UNIVERSIDADE DO MINHO
MIEI

SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO

2º Exercício
Programação em lógica estendida e Conhecimento Imperfeito

Grupo 2
João Nunes - a82300
Flávio Martins - a65277
Luís Braga - a82088
Luís Martins - a82298

Braga, Portugal
1 de Maio de 2019

Resumo

O presente relatório documenta a resolução do segundo exercício prático da unidade curricular de **Sistemas de Representação de Conhecimento e Raciocínio**.

Este trabalho tem o objectivo de consolidar matérias leccionadas na **UC** e pretende-se, também, motivar os alunos para a utilização da extensão à programação em lógica, usando o **PROLOG**, no contexto da representação de conhecimento imperfeito. Mais concretamente, pretende-se que seja desenvolvido um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área de prestação de cuidados de saúde pela realização de serviços de actos médicos.

Assim, esta fase de trabalho será uma continuação da anterior, fazendo as alterações necessárias de modo a ir de encontro com os objectivos.

Conteúdo

1 Introdução

No âmbito da unidade curricular de **Sistemas de Representação de Conhecimento e Raciocínio** resolveu-se o presente exercício, cujo tema é **Programação em lógica estendida e Conhecimento Imperfeito**.

No primeiro trabalho solicitado recorreu-se ao pressuposto do mundo fechado, que dita que toda a informação que não existe mencionada na base de dados é considerada falsa, ao pressuposto do domínio fechado, que afirma que não existem mais objectos no universo de discurso para além daqueles designados por constantes na base de dados. Contudo, surgiu a necessidade de evoluir esta representação de conhecimento para uma em que não se considere falso aquilo que não consta na base de dados. Com isto surge a extensão à programação em lógica e o uso do terceiro valor de verdade: o **desconhecido**.

Assim, cumprindo um dos objectivos principais, obtemos como resultado uma representação de conhecimento mais completa e mais próxima da realidade. Contudo, conceitos como o pressuposto do nomes únicos, que diz que duas constantes diferentes designam, necessariamente, duas entidades diferentes do universo de discurso, mantêm-se.

1.1 Estrutura do relatório

O relatório é composto por quatro partes:

- **Introdução** - onde descreve o tema principal do trabalho, justificação dos objectivos principais, onde, embora que brevemente, se fala do trabalho anterior, pois influenciou o desenvolvimento deste segundo trabalho;
- **Preliminares** - onde se apresentam conceitos importantes para o entendimento do exercício;
- **Resolução do exercício** - onde se dá a conhecer o trabalho realizado;
- **Conclusões e Sugestões** - onde se é feito um resumo dos resultados finais, são consideradas sugestões a adoptar em futuras evoluções do trabalho, fazendo uma análise crítica do trabalho realizado.

2 Preliminares

2.1 Trabalho passado

Tendo em conta que este é o segundo exercício prático da disciplina pode-se afirmar que a equipa de trabalho já realizou e entregou o primeiro trabalho prático.

Nesse primeiro trabalho prático, cujo tema é "**Programação em lógica e Invariantes**", desenvolveu-se um sistema de representação de conhecimento e raciocínio que possui a capacidade um universo de discurso na área da prestação de cuidados de saúde. No qual o conhecimento é dado na forma de:

- utente: (#IdUt, Nome, Idade, Cidade) $\rightsquigarrow \{V, F\}$
- serviço: (#IdServ, Descricao, Instituicao, Cidade) $\rightsquigarrow \{V, F\}$
- consulta: (Data, #IdUt, #IdServ, #IdM, Custo) $\rightsquigarrow \{V, F\}$
- medico: (#idM, Nome, Idade, #IdServ) $\rightsquigarrow \{V, F\}$

Após este trabalho prático o grupo de trabalho ficou mais experiente e assimilou conhecimentos do que é a programação em lógica e o que está por detrás do funcionamento desta. Foram também adquiridos conhecimentos quanto à utilização de invariantes para a inserção e remoção de conhecimento, mantendo assim a integridade da informação.

No exercício anterior o sistema desenvolvido funcionava segundo pressupostos como o do mundo fechado e, consequentemente, as respostas às questões feitas apenas poderiam ser verdadeiro ou falso, consoante existisse informação no sistema que provasse a veracidade de tais afirmações. Como tal, esta representação do conhecimento é irrealista e torna-se necessário actualizar o antigo sistema de conhecimento de maneira a que mantenha aberta a possibilidade de uma afirmação ser verdadeira ou falsa sem a caracterizar imediatamente como falsa, pois é erróneo atribuir tal valor de verdade se não há prova disso. Assim, surge um terceiro valor de verdade: o desconhecido; para além de assumirmos outros pressuposto quando tratamos de conhecimento.

2.2 Representação de Informação Incompleta

Tendo sido dito tudo isto teremos os seguintes valores de verdade:

- **Verdadeiro** - quando for possível provar uma questão Q;
- **Falso** - quando for possível provar a falsidade de uma questão $\neg Q$;
- **Desconhecido** - quando não for possível provar a questão Q nem a sua falsidade.

"*Os sistemas de Bases de Dados (BD's) e os sistemas de representação de conhecimento, lidam com aspectos concretos do mundo real e podem-se comparar nos termos em que dividem a utilização da informação. Ambos os sistemas distinguem as funções de representação – descrição do esquema conceptual e dos dados – e de computação – construção de respostas a questões e manipulação dos dados. [...] questionando a necessidade da adopção do Pressuposto do Mundo Fechado (PMF) em sistemas de representação de conhecimento.*"

Analide, Cesar & Neves, José. Representação de Informação Incompleta, pág.2

Adiante fala-se mais sobre estes tipos de tratamento e manipulação de informação.

2.2.1 Bases de Dados

Portanto, o tratamento de informação num sistema de base de dados é suportado pelos seguintes pressupostos:

- **Pressuposto do Mundo Fechado** - toda a informação que não existe mencionada na base de dados é considerada falsa;
- **Pressuposto dos Nomes Únicos** - duas constantes diferentes (que definam valores atómicos ou objectos) designam, necessariamente, duas entidades diferentes do universo de discurso;
- **Pressuposto do Domínio Fechado** - não existem mais objectos no universo de discurso para além daqueles designados por constantes na base de dados.

2.2.2 Sistemas de Representação de Conhecimento

Um sistema de representação de conhecimento baseia-se nos seguintes pressupostos:

- **Pressuposto do Mundo Aberto** - podem existir outros factos ou conclusões verdadeiros para além daqueles representados na base de conhecimento;
- **Pressuposto dos Nomes Únicos** - duas constantes diferentes (que definam valores atómicos ou objectos) designam, necessariamente, duas entidades diferentes do universo de discurso;
- **Pressuposto do Domínio Aberto** - podem existir mais objectos do universo de discurso para além daqueles designados pelas constantes da base de conhecimento.

2.2.3 Valores Nulos

Os valores nulos servem para discernir dois tipos de situações: aquelas em que as respostas a questões devem ser dadas como verdadeiras ou falsas ou como desconhecidas.

A seguir apresentamos mais detalhadamente cada um dos tipos de valores nulos.

Do Tipo Desconhecido ou do Tipo I

Este tipo apenas representa algo que não é conhecido, isto é, não existe nenhuma prova que comprove que tal seja verdadeiro ou falso.

Do Tipo Desconhecido, de um Conjunto de Valores ou do Tipo II

Este tipo de valor nulo distingue-se do anterior precisamente no facto de este representar um (ou mais) valores de um conjunto de valores bem determinados, não sendo apenas especificado qual dos valores concretizará a questão.

Do Tipo Não Permitido ou do Tipo III

Este tipo de valor nulo caracteriza tipos de dados que não se pretenda que surjam a base de conhecimento.

3 Descrição do Trabalho e Análise de Resultados

Tal como foi explícito no capítulo anterior, foi proposta a realização de um trabalho prático focado na temática de conhecimento imperfeito, num universo de discurso na área de prestação de cuidados médicos, tal como o trabalho prático anterior.

O trabalho foi desenvolvido tendo sempre em mente os seguinte tópicos:

- 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.

Sendo que, ao longo das seguintes secções, irão ser abordados mais concretamente cada um dos tópicos supracitados, e a sua aplicação concreta no trabalho desenvolvido.

3.1 Fontes de conhecimento

Embora se tenha mantido o universo de discurso, ocorreram algumas alterações em relação à fonte de conhecimento anterior.

No que toca utente, ao invés de possuir uma cidade irá possuir uma morada, sendo que o grosso das mudanças ocorrem no prestador (antigo serviço) e no cuidado (consulta). No prestador, consta o nome de quem prestou o serviço, a especialidade a instituição. No cuidado, existe informação acerca da data em que o cuidado foi prestado, dois id's distintos, do utente e da prestação, uma descrição do cuidado e por fim o seu custo.

Portanto, desta forma definiram-se as seguintes fontes de conhecimento:

- **utente:** #IdUt, Nome, Idade, Morada $\rightsquigarrow \{V, F, D\}$;
- **prestador:** #IdPrest, Nome, Especialidade, Instituição $\rightsquigarrow \{V, F, D\}$;
- **cuidado:** Data, #IdUt, #IdPrest, Descrição, Custo $\rightsquigarrow \{V, F, D\}$.

Tendo isto em conta, seguem-se as seguintes definições iniciais, correspondentes às bases de conhecimento anteriores, bem como definições auxiliares para a representação de conhecimento imperfeito.

```
:- dynamic(prestador/4).
:- dynamic(utente/4).
:- dynamic(cuidado/5).
:- dynamic(excecao/1).
:- dynamic(nulointerdito/1).
```

3.2 Representação do conhecimento positivo

Tal como no trabalho anterior, devido à mudança nas fontes de conhecimento, foi necessário proceder, mais uma vez, à elaboração da base de conhecimento, de forma a conseguir testar e verificar os predicados elaborados. Para tal foi necessário primeiramente adicionar o conhecimento positivo, referente a cada uma das fontes de conhecimento.

```
utente(1, "Luis Martins", 20, braga).
utente(2, "Francisco Lomba", 16, braga).
utente(3, "Joao Nunes", 20, braga).
utente(4, "Renato Silva", 34, porto).
utente(5, "Mateus Oliveira", 22, porto).
utente(6, "Joana Santos", 45, porto).
utente(7, "Mariana Moreira", 20, porto).
utente(8, "Maria Ralha", 36, braga).
utente(9, "Patricia Nogueira", 57, braga).
utente(10, "Carla Queiros", 23, porto).
```

É de notar que, no utente com o *IdUt=11*, não há conhecimento acerca da sua localização, ou seja, é desconhecido, este facto irá ter mais relevância nas secções posteriores.

```
prestador(1, joao, urgencia, hospital_sao_joao).
prestador(2, ricardo, terapia_fala, hospital_sao_joao).
prestador(3, flavio, fisioterapia, hospital_sao_joao).
prestador(4, luis, pediatria, hospital_sao_joao).
prestador(5, gervasio, ginecologia, hospital_sao_joao).
prestador(6, tomas, neurologia, hospital_Braga).
prestador(7, tiago, oncologia, hospital_Braga).
prestador(8, hugo, urgencia, hospital_Braga).
prestador(9, rui, cardiologia, hospital_Braga).
prestador(10, jose, fisioterapia, hospital_Braga).
```

```
cuidado('2019-01-23',1,2,"A",25.00).
cuidado('2019-01-23',2,10,"B",32.50).
cuidado('2019-01-23',3,9,"C",17.00).
cuidado('2019-01-23',4,3,"D",50.09).
cuidado('2019-01-23',5,1,"E",28.00).
cuidado('2019-01-23',6,5,"F",32.52).
cuidado('2019-01-23',7,4,"G",29.00).
cuidado('2019-01-23',8,7,"H",53.59).
cuidado('2019-01-23',9,8,"I",25.80).
cuidado('2019-01-23',10,6,"J",38.21).
cuidado('2019-01-24',1,8,"K",35.00).
cuidado('2019-01-24',2,9,"L",35.50).
cuidado('2019-01-24',3,10,"M",13.00).
cuidado('2019-01-24',4,3,"N",20.09).
cuidado('2019-01-24',5,2,"O",58.00).
```

```
cuidado('2019-01-24',6,1,"P",42.52).
cuidado('2019-01-24',7,5,"Q",59.00).
cuidado('2019-01-24',8,7,"R",43.59).
cuidado('2019-01-24',9,8,"S",26.30).
cuidado('2019-01-24',10,6,"T",39.11).
```

3.3 Representação do conhecimento negativo

Para além do conhecimento positivo, foi necessário também representar o conhecimento negativo, podendo este ser de dois tipos diferentes, negação por falha ou negação forte.

Como tal, e dada a importância do conhecimento negativo na *programação em lógica estendida*, foram definidos os seguintes conjuntos de informação:

```
-utente(Id, Nome, Idade, Cid) :-
    nao(utente(Id, Nome, Idade, Cid)),
    nao(excecao(utente(Id, Nome, Idade, Cid))).
-utente(20, "Rui Sousa", 40, prado).
-utente(21, "Hugo Boss", 33, barcelos).
```

Onde, por exemplo, pegando no caso do *"Rui Sousa"*, é possível descrever o seu caso como:

"É falso que o Rui Sousa de 40 anos e habitante de Prado seja um utente".

Prosseguindo, com os restantes casos.

```
-prestador(IdPrest, Nome, Esp, Inst) :-
    nao(prestador(IdPrest, Nome, Esp, Inst)),
    nao(excecao(prestador(IdPrest, Nome, Esp, Inst))).
-prestador(11, zeca, cardiologia, hospital_sao_joao).
-prestador(12, paulo, pediatria, hospital_Braga).
```

Sendo que neste caso, pretende-se representar a inexistência tanto de cardiologia no Hospital de São João, como de pediatria no Hospital de Braga.

```
-cuidado(Data, IdUt, IdPrest, Desc, Custo) :-
    nao(cuidado(Data, IdUt, IdPrest, Desc, Custo)),
    nao(excecao(cuidado(Data, IdUt, IdPrest, Desc, Custo))).

-cuidado('2019-01-27',4,2,"U",54).
```

No cuidado, a falsidade poderá ser entendida, e contextualizada, sendo utilizado para tal o extrato de código anterior, como:

"É falso que utente 4 tenha ido ao prestador 2 e que tenha pago 54 euros no dia 27 de Janeiro de 2019. "

3.4 Representação de conhecimento imperfeito

Como é possível observar, já foi possível representar o conhecimento cujas respostas às questões são necessariamente positivas ou negativas, contudo, é ainda necessário estabelecer uma distinção entre esse tipo de situações e outras situações que são simplesmente desconhecidas.

Como tal, são utilizados os valores nulos, sendo essa a estratégia de distinção para as situações desconhecidas, existindo portanto três tipos de conhecimento associados aos valores nulos.

- **Incerto:** Desconhecido, de um conjunto indeterminado de hipóteses (I);
- **Impreciso:** Desconhecido, mas de um conjunto determinado de hipóteses (II);
- **Interdito:** Desconhecido e não permitido conhecer (III).

3.4.1 Representação de conhecimento incerto

```
% Conjunto de excecoes associadas ao utente.
excecao(utente(Id, Nome, Idade, _)) :- utente(Id,Nome,Idade,incerto).
excecao(utente(Id, Nome, _, Cidade)) :- utente(Id, Nome, incerto, Cidade).
excecao(utente(Id, _, Idade, Cidade)) :- utente(Id, incerto, Idade, Cidade)
    .

utente(11, "Gervasio Mota", 24, incerto).
utente(15, "Jose Meireles", 32, incerto).

% Excecao associada ao prestador.
excecao(prestador(Id, _, Especi, Inst)) :- prestador(Id, incerto, Especi,
    Inst).

prestador(13, incerto, fisioterapia, hospital_Lisboa).

% Conjunto de excecoes associadas ao cuidado.
excecao(cuidado(Data, IdUt, IdPrest, _ , Custo)) :- cuidado(Data, IdUt,
    IdPrest, incerto, Custo).
excecao(cuidado(Data, IdUt, IdPrest, Desc , _)) :- cuidado(Data, IdUt,
    IdPrest, Desc, incerto).

cuidado('2019-01-28', 4, 8, "B", incerto).
cuidado('2019-01-28', 2, 5, incerto, 40.20).
```

No que toca à representação dos valores nulos de tipo I, foram criadas uma série de representações de conhecimento incerto para cada uma das fontes de conhecimento, sendo que, cada um dos casos pode ser descrito textualmente, contextualizando cada uma das decisões do grupo, começando com o primeiro valor nulo no utente:

"O utente José Meireles de 32 anos sofre de esquizofrenia paranoide, e como tal não revelou a sua localização."

Portanto, ao representar o valor nulo de I, no campo onde não existe informação, é necessário representar como um incerto. Para além disto, foi também necessário criar um conjunto de exceções associados ao utente, cada exceção para cada campo passível de ser desconhecido.

No que toca ao prestador:

"Ainda não se sabe o nome do prestador de fisioterapia no hospital de Lisboa."

Para este caso, tal como o anterior, foi necessário criar uma exceção para o campo cujo conhecimento sobre ele é desconhecido, de seguida, na representação do valor nulo de tipo I, no campo que corresponde ao nome, como este é incerto, representa-se pelo próprio incerto.

Para a representação do conhecimento do tipo incerto no cuidado, seguem-se os seguintes exemplos:

"A azáfama era tanta, que tanto o utente tanto a pessoa que passou a fatura não se lembra do valor dela."

"O médico esqueceu-se de colocar à descrição do cuidado que elaborou."

Na última fonte de conhecimento, ou seja, o cuidado, foram elaboradas duas representações do valor nulo de tipo II, pelo que, o seu funcionamento é análogo às anteriores. No primeiro caso, não há informação atual sobre o valor do cuidado (incerto), existindo para tal uma exceção associada a esse caso, para o segundo exemplo, não existe conhecimento acerca descrição do cuidado (incerto), pelo que, novamente foi criada uma exceção para este caso.

3.4.2 Representação do conhecimento impreciso

```
% Conjunto de excecoes associadas ao utente.
excecao(utente(12, "Otavio Silva", 38, porto)).
excecao(utente(12, "Otavio Silva", 39, porto)).
% Conjunto de excecoes associadas ao prestador.
excecao(prestador(14, joao, cardiologia, hospital_Lisboa)).
excecao(prestador(14, jose, cardiologia, hospital_Lisboa)).
% Conjunto de excecoes associadas ao cuidado.
excecao(cuidado('2019-01-27',5,2,"G",C)) :- C >= 40, C =< 100.
```

Relativamente à representação dos valores nulos de tipo II, existindo para tal um conjunto de representações deste tipo de valor para cada uma das fontes de conhecimento.

No que toca ao utente, existem duas hipóteses, sendo estas representadas por duas exceções uma vez que não há certeza relativamente ao conhecimento que está a ser expresso.

"O utente Ótavio Silva do Porto, ou tem 38 anos ou tem 39 anos."

O mesmo se aplica ao prestador, mais uma vez, não há certeza relativamente ao nome do prestador, portanto, é necessário representar as hipóteses possíveis por exceções, sendo possível descrever o cenário como:

"Não se sabe se o prestador de serviços de cardiologia do Hospital de Lisboa se chama João ou José."

O cuidado também foi alvo de uma representação de um valor do tipo nulo III, sendo que existem dúvidas relativamente ao custo do cuidado:

"Não se sabe o valor exato do cuidao, pensa-se que varie entre os 40 e 100 euros."

Portanto, tal como os anteriores, foi criada uma exceção para representar este valor nulo, onde o custo irá variar entre os 40 e 100 euros tal como a situação assim o indica.

3.4.3 Representação do conhecimento interdito

```
% Conjunto de excecoes associadas ao utente.
excecao(utente(Id, _, Idade, Cidade)) :- utente(Id, Nome, Idade, Cidade),
    nulointerdito(Nome).
excecao(utente(Id, Nome, _, Cid)) :- utente(Id, Nome, Idade, Cid),
    nulointerdito(Idade).
excecao(utente(Id, Nome, Idade, _)) :- utente(Id, Nome, Idade, Cidade),
    nulointerdito(Cidade).

utente(16, imp, 23, braga).
utente(13, "Joaquim Oliveira", imp, braga).
utente(14, "Mateus Ferreira", 21, imp).

% Conjunto de excecoes associadas ao prestador.
excecao(prestador(Id, _, Especi, Inst)) :- prestador(Id, Nome, Especi, Inst
    ), nulointerdito(Nome).

prestador(15, imp, neurologia, hospital_Lisboa).

% Conjunto de excecoes associadas ao cuidado.
excecao(cuidado(Data,IdUt, IdPrest, Desc, _)) :- cuidado(Data, IdUt,
    IdPrest, Desc, Cts), nulointerdito(Cts).
excecao(cuidado(Data,IdUt, IdPrest, _, Custo)) :- cuidado(Data, IdUt,
    IdPrest, Desc, Custo), nulointerdito(Desc).

cuidado('2019-01-25',10,6,"V",imp).
cuidado('2019-01-28', 3, 7, "A", imp).
cuidado('2019-01-28',2,8,imp,40).
```

No que diz respeito à representação do conhecimento interdito, ou seja, os valores nulo de tipo III, é necessário ter em conta a impossibilidade de representar certos tipos de valores associados à fonte de conhecimento, pelo que, será necessário criar um conjunto de exceções para tratar os casos relativos à fonte de conhecimento.

No utente, existem 3 situações possíveis, ou seja, impossibilidade se vir a saber o nome do utente, a sua idade e a localização, pelo que foram também criadas as 3 exceções correspondentes a cada um destes casos.

Para o prestador, existe a situação em que será impossível vir-se a conhecer o seu nome, onde, tal como o caso anterior, se criou uma exceção para tratar este tipo de ocorrências.

Para a última fonte de conhecimento ou seja, o cuidado, existem 2 casos que retratam a impossibilidade de se vir a conhecer o custo do cuidado elaborado, e um outro caso que retrata a impossibilidade de se vir a conhecer a descrição do cuidado, onde, do mesmo modo em que foram representadas as exceções anteriores, também foram acrescentadas exceções para estes dois tipos de situações.

3.5 Invariantes destinados à inserção e remoção de conhecimento

De modo a manter a base de conhecimento consistente, como tal, é necessário restringir e controlar a inserção e remoção do conhecimento, uma vez que uma inserção e remoção desmedida poderia trazer problemas para a base de conhecimento.

Portanto, foram criados invariantes estruturais relativos a cada uma das fontes de conhecimento.

3.5.1 Invariantes estruturais do utente

No que à inserção dos utentes, é necessário comparar o id do utente que se pretende adicionar com os id's dos utentes já inseridos na base de conhecimento, de seguida, verifica-se se o tamanho da lista é zero, uma vez que caso seja zero, então o utente que se pretende adicionar não existe na base de conhecimento.

```
1 +utente(Id,_,_,_) :: (solucoes(Id,(utente(Id,_,_,_)),L),
2             comprimento(L,0)).
```

Continuando na adição do conhecimento, também é necessário restringir a inserção de conhecimento negativo, sendo que, este funciona de uma maneira semelhante ao anterior.

```
1 +(-utente(Id,_,_,_)) :: (solucoes(Id,(-utente(Id,_,_,_)),L),
2             comprimento(L,0)).
```

De modo a impedir contradições na base de conhecimento, ao inserir conhecimento negativo é necessário verificar que a informação que se pretende adicionar não se encontra na base de conhecimento positiva do utente, portanto, o comprimento da lista final terá que ser zero.

```
1 +(-utente(_,Nome,Idade,Cidade)) :: (solucoes((Id),utente(Id,Nome,
                                    Idade,Cidade),L),
2                                       comprimento(L,0)).
```

Foram também criados invariantes para impedir a inserção por cima de conhecimento interdito já existente ou seja, valor nulo de tipo III, onde, são comparados cada um dos campos do utente que se pretende adicionar com os que já existem na base de conhecimento, e de seguida verifica-se se o campo tem valor nulo. Sendo que só permite a inserção do conhecimento caso o tamanho da lista final seja zero, uma vez que não conhecimento interdito acerca do utilizador que se pretende adicionar.

O mesmo processo foi elaborado para a inserção de conhecimento negativo.

```
1 +utente(Id,_,Idade,Cid) :: (solucoes((Id,Nome,Idade,Cid),(utente(Id,
    Nome,Idade,Cid),nulointerdito(Nome)),L),
2                   comprimento(L,0)).
3
4 +(-utente(Id,_,Idade,Cid)) :: (solucoes((Id,Nome,Idade,Cid),(utente(Id,
     Nome,Idade,Cid),nulointerdito(Nome)),L),
5                   comprimento(L,0)).
6
7 +utente(Id,Nome,_,Cid) :: (solucoes((Id,Nome,Idade,Cid),(utente(Id,
    Nome,Idade,Cid),nulointerdito(Idade)),L),
8                   comprimento(L,0)).
9
10 +(-utente(Id,Nome,_,Cid)) :: (solucoes((Id,Nome,Idade,Cid),(utente(Id,
    Nome,Idade,Cid),nulointerdito(Idade)),L),
11                  comprimento(L,0)).
```

```
12
13 +utente(Id,Nome,Idade,_) :: (solucoes((Id, Nome, Idade, Cidade),(utente(Id
       , Nome, Idade, Cidade), nulointerdito(Cidade)),L),
14                     comprimento(L,0)).
15
16 +(-utente(Id,Nome,Idade,_)) :: (solucoes((Id, Nome, Idade, Cidade),(utente
       (Id, Nome, Idade, Cidade), nulointerdito(Cidade)),L),
17                     comprimento(L,0)).
```

Para remover utentes, é necessário verificar que o utente que se pretende remover existe, de seguida, é também necessário verificar que não possui cuidados associados a ele, uma vez que se existem cuidados associados implicava problemas relacionados com dependência entre os id's.

Portanto, ao remover o utente primeiro compara-se o id do utente que se pretende adicionar com os que já existem na base de conhecimento e verifica-se se o comprimento da lista resultante é 1, de modo a verificar que o utente que se pretende adicionar existe na base de conhecimento, de seguida é necessário comparar o id do utente que se pretende adicionar com os id's dos utentes existentes relativos à base de conhecimento da consulta, sendo que o comprimento da lista resultante terá que ser zero, porque o utente que se pretende remover não pode ter consultas associadas a ele.

Na remoção de conhecimento negativo, a tarefa é simplificada, apenas é necessário verificar se o utente que se pretende remover existe na base de conhecimento negativa.

```
1 -utente(IdU,_,_,_) :: (solucoes(IdU,(utente(IdU,_,_,_)),L),
2                                 comprimento(L,1),
3                                   solucoes(IdU, cuidado(_,IdU,_,_,_),C),
4                                    comprimento(C,0)).
5
6 -(-utente(IdU,_,_,_)) :: (solucoes(IdU,(-utente(IdU,_,_,_)),L),
7                                 comprimento(L,1)).
```

3.5.2 Invariantes estruturais do prestador

A respeito dos invariantes estruturais do prestador, tal como foi feito anteriormente, na inserção de conhecimento relativo ao prestador, é necessário verificar que o prestador que se pretende adicionar não existe na base de conhecimento, sendo comparado os id's, no final verifica-se o tamanho lista resultante, tendo este de ser zero para se poder adicionar.

A inserção de conhecimento negativo decorre da mesma maneira que a inserção de conhecimento positivo.

```
1 +prestador(Id,_,_,_) :: (solucoes(Id,(prestador(Id,_,_,_)),L),
2             comprimento(L,0)).
3
4 +(-prestador(Id,_,_,_)) :: (solucoes(Id, (-prestador(Id,_,_,_)) ,L),
5             comprimento(L,0)).
```

Também é necessário verificar que não existem contradições na base de conhecimento no que toca ao prestador, portanto, ao inserir conhecimento negativo na base de conhecimento, é necessário verificar que o resto da informação negativa relacionada com o prestador não existe na base de conhecimento positiva do prestador, pelo que o resultado da lista resultante terá que ser zero.

```
1 +(-prestador(_,Nome,Esp,Inst)) :: ( solucoes( (Nome), prestador(_,
                                      Nome, Esp, Inst), L),
                                      comprimento(L,0)).
```

Tal como aconteceu com o utente, é necessário impedir a inserção de conhecimento sobre conhecimento já existente e interdito à base de conhecimento, onde são comparados os campos do prestador que se pretende adicionar com o conhecimento já existente na base de conhecimento, de seguida verifica-se se não existe campo nulo, sendo que o tamanho da lista resultante terá que ser zero para permitir a inserção do novo conhecimento.

```
+prestador(Id,_,Especi,Inst) :: ( solucoes( Id, ( prestador(Id, Nome,
    Especi, Inst), nulointerdito(Nome) ), L ),
                                    comprimento( L , 0 )).

+(-prestador(Id,_,Especi,Inst)) :: ( solucoes( Id, ( prestador(Id, Nome,
    Especi, Inst), nulointerdito(Nome) ), L ),
                                    comprimento( L , 0 )).
```

Na remoção do prestador, apenas é permitido remover o conhecimento positivo relativo ao prestador se este já existir na base de conhecimento, daí o comprimento da lista resultante ser 1, e se o prestador não tiver nenhum cuidado associado a ele, devido à dependência existente entre os id's no prestador e no cuidado, portanto o comprimento da lista resultante terá que ser zero. Ao remover o conhecimento negativo relativo ao prestador, apenas é necessário verificar se este já existe na base de conhecimento negativa do prestador.

```
-prestador(IdP,_,_,_) :: (solucoes(IdP,(prestador(IdP,_,_,_)),L),
                          comprimento(L,1),
                           solucoes(IdP, cuidado(_,_,IdP,_,_),M),
                            comprimento(M,0)).

-(-prestador(IdP,_,_,_)) :: (solucoes(IdP,(-prestador(IdP,_,_,_)),L),
                             comprimento(L,1)).
```

3.5.3 Invariantes estruturais do cuidado

Uma vez que a fonte de conhecimento do cuidado não possui nenhum identificador associado a ele mesmo, ao inserir conhecimento acerca do cuidado, é necessário verificar se a identificação do utente e da prestação existem na base de conhecimento, o mesmo princípio se aplica na inserção de conhecimento negativo sobre o cuidado.

```
+cuidado(_,IdUt,IdPrest,_,_) :: (utente(IdUt,_,_,_),
                                  prestador(IdPrest,_,_,_)).
+(-cuidado(_,IdUt,IdPrest,_,_)) :: (utente(IdUt,_,_,_),
                                    prestador(IdPrest,_,_,_)).
```

Não poderá haver contradições dentro da base de conhecimento, portanto ao adicionar conhecimento negativo sobre o cuidado, é necessário verificar se as informações que se pretendem adicionar não existem na base de conhecimento positiva do cuidado, pelo que o tamanho da lista resultante terá que ser zero para se permitir a inserção do novo conhecimento.

```
+(-cuidado(Data,IdUt,IdPrest,Desc,Custo)) :: ( solucoes( (Data,IdUt),
                    cuidado(Data, IdUt, IdPrest, Desc, Custo), L),
                    comprimento(L,0)).
```

Não é permitida a inserção sobre valores nulos já existentes do tipo III, portanto são então comparados os campos do cuidado que se pretende inserir com o conhecimento já inserido, pelo que se verificará também que não existem campos nulos, pelo que foram elaborados os seguintes invariantes.

```
+cuidado(Data,IdUt,IdPrest,Desc,_) :: (solucoes((Data, IdUt, IdPrest, Desc,
    Cs),(cuidado(Data, IdUt, IdPrest, Desc, Cs), nulointerdito(Cs)),L),
```

```
                comprimento(L,0)).

+(-cuidado(Data,IdUt,IdPrest,Desc,_)) :: (solucoes((Data, IdUt, IdPrest,
    Desc, Cs),(cuidado(Data, IdUt, IdPrest, Desc, Cs), nulointerdito(Cs)),L)
    ,
                    comprimento(L,0)).

+cuidado(Data,IdUt,IdPrest,_,Custo) :: (solucoes((Data, IdUt, IdPrest,
    Descs, Custo),(cuidado(Data, IdUt, IdPrest, Descs, Custo), nulointerdito
    (Descs)),L),
                    comprimento(L,0)).

+(-cuidado(Data,IdUt,IdPrest,_,Custo)) :: (solucoes((Data, IdUt, IdPrest,
    Descs, Custo),(cuidado(Data, IdUt, IdPrest, Descs, Custo), nulointerdito
    (Descs)),L),
                    comprimento(L,0)).
```

3.6 Evolução do conhecimento

De modo a permitir a evolução do conhecimento, foi necessário criar um predicado evolução que permite a inserção de conhecimento, sendo possível inserir conhecimento positivo, negativo, ou conhecimento imperfeito.

Tal como foi dito anteriormente no predicado evolução, após testar se o conhecimento que se pretende adicionar cumpre com os invariantes, de seguida testa a validade da lista e por fim adiciona o conhecimento.

```
% Evolucao do conhecimento positivo
evolucao(E, positivo) :- solucoes(I,+E::I,L),
                         teste(L),
                         assert(E).

% Evolucao do conhecimento negativo
evolucao(E, negativo) :- solucoes(I,+(-E)::I,L),
                         teste(L),
                         assert(-E).
```

Segue-se a inserção do conhecimento imperfeito, podendo ser adicionado conhecimento incerto, impreciso ou nulo interdito, contudo é de referir, que todo o processo de inserção e os predicados desenvolvidos, dependem do processo anterior de representação do conhecimento imperfeito.

No que toca à evolução dos valores nulos de tipo I, relativamente à fonte de conhecimento do utente, poderá haver incertezas no que toca ao nome, idade ou cidade, sendo então necessário os mecanismos necessários para inserir conhecimento incerto relativo a estes campos, sendo então inserido o utente com a dada incerteza no dado campo.

```
evolucao(utente(Id, _, Idade, Cid), incerto, nome) :- evolucao(utente(Id,
    incerto, Idade, Cid), positivo).

evolucao(utente(Id, Nome, _, Cid), incerto, idade) :- evolucao(utente(Id,
    Nome, incerto, Cid), positivo).

evolucao(utente(Id, Nome, Idade, _), incerto, cidade) :- evolucao(utente(Id
    , Nome, Idade, incerto), positivo).
```

O mesmo processo se aplica para o prestador e para o cuidado, no prestador é permitida a inserção de valores nulos do tipo II relativos ao nome do prestador, e no cuidado é permitida a inserção na descrição e no custo do cuidado, sendo que em cada um destes casos é inserido o conhecimento incerto com a marca da incerteza no dado tipo de evolução escolhido.

```
evolucao(prestador(Id, _, Esp, Inst)) :- evolucao(prestador(Id, incerto, Esp, Inst), positivo).

evolucao(cuidado(Data, IdUt, IdPrest, _, Custo), incerto, descricao) :- evolucao(cuidado(Data, IdUt, IdPrest, incerto, Custo), positivo).

evolucao(cuidado(Data, IdUt, IdPrest, Desc, _), incerto, custo) :- evolucao(cuidado(Data, IdUt, IdPrest, Desc, incerto), positivo).
```

Segue-se a evolução dos valores nulos do tipo II, ou seja, do conhecimento impreciso. Neste caso é necessário ter em conta as várias formas sobre as quais o conhecimento impreciso poderá ser expresso. Estes valores nulos podem ser expressos em várias hipóteses, podem também variar entre um dado intervalo, tal como foi visto no capítulo 3.4.2. Identificaram-se 3 tipos de inserir conhecimento imperfeito do tipo 2: variado, próximo e intervalo. Todos eles têm em comum que não podem existir na base de conhecimento positivo.

O próximo consiste em, dado um valor no argumento, multiplica-se por 0.96 e 1.04 esse valor e fica na forma de um intervalo, com a variável (número) a puder variar entre ambos. O intervalo consiste em dados dois números, a variável pretendida terá que variar nesse intervalo. E o variado, consiste numa lista de opções passadas pelo utilizador, por exemplo, o José tem 34 ou 35 anos, será passsado na forma de lista ([34,35]).

```
evolucao_impreciso(utente(Id, Nome, Idade), variado, cidade, [H|T]) :- nao(utente(Id,_,_,_)),
        assert(excecao(utente(Id, Nome, Idade, H))), evolucao_impreciso(utente(Id, Nome, Idade), variado, cidade, T).

evolucao_impreciso(utente(Id, Nome, Cidade), variado, idade, [H|T]) :- nao(utente(Id,_,_,_)),
        assert(excecao(utente(Id, Nome, H, Cidade))), evolucao_impreciso(utente(Id, Nome, Cidade), variado, idade, T).

evolucao_impreciso(utente(Id, Nome, Cidade), proximo, idade, V) :- nao(utente(Id,_,_,_)), M1 is V*0.96,
        M2 is V*1.04, assert(excecao(utente(Id,Nome,Idade,Cidade)) :- (Idade >=M1, Idade =< M2)).

evolucao_impreciso(utente(Id, Nome, Cidade), intervalo, idade, V1, V2) :- nao(utente(Id,_,_,_)), assert(excecao(utente(Id,Nome,Idade,Cidade)) :- (Idade =< V2, Idade >= V1)).
```

Tais comportamentos podem também ser expressos nas fontes de conhecimento do prestador e cuidado, e, como tal, no desenvolvimento do predicado para a evolução do conhecimento do valor nulo de tipo II também se teve cuidado de tratar estes casos para estas fontes de conhecimento.

```
evolucao_impreciso(prestador(Id, Esp, Inst), variado, nome, [H|T]) :- nao(prestador(Id,_,_,_)),
        assert(excecao(prestador(Id, H, Esp, Inst))),
    evolucao_impreciso(prestador(Id, Esp, Inst), variado, nome, T).

evolucao_impreciso(cuidado(Data, IdU, IdP, Desc), variado, custo, [H|T]) :-
     nao(cuidado(Data, IdU, IdP, Desc, _)),
```

```
 assert(excecao(cuidado(Data, IdU, IdP, Desc, H))), evolucao_impreciso(
    cuidado(Data, IdU, IdP, Desc), variado, custo, T).

evolucao_impreciso(cuidado(Data, IdU, IdP, Desc), proximo, custo, V) :- nao
    (cuidado(Data, IdU, IdP, Desc, _)),
                    M1 is V*0.96,
                    M2 is V*1.04,
                     assert(excecao(cuidado(Data,IdU,IdP,Desc,Custo)) :- (
    Custo >=M1, Custo =< M2)).

evolucao_impreciso(cuidado(Data, IdU, IdP, Desc), intervalo, custo, V1, V2)
     :- nao(cuidado(Data, IdU, IdP, Desc, _)),
         assert(excecao(cuidado(Data,IdU,IdP,Desc,Custo)) :- (Custo >=V1,
    Custo =< V2)).
```

Relativamente à inserção de conhecimento com valor nulo de tipo III, e uma vez que cada campo está sujeito a ser representado por uma impossibilidade, foi desenvolvido o predicado evolução para estes casos, onde se adiciona o conhecimento interdito como conhecimento positivo e de seguida coloca-se o determinado campo como nulo interdito, caso cumpra o teste.

```

evolucao(utente(Id, Nome, Idade, Cid), interdito, nome) :-
      evolucao(utente(Id, Nome, Idade, Cid), positivo),
      testaInterdito(Nome).


evolucao(utente(Id, Nome, Idade, Cid), interdito, idade) :-
      evolucao(utente(Id, Nome, Idade, Cid), positivo),
      testaInterdito(Idade).


evolucao(utente(Id, Nome, Idade, Cid), interdito, cidade) :-
       evolucao(utente(Id, Nome, Idade, Cid), positivo),
       testaInterdito(Cid).


evolucao(prestador(Id, Nome, Esp, Inst), interdito, nome):-
        evolucao(prestador(Id, Nome, Esp, Inst), positivo),
        testaInterdito(Nome).


evolucao(cuidado(Data, IdUt, IdPrest, Desc, Custo), interdito, descricao)
    :-
        evolucao(cuidado(Data, IdUt, IdPrest, Desc, Custo), positivo),
        testaInterdito(Desc).

evolucao(cuidado(Data, IdUt, IdPrest, Desc, Custo), interdito, custo) :-
        evolucao(cuidado(Data, IdUt, IdPrest, Desc, Custo), positivo),
        testaInterdito(Custo).
```

De modo a desenvolver este predicado, foi também necessário criar alguns predicados adicionais, de modo a auxiliar na tarefa de construção, tais como:

```
teste([]).
teste([X|Y]) :- X , teste(Y).

testaInterdito(X) :- nao(nulointerdito(X)), assert(nulointerdito(X)).
testaInterdito(X) :- nulointerdito(X).
```

Onde o primeiro predicado, tal como o nome indica é utilizado para testar a validade de uma lista, sendo que o segundo é utilizado para adicionar valores nulos interditos caso não existam na base de conhecimento.

Outra forma de evoluir o conhecimento é atualizar o conhecimento já existente, como tal, foi elaborado o predicado *update* que remove o conhecimento antigo através do retract e adiciona o conhecimento atualizado, através do assert.

```
update(Remover, Novo) :- retract(Remover), assert(Novo).
```

Como tal, este predicado foi utilizado como auxiliar, no desenvolvimento de um predicado novo capaz de atualizar o conhecimento incerto onde, pegando no caso do utente, é apenas necessário indicar a identificação do utente que se pretende atualizar, e o campo que se pretende atualizar, sendo retirado o utente corresponde a esse identificador passado como argumento, de seguida é utilizado o predicado *update* para atualizar o conhecimento.

Portanto, para o utente existem os seguintes predicados destinados a atualização da informação dos valores nulos de tipo I.

```
update_Utente(Id, Nome, nome) :-
    utente(Id, OldName, Idade, Cidade),
    OldName == incerto,
    update(utente(Id, OldName, Idade, Cidade),
           utente(Id, Nome, Idade, Cidade)).

update_Utente(Id, Idade, idade) :-
    utente(Id, Nome, OldAge, Cidade),
    OldAge == incerto,
    update(utente(Id, Nome, OldAge, Cidade),
           utente(Id, Nome, Idade, Cidade)).

update_Utente(Id, Cidade, cidade) :-
    utente(Id, Nome, Idade, OldCity),
    OldCity == incerto,
    update(utente(Id, Nome, Idade, OldCity),
           utente(Id, Nome, Idade, Cidade)).
```

Para o prestador e para o cuidado, foram desenvolvidos os seguintes predicados com o mesmo intuito de atualizar o conhecimento incerto:

```
update_Prestador(Id, Nome, nome) :-
    prestador(Id, OldName, Esp, Inst),
    OldName == incerto,
    update(prestador(Id,OldName,Esp,Inst),
           prestador(Id, Nome, Esp, Inst)).

update_Cuidado(Data, IdUt, IdPrest, NewDesc, descricao) :-
    cuidado(Data, IdUt, IdPrest, Desc, Custo),
    Desc == incerto,
    update(cuidado(Data, IdUt, IdPrest, Desc, Custo),
           cuidado(Data, IdUt, IdPrest, NewDesc, Custo)).

update_Cuidado(Data, IdUt, IdPrest, NewCusto, custo) :-
    cuidado(Data, IdUt, IdPrest, Desc, Custo),
    Custo == incerto,
    update(cuidado(Data, IdUt, IdPrest, Desc, Custo),
           cuidado(Data, IdUt, IdPrest, Desc, NewCusto)).
```

3.7 Involução do conhecimento

Tal como seria de esperar, uma vez que é possível inserir e atualizar o conhecimento, também é possível *"involuir"* o conhecimento, ou seja, retroceder o conhecimento.

Foi necessário criar o predicado involução de modo a permitir o recuo no que toca ao conhecimento. Tal como acontece no evolução para o conhecimento positivo e negativo, no involução para este tipo de conhecimento, primeiramente testa-se os invariantes de remoção e a validade da lista resultante, caso passem em ambos, então retrocede-se nesse conhecimento, removendo esse conhecimento.

```
% Involucao do conhecimento positivo
involucao(E, positivo) :- solucoes(I,-E::I,L),
                  teste(L),
                  remove(E).

% Involucao do conhecimento negativo
involucao(E, negativo) :- solucoes(I, -(-E)::I, L),
                          teste(L),
                          remove(E).
```

Continuando para o paradigma do conhecimento imperfeito, também é possível retroceder no que toca aos valores nulos de tipo I, II ou III.

De modo a regredir no conhecimento do tipo incerto, por exemplo no utente, primeiramente é necessário passar como argumento ao predicado o identificador do utente sobre o qual se pretende que o conhecimento seja retrocedido, de seguida e conforme o parâmetro cuja incerteza está associado ao utente, é utilizado o predicado de involução supracitado para remover esse conhecimento acerca do dado utente.

```
involucao(utente(Id), incerto, nome) :-
    utente(Id, incerto, Idade, Cidade),
    involucao(utente(Id, incerto, Idade, Cidade), positivo).

involucao(utente(Id), incerto, idade) :-
    utente(Id, Nome, incerto, Cidade),
    involucao(utente(Id, Nome, incerto, Cidade), positivo).

involucao(utente(Id), incerto, cidade) :-
    utente(Id, Nome, Idade, incerto),
    involucao(utente(Id, Nome, Idade, incerto), positivo).
```

O mesmo princípio se aplica no que toca ao prestador e ao cuidado.

```
involucao(prestador(Id), incerto, nome) :-
    prestador(Id, incerto, Esp, Inst),
    involucao(prestador(Id, incerto, Esp, Inst), positivo).

involucao(cuidado(Data, IdU, IdP, Custo), incerto, descricao) :-
    cuidado(Data, IdU, IdP, incerto, Custo),
    involucao(prestador(Data, IdU, IdP, incerto, Custo), positivo).

involucao(cuidado(Data, IdU, IdP, Desc), incerto, custo) :-
    cuidado(Data, IdU, IdP, Desc, incerto),
    involucao(prestador(Data, IdU, IdP, Desc, incerto), positivo).
```

De modo a retroceder no que toca ao conhecimento impreciso, e uma vez que este poderá ser expresso de várias maneiras, é necessário ter em conta os vários casos sobre o qual este conhecimento poderá ser expresso. Portanto, foi desenvolvido o seguinte predicado auxiliar cujo intuito é remover as exceções associadas a este tipo de conhecimento.

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

Tendo definido o predicado anterior, é agora possível representar a extensão do predicado associado à involução neste tipo de valor nulo, começando no utente.

```
involucao_impreciso(utente(Id), variado) :-
    solucoes( excecao(utente(Id, Nome, Idade, Cidade)),
    excecao(utente(Id, Nome, Idade, Cidade)), L),
    comprimento(L,N),
    N > 0,
    retirarExcecoes(L).


involucao_impreciso(utente(Id), proximo, V) :-
    M1 is V*0.96,
    M2 is V*1.04,
    remove(excecao(utente(Id, _, Idade, _) :-
    M1=<Idade, M2>=Idade)).

involucao_impreciso(utente(Id), intervalo, M1, M2) :-
    remove(excecao(utente(Id, _, Idade, _) :-
    M1=<Idade, M2>=Idade)).
```

Para o prestador e para o cuidado, foi também estendido o predicado involução.

```
involucao_impreciso(cuidado(Data, IdU, IdP, Desc),variado, custo) :-
    solucoes( excecao(cuidado(Data, IdU, IdP, Desc,
    Custo)), excecao(prestador(Data, IdU, IdP, Custo)),L),
    comprimento(L, N),
    N > 0,
    retirarExcecoes(L).

involucao_impreciso(cuidado(Data, IdU, IdP, Desc), proximo, V) :-
    M1 is V*0.96,
    M2 is V*1.04,
    remove(excecao(cuidado(Data, IdU, IdP, Desc, Custo)):-
    Custo=< M2, Custo >= M1).

involucao_impreciso(cuidado(Data, IdU, IdP, Desc), intervalo, M1, M2) :-
    remove(excecao(cuidado(Data, IdU, IdP, Desc, Custo)) :-
    Custo=< M2, Custo >= M1).
```

Para os valores nulos de tipo III, é necessário ter em conta os vários casos onde o valor nulo interdito poderá estar como parâmetro, portanto, no predicado, e tomando como exemplo o utente mais uma vez, terá que se passar o seu identificador, sendo depois retirado o utente e todos os seus parâmetros associados, de seguida é aplicado o predicado *nulointerdito*, apresentado anteriormente, só depois é que é permitida a involução desse dado utente.

```
involucao(utente(Id), interdito, nome) :-
    utente(Id, Nome, Idade, Cidade),
    nulointerdito(Nome),
    involucao(utente(Id, Nome, Idade, Cidade), positivo).

involucao(utente(Id), interdito, idade) :-
    utente(Id, Nome, Idade, Cidade),
    nulointerdito(Idade),
    involucao(utente(Id, Nome, Idade, Cidade), positivo).

involucao(utente(Id), interdito, cidade) :-
```

```
    utente(Id, Nome, Idade, Cidade),
    nulointerdito(Cidade),
    involucao(utente(Id, Nome, Idade, Cidade), positivo).
```

A mesma linha de pensamento se aplica no prestador e no cuidado.

```
involucao(prestador(Id), interdito) :-
    prestador(Id, Nome, Esp, Inst),
    nulointerdito(Nome),
    involucao(prestador(Id, Nome, Esp, Inst), positivo).

involucao(cuidado(Data, IdU, IdP, Custo), interdito, descricao) :-
    cuidado(Data, IdU, IdP, Desc, Custo),
    nulointerdito(Desc),
    involucao(cuidado(Data, IdU, IdP, Desc, Custo), positivo).

involucao(cuidado(Data, IdU, IdP, Desc), interdito, custo) :-
    cuidado(Data, IdU, IdP, Desc, Custo),
    nulointerdito(Custo),
    involucao(cuidado(Data, IdU, IdP, Desc, Custo), positivo).
```

3.8 Sistema de inferência

Devido à relativa complexidade dos sistemas apresentados anteriormente, foi necessário desenvolver um sistema de inferência adequado e capaz de apresentar conclusões viáveis, consoante a questão apresentada, que poderão ser **verdadeiras**, **falsas** ou **desconhecidas**.

A maneira como as questões são discernidas entre as suas conclusões possíveis é se, caso exista conhecimento acerca da questão da base de conhecimento então a resposta à questão será verdadeiro, caso exista uma negação explícita do conhecimento acerca da questão, então a resposta será falso. E por fim, caso não existe conhecimento a confirmar ou a negar a questão na base de conhecimento, então será desconhecido.

Portanto, tendo em conta os princípios anteriores, foi desenvolvido o seguinte predicado que incorpora essa linha de pensamento.

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

4 Conclusões e Sugestões

Após a elaboração deste trabalho prático, o seu valor em termos da consolidação da matéria foi incomensurável, uma vez que permitiu com que o grupo, para além de melhor perceber e entender os conceitos relacionados com o conhecimento imperfeito, bem como os invariantes estruturais, entendesse de uma melhor maneira a utilidade destes conhecimentos em situações práticas.

Contudo, a elaboração deste trabalho não veio sem dificuldades, por exemplo, no que toca a escolha da melhor maneira para implementar o conhecimento imperfeito, sendo que só após alguns retrocessos e avanços por parte do grupo, é que se chegou a uma conclusão sobre a melhor maneira de se avançar.

O resultado final, nos olhos do grupo, é bastante satisfatório e completo pelo que se espera que assim se mantenha no próximo trabalho prático que se adivinha.