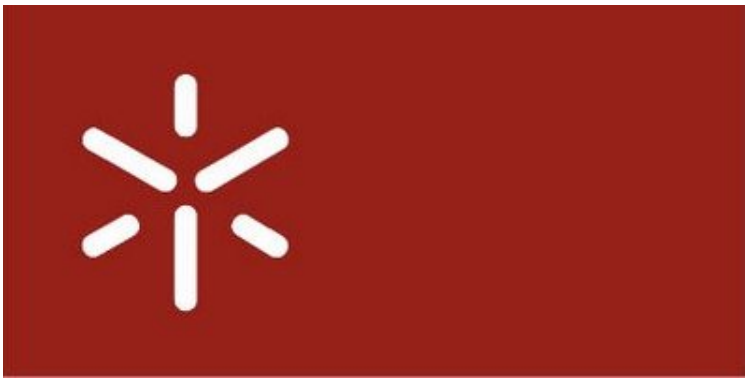

Universidade do Minho

Mestrado Integrado de Engenharia Informática

Sistemas de Representação Conhecimento e Raciocínio

Exercício 2 - Programação em lógica estendida e Conhecimento imperfeito

Trabalho realizado por:

Adriana Guedes	A74545
Guilherme Guerreiro	A73860
Marco Barbosa	A75278
Ricardo Certo	A75315

GRUPO 1

9 de Abril de 2017

Resumo

O presente relatório documenta o segundo exercício prático da Unidade Curricular de Sistemas de Representação Conhecimento e Raciocínio.

Nesta fase o objetivo será construir um mecanismo de representação de conhecimento com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde. O objetivo é demonstrar as funcionalidades subjacentes à programação em lógica estendida e à representação de conhecimento imperfeito, recorrendo à temática dos valores nulos. Em termos gerais, foi usada a linguagem PROLOG, esta que utiliza um conjunto de factos, predicados e regras de derivação de lógica.

Conteúdo

1 Introdução **3**
1.1 Motivação e Objetivos . . . 3
1.2 Estrutura do relatório . . . 3

2 Preliminares **4**
2.1 Representação de Informação Incompleta . . . 4
2.2 Sistema de inferência capaz de implementar os mecanismos de raciocínio de Programação em Lógica Estendida . . . 5

3 Descrição do trabalho e análise de resultados **6**
3.1 Base de conhecimento . . . 6
3.1.1 Utentes . . . 6
3.1.2 Atos Médicos . . . 6
3.1.3 Cuidados Prestados . . . 7
3.2 Conhecimento Positivo . . . 7
3.3 Conhecimento Negativo . . . 7
3.3.1 Conhecimento Negativo por falha . . . 7
3.3.2 Negação Forte . . . 8
3.4 Conhecimento Imperfeito . . . 8
3.4.1 Valor nulo tipo I . . . 8
3.4.2 Valor nulo tipo II . . . 8
3.4.3 Valor nulo tipo III . . . 9
3.5 Evolução do Conhecimento . . . 9
3.5.1 Adição do Conhecimento . . . 9
3.5.2 Remoção do Conhecimento . . . 9
3.6 Invariantes . . . 10
3.7 Outputs . . . 11
3.7.1 Exemplo de output de conhecimento perfeito . . . 11
3.7.2 Exemplo de output de conhecimento Imperfeito . . . 12
3.7.3 Exemplo de output de conhecimento Inexistente . . . 12
3.7.4 Exemplo de output de conhecimento Interdito . . . 12
3.7.5 Teste do invariante que não permite remover um cuidado se estiver associado a um ato . . . 12

4 Conclusão **14**

1 Introdução

A extensão à programação em lógica surge da necessidade de abandonar conceitos restritos obrigatoriamente associados à programação em lógica abordada no primeiro trabalho prático. Posto isto, possibilita o abandono do mundo fechado e do conhecimento perfeito, onde apenas se assume verdadeiro aquilo que é conhecido (o que a base de conhecimento contempla) e assume que não existem mais objetos do que aqueles presentes na base de conhecimento. Assim, consideramos três valores de verdade: verdadeiro, falso e desconhecido.

Para além das alterações referidas a cima, a extensão à programação em lógica introduz o conceito de negação forte, não excluído a existência da negação por falha na prova (predicado *nao*).

A negação forte possibilita a representação de conhecimento negativo e é mais completa em relação à negação fraca (negação por falha na prova), pois só declara o seu valor de verdade quando de facto existe prova para tal, já a negação fraca, perante falta de conhecimento considera, pelo pressuposto de mundo fechado que se este não existe então o seu valor é falso o que não é de todo correto para situações reais onde o pressuposto mundo fechado não é aplicável.

Apesar destas alterações na passagem de programação em lógica para a sua extensão os restantes conceitos mantêm-se, como é o caso do pressuposto dos nomes únicos.

1.1 Motivação e Objetivos

Depois da conclusão do primeiro exercício prático, pretendemos neste segundo continuar com o trabalho e objetivos definidos anteriormente. Assim, pretende-se aprofundar a consolidar os conteúdos abordados nas aulas, desenvolvendo um sistema de representação de conhecimento e raciocínio o mais próximo da realidade possível.

1.2 Estrutura do relatório

O presente relatório é constituído por 4 partes. A primeira parte corresponde à introdução do segundo exercício prático. Na segunda parte iremos apresentar conceitos que consideramos importantes o leitor saber para entender o exercício bem como a sua resolução. A terceira secção é dedicada à resolução do problema e a explicação da mesma. Por fim, apresentamos a conclusão desta fase do trabalho.

2 Preliminares

2.1 Representação de Informação Incompleta

Mesmo antes de abordarmos o problema proposto e as linguagem de programação lógica *PROLOG* propriamente dita, convém debruçarmos e focarmos um pouco da nossa atenção sobre os pressupostos em que este tipo de linguagens se baseiam e fundamentam. Assim, tomemos os dois tipos de armazenamento de informação utilizados por um sistema computacional:

- **Base de Dados**

  Uma base de dados que sirva de suporte a uma linguagem de programação lógica/manipulação de informação confere-lhe as seguintes características: toda a informação que não se encontra referenciada na base de dados é considerada falsa, partindo por isso de um *Pressuposto de Mundo Fechado*; por outro lado, para além de duas constantes diferentes representarem duas entidades necessariamente diferentes nesse mesmo universo de representação (*Pressuposto dos Nomes Únicos*), parte-se também do princípio da não existência de mais objetos nesse universo, para além daqueles que se encontram representados por constantes na base de dados sobra a qual a linguagem se suporta (*Pressuposto do Domínio Fechado*).

- **Sistema de Representação de Conhecimento**

  Um Sistema de Representação de Conhecimento que sirva de suporte a uma linguagem de programação lógica/manipulação de informação confere-lhe as seguintes características: é considerada a possibilidade de existência de outros factos verdadeiros para além daqueles que se encontram enunciados na base de conhecimento, havendo por isso um *Pressuposto de Mundo Aberto*; por outro lado e à semelhança do que acontece numa linguagem de manipulação de informação assente numa Base de Dados, existe um *Pressuposto de Nomes Únicos*, na medida em que duas constantes diferentes representarem duas entidades necessariamente diferentes nesse mesmo universo de representação; por fim, um Sistema de Representação de Conhecimento considera a hipótese de existirem mais objetos no universo da representação, para além daqueles que já se encontram especificados na base de conhecimento (*Pressuposto de Mundo Aberto*).

Após esta breve abordagem dos pressupostos inerentes aos dois possíveis sistemas sobre os quais uma linguagem de manipulação de informação poder-se-á alicerçar, podemos compreender facilmente que a Programação em Lógica clássica (por ser baseada em pressupostos tais como o de um *Mundo Fechado* e de *Domínio Fechado*) apresenta algumas limitações ao tipo de processamento necessário para tratar informação incompleta. Os Sistemas de Representação de Conhecimento reais, no entanto, não trabalham com estes pressupostos, pois contextualizam situações em que não existe informação completa acerca do problema em questão, sendo por isso importante representa-las para além do *verdadeiro* e do *falso*.

Assim, a solução encontrada parte por se utilizar a Programação em Lógica como ferramenta para a representação de conhecimento, mas agora introduzindo-se uma extensão a esta forma de representação, de modo a que seja possível representar-se o *desconhecido*. Este tipo de extensão faz-se pela introdução, na linguagem de representação, de dois tipos de negação: a negação por falha na prova, denotada pelo termo *não*, e a negação forte ou clássica, denotada por '-'.

Com a representação de informação negativa de forma explícita, ao contrário de se assumir que o que não se conhece é falso (ou seja, um mundo fechado), passamos a representar na base de conhecimento algo que sabemos ser falso.

2.2 Sistema de inferência capaz de implementar os mecanismos de raciocínio de Programação em Lógica Estendida

Ao adicionar-se a capacidade para representação e raciocínio sobre informação incompleta a um sistema, a sua base de conhecimento passa a poder descrever o mundo real de forma muito mais correta e a interpretação de uma questão Q que lhe seja colocada pode ser definida da seguinte forma:

- **Verdadeira**: quando for possível provar a questão Q na base de conhecimento;
- **Falsa**: quando for possível provar a falsidade de uma questão -Q na base de conhecimento;
- **Desconhecida**: quando não for possível provar a questão Q nem a questão -Q

Este tipo de interpretação/inferência a uma questão colocada num programa em Lógica Estendida advém da seguinte extensão do predicado *demo*:

```
demo( Questao,verdadeiro ) :- Questao.

demo( Questao, falso ) :- -Questao.

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

Assim, como podemos perceber pelo código *PROLOG* apresentado anteriormente, o interpretador, quando interrogado sobre um predicado, apresenta 3 respostas possíveis: verdadeiro, falso ou desconhecido. Caso exista prova da veracidade da questão na base de conhecimento, a resposta será *verdadeira*, enquanto que se houver uma negação forte da questão na base de conhecimento a resposta será *falsa*. Por fim, na situação em que a questão nem se encontra como verídica na base de conhecimento nem explicitamente demarcada como falsa (através da negação forte), ela cai no domínio do *desconhecido*, ou seja , não se pode ser conclusivo.

3 Descrição do trabalho e análise de resultados

3.1 Base de conhecimento

A base de conhecimento é constituída por um conjunto de axiomas acerca de atos médicos prestados a dados utentes numa determinada instituição. Durante a realização desta segunda etapa ouve necessidade de efetuar alterações na nossa base de conhecimento em relação á primeira etapa, pois para esta fase adicionamos um novo tipo de conhecimento á nossa base que é o conhecimento imperfeito. Este tipo de conhecimento é divido em três diferentes tipos de valores nulos: do tipo desconhecido, do tipo desconhecido de um conjunto dado de valores e do tipo não permitido. Sendo assim a nossa base de conhecimento ficou capaz de representar conhecimento verdadeiro , falso e desconhecido.

3.1.1 Utentes

Base de conhecimento respetiva ao predicado utente:

```
% Extensao do predicado utente : IdUt, Nome, Idade, Rua, Cidade, Contacto -> { V, F , D }

utente( 1,'Carlos',35,'Rua D.Pedro V','Braga','253456789').
utente( 2,'Joao',12,'Rua da Ramada','Guimaraes','929876543').
utente( 3,'Julio',89,'Rua das Victorias','Guimaraes','935436789').
utente( 4,'Ana',25,'Rua Conde Almoster','Lisboa','913456789').
utente( 5,'Carolina',50,'Rua do Caires','Braga','253987654').
utente( 6,'Joana',26,'Av.da Boavista','Porto','961234567').
utente( 7,'Fernando',65,'Rua do Loureiro','Viana do Castelo','966668569').
utente( 8,'Rute',18,'Av. da Liberdade','Braga','916386423').
utente( 9,'Maciel',45,rua_desconhecida,cidade_desconhecida,'935731290').
utente( 10,'Filipa',35,rua_desconhecida,cidade_desconhecida,contacto_desconhecido).
utente( 11,'Mauro',76,'Rua Gil Vicente','Montalegre',contacto_desconhecido).
utente( 12,'Laura',90,'Rua Fernando Mendes',cidade_desconhecida,'244000045').
utente( 13,'Jaime',50,rua_desconhecida,'Barcelos','914768180').
utente( 14,'Lourenço',idade_desconhecida,'Rua da Boavista','Guimaraes','926306127').
utente( 15,nome_desconhecido,16,'Rua Monsenhor de Melo','Vilamoura','936150873').
```

3.1.2 Atos Médicos

Base de conhecimento respetiva ao predicado atos:

```
% Extensao do predicado ato_medico:  Data, IdUt, IdServ, CorPulseira, Médico, Custo -> { V, F, D }

atos( '02-01-17', 15, 10, 'Verde', 'Dra.Sara', 17.5).
atos( '24-02-17', 8, 4, 'Sem_pulseira', 'Dr.Mourao', 4).
atos( '25-02-17', 1, 2, 'Sem_pulseira', 'Dr.Barroso', 12).
atos( '28-01-17', 13, 9, 'Laranja', 'Dr.Tomas', 5).
atos( '10-02-17', 4, 3, 'Sem_pulseira', 'Dr.Falcão', 20).
atos( '19-03-17', 3, 8, 'Amarela', 'Dr.Bones', 50).
atos( '11-01-17', 1, 1, 'Sem_pulseira', 'Dr.Pardal', 2).
atos( '12-02-17', 5, 8, 'Sem_pulseira', 'Dra.Teresa', 13.75).
atos( '20-03-17', 11, 11, 'Sem_pulseira', 'Dra.Marta', 13).
atos( '27-01-17', 2, 7, 'Amarela', 'Dr.Pedro Martins', 11).
atos( '01-01-17', 6, 6, 'Laranja', 'Dr.Reveillon', 50).
```

```
atos( '03-03-17', 3, 1, 'Sem_pulseira', 'Dra.Candida', 45).
atos( '08-03-17', 9, 9, 'Vermelha', 'Dr.Lima', 50).
atos( '14-02-17', 14, 7, 'Amarela', 'Dra.Mafalda', 23).
atos( '30-01-17', 7, 5, cor_desconhecida, 'Dr.Quimio', custo_desconhecido).
atos( '01-02-17', 1, 6, 'Verde', 'Dra.Luisa', custo_desconhecido).
atos( '01-03-17', 10, 6, 'Verde', medico_desconhecido, 25.5).
atos( '10-03-17', 12, 11, cor_desconhecida, 'Dr.Luis', 12.50).
```

3.1.3 Cuidados Prestados

Base de conhecimento respetiva ao predicado cuidado_prestado:

```
% Extensao do predicado cuidado_prestado: IdServ, Descrição, Instituição, Cidade -> { V, F, D }

cuidado_prestado( 1,'Pediatria','Hospital Privado de Braga','Braga').
cuidado_prestado( 2,'Cardiologia','Hospital de Braga','Braga').
cuidado_prestado( 3,'Ortopedia','Hospital de Braga','Braga').
cuidado_prestado( 4,'Oftalmologia','Hospital de Braga','Braga').
cuidado_prestado( 5,'Oncologia','IPO','Porto').
cuidado_prestado( 6,'Urgência','Hospital de Santa Maria','Porto').
cuidado_prestado( 7,'Urgência','Hospital da Luz','Guimaraes').
cuidado_prestado( 8,'Neurologia','Centro Hospitalar Sao Joao','Porto').
cuidado_prestado( 9,'Urgência','Hospital de Braga','Braga').
cuidado_prestado( 10,'Urgência','Hospital Lusiadas','Faro').
cuidado_prestado( 11,'Otorrinolaringologia','Hospital de Vila Real','Vila Real').
cuidado_prestado( 12,'Podologia',instituicao_desconhecida,'Vila Real').
```

3.2 Conhecimento Positivo

A representação do conhecimento positivo é feito baseado nos factos, como podemos constatar no seguinte exemplo:

```
cuidado_prestado( 1,'Pediatria','Hospital Privado de Braga','Braga').
atos( '10-02-17', 4, 3, 'Sem_pulseira', 'Dr.Falcão', 20).
utente( 1,'Carlos',35,'Rua D.Pedro V','Braga','253456789').
```

3.3 Conhecimento Negativo

O conhecimento negativo pode ser representado de duas formas, ou utilizando a negação por falha ou utilizando a negação forte.

3.3.1 Conhecimento Negativo por falha

```
-utente(Id,Nome,Idade,Morada) :-
    nao(utente(Id,Nome,Idade,Morada)) ,
    nao(excecao(utente(Id,Nome,Idade,Morada))).

-atos(D,IdUt,IdServ,C,M,Ct) :-
     nao(atos(D,IdUt,IdServ,C,M,Ct)) ,
```

```
    nao(excecao(atos(D,IdUt,IdServ,C,M,Ct))).


-cuidado_prestado(Id,D,I,C) :-
    nao(cuidado_prestado(Id,D,I,C)) ,
    nao(cuidado_prestado(excecao(Id,D,I,C))).
```

3.3.2 Negação Forte

```
-utente( 19,'Joaquina',75,'Av.da Liberdade','Lisboa','962525258').

-atos( '05-04-17', 10, 10, 'Vermelho', 'Dr.Luis', 29 ).

-cuidado_prestado( 13,'Radiografia','Clínica Hospitalar de Faro','Faro').
```

3.4 Conhecimento Imperfeito

O Conhecimento Imperfeito caracteriza-se pela utilização de valores nulos de todos os tipos. Os valores nulos servem para nos ajudar na enumeração de casos, para os quais queremos diferenciar entre situações em que as respostas deverão ser do tipo verdadeiro ou falso ou desconhecidas.

3.4.1 Valor nulo tipo I

Para o caso dos utentes usamos o axioma **utente(I,N,Idd,R,C,Con)**, para representarmos um dado utente que possua uma idade que não seja conhecida na nossa base de conhecimento, e para isso usamos os valores **nulos do tipo desconhecido**. De seguida apresentamos um exemplo para os atos , para os utentes e para os cuidados prestados:

```
% idade desconhecida
excecao(utente(I,N,Idd,R,C,Con)) :-
utente(I,N,idade_desconhecida,R,C,Con).

% Cor Pulseira desconhecida
excecao(atos(D,IdUt,IdServ,C,M,Ct)) :-
 atos(D,IdUt,IdServ,cor_desconhecida,M,Ct).

% Instituição desconhecida
excecao(cuidado_prestado(Id,D,I,C)) :-
cuidado_prestado(Id,D,instituicao_desconhecida,C).
```

3.4.2 Valor nulo tipo II

O valor nulo do tipo II pode ser definido como o valor nulo impreciso, pois sabemos que uma das opções é verdadeira mas não sabemos qual ou sabemos que a opção é verdadeira dentro de uma gama de valores. Por exemplo, para o ato realizado no dia 07/04/2017 sabemos que teve um preço compreendido entre 15 e 35 euros. De seguida apresentamos um exemplo para os atos , para os utentes e para os cuidados prestados:

```
% O ato realiado no dia 07/04/2017 custou entre 15 e 35 euros
```

```
atos( '07-04-17', 2, 1, 'Amarela', 'Dr.Mike', C) :- C >= 15 , C =< 35.

% um utente com o mesmo ID pode estar registado com diferentes nomes do seu nome completo
excecao(utente( 18,'Carlos',33,'Avenida 25 de Abril','Santarém','935694789')).
excecao(utente( 18,'Carlos Alberto',33,'Avenida 25 de Abril','Santarém','935694789')).

% um cuidado pode ter prestado em dois hospitais distintos
excecao(cuidado_prestado( 13,'Pediatria','Hospital Privado do Algarve','Faro')).
excecao(cuidado_prestado( 13,'Pediatria','Hospital de Faro','Faro')).
```

3.4.3 Valor nulo tipo III

O valor nulo do tipo III pode ser definido como o valor nulo não permitido, pois na nossa base de dados podemos possuir parametros em que não se permite conhecer por exemplo um contacto de uma utente ou até mesmo um ato médico em que não se pode conhecer o médico que o realizou. De seguida apresentamos um exemplo para os atos e para os utentes.

```
% utente 16 tem contacto que ninguém pode conhecer

utente( 16,'Djalo',30,'Rua Tangente II','Arouca',contacto_desconhecido).
excecao(utente(I,N,Id,R,C,Ct)) :- utente(I,N,Id,R,C,contacto_desconhecido).
nulo(contacto_desconhecido).
+utente(16,'Djalo',30,'Rua Tangente II','Arouca',contacto_desconhecido) :: (solucoes((16,'Djalo',30,'
comprimento( S,N ) ,
N == 0).


% ato do dia 1-04-2017 tem um medico que ninguém pode conhecer
atos( '01-04-17', 2, 10, 'Verde', medico_desconhecido, 27.5).
excecao(atos(D,IdUt,IdS,C,Dt,P)) :- atos(D,IdUt,IdS,C,medico_desconhecido,P).
nulo(medico_desconhecido).
+atos(D,IdUt,IdS,C,Dt,P) :: ( solucoes((D,IdUt,IdS,C,Dt,P), (atos('01-04-17', 2, 10, 'Verde', Dt, 27.
comprimento( S,N ),
N == 0).
```

3.5 Evolução do Conhecimento

Para que fosse possíivel a inserção e remoção de conhecimento na base desenvolvemos o teorema evolucao, teorema este que verifica os invariantes que nós definimos, pois só assim podemos garantir que não é adicionada á base informação repetida, e caso não seja informação repetida adiciona na base.

3.5.1 Adição do Conhecimento

```
registaUtentes(ID,NM,I,RU,CDD,CNT) :-  evolucao(utente(ID,NM,I,RU,CDD,CNT)).
registaCuidados(ID,D,I,C) :- evolucao(cuidado_prestado(ID,D,I,C)).
registaAtos(D,IDUT,IDS,CP,MDC,C) :- evolucao(atos(D,IDUT,IDS,CP,MDC,C)).
```

3.5.2 Remoção do Conhecimento

O ato de remoção de um utente da base de conhecimento têm uma implicação que é só podermos remover um utente aquando todos os seus atos forem também removidos da base de conhecimento.

```
removeUtentes(U) :- solucoes((D,U,IDS),atos(D,U,IDS,_,_,_),R),
          removeTodosAtos(R),
          retroceder(utente(U,N,I,RU,CDD,CNT)).

removeTodosAtos([]).
removeTodosAtos([(D,IDUT,IDS)]) :- removeAtos(D,IDUT,IDS).
removeTodosAtos([(D,IDUT,IDS)|As]) :- removeAtos(D,IDUT,IDS),
                        removeTodosAtos(As).


removeCuidados(I) :- retroceder(cuidado_prestado(I,D,C,Cid)).


removeAtos(D,IDUT,IDS) :- retroceder(atos(D,IDUT,IDS,_,_,_)).
```

3.6 Invariantes

Tal como referido anteriormente, foi necessário criarmos invariantes para garantir que não era adicionada à base informação repetida.

```
% ------------------------------------------------------------
% Invariante Estrutural para utente:
% (não permite a inserção de conhecimento repetido, utentes com o mesmo ID)

+utente(I,Nome,IDD,RU,CDD,CNT) :: (solucoes(I,(utente(I,_,_,_,_,_)),L),
                        comprimento(L,N),
                        N == 1).


% ------------------------------------------------------------
% Invariante Estrutural para cuidado_prestado:
% (não permite a inserção de conhecimento repetido, cuidados prestados com o mesmo ID)

+cuidado_prestado(ID,D,I,X) ::(solucoes(ID,(cuidado_prestado(ID,_,_,_)),L),
                                comprimento(L,N),
                                 N == 1).

% ------------------------------------------------------------
% Invariante Estrutural para cuidado_prestado:
%(não permite a inserção de conhecimento repetido, atos com o mesmo ID)

+atos(D,IDUT,IDS,CP,MDC,C) :: (solucoes((D,IDUT,IDS),(atos(D,IDUT,IDS,_,_,_)),L),
                               comprimento(L,N),
                               N == 1).

% ------------------------------------------------------------
% Invariante que certifica a existência de um ID de utente e de um ID serviço para um ato

+atos(D,IDUT,IDS,CP,MDC,C) :: (utente(IDUT,_,_,_,_,_),
```

```
                                cuidado_prestado(IDS,_,_,_)).
```

Acrescentamos também um invariante para que não seja possível remover um cuidado se este tiver um ato a si associado e acrecentámos outro que não permite a remoção de um ato médico caso esteja um utente associado a este.

```
%-----------------------------------------------------------------------------


% não é possível remover um cuidado prestado se tiver um ato associado

-cuidado_prestado(ID,D,I,X) :: (solucoes(ID,atos(_,_,ID,_,_,_),L),
                                comprimento(L,N),
                                N==0).

% não é possível remover um ato se tiver um utente associado

-atos(D,IdUt,IdServ,C,M,Ct) :: (solucoes(IdUt,utente(IdUt,_,_,_),L),
                                comprimento(L,N),
                                N==0).
```

3.7 Outputs

```
| ?- listing(utente).

utente(1, 'Carlos', 35, 'Rua D.Pedro V', 'Braga', '253456789').
utente(2, 'Joao', 12, 'Rua da Ramada', 'Guimaraes', '929876543').
utente(3, 'Julio', 89, 'Rua das Victorias', 'Guimaraes', '935436789').
utente(4, 'Ana', 25, 'Rua Conde Almoster', 'Lisboa', '913456789').
utente(5, 'Carolina', 50, 'Rua do Caires', 'Braga', '253987654').
utente(6, 'Joana', 26, 'Av.da Boavista', 'Porto', '961234567').
utente(7, 'Fernando', 65, 'Rua do Loureiro', 'Viana do Castelo', '966668569').
utente(8, 'Rute', 18, 'Av. da Liberdade', 'Braga', '916386423').
utente(9, 'Maciel', 45, rua_desconhecida, cidade_desconhecida, '935731290').
utente(10, 'Filipa', 35, rua_desconhecida, cidade_desconhecida, contacto_desconhecido).
utente(11, 'Mauro', 76, 'Rua Gil Vicente', 'Montalegre', contacto_desconhecido).
utente(12, 'Laura', 90, 'Rua Fernando Mendes', cidade_desconhecida, '244000045').
utente(13, 'Jaime', 50, rua_desconhecida, 'Barcelos', '914768180').
utente(14, 'Lourenço', idade_desconhecida, 'Rua da Boavista', 'Guimaraes', '926306127').
utente(15, nome_desconhecido, 16, 'Rua Monsenhor de Melo', 'Vilamoura', '936150873').
utente(19, 'Joaquina', idade_desconhecida, 'Av.da Liberdade', 'Lisboa', '962525258').
utente(16, 'Djalo', 30, 'Rua Tangente II', 'Arouca', nxpto).
utente(17, 'Maria de Jesus', ixpto, 'Rua dos Videiras', 'Estoril', '922225014').
```

3.7.1 Exemplo de output de conhecimento perfeito

```
| ?- demo(utente(1,'Carlos',35,'Rua D.Pedro V','Braga','253456789'),R).
R = verdadeiro ?
yes
```

3.7.2 Exemplo de output de conhecimento Imperfeito

```
| ?- demo(utente(9,'Maciel',45,'Rua dos Amores','Braga', '935731290'),R).
R = desconhecido ?
yes
```

3.7.3 Exemplo de output de conhecimento Inexistente

```
| ?- demo(utente(9,'Hugo',45,a,a,'a'), R).
R = desconhecido ?
yes
```

3.7.4 Exemplo de output de conhecimento Interdito

Um utente registado tem uma idade incerta mas sabemos que não 75 anos

```
| ?- demo(utente( 19,'Joaquina',75,'Av.da Liberdade','Lisboa','962525258'),R).
R = falso ?
yes
```

3.7.5 Teste do invariante que não permite remover um cuidado se estiver associado a um ato

```
| ?- listing(atos).
atos('02-01-17', 15, 10, 'Verde', 'Dra.Sara', 17.5).
atos('24-02-17', 8, 4, 'Sem_pulseira', 'Dr.Mourao', 4).
atos('25-02-17', 1, 2, 'Sem_pulseira', 'Dr.Barroso', 12).
atos('28-01-17', 13, 9, 'Laranja', 'Dr.Tomas', 5).
atos('10-02-17', 4, 3, 'Sem_pulseira', 'Dr.FalcÃ£o', 20).
atos('19-03-17', 3, 8, 'Amarela', 'Dr.Bones', 50).
atos('11-01-17', 1, 1, 'Sem_pulseira', 'Dr.Pardal', 2).
atos('12-02-17', 5, 8, 'Sem_pulseira', 'Dra.Teresa', 13.75).
atos('20-03-17', 11, 11, 'Sem_pulseira', 'Dra.Marta', 13).
atos('27-01-17', 2, 7, 'Amarela', 'Dr.Pedro Martins', 11).
atos('01-01-17', 6, 6, 'Laranja', 'Dr.Reveillon', 50).
atos('03-03-17', 3, 1, 'Sem_pulseira', 'Dra.Candida', 45).
atos('08-03-17', 9, 9, 'Vermelha', 'Dr.Lima', 50).
atos('14-02-17', 14, 7, 'Amarela', 'Dra.Mafalda', 23).
atos('30-01-17', 7, 5, cor_desconhecida, 'Dr.Quimio', custo_desconhecido).
atos('01-02-17', 1, 6, 'Verde', 'Dra.Luisa', custo_desconhecido).
atos('01-03-17', 10, 6, 'Verde', medico_desconhecido, 25.5).
atos('10-03-17', 12, 11, cor_desconhecida, 'Dr.Luis', 12.5).
atos('05-04-17', 10, 10, cor_desconhecida, 'Dr.Luis', 29).
atos('01-04-17', 2, 10, 'Verde', medxpto, 27.5).

yes
| ?- removeCuidados(1).
no

| ?- registaCuidados(14,'Dermatologia','Hospital de Braga','Braga').
yes
| ?- removeCuidados(14).
```

```
yes
```

4 Conclusão

Em jeito de conclusão, a ênfase recai maioritariamente sobre a problemática da representação de informação inconclusiva/desconhecida, ou seja, a transição de uma base de conhecimento que utiliza a Programação em Lógica Clássica como forma de representação de conhecimento (que fora desenvolvida no primeiro trabalho) para uma outra base que se suporta numa Programação em Lógica Estendida para representação do conhecimento.

Esta transição assenta, sobretudo, na introdução de dois tipos de negação na linguagem de representação utilizada que foi o *PROLOG* : por um lado, a negação por falha na prova, representada pelo termo 'nao' e, por outro lado, a negação forte , representada por '-'. Desta forma, conseguimos efetuar a distinção entre situações que são falsas, situações verdadeiras e situações para as quais não existe prova de que sejam verdadeiras.

Para os casos em que se trata de informação incompleta, estão também definidos 3 tipos de valores, designados por *Valor Nulo Tipo I, Valor Nulo Tipo II* e *Valor Nulo Tipo III*, respetivamente: aqueles valores que são incertos (são desconhecidos), os que são imprecisos (desconhecidos mas dentro de uma gama de valores) e os que são interditos (não existe qualquer hipótese da base de conhecimento tirar ilações ou vir a descobrir certas informações). Na prática, encontram-se determinadas implementações no Programa em Lógica Estendida que foi desenvolvido que procuram lidar e tratar destes tipos de valores (quando aparecem), e, após este breve resumo acerca do trabalho desenvolvido, conseguimos abranger o aspeto do conhecimento imperfeito em *PROLOG*.