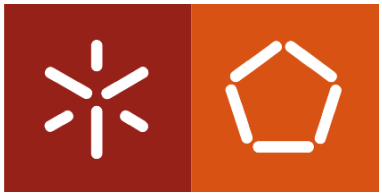


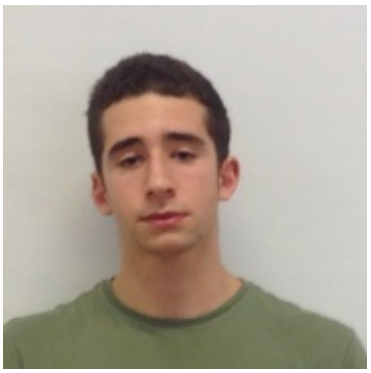
UNIVERSIDADE DO MINHO
Mestrado Integrado em Engenharia Informática
Sistemas de Representação de Conhecimento e Raciocínio

TRABALHO PRÁTICO 2

Programação em lógica estendida e conhecimento imperfeito.

Grupo 31:

Gonçalo Pereira : A74413



António Silva : A73827

André Diogo : A75505

Braga, 9 de Abril de 2017

Conteúdo

Introdução

O objetivo deste trabalho terá sido o de extensão do conhecimento enunciado no primeiro trabalho prático, através da linguagem `PROLOG`, representando conhecimento imperfeito já com recorrência a valores nulos e/ou outros mecanismos próprios para tal. De modo a que tal fosse possível, recorremos à extensão do conhecimento para abarcar o valor de "desconhecido", usado na eventualidade não se saber o valor de verdade:

- Utente, caracterizado por `#IDUtente, Nome, Idade, Morada -> {V, F, D}`
- Cuidado prestado, caracterizado por `#IDServiço, Descrição, Instituição, Cidade -> {V, F, D}`
- Ato médico, caracterizado por `Data, #IDUtente, #IDServiço, Custo -> {V, F, D}`

Como tal, este sistema deverá respeitar todas as noções definidas para o trabalho prático anterior (limitação pelos invariantes), assim como deverá também fazer uma correta representação de conhecimento imperfeito, através de valores nulos (e não só), de modo a representar, virtual e corretamente, um universo de prestação de cuidados de saúde a utentes registados através de atos médicos.

Uma vez mais, interpretou-se o enunciado de tal forma que se considerou que um ato médico potencia múltiplos cuidados prestados, mas que estes cuidados prestados tem uma identificação de serviço igual entre eles para o mesmo ato médico e estes cuidados são também únicos ao utente a quem os cuidados foram prestados. Os utentes serão também únicos, identificados pelo seu identificador e o ato médico terá então para cada identificador de serviço apenas um possível utente para esse serviço.

Considerou-se ainda que um ato médico pode ser efetuado parcialmente em múltiplos estabelecimentos, caso uma pessoa seja transferida para outro estabelecimento para prestar cuidados que não são possíveis no primeiro. Elucidar-se-á tal sistema nas seguintes páginas.

Tratamento da conjunção de predicados extendidos a conhecimento imperfeito

De modo a seguir a convenção utilizada nas aulas práticas, e com o intuito de estender a capacidade de raciocínio sobre os nossos predicados, decidimos exprimir a conjunção de valores de verdade agora com o desconhecido da seguinte forma:

	Verdadeiro	Falso	Desconhecido
Verdadeiro	Verdadeiro	Falso	Desconhecido
Falso	Falso	Falso	Falso
Desconhecido	Desconhecido	Falso	Desconhecido

Representar conhecimento positivo e negativo

Para representarmos conhecimento positivo e negativo, adaptámos a convenção de se utilizar o sinal de subtração para representação de conhecimento negativo, sendo que, na representação de conhecimento positivo, fizemos uso do predicado utilizado no primeiro trabalho prático, sem quaisquer caracteres adicionais.

- POSITIVO:

```
cuidadoprestado(2, 'Cirurgia', 'Hospital dos Covoes', 'Coimbra').
```

- NEGATIVO:

```
-cuidadoprestado(2, 'Cirurgia', 'Hospital dos Covoes', 'Braga').
```

Representar casos de conhecimento imperfeito, pela utilização de valores nulos de todos os tipos estudados

Na representação de casos de conhecimento imperfeito, adotámos a convenção utilizada nas aulas práticas, através da utilização de valores nulos (e não só), de modo a explicitar os três tipos de conhecimento de forma correta:

- Conhecimento Incerto:

```
atomedico(desconhecido2,3,87,23.95).

excecao(atomedico(P,3,87,23.95)) :-
    atomedico(desconhecido2,3,87,23.95).
```

- Conhecimento Impreciso:

```
excecao(cuidadoprestado(3,'Vacinação Tétano','USF Gualtar','Braga')).
```

- Conhecimento Interdito:

```
cuidadoprestado(3,'Nascimento',desconhecido1,'Kiev').

excecao(cuidadoprestado(3,'Nascimento',P,'Kiev')) :-
    cuidadoprestado(3,'Nascimento',desconhecido1,'Kiev').

nulo(desconhecido1).
```

Manipular invariantes que designem restrições à inserção e à remoção de conhecimento do sistema

A manipulação de invariantes foi feita do mesmo modo que no trabalho prático anterior, isto é, a não permissão de evolução na eventualidade de já existir um duplicado, ou a não regressão de conhecimento caso algo dependesse da existência de um outro predicado. Porém, e como existiu um desenvolvimento de maior escala de modo a abranger os valores desconhecidos, utilizámos, também, certos predicados para refletir sobre o conhecimento imperfeito e considerar a existência de valores nulos.

Foram adicionados invariantes também para os casos de inserção de conhecimento certo mas negativo.

Exemplificando, para os `cuidadosprestados`:

```
+cuidadoprestado(ID,Serv,Estab,Local) :: (
  nao(-cuidadoprestado(ID,Serv,Estab,Local)),
  nao(cuidadoprestado(ID,Serv,Estab,Local)),
  naonulocuidadoprestado(ID)
).

naonulocuidadoprestado(ID) :-
  cuidadoprestado(ID,ServB,EstabB,LocalB),
  nao(nulo(ServB)),
  nao(nulo(EstabB)),
  nao(nulo(LocalB)).

naonulocuidadoprestado(ID) :-
  nao(cuidadoprestado(ID,ServB,EstabB,LocalB)).

+(-cuidadoprestado(ID,Serv,Estab,Local)) :: (
  nao(-cuidadoprestado(ID,Serv,Estab,Local)),
  nao(cuidadoprestado(ID,Serv,Estab,Local)),
  naonulocuidadoprestado(ID)
).
```

Este invariante, por exemplo, força a que um cuidado prestado quer seja conhecimento positivo ou negativo só possa ser inserido caso não haja conhecimento certo que o contradiga, nem conhecimento certo que tornasse a inserção numa repetição. Não pode ainda haver um cuidado prestado que interdite a sua introdução.

Lidar com a problemática da evolução do conhecimento, criando os procedimentos adequados

Esta tarefa, de carácter mais árduo e verboso, requeria a representação de cada um dos casos práticos, de modo a remeter para todas e quaisquer possibilidades de existência de valor nulo nos predicados. Como tal, tratamos cada caso individualmente,

remetendo para o teste da sua existência e verificação das exceções, não esquecendo o tipo de conhecimento que se trataria na evolução.

Como cada predicado tem as suas próprias caraterísticas e reflete um conhecimento de caráter necessariamente diferente de o representado pelos demais predicados, tivemos de tratar os casos de exceção por extensão para cada um deles.

Para tal, utilizámos predicados de evolução específicos para cada tipo de exceção possível de se inserir para cada um dos três predicados que compõem a nossa base de conhecimento.

De modo semelhante também fabricámos predicados de regressão específicos para cada tipo de exceção possível para cada um dos três predicados referidos.

Finalmente, na evolução de conhecimento certo acrescentámos um predicado adicional que remove as exceções que são eliminadas por virtude do acréscimo de conhecimento certo.

Como alguns dos predicados lidam com possibilidade de haver menos incerteza ou mais incerteza, menos imprecisão ou maior imprecisão e interdições parciais além da mera representação de um predicado como inteiramente incerto,impreciso ou interdito, alguns dos predicados tornam-se tão verbosos e naturalmente expressos por extensão que se verificou praticamente inútil o isolamento de verificações comuns para invariantes sobre inserção de conhecimento imperfeito. Isto iria requerer ainda mais predicados específicos para cada um deles, e iterar além disso sobre cada uma destas verificações que em regra nem constituem a parte mais extensa do predicado (lidam estas partes mais extensas frequentemente com a remoção/substituição de outras exceções já existentes).

Não considerámos a possibilidade de evoluir conhecimento impreciso definido por compreensão pois torna-se incrivelmente difícil testar que conhecimento já existente incide sobre a mesma gama de valores e implica a possível regressão de muito conhecimento, com as consequências que isso traz sobre a verificação da consistência da base de conhecimento.

```
%Inserir um valor de utente impreciso requer que não haja conhecimento negativo igual
%ao que se quer inserir, não pode já existir este valor impreciso e não pode
%haver uma interdição a mexer com este ID. Se houver um valor incerto, a imprecisão
%remove-o.

evolucaoImpreciso(utente(ID,Nome,Idade,Morada)) :-
  nao(-utente(ID,Nome,Idade,Morada)),
  nao(excecao(utente(ID,Nome,Idade,Morada))),
  testaNulo(utente(ID,Nome,Idade,Morada)).

testaNulo(utente(ID,Nome,Idade,Morada)) :-
```

```
  utente(ID,NomeB,IdadeB,MoradaB),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(nulo(NomeB)),
  nao(nulo(IdadeB),
  nao(nulo(MoradaB))),
  clause(excecao(utente(ID,NomeB,IdadeB,MoradaB)),B,R),
  retract(utente(ID,NomeB,IdadeB,MoradaB)),
  retract((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-B)),
  assert(excecao(utente(ID,Nome,Idade,Morada))).

testaNulo(utente(ID,Nome,Idade,Morada)) :-
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  assert(excecao(utente(ID,Nome,Idade,Morada))).


%Inserir um valor de utente interdito necessita que não haja incerteza nem imprecisão
%nem um utente já existente.
evolucaoNulo(utente(ID,Nulo,Idade,Morada),Nulo) :-
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(excecao(utente(ID,NomeC,IdadeC,MoradaC))),
  assert((excecao(utente(ID,NomeD,IdadeD,MoradaD)):- utente(ID,Nulo,Idade,Morada))),
  assert(utente(ID,Nulo,Idade,Morada)),
  assert(nulo(Nulo)).

evolucaoNulo(utente(ID,Nome,Nulo,Morada),Nulo) :-
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(excecao(utente(ID,NomeC,IdadeC,MoradaC))),
  assert((excecao(utente(ID,NomeD,IdadeD,MoradaD)):- utente(ID,Nome,Nulo,Morada))),
  assert(utente(ID,Nome,Nulo,Morada)),
  assert(nulo(Nulo)).

evolucaoNulo(utente(ID,Nome,Idade,Nulo),Nulo) :-
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(excecao(utente(ID,NomeC,IdadeC,MoradaC))),
  assert((excecao(utente(ID,NomeD,IdadeD,MoradaD)):- utente(ID,Nome,Idade,Nulo))),
  assert(utente(ID,Nome,Idade,Nulo)),
  assert(nulo(Nulo)).

evolucaoNulo(utente(ID,Nulo,Nulo,Morada),Nulo) :-
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(excecao(utente(ID,NomeC,IdadeC,MoradaC))),
  assert((excecao(utente(ID,NomeD,IdadeD,MoradaD)):- utente(ID,Nulo,Nulo,Morada))),
  assert(utente(ID,Nulo,Nulo,Morada)),
  assert(nulo(Nulo)).

evolucaoNulo(utente(ID,Nulo,Idade,Nulo),Nulo) :-
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(excecao(utente(ID,NomeC,IdadeC,MoradaC))),
```

```
  assert((excecao(utente(ID,NomeD,IdadeD,MoradaD)):- utente(ID,Nulo,Idade,Nulo))),
  assert(utente(ID,Nulo,Idade,Nulo)),
  assert(nulo(Nulo)).

evolucaoNulo(utente(ID,Nome,Nulo,Nulo),Nulo) :-
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(excecao(utente(ID,NomeC,IdadeC,MoradaC))),
  assert((excecao(utente(ID,NomeD,IdadeD,MoradaD)):- utente(ID,Nome,Nulo,Nulo))),
  assert(utente(ID,Nome,Nulo,Nulo)),
  assert(nulo(Nulo)).


evolucaoNulo(utente(ID,Nulo,Nulo,Nulo),Nulo) :-
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(excecao(utente(ID,NomeC,IdadeC,MoradaC))),
  assert((excecao(utente(ID,NomeD,IdadeD,MoradaD)):- utente(ID,Nulo,Nulo,Nulo))),
  assert(utente(ID,Nulo,Nulo,Nulo)),
  assert(nulo(Nulo)).



%Evolução de incerto para utente.
%Não pode haver já um utente. Não pode haver já imprecisão para esse ID.
%Não pode haver nulos para esse ID, não pode existir um predicado incerto igual
%nem um predicado menos incerto do que o que se quer inserir.
evolucaoIncerto(utente(ID,Nulo,Idade,Morada),Nulo) :-
  nao(excecao(ID,NomeC,IdadeC,MoradaC)),
  nao(excecao(utente(ID,NomeC,IdadeC,MoradaC))),
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
  utente(ID,Nulo,Idade,Morada))),
  assert(utente(ID,Nulo,Idade,Morada)).

evolucaoIncerto(utente(ID,Nulo,Idade,Morada),Nulo) :-
  utente(ID,NomeB,IdadeB,MoradaB),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(nulo(NomeB)),
  nao(nulo(IdadeB)),
  nao(nulo(MoradaB)),
  nao(excecao(utente(ID,NomeB,Idade,Morada))),
  clause(excecao(utente(ID,NomeB,IdadeB,MoradaB)),B,R),
  retract((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-B)),
  retract(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
  utente(ID,Nulo,Idade,Morada))),
  assert(utente(ID,Nulo,Idade,Morada)).

%Não há excecoes nem utentes, insere
```

```
evolucaoIncerto(utente(ID,Nome,Nulo,Morada),Nulo) :-
  nao(excecao(ID,NomeC,IdadeC,MoradaC)),
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
  utente(ID,Nome,Nulo,Morada))),
  assert(utente(ID,Nome,Nulo,Morada)).

%Há utente e não interdita e é mais incerto, insere.
evolucaoIncerto(utente(ID,Nome,Nulo,Morada),Nulo) :-
  utente(ID,NomeB,IdadeB,MoradaB),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(nulo(NomeB)),
  nao(nulo(IdadeB)),
  nao(nulo(MoradaB)),
  nao(excecao(ID,Nome,IdadeB,Morada)),
  clause(excecao(utente(ID,NomeB,IdadeB,MoradaB)),B,R),
  retract((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-B)),
  retract(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
  utente(ID,Nome,Nulo,Morada))),
  assert(utente(ID,Nome,Nulo,Morada)).

evolucaoIncerto(utente(ID,Nome,Idade,Nulo),Nulo) :-
  nao(excecao(ID,NomeC,IdadeC,MoradaC)),
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
  utente(ID,Nome,Idade,Nulo))),
  assert(utente(ID,Nome,Idade,Nulo)).

evolucaoIncerto(utente(ID,Nome,Idade,Nulo),Nulo) :-
  utente(ID,NomeB,IdadeB,MoradaB),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(nulo(NomeB)),
  nao(nulo(IdadeB)),
  nao(nulo(MoradaB)),
  nao(excecao(ID,Nome,Idade,MoradaB)),
  clause(excecao(utente(ID,NomeB,IdadeB,MoradaB)),B,R),
  retract((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-B)),
  retract(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
  utente(ID,Nome,Idade,Nulo))),
  assert(utente(ID,Nome,Idade,Nulo)).



evolucaoIncerto(utente(ID,Nulo,Idade,Nulo),Nulo) :-
  nao(excecao(ID,NomeC,IdadeC,MoradaC)),
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
```

```
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
  utente(ID,Nulo,Idade,Nulo))),
  assert(utente(ID,Nulo,Idade,Nulo)).

evolucaoIncerto(utente(ID,Nulo,Idade,Nulo),Nulo) :-
  utente(ID,NomeB,IdadeB,MoradaB),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(nulo(NomeB)),
  nao(nulo(IdadeB)),
  nao(nulo(MoradaB)),
  NomeB==IdadeB,
  IdadeB==MoradaB,
  clause(excecao(utente(ID,NomeB,IdadeB,MoradaB)),B,R),
  retract((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-B)),
  retract(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeC,IdadeC,MoradaC)):-
  utente(ID,Nulo,Idade,Nulo))),
  assert(utente(ID,Nulo,Idade,Nulo)).



evolucaoIncerto(utente(ID,Nulo,Nulo,Morada),Nulo) :-
  nao(excecao(ID,NomeC,IdadeC,MoradaC)),
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
  utente(ID,Nulo,Nulo,Morada))),
  assert(utente(ID,Nulo,Nulo,Morada)).

evolucaoIncerto(utente(ID,Nulo,Nulo,Morada),Nulo) :-
  utente(ID,NomeB,IdadeB,MoradaB),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(nulo(NomeB)),
  nao(nulo(IdadeB)),
  nao(nulo(MoradaB)),
  NomeB==IdadeB,
  IdadeB==MoradaB,
  clause(excecao(utente(ID,NomeB,IdadeB,MoradaB)),B,R),
  retract((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-B)),
  retract(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeC,IdadeC,MoradaC)):-
  utente(ID,Nulo,Nulo,Morada))),
  assert(utente(ID,Nulo,Nulo,Morada)).


evolucaoIncerto(utente(ID,Nome,Nulo,Nulo),Nulo) :-
  nao(excecao(ID,NomeC,IdadeC,MoradaC)),
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
```

```
  utente(ID,Nome,Nulo,Nulo))),
  assert(utente(ID,Nome,Nulo,Nulo)).

evolucaoIncerto(utente(ID,Nome,Nulo,Nulo),Nulo) :-
  utente(ID,NomeB,IdadeB,MoradaB),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(nulo(NomeB)),
  nao(nulo(IdadeB)),
  nao(nulo(MoradaB)),
  NomeB==IdadeB,
  IdadeB==MoradaB,
  clause(excecao(utente(ID,NomeB,IdadeB,MoradaB)),B,R),
  retract((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-B)),
  retract(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeC,IdadeC,MoradaC)):-
  utente(ID,Nome,Nulo,Nulo))),
  assert(utente(ID,Nome,Nulo,Nulo)).

evolucaoIncerto(utente(ID,Nulo,Nulo,Nulo),Nulo) :-
  nao(excecao(ID,NomeC,IdadeC,MoradaC)),
  nao(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-
  utente(ID,Nulo,Nulo,Nulo))),
  assert(utente(ID,Nulo,Nulo,Nulo)).

evolucaoIncerto(utente(ID,Nulo,Nulo,Nulo),Nulo) :-
  utente(ID,NomeB,IdadeB,MoradaB),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  nao(nulo(NomeB)),
  nao(nulo(IdadeB)),
  nao(nulo(MoradaB)),
  nao(utente(ID,NomeB,NomeB,NomeB)),
  clause(excecao(utente(ID,NomeB,IdadeB,MoradaB)),B,R),
  retract((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-B)),
  retract(utente(ID,NomeB,IdadeB,MoradaB)),
  assert((excecao(utente(ID,NomeC,IdadeC,MoradaC)):-
  utente(ID,Nulo,Nulo,Nulo))),
  assert(utente(ID,Nulo,Nulo,Nulo)).

%Regressão de exceções de utente
regressaoImpreciso(utente(ID,Nome,Idade,Morada)) :-
  excecao(utente(ID,Nome,Idade,Morada)),
  retract(excecao(utente(ID,Nome,Idade,Morada))).

regressaoInterdito(utente(ID,Nome,Idade,Morada)) :-
  utente(ID,Nome,Idade,Morada),
  excecao(utente(ID,Nome,Idade,Morada)),
  regressaonulo(utente(ID,Nome,Idade,Morada)),
```

```
  clause(excecao(utente(ID,Nome,Idade,Morada)),B,R),
  retract((excecao(utente(ID,Nome,Idade,Morada)):-B)),
  retract(utente(ID,Nome,Idade,Morada)).

regressaonulo(utente(ID,Nome,Idade,Morada)) :-
  nulo(Nome).

regressaonulo(utente(ID,Nome,Idade,Morada)) :-
  nulo(Idade).

regressaonulo(utente(ID,Nome,Idade,Morada)) :-
  nulo(Morada).

regressaoIncerteza(utente(ID,Nome,Idade,Morada)) :-
  utente(ID,Nome,Idade,Morada),
  excecao(utente(ID,Nome,Idade,Morada)),
  nao(nulo(Nome)),
  nao(nulo(Idade)),
  nao(nulo(Morada)),
  clause(excecao(utente(ID,Nome,Idade,Morada)),B,R),
  retract((excecao(utente(ID,Nome,Idade,Morada)):-B)),
  retract(utente(ID,Nome,Idade,Morada)).


%Evolução de exceções de cuidados prestados

%Impreciso no cuidado prestado implica inserção desde que não haja negação,
%repetição, interdição ou conhecimento igual mas certo.
%Se houver uma incerteza é irremovível porque há imensos cuidados prestados.
%Can't be sure what you're unsure of.
evolucaoImpreciso(cuidadoprestado(IDServ,Serv,Estab,Local)) :-
  nao(-cuidadoprestado(IDServ,Serv,Estab,Local)),
  nao(cuidadoprestado(IDServ,Serv,Estab,Local)),
  nao(excecao(cuidadoprestado(IDServ,Serv,Estab,Local))),
  testaNulo(cuidadoprestado(IDServ,Serv,Estab,Local)).

testaNulo(cuidadoprestado(IDServ,Serv,Estab,Local)) :-
  cuidadoprestado(IDServ,ServB,EstabB,LocalB),
  excecao(cuidadoprestado(IDServ,ServB,EstabB,LocalB)),
  nao(nulo(ServB)),
  nao(nulo(EstabB),
  nao(nulo(LocalB))),
  assert(excecao(cuidadoprestado(IDServ,Serv,Estab,Local))).

testaNulo(cuidadoprestado(IDServ,Serv,Estab,Local)) :-
  nao(cuidadoprestado(IDServ,ServB,EstabB,LocalB)),
  assert(excecao(cuidadoprestado(IDServ,Serv,Estab,Local))).
```

```
%Interdição no cuidado prestado implica não haver mais possibilidade de conhecer
%mais cuidados prestados para um ato médico a partir do ponto em que é inserido.
%Só não pode haver já uma interdição.
evolucaoNulo(cuidadoprestado(IDServ,Nome,Estab,Local)) :-
  findall(cuidadoprestado(IDServ,ServB,EstabB,LocalB),
  cuidadoprestado(IDServ,ServB,EstabB,LocalB),LCP),
  testaNulo(LCP),
  assert((excecao(cuidadoprestado(IDServC,ServC,EstabC,LocalC)):-
                  cuidadoprestado(IDServ,Nome,Estab,Local))),
  assert(cuidadoprestado(IDServ,Nulo,Estab,Local)),
  assert(nulo(Nulo)).


testaNulo([cuidadoprestado(IDServ,Serv,Estab,Local)|T]) :-
  excecao(cuidadoprestado(IDServ,Serv,Estab,Local)),
  nao(nulo(Serv)),
  nao(nulo(Estab)),
  nao(nulo(Local)),
  testaNulo(T).

testaNulo([cuidadoprestado(IDServ,Serv,Estab,Local)|T]) :-
  nao(excecao(cuidadoprestado(IDServ,Serv,Estab,Local))).


testaNulo([]).


%Incerteza só pode ser inserida se não houver uma interdição mas é sempre válido
%porque não há maneira de saber quando uma incerteza
%é concretizada e se não é simplesmente um novo cuidado prestado diferente.
%No entanto não consideramos possível refinar incerteza para um dado valor nulo,
%a incerteza é expressamente substituída.
evolucaoIncerto(cuidadoprestado(IDServ,Nulo,Estab,Local),Nulo) :-
  cuidadoprestado(IDServ,ServB,EstabB,LocalB),
  excecao(cuidadoprestado(IDServ,ServB,EstabB,LocalB)),
  nao(nulo(ServB)),
  nao(nulo(EstabB)),
  nao(nulo(LocalB)),
  incertocuidadoprestado(IDServ,Nulo),
  assert((excecao(cuidadoprestado(IDServ,Serv,Estab,Local)):-
  cuidadoprestado(IDServ,Serv,Estab,Local))),
  assert(cuidadoprestado(IDServ,Serv,Estab,Local)).

incertocuidadoprestado(IDServ,Nulo) :-
  cuidadoprestado(IDServ,ServB,Nulo,LocalB),
  clause(cuidadoprestado(IDServ,ServB,Nulo,LocalB),B,R),
  retract((cuidadoprestado(IDServ,ServB,Nulo,LocalB):-B)),
```

```
  retract(cuidadoprestado(IDServ,ServB,Nulo,LocalB)).

incertocuidadoprestado(IDServ,Nulo) :-
  cuidadoprestado(IDServ,Nulo,EstabB,LocalB),
  clause(cuidadoprestado(IDServ,Nulo,EstabB,LocalB),B,R),
  retract((cuidadoprestado(IDServ,Nulo,EstabB,LocalB):-B)),
  retract(cuidadoprestado(IDServ,Nulo,EstabB,LocalB)).

incertocuidadoprestado(IDServ,Nulo) :-
  cuidadoprestado(IDServ,ServB,EstabB,Nulo),
  clause(cuidadoprestado(IDServ,ServB,EstabB,Nulo),B,R),
  retract((cuidadoprestado(IDServ,ServB,EstabB,Nulo):-B)),
  retract(cuidadoprestado(IDServ,ServB,EstabB,Nulo)).

incertocuidadoprestado(IDServ,Nulo) :-
  cuidadoprestado(IDServ,Nulo,EstabB,Nulo),
  clause(cuidadoprestado(IDServ,Nulo,EstabB,Nulo),B,R),
  retract((cuidadoprestado(IDServ,Nulo,EstabB,Nulo):-B)),
  retract(cuidadoprestado(IDServ,Nulo,EstabB,Nulo)).

incertocuidadoprestado(IDServ,Nulo) :-
  cuidadoprestado(IDServ,ServB,Nulo,Nulo),
  clause(cuidadoprestado(IDServ,ServB,Nulo,Nulo),B,R),
  retract((cuidadoprestado(IDServ,ServB,Nulo,Nulo):-B)),
  retract(cuidadoprestado(IDServ,ServB,Nulo,Nulo)).

incertocuidadoprestado(IDServ,Nulo) :-
  cuidadoprestado(IDServ,Nulo,Nulo,Nulo),
  clause(cuidadoprestado(IDServ,Nulo,Nulo,Nulo),B,R),
  retract((cuidadoprestado(IDServ,Nulo,Nulo,Nulo):-B)),
  retract(cuidadoprestado(IDServ,Nulo,Nulo,Nulo)).

incertocuidadoprestado(IDServ,Nulo) :-
  cuidadoprestado(IDServ,Nulo,Nulo,LocalB),
  clause(cuidadoprestado(IDServ,Nulo,Nulo,LocalB),B,R),
  retract((cuidadoprestado(IDServ,Nulo,Nulo,LocalB):-B)),
  retract(cuidadoprestado(IDServ,Nulo,Nulo,LocalB)).

incertocuidadoprestado(IDServ,Nulo).


%Regressão de exceções de cuidados prestados

regressaoImpreciso(cuidadoprestado(IDServ,Serv,Estab,Local)) :-
  excecao(cuidadoprestado(IDServ,Serv,Estab,Local)),
  retract(excecao(cuidadoprestado(IDServ,Serv,Estab,Local))).

regressaoInterdito(cuidadoprestado(IDServ,Serv,Estab,Local)) :-
```

```
  cuidadoprestado(IDServ,Serv,Estab,Local),
  excecao(cuidadoprestado(IDServ,Serv,Estab,Local)),
  regressaonulo(cuidadoprestado(IDServ,Serv,Estab,Local)),
  clause(excecao(utente(IDServ,Serv,Estab,Local)),B,R),
  retract((excecao(utente(IDServ,Serv,Estab,Local)):-B)),
  retract(utente(IDServ,Serv,Estab,Local)).

regressaonulo(cuidadoprestado(IDServ,Serv,Estab,Local)) :-
  nulo(Serv).

regressaonulo(cuidadoprestado(IDServ,Serv,Estab,Local)) :-
  nulo(Estab).

regressaonulo(cuidadoprestado(IDServ,Serv,Estab,Local)) :-
  nulo(Local).

regressaoIncerteza(cuidadoprestado(IDServ,Serv,Estab,Local)) :-
  cuidadoprestado(IDServ,Serv,Estab,Local),
  excecao(cuidadoprestado(IDServ,Serv,Estab,Local)),
  nao(nulo(Serv)),
  nao(nulo(Estab)),
  nao(nulo(Local)),
  clause(excecao(cuidadoprestado(IDServ,Serv,Estab,Local)),B,R),
  retract((excecao(cuidadoprestado(IDServ,Serv,Estab,Local)):-B)),
  retract(cuidadoprestado(IDServ,Serv,Estab,Local)).



%Evolução de exceções de atos médicos

evolucaoImpreciso(atomedico(Data,IDUtente,IDServ,Custo)) :-
  nao(-atomedico(Data,IDUtente,IDServ,Custo)),
  nao(excecao(atomedico(Data,IDUtente,IDServ,Custo))),
  testaNulo(atomedico(Data,IDUtente,IDServ,Custo)).

testaNulo(atomedico(Data,IDUtente,IDServ,Custo)) :-
  atomedico(DataB,IDUtente,IDServ,CustoB),
  excecao(atomedico(DataB,IDUtente,IDServ,CustoB)),
  nao(nulo(DataB)),
  nao(nulo(CustoB)),
  retract(atomedico(DataB,IDUtente,IDServ,CustoB)),
  clause(excecao(atomedico(DataB,IDUtente,IDServ,CustoB)),B,R),
  retract((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-B)),
  assert(excecao(atomedico(Data,IDUtente,IDServ,Custo))).

testaNulo(atomedico(Data,IDUtente,IDServ,Custo)) :-
  nao(atomedico(DataB,IDUtente,IDServ,CustoB)),
```

```
  assert(excecao(atomedico(Data,IDUtente,IDServ,Custo))).

%-----------------------------------------------------

evolucaoNulo(atomedico(Nulo,IDUtente,IDServ,Custo),Nulo) :-
  nao(atomedico(DataB,IDUtente,IDServ,CustoB)),
  nao(excecao(atomedico(DataC,IDUtente,IDServ,CustoC))),
  assert((excecao(atomedico(DataD,IDUtente,IDServ,CustoD)):-
                  atomedico(Nulo,IDUtente,IDServ,Custo))),
  assert(atomedico(Nulo,IDUtente,IDServ,Custo)),
  assert(nulo(Nulo)).

evolucaoNulo(atomedico(Data,IDUtente,IDServ,Nulo),Nulo) :-
  nao(atomedico(DataB,IDUtente,IDServ,CustoB)),
  nao(excecao(atomedico(DataC,IDUtente,IDServ,CustoC))),
  assert((excecao(atomedico(DataD,IDUtente,IDServ,CustoD)):-
                  atomedico(Data,IDUtente,IDServ,Nulo))),
  assert(atomedico(Data,IDUtente,IDServ,Nulo)),
  assert(nulo(Nulo)).

evolucaoNulo(atomedico(Nulo,IDUtente,IDServ,Nulo),Nulo) :-
  nao(atomedico(DataB,IDUtente,IDServ,CustoB)),
  nao(excecao(atomedico(DataC,IDUtente,IDServ,CustoC))),
  assert((excecao(atomedico(DataD,IDUtente,IDServ,CustoD)):-
                  atomedico(Nulo,IDUtente,IDServ,Nulo))),
  assert(atomedico(Nulo,IDUtente,IDServ,Nulo)),
  assert(nulo(Nulo)).

%-----------------------------------------------------

evolucaoIncerto(atomedico(Nulo,IDUtente,IDServ,Custo),Nulo) :-
  nao(excecao(DataC,IDUtente,IDServ,CustoC)),
  nao(excecao(atomedico(DataC,IDUtente,IDServ,CustoC))),
  assert((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-
  atomedico(Nulo,IDUtente,IDServ,Custo))),
  assert(atomedico(Nulo,IDUtente,IDServ,Custo)).

evolucaoIncerto(atomedico(Nulo,IDUtente,IDServ,Custo),Nulo) :-
  atomedico(DataB,IDUtente,IDServ,CustoB),
  excecao(atomedico(DataB,IDUtente,IDServ,CustoB)),
  nao(nulo(DataB)),
  nao(nulo(CustoB)),
  nao(excecao(atomedico(DataB,IDUtente,IDServ,CustoB))),
  clause(excecao(atomedico(DataB,IDUtente,IDServ,CustoB)),B,R),
  retract((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-B)),
  retract(atomedico(DataB,IDUtente,IDServ,CustoB)),
  assert((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-
                  atomedico(Nulo,IDUtente,IDServ,CustoB))),
```

```
  assert(atomedico(Nulo,IDUtente,IDServ,Custo)).



evolucaoIncerto(atomedico(Data,IDUtente,IDServ,Nulo),Nulo) :-
  nao(excecao(DataC,IDUtente,IDServ,CustoC)),
  nao(excecao(atomedico(DataC,IDUtente,IDServ,CustoC))),
  assert((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-
  atomedico(Data,IDUtente,IDServ,Nulo))),
  assert(atomedico(Data,IDUtente,IDServ,Nulo)).

evolucaoIncerto(atomedico(Data,IDUtente,IDServ,Nulo),Nulo) :-
  atomedico(DataB,IDUtente,IDServ,CustoB),
  excecao(atomedico(DataB,IDUtente,IDServ,CustoB)),
  nao(nulo(DataB)),
  nao(nulo(CustoB)),
  nao(excecao(atomedico(DataB,IDUtente,IDServ,CustoB))),
  clause(excecao(atomedico(DataB,IDUtente,IDServ,CustoB)),B,R),
  retract((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-B)),
  retract(atomedico(DataB,IDUtente,IDServ,CustoB)),
  assert((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-
  atomedico(Data,IDUtente,IDServ,Nulo))),
  assert(atomedico(Data,IDUtente,IDServ,Nulo)).



evolucaoIncerto(atomedico(Nulo,IDUtente,IDServ,Nulo),Nulo) :-
  nao(excecao(DataC,IDUtente,IDServ,CustoC)),
  nao(excecao(atomedico(DataC,IDUtente,IDServ,CustoC))),
  assert((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-
  atomedico(Nulo,IDUtente,IDServ,Nulo))),
  assert(atomedico(Nulo,IDUtente,IDServ,Nulo)).

evolucaoIncerto(atomedico(Nulo,IDUtente,IDServ,Nulo),Nulo) :-
  atomedico(DataB,IDUtente,IDServ,CustoB),
  excecao(atomedico(DataB,IDUtente,IDServ,CustoB)),
  nao(nulo(DataB)),
  nao(nulo(CustoB)),
  DataB==CustoB,
  clause(excecao(atomedico(DataB,IDUtente,IDServ,CustoB)),B,R),
  retract((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-B)),
  retract(atomedico(DataB,IDUtente,IDServ,CustoB)),
  assert((excecao(atomedico(DataB,IDUtente,IDServ,CustoB)):-
  atomedico(Nulo,IDUtente,IDServ,Nulo))),
  assert(atomedico(Nulo,IDUtente,IDServ,Nulo)).

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

```
%Regressão de exceções de cuidados prestados

regressaoImpreciso(atomedico(Data,IDUtente,IDServ,Custo)) :-
  excecao(atomedico(Data,IDUtente,IDServ,Custo)),
  retract(atomedico(Data,IDUtente,IDServ,Custo)).

regressaoInterdito(atomedico(Data,IDUtente,IDServ,Custo)) :-
  atomedico(Data,IDUtente,IDServ,Custo),
  excecao(atomedico(Data,IDUtente,IDServ,Custo)),
  regressaonulo(atomedico(Data,IDUtente,IDServ,Custo)),
  clause(excecao(atomedico(Data,IDUtente,IDServ,Custo)),B,R),
  retract((excecao(atomedico(Data,IDUtente,IDServ,Custo)):-B)),
  retract((atomedico(Data,IDUtente,IDServ,Custo)).

regressaonulo(atomedico(Data,IDUtente,IDServ,Custo)) :-
  nulo(Data).

regressaonulo(atomedico(Data,IDUtente,IDServ,Custo)) :-
  nulo(Custo).

regressaoIncerteza(atomedico(Data,IDUtente,IDServ,Custo)) :-
  atomedico(Data,IDUtente,IDServ,Custo),
  excecao(atomedico(Data,IDUtente,IDServ,Custo)),
  nao(nulo(Data)),
  nao(nulo(Custo)),
  clause(excecao(atomedico(Data,IDUtente,IDServ,Custo)),B,R),
  retract((excecao(atomedico(Data,IDUtente,IDServ,Custo)):-B)),
  retract(atomedico(Data,IDUtente,IDServ,Custo)).

%Evolucao de conhecimento certo positivo e negativo

evolucao(excecao(P)) :-
  fail.

evolucao( P ) :-
  findall(Inv,+P::Inv,LInv),
  testa(LInv),
  removedesconhecido(P),
  assert(P).

removedesconhecido(utente(ID,Nome,Idade,Morada)) :-
  utente(ID,NomeB,IdadeB,MoradaB),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  clause(excecao(utente(ID,NomeB,IdadeB,MoradaB)),B,R),
  retract((excecao(utente(ID,NomeB,IdadeB,MoradaB)):-B)),
  retract(utente(ID,NomeB,IdadeB,MoradaB)).

removedesconhecido(utente(ID,Nome,Idade,Morada)) :-
```

```
  findall(excecao(utente(ID,NomeB,IdadeB,MoradaB)),
  excecao(utente(ID,NomeB,IdadeB,MoradaB)),LE),
  removedesconhecido(LE).

removedesconhecido([excecao(utente(ID,Nome,Idade,Morada))|T]) :-
  clause(excecao(utente(ID,Nome,Idade,Morada)),B,R),
  retract((excecao(utente(ID,Nome,Idade,Morada)):-B)),
  removedesconhecido(T).

removedesconhecido(-utente(ID,Nome,Idade,Morada)) :-
  findall(excecao(utente(ID,Nome,Idade,Morada)),
  excecao(utente(ID,Nome,Idade,Morada)),LE),
  removedesconhecido(LE).

removedesconhecido(cuidadoprestado(ID,Serv,Estab,Local)) :-
  findall(excecao(cuidadoprestado(ID,Serv,Estab,Local)),
          excecao(cuidadoprestado(ID,Serv,Estab,Local)),
          LE),
  removedesconhecido(LE).

removedesconhecido(-cuidadoprestado(ID,Serv,Estab,Local)) :-
  findall(excecao(cuidadoprestado(ID,Serv,Estab,Local)),
          excecao(cuidadoprestado(ID,Serv,Estab,Local)),
          LE),
  removedesconhecido(LE).

removedesconhecido([excecao(cuidadoprestado(ID,Serv,Estab,Local))|T]) :-
  clause(excecao(cuidadoprestado(ID,Serv,Estab,Local)),B,R),
  retract((excecao(cuidadoprestado(ID,Serv,Estab,Local)):-B)).

removedesconhecido(atomedico(Data,IDUtente,IDServ,Preco)) :-
  atomedico(DataB,IDUtente,IDServ,PrecoB),
  excecao(atomedico(DataB,IDUtente,IDServ,PrecoB)),
  clause(excecao(atomedico(DataB,IDUtente,IDServ,PrecoB)),B,R),
  retract((excecao(atomedico(DataB,IDUtente,IDServ,PrecoB)):-B)),
  retract(atomedico(DataB,IDUtente,IDServ,PrecoB)).

removedesconhecido(atomedico(Data,IDUtente,IDServB,PrecoB)) :-
  findall(excecao(atomedico(DataB,IDUtente,IDServ,PrecoB)),
          excecao(atomedico(DataB,IDUtente,IDServ,PrecoB)),LE),
  removedesconhecido(LE).

removedesconhecido([excecao(atomedico(Data,IDUtente,IDServ,Preco))|T]) :-
  clause(excecao(atomedico(Data,IDUtente,IDServ,Preco)),B,R),
  retract((excecao(atomedico(Data,IDUtente,IDServ,Preco)):-B)),
  removedesconhecido(T).

removedesconhecido([]).
```

Devido a estas considerações tomadas sobre permitir e manipular conhecimentos imperfeitos de vários graus de imperfeição, considerámos, como é possível verificar acima, para os cuidados prestados que:

- A interdição de conhecimento de qualquer grau implicaria a impossibilidade de evoluir mais conhecimento sobre cuidados prestados para o ato médico em questão.
- O conhecimento incerto é irremovível pela evolução de mais conhecimento, e caso esta evolução seja de caráter incerto com o mesmo valor nulo o atual é substituído.

Considerámos tanto para os utentes como para os atos médicos que:

- A interdição de conhecimento de qualquer grau implicaria a impossibilidade de evoluir mais conhecimento para um dado ID
- Apenas se pode inserir conhecimento incerto de um grau menos incerto que o já existente, neste caso substituindo o já existente e apenas no caso de não haver conhecimento certo ou já haver uma interdição ao conhecimento que se quer inserir.

Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas

Para completar a nossa interpretação da conjunção de valores de verdade, remetemos para os seguintes raciocínios de modo a que o conhecimento fosse representado (e interpretado) de forma correta. Como tal, desenvolvemos o predicado `demo`, utilizado nas aulas práticas, de modo a que fosse possível a sua extensão a mais do que uma pergunta, e, consecutivamente, a albergar o valor de desconhecido, de acordo com a tabela de verdade já explicitada anteriormente:

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

demoCj([H],P) :-
  demo(H,P).

demoCj([H|L],verdadeiro) :-
  demo(H,verdadeiro),
  demoCj(L,verdadeiro).

demoCj([H|L],falso) :-
  demo(H,falso),
```

```
  demoCj(L,verdadeiro).

demoCj([H|L],falso) :-
  demo(H,falso),
  demoCj(L,falso).

demoCj([H|L],falso) :-
  demo(H,falso),
  demoCj(L,desconhecido).

demoCj([H|L],desconhecido) :-
  demo(H,verdadeiro),
  demoCj(L,desconhecido).

demoCj([H|L],desconhecido) :-
  demo(H,desconhecido),
  demoCj(L,desconhecido).
```

Conclusão

Com este trabalho registou-se um melhoramento geral das capacidades do grupo de expressar conhecimento imperfeito através dos mecanismos de extensão à programação lógica aprendidos durante as aulas teóricas e práticas. Porém, e em certos casos, revelou-se bastante maçadora e extensiva a representação das múltiplas dezenas de raciocínios para cada um dos tipos de conhecimento enunciados inicialmente, de modo a perfazer todos os casos de nulidade nos parâmetros (isto é, desconhecimento). Só assim seria possível a correta funcionalidade de todo o sistema, abrangendo todas as hipóteses.