidado(1, analises, instituicao(hpbraga,braga)).
cuidado(2, tac, instituicao(hsjoao,porto)).
cuidado(3, parto, instituicao(hpbraga,braga)).
cuidado(4, ortopedia, instituicao(hviana,viana)).
cuidado(5, doar-sangue, instituicao(hpbraga,braga)).
cuidado(6, raioX, instituicao(hporto,porto)).
cuidado(7, medicina-geral, instituicao(hporto,porto)).
cuidado(8, parto, instituicao(hfaro,faro)).
cuidado(9, ecografia, instituicao(hfaro,faro)).
cuidado(10, quimioterapia, instituicao(hsjoao,porto)).
cuidado(12, pediatria, xpto003).
cuidado(13, xpto004, instituicao(hsjoao,porto)).
cuidado(14, oftalmologia, instituicao(hporto,porto)).

?- listing(excecao).
excecao(cuidado(A,B,_)) :-
        cuidado(A, B, xpto003).
excecao(cuidado(A,_,B)) :-
        cuidado(A, xpto004, B).
```

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

O ato médico com *#IdA* 16 ocorrido no dia 01-04-2017 com *#IdUt* 13 e *#IdServ* 10 custou 500 euros.

Resultado:

```
?- evoluirConhecimento(ato(16,data(1,4,2017),13,10,500)).
no

?- listing(ato).
ato(1, data(1,2,1996), 3, 3, 10).
ato(2, data(15,3,2017), 1, 2, 15).
ato(3, data(17,4,1997), 4, 4, 5).
ato(4, data(15,3,2007), 1, 5, 0).
ato(5, data(15,3,2007), 2, 5, 0).
ato(6, data(15,3,2007), 3, 2, 0).
ato(7, data(16,3,2017), 5, 6, 12).
ato(8, data(16,3,2007), 6, 9, 20).
ato(9, data(16,3,2007), 3, 1, 40).
ato(11, xpto006, 6, 2, 80).
ato(12, data(3,6,2007), 6, 2, xpto007).
ato(13, data(xpto008,3,2007), 5, 3, 20).
ato(16, data(1,4,2017), 13, 10, xpto009).

?- listing(nulo).
nulo(xpto002).
nulo(xpto009).
```

Conclusão e aspetos a melhorar

Como já foi referido anteriormente, o principal objetivo desta segunda fase do projeto consistiu no desenvolvimento da nossa anterior Base de Conhecimento, para que esta pudesse representar, não só o conhecimento perfeito, mas também o conhecimento imperfeito, podendo ele ser incerto, impreciso ou interdito.

Conseguimos obter ao longo da realização deste trabalho prático as faculdades necessárias à compreensão, gestão e organização de toda a informação presente na Base de Conhecimento, assim como todos predicados de forma natural. Além do `demo`, criamos um `demo2` que testa a junção de dois predicados, retornando o valor de verdade associado a esta associação. Os operadores lógicos de junção de predicados que utilizamos são a conjunção, disjunção, disjunção exclusiva, implicação e equivalência.

Foram implementados predicados que permitem a inserção e evolução da Base de Conhecimento. Como trabalho futuro pretende-se aprimorar estes predicados por forma a tornar mais acessível a sua utilização uma vez que no caso da inserção é necessário fornecer os identificadores e no caso da evolução só ser suportada para um parâmetro de imperfeição.

Deste modo, concluímos que fizemos a implementação destes conceitos de uma forma correta, e que fomos capazes de atingir todos os objetivos propostos e ainda aos nossos requisitos pessoais enquanto grupo.

A. Código fonte

```
%--------------------------------- - - - - - - - - - -  -  -  -  -
% SIST. REPR. CONHECIMENTO E RACIOCINIO - MiEI/3

%--------------------------------- - - - - - - - - - -  -  -  -  -
% Resolução do Exercício prático 2
%--------------------------------- - - - - - - - - - -  -  -  -  -
% SICStus PROLOG: Declaracoes iniciais

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

%--------------------------------- - - - - - - - - - -  -  -  -  -
% SICStus PROLOG: definicoes iniciais

:- op( 900,xfy,'::' ).
:- op(500,yfx,'and' ).
:- op(400,yfx,'or' ).
:- op(300,yfx,'xor' ).
:- op(600,yfx,'=>' ).
:- op(700,yfx,'<=>' ).
:- dynamic utente/4.
:- dynamic ato/5.
:- dynamic instituicao/2.
:- dynamic cuidado/3.
:- dynamic nulo/1.
:- dynamic excecao/1.
:- dynamic '-'/1.
:- dynamic '::'/2.
:- dynamic counter_idu/1.
:- dynamic counter_ids/1.
:- dynamic counter_ida/1.
:- dynamic counter_xpto/1.

counter_idu(13).
counter_ids(13).
counter_ida(16).
counter_xpto(48).

increment( idu ) :-
    retract( counter_idu( C ) ),
    C1 is C + 1,
    assert( counter_idu( C1 ) ) .
increment( ids ) :-
    retract( counter_ids( C ) ),
    C1 is C + 1,
    assert( counter_ids( C1 ) ) .
increment( ida ) :-
    retract( counter_ida( C ) ),
    C1 is C + 1,
    assert( counter_ida( C1 ) ) .
increment( xpto ) :-
    retract( counter_xpto( C ) ),
    C1 is C + 1,
    assert( counter_xpto( C1 ) ) .
```

```
getIncXPTO( XPTO ) :-
    increment( xpto ),
    counter_xpto( XPTO2 ),
    name( XPTO,[120,112,116,111,XPTO2] ).

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

getIncIDS( IDS ) :-
    increment( ids ),
    counter_ids( IDS ).

getIncIDA( IDA ) :-
    increment( ida ),
    counter_ida( IDA ).

%-------------------------------- - - - - - - - - - -  -  -  -  -
% Extensao do predicado utente:
% IdUt, Nome, Idade, Morada -> {V,F,D}

% ------- Conhecimento Perfeito Positivo -------%

utente( 1,diogo,21,braga ).
utente( 2,rui,20,braga ).
utente( 3,esm,21,prado ).
utente( 4,miguel,22,viana ).
utente( 5,joao,26,guimaraes ).
utente( 6,lisandra,25,fafe ).
utente( 7,paulo,24,braganca ).

% ------- Conhecimento Perfeito Negativo -------%

%% O utente manuel com id 8, 20 anos, nega ser de bragança.

-utente( 8,manuel,20,braganca ).

%% O utente com o id 9, chamada anastacia de felgueiras nega ter
↪  30 anos.

-utente( 9,anastacia,30,felgueiras ).

% ------- Conhecimento Imperfeito Incerto ------%

%% Desconhece-se a idade do utente com o IdUt 10, de nome maria e
↪  residente na guarda.

utente( 10 ,maria,xpto001,guarda ).
excecao( utente( IDU,N,I,M ) ) :-
    utente( IDU,N,xpto001,M ).

% ------- Conhecimento Imperfeito Impreciso ------%

%% A joana, utente com IdUt 11 e com 22 anos de idade, mora em
↪  braga ou em guimaraes.

excecao( utente( 11,joana,22,braga ) ).
excecao( utente( 11,joana,22,guimaraes ) ).
```

```
%%  Não se sabe ao certo a  idade do mauricio, com o IdUt 12 e
↪  residente em lisboa. Apenas se sabe que possui entre 18 a 24
↪  anos .

excecao( utente( 12,mauricio,I,lisboa ) ) :-
    I >= 18, I =< 24.

% -------  Conhecimento Imperfeito Interdito ------%

%% O utente com o IdUt 13, de nome trump, com 70 anos, exige que
↪  não se saiba a sua morada.

utente( 13,trump,70,xpto002 ).
excecao( utente( IDU,N,I,L ) ) :-
    utente( IDU,N,I,xpto002 ).
nulo( xpto002 ).
+utente(IDU,N,I,M)::(solucoes(X,(utente(13,_,_,X),nao(nulo(X))),S),
                     comprimento( S,L ),
                     L == 0 ).

% Adoção do Pressuposto do Mundo Fechado com consideração de
↪  exceções

-utente( IDU,N,I,M ) :-
    nao( utente( IDU,N,I,M ) ),
    nao( excecao( utente( IDU,N,I,M ) ) ).

% O Utente a remover não pode ter atos associados

-utente( ID,N,I,M ) :: ( solucoes( ID,ato( _,_,ID,_,_ ),S ),
                         comprimento( S,L ),
                         L == 0 ).


%--------------------------------- - - - - - - - - - -  -  -  -  -
% Extensao do predicado cuidado:
% IdServ, Descrição, Instituicao -> {V,F,D}

% ------- Conhecimento Perfeito Positivo -------%

cuidado( 1,analises,instituicao( hpbraga,braga ) ).
cuidado( 2,tac,instituicao( hsjoao,porto ) ).
cuidado( 3,parto,instituicao( hpbraga,braga ) ).
cuidado( 4,ortopedia,instituicao( hviana,viana ) ).
cuidado( 5,doar-sangue,instituicao( hpbraga,braga ) ).
cuidado( 6,raioX,instituicao( hporto,porto ) ).
cuidado( 7,medicina-geral,instituicao( hporto,porto ) ).
cuidado( 8,parto,instituicao( hfaro,faro ) ).
cuidado( 9,ecografia,instituicao( hfaro,faro ) ).
cuidado( 10,quimioterapia,instituicao( hsjoao,porto ) ).

% ------- Conhecimento Perfeito Negativo -------%

%% A instituição hfaro nao presta o cuidado ortopedia.

-cuidado( 11,ortopedia,instituicao( hfaro,faro ) ).

% -------  Conhecimento Imperfeito Incerto ------%

%%  É desconhecido o nome da instituição que presta o serviço
%%  com o IdServ 12 , designado por pediatria.
```

```
cuidado( 12,pediatria,xpto003 ).
excecao( cuidado( IdS,D,I ) ) :-
    cuidado( IdS,D,xpto003 ).

%%  É desconhecida a descrição do cuidado com o IdServ 12 ,
%%  prestado no hsjoao no porto.

cuidado( 13,xpto004,instituicao( hsjoao,porto )).
excecao( cuidado( IdS,D,I ) ) :-
    cuidado( IdS,xpto004,I ).


% -------  Conhecimento Imperfeito Impreciso ------%

%%  O cuidado com o IdServ 14, designado por oftalmologia,
%%  é prestado em uma das seguintes instituições: hsjoao ou
 ↪  hporto.

excecao( cuidado( 14,oftalmologia,instituicao( hsjoao,porto ) ) ).
excecao( cuidado( 14,oftalmologia,instituicao( hporto,porto ) ) ).

% Adoção do Pressuposto do Mundo Fechado com consideração de
 ↪  exceções

-cuidado( IdS,D,I ) :-
    nao( cuidado( IdS,D,I ) ),
    nao( excecao( cuidado( IdS,D,I ) ) ).

% Não permitir a insercao de cuidados repetidos numa instituição

+cuidado( ID,D,I ) :: ( solucoes( D,cuidado( _,D,I ),S ),
                        comprimento( S,L ),
                        L == 1 ).

% A instituição deve exisir na base de conhecimento

+cuidado( ID,D,instituicao( I,L ) ) :: instituicao( I,L ).

% O Cuidado removido não pode ter atos medicos associados

-cuidado( ID,D,I ) :: ( solucoes( ID,ato( _,_,_,ID,_ ),S ),
                        comprimento( S,L ),
                        L == 0 ).


%--------------------------------- - - - - - - - - - -  -  -  -  -
% Extensao do predicado Instituição: Nome, Cidade -> {V,F,D}

% ------- Conhecimento Perfeito Positivo -------%

instituicao( hpbraga,braga ).
instituicao( hsjoao,porto ).
instituicao( hviana,viana ).
instituicao( hporto,porto ).
instituicao( hfaro,faro ).

% ------- Conhecimento Perfeito Negativo -------%

%% Em braga nao existe nenhuma instituicao com o nome hsjoao.

-instituicao( hsjoao,braga ).
```

```
% -------  Conhecimento Imperfeito Incerto ------%

%% Desconhece-se a localidade do hsmaria.

instituicao( hsmaria,xpto005 ).
excecao( instituicao( I,L ) ) :-
    instituicao( I,xpto005 ).

% -------  Conhecimento Imperfeito Impreciso ------%

%% Não se sabe ao certo se a instituicao hsjose se encontra
↪  localizada em braga ou no porto.

excecao(instituicao(hsjose,braga)).
excecao(instituicao(hsjose,porto)).

% Adoção do Pressuposto do Mundo Fechado com consideração de
↪  exceções

-instituicao( N,C ) :- nao( instituicao( N,C ) ),
                       nao( excecao( instituicao( N,C ))).

% Não permite remoção se estiver associado algum cuidado prestado

-instituicao( I,C ) :: ( solucoes( I,cuidado(_,_,I,_),S ),
                         comprimento( S,L ),
                         L == 0 ).


%--------------------------------- - - - - - - - - - -  -  -  -  -
% Extensao do predicado ato medico:
% Data, IdUtente, IdServico, Custo -> {V,F,D}

% ------- Conhecimento Perfeito Positivo -------%

ato( 1,data( 1,2,1996 ),3,3,10 ).
ato( 2,data( 15,3,2017 ),1,2,15 ).
ato( 3,data( 17,4,1997 ),4,4,5 ).
ato( 4,data( 15,3,2007 ),1,5,0 ).
ato( 5,data( 15,3,2007 ),2,5,0 ).
ato( 6,data( 15,3,2007 ),3,2,0 ).
ato( 7,data( 16,3,2017 ),5,6,12 ).
ato( 8,data( 16,3,2007 ),6,9,20 ).
ato( 9,data( 16,3,2007 ),3,1,40 ).

% ------- Conhecimento Perfeito Negativo -------%

%% Sabe-se que o ato com ID 10 ocorrido no dia 31-08-2000, com
↪  IdUt 3 e IDServ 7, não teve um custo de 50 euros.

-ato( 10 ,data( 31,8,2000 ),3,7,50 ).

% ------- Conhecimento Imperfeito Incerto -------%

%%  Desconhece-se a data em que foi prestado o ato medico com
↪  IdAto 11, realizado pelo utente com o IdUt 6, relativo ao
↪  cuidado com IdServ 2 com um custo de 80 euros.

ato( 11,xpto006,6,2,80 ).
excecao( ato( ID,D,IDU,IdS,P ) ) :-
    ato( ID,xpto006,IDU,IdS,P ).
```

```
%% Desconhece-se o custo do ato medico com ocorrido em 03-06-2007,
↪  com IDServ 2 e IdUt 6, identificado pelo IDAto 12.

ato( 12,data(3,6,2007 ),6,2,xpto007 ).
excecao( ato( ID,D,IDU,IDS,P ) ) :-
    ato( ID,D,IDU,IDS, xpto007).

%% Desconhece-se o dia em que foi prestado o cuidado IdServ 3,
↪  pelo utente com o IdUt 5, com um custo de 20 euros. Apenas se
↪  sabe o ato com IDAto 13 foi prestado em março de 2007.

ato( 13,data( xpto008,3,2007 ),5,3,20 ).
excecao( ato( ID,data( D,M,A ),IDU,IdS,P ) ) :-
    ato( ID,data( xpto008,M,A ),IDU,IdS,P ).

% ------- Conhecimento Imperfeito Impreciso -------%

%% O ato medico com IdAto 14 ocorrido em 03-06-2007, com o IdUt 6
↪  e IdServ 9 custou entre 10 a 25 euros.

excecao( ato( 14,data(3,6,2007 ),6,9,P ) ) :-
    P >= 10, P =< 25.

%% O ato medico com IdAto 15 ocorrido em 04-06-2007, com o IdUt 7
↪  e IdServ 1 custou entre 20 ou 30 euros.

excecao( ato( 15,data(4,6,2007 ),6,9,20 ) ).
excecao( ato( 15,data(4,6,2007 ),6,9,30) ).

% ------- Conhecimento Imperfeito Interdito -------%

%% O utente com o IdUt 13, usufruiu do cuidado medico com o IdServ
↪  10 em 01-04-2017, contudo pretende que não se saiba quanto
↪  pagou por este ato identificado pelo IdAto 16.

ato( 16,data(1,4,2017),13,10,xpto009 ).
excecao( ato( ID,D,IDU,IdS,P ) ) :-
    ato( ID,D,IDU,IdS,xpto009 ).
nulo( xpto009 ).
+ato(ID,D,IDU,IdS,P)::(solucoes(P,(ato(16,_,_,_,P),nao(nulo(P))),S),
                       comprimento( S,N ),
                       N == 0).

% Adoção do Pressuposto do Mundo Fechado com consideração de
↪  exceções

-ato( ID,D,IDU,IDS,C ) :- nao( ato( ID,D,IDU,IDS, C )),
                          nao( excecao( ato( ID,D,IDU,IDS, C ) )).

% Não permitir inserir atos com IdUt não registados

+ato( ID,D,IDU,IDS,C ) :: ( solucoes( IDU,utente( IDU,_,_,_ ),S ),
                            comprimento( S,L ),
                            L == 1 ).

% Não permitir inserir atos com IdServ não registados

+ato( ID,D,IDU,IDS,C ) :: ( solucoes( IDS,cuidado( IDS,_,_ ),S ),
                            comprimento( S,L ),
                            L == 1 ).
```

```
%------------------ Predicados Auxiliares -------------------

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

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


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

demo2( (Q1 and Q2),verdadeiro ) :-
    demo( Q1,verdadeiro ),
    demo( Q2,verdadeiro ).
demo2( (Q1 and Q2),falso ) :-
    demo( Q1,falso ).
demo2( (Q1 and Q2),falso ) :-
    demo( Q2,falso ).
demo2( (Q1 and Q2),desconhecido ).

demo2( (Q1 or Q2),verdadeiro ) :-
    demo( Q1,verdadeiro ).
demo2( (Q1 or Q2),verdadeiro ) :-
    demo( Q2,verdadeiro ).
demo2( (Q1 or Q2),falso ) :-
    demo( Q1,falso ),
    demo( Q2,falso ).
demo2( (Q1 or Q2),desconhecido ).

demo2( (Q1 xor Q2),falso ) :-
    demo( Q1,falso ),
    demo( Q2,falso ).
demo2( (Q1 xor Q2),falso ) :-
    demo( Q1,verdadeiro ),
    demo( Q2,verdadeiro ).
demo2( (Q1 xor Q2),verdadeiro ) :-
    demo( Q1, verdadeiro ),
    demo( Q2, falso ).
demo2( (Q1 xor Q2),verdadeiro ) :-
    demo( Q1, falso ),
    demo( Q2, verdadeiro ).
demo2( (Q1 xor Q2), desconhecido ).

demo2( (Q1 => Q2),falso ) :-
    demo( Q1,verdadeiro ),
    demo(Q2, falso).
demo2( (Q1 => Q2),verdadeiro ) :-
    demo( Q1,falso ).
demo2( (Q1 => Q2),verdadeiro ) :-
    demo( Q2,verdadeiro ).
demo2( (Q1 => Q2),desconhecido ).


demo2( (Q1 <=> Q2),verdadeiro ) :-
    demo( Q1,verdadeiro ),
```

```
    demo( Q2,verdadeiro ).
demo2( (Q1 <=> Q2),verdadeiro ) :-
    demo( Q1,falso ),
    demo( Q2,falso ).
demo2( (Q1 <=> Q2),falso ) :-
    demo( Q1,verdadeiro ),
    demo( Q2,falso ).
demo2( (Q1 <=> Q2),falso ) :-
    demo( Q1,falso ),
    demo( Q2,verdadeiro ).
demo2( (Q1 <=> Q2),desconhecido ).


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

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


%-------------------------------- - - - - - - - - - - - - - - -
% 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).


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

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


%-------------------------------- - - - - - - - - - - - - - - -
% 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 ).



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

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

```
%------------------------------ - - - - - - - - - - - - - - - -
% Extensao do predicado solucoes

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


%------------------------------ - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a evolução de conhecimento

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


%------------------------------ - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a evolução de conhecimento
↪  perfeito partindo de conhecimento imperfeito impreciso do
↪  utente

evoluirConhecimento( utente( ID,N,I,M )) :-
    demo(utente(ID,N,I,M),desconhecido),
    solucoes( ((excecao( utente( ID,N,X,M) ) :- X >= LI, X =<
    ↪  LS)),(excecao( utente( ID,N,I,M) )),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    evolucao( utente( ID,N,I,M ) ).

evoluirConhecimento( utente( ID,N,I,M )) :-
    demo(utente(ID,N,I,M),desconhecido),
    solucoes((excecao( utente( ID,NU,IU,MU ))),(excecao( utente(
    ↪  ID,NU,IU,MU ))),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    evolucao( utente( ID,N,I,M ) ).

% partindo de conhecimento imperfeito impreciso do cuidado

evoluirConhecimento( cuidado( ID,D,I )) :-
    demo(cuidado( ID,D,I ), desconhecido),
    solucoes( (excecao( cuidado( ID,DC,IC ))),(excecao( cuidado(
    ↪  ID,DC,IC ))),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    evolucao( cuidado( ID,D,I )).

% partindo de conhecimento imperfeito impreciso da instituicao

evoluirConhecimento( instituicao( D,I )) :-
    demo(instituicao( D,I ), desconhecido),
    solucoes( (excecao( instituicao( D,II ))),(excecao(
    ↪  instituicao( D,II ))),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    assert(instituicao( D,I )).

% partindo de conhecimento imperfeito impreciso do ato

evoluirConhecimento(ato( ID,data( D,M,A ),IDU,IDC,C)) :-
    demo(ato( ID,data( D,M,A ),IDU,IDC,C),desconhecido),
```

```
    solucoes( ((excecao( ato( ID,data( X,M,A ),IDU,IDC,C) ) :- X
    ↪  >= LI, X =< LS)),(excecao( ato( ID,data(
    ↪  D,M,A),IDU,IDC,C))),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    evolucao( ato( ID,data( D,M,A ),IDU,IDC,C)).

evoluirConhecimento(ato( ID,data( D,M,A ),IDU,IDC,C)) :-
    demo(ato( ID,data( D,M,A ),IDU,IDC,C),desconhecido),
    solucoes( ((excecao( ato( ID,data( D,X,A ),IDU,IDC,C) ) :- X
    ↪  >= LI, X =< LS)),(excecao( ato( ID,data(
    ↪  D,M,A),IDU,IDC,C))),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    evolucao( ato( ID,data( D,M,A ),IDU,IDC,C)).

evoluirConhecimento(ato( ID,data( D,M,A ),IDU,IDC,C)) :-
    demo(ato( ID,data( D,M,A ),IDU,IDC,C),desconhecido),
    solucoes( ((excecao( ato( ID,data( D,M,X ),IDU,IDC,C) ) :- X
    ↪  >= LI, X =< LS)),(excecao( ato( ID,data(
    ↪  D,M,A),IDU,IDC,C))),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    evolucao( ato( ID,data( D,M,A ),IDU,IDC,C)).

evoluirConhecimento(ato( ID,D,IDU,IDC,C)) :-
    demo(ato( ID,D,IDU,IDC,C),desconhecido),
    solucoes( ((excecao( ato( ID,D,IDU,IDC,X) ) :- X >= LI, X =<
    ↪  LS)),(excecao( ato( ID,D,IDU,IDC,C))),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    evolucao( ato( ID,D,IDU,IDC,C)).

evoluirConhecimento( ato( ID,D,IDU,IDC,C)) :-
    demo(ato( ID,D,IDU,IDC,C), desconhecido),
    solucoes( (excecao( ato( ID,DA,IDA,IDSA,CA))),(excecao( ato(
    ↪  ID,DA,IDA,IDSA,CA))),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    evolucao( ato( ID,D,IDU,IDC,C)).


%-------------------------------- - - - - - - - - - -  -  -  -  -
% Extensão do predicado que permite a evolução de conhecimento
↪  perfeito partindo de conhecimento imperfeito incerto do utente

evoluirConhecimento( utente( ID,N,I,M ) ) :-
    demo(utente(ID,N,I,M),desconhecido),
    solucoes( (excecao( utente( ID,N,I,M ) ) :- utente( ID,X,I,M
    ↪  )),(utente( ID,X,I,M ),nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeUtente( ID ),
    evolucao( utente( ID,N,I,M ) ).
evoluirConhecimento( utente( ID,N,I,M ) ) :-
    demo(utente(ID,N,I,M),desconhecido),
    solucoes( (excecao( utente( ID,N,I,M ) ) :- utente( ID,N,X,M
    ↪  )),(utente( ID,N,X,M ),nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeUtente( ID ),
    evolucao( utente( ID,N,I,M ) ).
```

```
evoluirConhecimento( utente( ID,N,I,M ) ) :-
    demo(utente(ID,N,I,M),desconhecido),
    solucoes( (excecao( utente( ID,N,I,M ) ) :- utente( ID,N,I,X
    ↪  )),(utente( ID,N,I,X ),nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeUtente( ID ),
    evolucao( utente( ID,N,I,M ) ).

% partindo de conhecimento imperfeito incerto do cuidado

evoluirConhecimento( cuidado( ID,D,I )) :-
    demo(cuidado( ID,D,I ),desconhecido),
    solucoes( (excecao( cuidado( ID,D,I) ) :- cuidado(
    ↪  ID,X,I)),(cuidado( ID,X,I ),nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeCuidado( ID ),
    evolucao( cuidado( ID,D,I ) ).
evoluirConhecimento( cuidado( ID,D,I )) :-
    demo(cuidado( ID,D,I ),desconhecido),
    solucoes( (excecao( cuidado( ID,D,I) ) :- cuidado(
    ↪  ID,D,X)),(cuidado( ID,D,X ),nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeCuidado( ID ),
    evolucao( cuidado( ID,D,I ) ).

% partindo de conhecimento imperfeito incerto da instituicao

evoluirConhecimento( instituicao( D,I)) :-
    demo(instituicao(D,I),desconhecido),
    solucoes( (excecao( instituicao( D,X) ) :- instituicao(
    ↪  D,X)),(instituicao( D,X),nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeInst( D ),
    evolucao( instituicao( D,I)).

% partindo de conhecimento imperfeito incerto do ato

evoluirConhecimento( ato( ID,data(D,M,A),IDU,IDS,C ) ) :-
    demo(  ato( ID,data(D,M,A),IDU,IDS,C ),desconhecido),
    solucoes( (excecao(   ato( ID,data(D,M,A),IDU,IDS,C ) ) :-
    ↪  ato( ID,data(X,M,A),IDU,IDS,C )),(  ato(
    ↪  ID,data(X,M,A),IDU,IDS,C ),nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeAto( ID ),
    evolucao(  ato( ID,data(D,M,A),IDU,IDS,C ) ).

evoluirConhecimento( ato( ID,data(D,M,A),IDU,IDS,C ) ) :-
    demo(  ato( ID,data(D,M,A),IDU,IDS,C ),desconhecido),
    solucoes( (excecao(   ato( ID,data(D,M,A),IDU,IDS,C ) ) :-
    ↪  ato( ID,data(D,X,A),IDU,IDS,C )),(  ato(
    ↪  ID,data(D,X,A),IDU,IDS,C ),nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeAto( ID ),
    evolucao(  ato( ID,data(D,M,A),IDU,IDS,C ) ).

evoluirConhecimento( ato( ID,data(D,M,A),IDU,IDS,C ) ) :-
```

```
    demo(  ato( ID,data(D,M,A),IDU,IDS,C ),desconhecido),
    solucoes( (excecao(   ato( ID,data(D,M,A),IDU,IDS,C ) ) :-
    ↪  ato( ID,data(D,M,X),IDU,IDS,C )),(  ato(
    ↪  ID,data(D,M,X),IDU,IDS,C ),nao( nulo( X ) ) ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeAto( ID ),
    evolucao(  ato( ID,data(D,M,A),IDU,IDS,C ) ).

evoluirConhecimento( ato( ID,D,IDU,IDS,C ) ) :-
    demo( ato( ID,D,IDU,IDS,C ),desconhecido),
    solucoes( (excecao(  ato( ID,D,IDU,IDS,C ) ) :-  ato(
    ↪  ID,X,IDU,IDS,C)),( ato( ID,X,IDU,IDS,C ),nao( nulo( X ) )
    ↪  ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeAto( ID ),
    evolucao( ato( ID,D,IDU,IDS,C )  ).

evoluirConhecimento( ato( ID,D,IDU,IDS,C ) ) :-
    demo( ato( ID,D,IDU,IDS,C ),desconhecido),
    solucoes( (excecao(  ato( ID,D,IDU,IDS,C ) ) :-  ato(
    ↪  ID,D,X,IDS,C)),( ato( ID,D,X,IDS,C ),nao( nulo( X ) )
    ↪  ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeAto( ID ),
    evolucao( ato( ID,D,IDU,IDS,C )  ).

evoluirConhecimento( ato( ID,D,IDU,IDS,C ) ) :-
    demo( ato( ID,D,IDU,IDS,C ),desconhecido),
    solucoes( (excecao(  ato( ID,D,IDU,IDS,C ) ) :-  ato(
    ↪  ID,D,IDU,X,C)),( ato( ID,D,IDU,X,C ),nao( nulo( X ) )
    ↪  ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeAto( ID ),
    evolucao( ato( ID,D,IDU,IDS,C )  ).

evoluirConhecimento( ato( ID,D,IDU,IDS,C ) ) :-
    demo( ato( ID,D,IDU,IDS,C ),desconhecido),
    solucoes( (excecao(  ato( ID,D,IDU,IDS,C ) ) :-  ato(
    ↪  ID,D,IDU,IDS,X)),( ato( ID,D,IDU,IDS,X),nao( nulo( X ) )
    ↪  ),LEXC ),
    comprimento(LEXC,S), S > 0,
    removeAll( LEXC ),
    removeAto( ID ),
    evolucao( ato( ID,D,IDU,IDS,C )  ).


%--------------------------------- - - - - - - - - - - -  -  -  -  -
% Extensão do predicado que permite a evolução de conhecimento
% perfeito positivo

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

```
%-------------------------------- - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a evolução de conhecimento
% perfeito positivo

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


%-------------------------------- - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a evolução de conhecimento
% imperfeito do tipo incerto

inserirIncerto( utente( ID,I,M ),nome ) :-
    getIncXPTO( XPTO ),
    evolucao( utente( ID,XPTO,I,M ) ),
    evolucao( (excecao( utente( IDU,NU,IU,MU ) ) :- utente(
    ↪  IDU,XPTO,IU,MU )) ).
inserirIncerto( utente( ID,N,M ),idade ) :-
    getIncXPTO( XPTO ),
    evolucao( utente( ID,N,XPTO,M ) ),
    evolucao( (excecao( utente( IDU,NU,IU,MU ) ) :- utente(
    ↪  IDU,NU,XPTO,MU )) ).
inserirIncerto( utente( ID,N,I ),morada ) :-
    getIncXPTO( XPTO ),
    evolucao( utente( ID,N,I,XPTO ) ),
    evolucao( (excecao( utente( IDU,NU,IU,MU ) ) :- utente(
    ↪  IDU,NU,IU,XPTO )) ).

inserirIncerto( cuidado( ID,I ),descricao ) :-
    getIncXPTO( XPTO ),
    evolucao( cuidado( ID,XPTO,I ) ),
    evolucao( (excecao( cuidado( IDC,DC,IC ) ) :- cuidado(
    ↪  IDC,XPTO,IC )) ).
inserirIncerto( cuidado( ID,D ),instituicao ) :-
    getIncXPTO( XPTO ),
    evolucao( cuidado( ID,D,XPTO ) ),
    evolucao( (excecao( cuidado( IDC,DC,IC ) ) :- cuidado(
    ↪  IDC,DC,XPTO )) ).

inserirIncerto( instituicao( D ),cidade ) :-
    getIncXPTO( XPTO ),
    evolucao( instituicao( D,XPTO ) ),
    evolucao( (excecao( instituicao( DI,CI ) ) :- instituicao(
    ↪  DI,XPTO )) ).

inserirIncerto( ato( ID,data( M,A ),IDU,IDS,C ),dia ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,data(XPTO,M,A),IDU,IDS,C ) ),
    evolucao( (excecao( ato( IDA,data(DA,MA,AA),IDUA,IDSA,CA ) )
    ↪  :- ato( IDA,data(XPTO,MA,AA),IDUA,IDSA,CA )) ).
inserirIncerto( ato( ID,data( D,A) ,IDU,IDS,C ),mes ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,data(D,XPTO,A),IDU,IDS,C ) ),
    evolucao( (excecao( ato( IDA,data(DA,MA,AA),IDUA,IDSA,CA ) )
    ↪  :- ato( IDA,data(DA,XPTO,AA),IDUA,IDSA,CA )) ).
inserirIncerto( ato( ID,data( D,M ),IDU,IDS,C ),ano ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,data(D,M,XPTO),IDU,IDS,C ) ),
```

```
    evolucao( (excecao( ato( IDA,data(DA,MA,AA),IDUA,IDSA,CA ) )
     ↪  :- ato( IDA,data(DA,MA,XPTO),IDUA,IDSA,CA )) ).
inserirIncerto( ato( ID,IDU,IDS,C ),data ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,XPTO,IDU,IDS,C ) ),
    evolucao( (excecao( ato( IDA,DA,IDUA,IDSA,CA ) ) :- ato(
     ↪  IDA,XPTO,IDUA,IDSA,CA )) ).
inserirIncerto( ato( ID,D,IDS,C ),utente ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,D,XPTO,IDS,C ) ),
    evolucao( (excecao( ato( IDA,DA,IDUA,IDSA,CA ) ) :- ato(
     ↪  IDA,DA,XPTO,IDSA,CA )) ).
inserirIncerto( ato( ID,D,IDU,C ),cuidado ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,D,IDU,XPTO,C ) ),
    evolucao( (excecao( ato( IDA,DA,IDUA,IDSA,CA ) ) :- ato(
     ↪  IDA,DA,IDUA,XPTO,CA )) ).
inserirIncerto( ato( ID,D,IDS,C ),custo ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,D,IDU,IDS,XPTO ) ),
    evolucao( (excecao( ato( IDA,DA,IDUA,IDSA,CA ) ) :- ato(
     ↪  IDA,DA,IDUA,IDSA,XPTO )) ).


%-------------------------------- - - - - - - - - - -  -  -  -  -
% Extensão do predicado que permite a evolução de conhecimento
% imperfeito do tipo impreciso

inserirImpreciso( utente( ID,N,[I1 - I2],M ) ) :-
    evolucao( (excecao( utente( ID,N,I,M ) ) :- I >= I1, I =< I2)
     ↪  ).
inserirImpreciso( utente( ID,[],I,M ) ).
inserirImpreciso( utente( ID,N,[],M ) ).
inserirImpreciso( utente( ID,N,I,[] ) ).
inserirImpreciso( utente( ID,[N|NS],I,M ) ) :-
    evolucao( excecao( utente( ID,N,I,M ) ) ),
    inserirImpreciso( utente( ID,NS,I,M ) ).
inserirImpreciso( utente( ID,N,[I|IS],M ) ) :-
    evolucao( excecao( utente( ID,N,I,M ) ) ),
    inserirImpreciso( utente( ID,N,IS,M ) ).
inserirImpreciso( utente( ID,N,I,[M|MS] ) ) :-
    evolucao( excecao( utente( ID,N,I,M ) ) ),
    inserirImpreciso( utente( ID,N,I,MS ) ).

inserirImpreciso( cuidado( ID,[],I ) ).
inserirImpreciso( cuidado( ID,D,[] ) ).
inserirImpreciso( cuidado( ID,[D|DS],I ) ) :-
    evolucao( excecao( cuidado( ID,D,I ) ) ),
    inserirImpreciso( cuidado( ID,DS,I ) ).
inserirImpreciso( cuidado( ID,D,[I|IS] ) ) :-
    evolucao( excecao( cuidado( ID,D,I ) ) ),
    inserirImpreciso( cuidado( ID,D,IS ) ).

inserirImpreciso( instituicao( D,[] ) ).
inserirImpreciso( instituicao( D,[C|CS] ) ) :-
    evolucao( excecao( instituicao( D,C ) ) ),
    inserirImpreciso( instituicao( D,CS ) ).

inserirImpreciso( ato( ID,D,IDU,IDS,[C1 - C2] ) ) :-
    evolucao( (excecao( ato( ID,D,IDU,IDS,C ) ) :- C >= C1, C =<
     ↪  C2) ).
inserirImpreciso( ato( ID,data( [D1 - D2],M,A ),IDU,IDS,C ) ) :-
```

```
    evolucao( (excecao( ato( ID,data( D,M,A ),IDU,IDS,C ) ) :- D
     ↪  >= D1, D =< D2) ).
inserirImpreciso( ato( ID,data( D,[M1 - M2],A ),IDU,IDS,C ) ) :-
    evolucao( (excecao( ato( ID,data( D,M,A ),IDU,IDS,C ) ) :- M
     ↪  >= M1, M =< M2) ).
inserirImpreciso( ato( ID,data( D,M,[A1 - A2] ),IDU,IDS,C ) ) :-
    evolucao( (excecao( ato( ID,data( D,M,A ),IDU,IDS,C ) ) :- A
     ↪  >= A1, A =< A2) ).
inserirImpreciso( ato( ID,[],IDU,IDS,C ) ).
inserirImpreciso( ato( ID,D,[],IDS,C ) ).
inserirImpreciso( ato( ID,D,IDU,[],C ) ).
inserirImpreciso( ato( ID,D,IDU,IDS,[] ) ).
inserirImpreciso( ato( ID,data( [],M,A ),IDU,IDS,C ) ).
inserirImpreciso( ato( ID,data( D,[],A ),IDU,IDS,C ) ).
inserirImpreciso( ato( ID,data( D,M,[] ),IDU,IDS,C ) ).
inserirImpreciso( ato( ID,[D|DS],IDU,IDS,C ) ) :-
    evolucao( excecao( ato( ID,D,IDU,IDS,C ) ) ),
    inserirImpreciso( ato( ID,DS,IDU,IDS,C ) ).
inserirImpreciso( ato( ID,D,[IDU|IDUS],IDS,C ) ) :-
    evolucao( excecao( ato( ID,D,IDU,IDS,C ) ) ),
    inserirImpreciso( ato( ID,D,IDUS,IDS,C ) ).
inserirImpreciso( ato( ID,D,IDU,[IDS|IDSS],C ) ) :-
    evolucao( excecao( ato( ID,D,IDU,IDS,C ) ) ),
    inserirImpreciso( ato( ID,D,IDU,IDSS,C ) ).
inserirImpreciso( ato( ID,D,IDU,IDS,[C|CS] ) ) :-
    evolucao( excecao( ato( ID,D,IDU,IDS,C ) ) ),
    inserirImpreciso( ato( ID,D,IDU,IDS,CS ) ).
inserirImpreciso( ato( ID,data( [D|DS],M,A ),IDU,IDS,C ) ) :-
    evolucao( excecao( ato( ID,data( D,M,A ),IDU,IDS,C ) ) ),
    inserirImpreciso( ato( ID,data( DS,M,A ) ),IDU,IDS,C ).
inserirImpreciso( ato( ID,data( D,[M|MS],A ),IDU,IDS,C ) ) :-
    evolucao( excecao( ato( ID,data( D,M,A ),IDU,IDS,C ) ) ),
    inserirImpreciso( ato( ID,data( D,MS,A ) ),IDU,IDS,C ).
inserirImpreciso( ato( ID,data( D,M,[A|AS] ),IDU,IDS,C ) ) :-
    evolucao( excecao( ato( ID,data( D,M,A ),IDU,IDS,C ) ) ),
    inserirImpreciso( ato( ID,data( D,M,AS ) ),IDU,IDS,C ).


%-------------------------------- - - - - - - - - - -  -  -  -  -
% Extensão do predicado que permite a evolução de conhecimento
% imperfeito do tipo interdito

inserirInterdito( utente( ID,I,M ),nome ) :-
    getIncXPTO( XPTO ),
    evolucao( utente( ID,XPTO,I,M ) ),
    evolucao( (excecao( utente( IDU,NU,IU,MU ) ) :- utente(
     ↪  IDU,XPTO,IU,MU )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+utente( IDU,NU,IU,MU ) :: (solucoes( X,(utente(
     ↪  ID,X,_,_ ),nao( nulo( X ) )),S ),
                                           comprimento( S,L ),
                                           L == 0 )) ).
inserirInterdito( utente( ID,I,M ),idade ) :-
    getIncXPTO( XPTO ),
    evolucao( utente( ID,N,XPTO,M ) ),
    evolucao( (excecao( utente( IDU,NU,IU,MU ) ) :- utente(
     ↪  IDU,NU,XPTO,MU )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+utente( IDU,NU,IU,MU ) :: (solucoes( X,(utente(
     ↪  ID,_,X,_ ),nao( nulo( X ) )),S ),
                                           comprimento( S,L ),
                                           L == 0 )) ).
```

```
inserirInterdito( utente( ID,I,M ),morada ) :-
    getIncXPTO( XPTO ),
    evolucao( utente( ID,N,I,XPTO ) ),
    evolucao( (excecao( utente( IDU,NU,IU,MU ) ) :- utente(
    ↪ IDU,NU,IU,XPTO )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+utente( IDU,NU,IU,MU ) :: (solucoes( X,(utente(
    ↪ ID,_,_,X ),nao( nulo( X ) )),S ),
                                          comprimento( S,L ),
                                          L == 0 )) ).

inserirInterdito( cuidado( ID,I ),descricao ) :-
    getIncXPTO( XPTO ),
    evolucao( cuidado( ID,XPTO,I ) ),
    evolucao( (excecao( cuidado( IDC,DC,IC ) ) :- cuidado(
    ↪ IDC,XPTO,IC )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+cuidado( IDC,DC,IC ) :: (solucoes( X,(cuidado(
    ↪ ID,X,_ ),nao( nulo( X ) )),S ),
                                        comprimento( S,L ),
                                        L == 0 )) ).
inserirInterdito( cuidado( ID,D ),instituicao ) :-
    getIncXPTO( XPTO ),
    evolucao( cuidado( ID,D,XPTO ) ),
    evolucao( (excecao( cuidado( IDC,DC,IC ) ) :- cuidado(
    ↪ IDC,DC,XPTO )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+cuidado( IDC,DC,IC ) :: (solucoes( X,(cuidado(
    ↪ ID,_,X ),nao( nulo( X ) )),S ),
                                        comprimento( S,L ),
                                        L == 0 )) ).

inserirInterdito( instituicao( D ),cidade ) :-
    getIncXPTO( XPTO ),
    evolucao( instituicao( D,XPTO ) ),
    evolucao( (excecao( instituicao( DI,CI ) ) :- instituicao(
    ↪ DI,XPTO )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+instituicao( DI,CI ) :: (solucoes( X,(instituicao(
    ↪ DI,X ),nao( nulo( X ) )),S ),
                                        comprimento( S,L ),
                                        L == 0 )) ).

inserirInterdito( ato( ID,IDU,IDS,C ),data ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,XPTO,IDU,IDS,C ) ),
    evolucao( (excecao( ato( IDA,DA,IDUA,IDSA,CA ) ) :- ato(
    ↪ IDA,XPTO,IDUA,IDSA,CA )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+ato( IDA,DA,IDUA,IDSA,CA ) :: (solucoes( X,(ato(
    ↪ ID,X,_,_,_ ),nao( nulo( X ) )),S ),
                                              comprimento( S,L ),
                                              L == 0 )) ).
inserirInterdito( ato( ID,D,IDS,C ),utente ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,D,XPTO,IDS,C ) ),
    evolucao( (excecao( ato( IDA,DA,IDUA,IDSA,CA ) ) :- ato(
    ↪ IDA,DA,XPTO,IDSA,CA )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+ato( IDA,DA,IDUA,IDSA,CA ) :: (solucoes( X,(ato(
    ↪ ID,_,X,_,_ ),nao( nulo( X ) )),S ),
                                              comprimento( S,L ),
```

```
                                                    L == 0 )) ).
inserirInterdito( ato( ID,D,IDU,C ),cuidado ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,D,IDU,XPTO,C ) ),
    evolucao( (excecao( ato( IDA,DA,IDUA,IDSA,CA ) ) :- ato(
     ↪  IDA,DA,IDUA,XPTO,CA )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+ato( IDA,DA,IDUA,IDSA,CA ) :: (solucoes( X,(ato(
     ↪  ID,_,_,X,_ ),nao( nulo( X ) )),S ),
                                                    comprimento( S,L ),
                                                    L == 0 )) ).
inserirInterdito( ato( ID,D,IDS,C ),custo ) :-
    getIncXPTO( XPTO ),
    evolucao( ato( ID,D,IDU,IDS,XPTO ) ),
    evolucao( (excecao( ato( IDA,DA,IDUA,IDSA,CA ) ) :- ato(
     ↪  IDA,DA,IDUA,IDSA,XPTO )) ),
    evolucao( nulo( XPTO ) ),
    evolucao( (+ato( IDA,DA,IDUA,IDSA,CA ) :: (solucoes( X,(ato(
     ↪  ID,_,_,_,X ),nao( nulo( X ) )),S ),
                                                    comprimento( S,L ),
                                                    L == 0 )) ).


%--------------------------------- - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a inserção de conhecimento

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


%--------------------------------- - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a involução do conhecimento

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


%--------------------------------- - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a remoção de conhecimento

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


%--------------------------------- - - - - - - - - - - - - - - - -
% Extensão do predicado que testa uma lista de termos

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

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

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


%--------------------------------- - - - - - - - - - - - - - - - -
% Extensão do predicado que permite a evolução de uma lista de
 ↪  Termos

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


%--------------------------------- - - - - - - - - - - - - - - - -
 ↪  -
% Predicado «comprimento» que calcula o número de elementos
% existentes numa lista

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


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

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


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

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


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

removeUtente( ID ) :-
    solucoes( utente( IDU,NU,IU,MU ),utente( ID,NU,IU,MU ),LUT ),
    comprimento( LUT,L ),
    L > 0,
    removeAll( LUT ).

%--------------------------------- - - - - - - - - - - - - - - - -
% Extensao do predicado removeCuidado: IDC -> {V,F}

removeCuidado( ID ) :-
    solucoes( cuidado( IDC,DC,IC),cuidado( ID,DC,IC),LC ),
    comprimento( LC,L ),
    L > 0,
```

```
    removeAll( LC).


%-------------------------------- - - - - - - - - - -  -  -  -  -
% Extensao do predicado removeInst: D,I -> {V,F}

removeInst( D) :-
    solucoes( instituicao(DI,II),instituicao( D,II),LI),
    comprimento( LI,L ),
    L > 0,
    removeAll( LI).


%-------------------------------- - - - - - - - - - -  -  -  -  -
% Extensao do predicado removeCuidado: IDA -> {V,F}

removeAto( ID ) :-
    solucoes( ato( IDA,DA,IC,IS,C),ato( ID,DA,IC,IS,C),LA ),
    comprimento( LA,L ),
    L > 0,
    removeAll( LA).


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

naoExiste(utente(ID,N,I,M)) :-
    solucoes( ID, utente(ID,NU,IU,MU), L),
    comprimento( L,S ),
       S == 0.
naoExiste(instituicao(D,I)) :-
    solucoes( D, utente(D,I), L),
    comprimento( L,S ),
       S == 0.
naoExiste(cuidado(ID,D,I)) :-
    solucoes( ID, cuidado(ID,DC,IC), L),
    comprimento( L,S ),
       S == 0.
naoExiste(ato(ID,D,IDC,IDS,C)):-
    solucoes( ID, ato(ID,DA,IDCA,IDSA,CA), L),
    comprimento( L,S ),
       S == 0.


%-------------------------------- - - - - - - - - - -  -  -  -  -
% Extensao do predicado naoExisteNT: ID -> {V,F}

naoExisteNT(-utente(ID,N,I,M)) :-
    solucoes( ID, -utente(ID,NU,IU,MU), L),
    comprimento( L,S ),
       S < 2.
naoExisteNT(-instituicao(D,I)) :-
    solucoes( D, -instituicao(D,II), L),
    comprimento( L,S ),
       S < 2.
naoExisteNT(-cuidado(ID,D,I)) :-
    solucoes( ID, -cuidado(ID,DC,IC), L),
    comprimento( L,S ),
       S < 2.
naoExisteNT(-ato(ID,D,IDC,IDS,C)):-
    solucoes( ID, -ato(ID,DA,IDCA,IDSA,CA), L),
    comprimento( L,S ),
```

```
        S < 2.


%-------------------------------- - - - - - - - - - -  -  -  -  -
% Extensao do predicado existeE: ID -> {V,F}

existeE(utente(ID,N,I,M)) :-
    retract((excecao(utente(ID,N,X,M)):- X>=Y,X=<Z)),
    assert((excecao(utente(ID,N,X,M)):- X>=Y,X=<Z)).
existeE(utente(ID,N,I,M)) :-
    solucoes( (excecao( utente( ID,NU,IU,MU ))),(excecao( utente(
    ↪  ID,NU,IU,MU ))),LEXC ),
    comprimento(LEXC,S), S > 0.
existeE(cuidado(ID,D,I)) :-
    solucoes(
    ↪  (excecao(cuidado(ID,DC,IC))),(excecao(cuidado(ID,DC,IC))),LEXC
    ↪  ),
    comprimento(LEXC,S), S > 0.
existeE(instituicao(D,I)) :-
    solucoes(
    ↪  (excecao(instituicao(D,II))),(excecao(instituicao(D,II))),LEXC
    ↪  ),
    comprimento(LEXC,S), S > 0.
existeE(ato(ID,D,IDU,IDS,C)) :-
    retract((excecao(ato(ID,D,IDU,IDS,X)):- X>=Y,X=<Z)),
    assert((excecao(ato(ID,D,IDU,IDS,X)):- X>=Y,X=<Z)).
existeE(ato(ID,data( D,M,A ),IDU,IDS,C)) :-
    retract((excecao(ato(ID,data( X,M,A ),IDU,IDS,C)):-
    ↪  X>=Y,X=<Z)),
    assert((excecao(ato(ID,data( X,M,A ),IDU,IDS,C)):-
    ↪  X>=Y,X=<Z)).
existeE(ato(ID,data( D,M,A ),IDU,IDS,C)) :-
    retract((excecao(ato(ID,data( D,X,A ),IDU,IDS,C)):-
    ↪  X>=Y,X=<Z)),
    assert((excecao(ato(ID,data( D,X,A ),IDU,IDS,C)):-
    ↪  X>=Y,X=<Z)).
existeE(ato(ID,data( D,M,A ),IDU,IDS,C)) :-
    retract((excecao(ato(ID,data( D,M,X ),IDU,IDS,C)):-
    ↪  X>=Y,X=<Z)),
    assert((excecao(ato(ID,data( D,M,X ),IDU,IDS,C)):-
    ↪  X>=Y,X=<Z)).
existeE(ato(ID,D,IDU,IDS,C)) :-
    solucoes( (excecao(
    ↪  ato(ID,DA,IDA,IDSA,CA))),(excecao(ato(ID,DA,IDA,IDSA,CA))),LEXC
    ↪  ),
    comprimento(LEXC,S), S > 0.
```

Bibliografia

[1] ANTONIOU, G. *Nonmonotonic Reasoning*. Artificial Intelligence Series. The MIT Press, 1997.

[2] CÉSAR ANALIDE, J. N. Representação de informação incompleta. *Actas de 1ª CAPSI, Conferência da Associação Portuguesa de Sistemas de Informação* (2000).

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

[4] DA COSTA LIMA, L. *Apoio à Tomada de Decisão em Grupo na Área da Saúde*. PhD thesis, Universidade de Trás-os-Montes e Alto Douro, 2009.