

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho de Grupo – 2º Exercício

MIEI - 3º Ano, 2º Semestre

Universidade do Minho

Ano letivo 2016/2017

Ana Isabel Castro
(a55522)

Joana Miguel
(a57127)

Lúcia Abreu
(a71634)

Braga, 9 de Abril de 2017

Resumo

Este trabalho consistiu em desenvolver e adicionar funcionalidades e conhecimento ao trabalho realizado no Exercício 1: um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar o contexto das clínicas SaúdeMais, na sua área de prestação de cuidados de saúde pela realização de serviços de atos médicos, as consultas. As clínicas SaúdeMais estão distribuídas pelas principais cidades do norte de Portugal e prestam diversos serviços na área da saúde.

Para tal teve-se que demonstrar as funcionalidades subjacentes à programação em lógica estendida e à representação de conhecimento imperfeito, recorrendo à temática dos valores nulos. Para além do conhecimento perfeito positivo teve-se que adicionar a base de conhecimento exemplos de conhecimento perfeito negativo, e de conhecimento imperfeito incerto, impreciso e interdito.

Portanto recorreu-se à Programação em Lógica Estendida para representar as situações em que o conhecimento é desconhecido. Dado isto, criou-se novos invariantes de modo a lidar com os diferentes tipos de representação de conhecimento, assim como alterar a forma como o sistema evolui e regride, isto é, garantir que este sabe como tratar os diferentes tipos de conhecimento. Por fim, desenvolver um sistema que seja capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas, ou seja, que seja capaz de inferir se uma questão ou um conjunto de questões são verdadeiras, falsas, ou desconhecidas.

Conteúdo

1	Introdução	4
2	Caso de Estudo	6
3	Representação de Conhecimento Perfeito	9
3.1	Conhecimento Perfeito Positivo	9
3.1.1	Instituição	9
3.1.2	Serviço	10
3.1.3	Médico	10
3.1.4	Utente	10
3.1.5	Ato Médico	10
3.2	Conhecimento Perfeito Negativo	11
3.2.1	Instituição	11
3.2.2	Serviço	11
3.2.3	Médico	12
3.2.4	Utente	12
3.2.5	Ato Médico	12
4	Representação de Conhecimento Imperfeito	13
4.1	Conhecimento Imperfeito Incerto	13
4.1.1	Serviço	13
4.1.2	Médico	14
4.1.3	Utente	14
4.1.4	Ato Médico	14
4.2	Conhecimento Imperfeito Impreciso	15
4.2.1	Instituição	15
4.2.2	Serviço	15
4.2.3	Médico	16
4.2.4	Utente	16
4.2.5	Ato Médico	16
4.3	Conhecimento Imperfeito Interdito	17

4.3.1	Serviço	17
4.3.2	Médico	17
4.3.3	Utente	18
4.3.4	Ato Médico	19
5	Invariantes	20
5.1	Invariantes de Inserção	20
5.1.1	Instituição	20
5.1.2	Serviço	21
5.1.3	Médico	22
5.1.4	Utente	22
5.1.5	Consulta	23
5.1.6	Invariantes Gerais de Inserção	28
5.2	Invariantes de Remoção	28
5.2.1	Instituição	28
5.2.2	Serviço	29
5.2.3	Médico	29
5.2.4	Utente	29
6	Evolução do Conhecimento	30
6.1	Predicado evolução	30
6.1.1	Evoluir com conhecimento perfeito positivo	30
6.1.2	Evoluir com conhecimento perfeito negativo	33
6.1.3	Evoluir com conhecimento imperfeito do tipo I	35
6.1.4	Evoluir com conhecimento imperfeito do tipo II	35
6.1.5	Evoluir com conhecimento imperfeito do tipo III	36
6.2	Predicado regressão	37
7	Sistema de Inferência	39
7.1	Implementação em Prolog	40
8	Predicados Extra	41
8.1	Determinar a média de idade dos utentes	41
8.2	Número de consultas de um serviço num determinado ano	41
8.3	Serviço mais consultado de uma instituição num determinado ano	42
9	Testes e Resultados	43
9.1	Consultas à base de conhecimento	43
9.2	Predicados extra (funcionalidades)	45
9.2.1	Determinar a média de idade dos utentes	46
9.2.2	Número de consultas de um serviço num determinado ano	46

9.2.3	Serviço mais consultado de uma instituição num determinado ano	46
9.3	Evolução da base de conhecimento	46
10	Conclusão	49
	Bibliografia	50

Capítulo 1

Introdução

O segundo trabalho de grupo da unidade curricular de **Sistemas de Representação de Conhecimento e Raciocínio** do 3º ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho consiste no desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde pela realização de serviços de atos médicos.

Com este trabalho pretende-se continuar o que foi iniciado no trabalho anterior, utilizando a extensão à programação em lógica, e usando a linguagem de programação em lógica PROLOG, no âmbito da representação de **conhecimento imperfeito**.

Anteriormente, o sistema de representação de conhecimento implementado baseava-se no **Pressuposto do Mundo Fechado**. Segundo este pressuposto, tudo aquilo que não pode ser provado como sendo Verdade, é considerado Falso. Porém, nem sempre se pode assumir que a informação representada é a única que é válida.

Assim, neste trabalho, o sistema de conhecimento será capaz de raciocinar segundo os 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.

Como se assume a possibilidade da existência de informação válida que não está representada no sistema de representação de conhecimento, é necessário representar esse conhecimento incompleto para além do Verdadeiro e Falso. Para tal recorreu-se à **Programação em Lógica Estendida** para representar as situações em que o conhecimento é **desconhecido**. Torna-se então possível distinguir três tipos de conclusões para uma questão: esta pode ser **verdadeira**, **falsa** ou, quando não existe informação que permita inferir uma ou outra das conclusões anteriores, a resposta à questão será **desconhecida**.

Os principais **objetivos** com este trabalho prático são:

- Desenvolvimento de um sistema que seja capaz de representar as diversas ocorrências de uma instituição de saúde, tais como cuidados prestados e atos médicos, e respectivos intervenientes: os médicos e utentes.
- Representação de **conhecimento positivo e negativo**.
- Representação de casos de **conhecimento imperfeito**, pela utilização de valores nulos dos tipos: **incerto, impreciso e interdito**.
- Manipulação de **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.

Capítulo 2

Caso de Estudo



O Grupo SaúdeMais é uma empresa que inicialmente possuía uma rede de clínicas situadas no Norte de Portugal, nomeadamente:

- **csmbraga**, situada em Braga
- **csmporto**, situada no Porto
- **csmviana**, situada em Viana do Castelo
- **csmguimaraes**, situada em Guimarães

Que foram aumentando com o passar do tempo, e devido ao aumento da procura, tornando-se uma referência a nível nacional, com clínicas espalhadas por todo o país.

Estas clínicas situam-se em locais estratégicos, respondendo às exigências das metrópoles mais habitadas do país e possuindo, assim, um grande número de utentes que recorrem aos serviços das mesmas.

Num sistema desta dimensão, e para representar de forma prática e de grande utilidade todos os acontecimentos, circunstâncias e informações próprias de cada instituição de saúde, prevendo, antecipando e assegurando as funcionalidades que serão úteis e favoráveis ao funcionamento da instituição, é essencial que se possa representar factos que ainda não estão representados, e factos que ainda não estão na base de conhecimento. O Pressuposto do Mundo Fechado não o permite, pois tudo o que não está representado é considerado falso. Então, recorreu-se à Extensão da Programação em Lógica e aos seguintes valores nulos para melhor representar o sistema de conhecimento:

- **Conhecimento Imperfeito Incerto**

- Tal a dimensão da rede de clínicas, por vezes não se sabe qual a instituição onde determinado serviço é prestado, ou sabendo em que instituição é prestado um serviço, não se sabe a cidade em que se localiza;
- Por vezes, por lapso burocrático, não se sabe o serviço a que determinado médico está associado, nem saber qual o médico que deu uma determinada consulta.;
- Não se sabe qual a idade ou a morada de determinados utentes, que por terem uma idade avançada, podem não se lembrar dos seus dados pessoais; essa informação não é vital ao funcionamento da clínica;
- Sabendo a data de uma consulta, não saber a hora a que esta se realizou, pois a memória não permite lembrar esse detalhe.

- **Conhecimento Imperfeito Impreciso**

- Não se saber a localização precisa de uma instituição, sabendo-se que fica numa de duas localidades muito próximas;
- Não se saber qual a especialidade de um dado médico, sabendo-se que é uma de duas com pronúncia muito parecida, o que torna difícil de saber qual delas é;
- Não se saber em que instituição se disponibiliza determinado serviço, sabendo-se que uma de duas instituições o disponibiliza porque ambas possuem essa especialidade;
- Por lapso dos serviços administrativos, não se sabe se determinado médico presta um de dois serviços cujo id é consecutivo;
- Não se saber ao certo a idade de um utente, apenas um intervalo de anos;
- Não se saber a morada de um utente, apenas se sabe que é uma de duas com pronúncia muito parecida.
- Estima-se que o custo da consulta seja de 50 euros, mas pode ser 100 euros, depende da realização ou não de um exame adicional;
- Sabe-se que o custo de uma consulta está entre 20 a 30 euros devido a:
 - * Plano de Saúde
 - * Desconto
 - * Seguro de Saúde paga parte do valor, então não se sabe o custo
 - * A Segurança Social / ADSE paga uma percentagem, então não se sabe o custo

- **Conhecimento Imperfeito Interdito**

- Não se sabe a morada do utente nem nunca se virá conhecer porque:
 - * O utente quer confidencialidade
 - * O utente pode estar ao abrigo da proteção de testemunhas
 - * O utente pode ser vítima de violência doméstica
 - * O utente pode ser uma celebridade e não querer fornecer tal informação
 - * O utente pode ser idoso e não se lembrar
- Não se sabe nem se virá a conhecer o custo da consulta porque:
 - * O utente não quer que se saiba o custo da consulta
 - * O médico não quer que se saiba o custo da consulta
- Não se poderá vir a saber qual a instituição que disponibiliza determinado serviço, pois pode ser ilegal em Portugal

Sendo esta uma rede de clínicas de grande dimensão, e com um número elevado de utentes e cuidados prestados, é essencial para o bom funcionamento das instituições que seja possível esta flexibilidade de registo dos atos médicos prestados, assim como a informação sobre os utentes e médicos.

Capítulo 3

Representação de Conhecimento Perfeito

Neste projeto é abordada a representação do Conhecimento por duas perspectivas: **Conhecimento Perfeito** e Conhecimento Imperfeito.

Na representação de **Conhecimento Perfeito** é considerado que o conhecimento possui os seguintes pressupostos:

- **Pressuposto do Mundo Fechado** – toda a informação que não existe mencionada 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.

Assim, a representação do conhecimento perfeito e as suas respetivas cláusulas possuem como contradomínio do predicado apenas os valores de prova, **verdadeiro** e **falso**.

Neste capítulo iremos abordar o Conhecimento **Perfeito**, que se divide em Conhecimento **Positivo** e Conhecimento **Negativo**.

3.1 Conhecimento Perfeito Positivo

A representação de Conhecimento Perfeito Positivo, que corresponde à Base de Conhecimento do Trabalho 1, é feita através de factos que possuímos, e informação completa acerca dos mesmos.

De seguida estão detalhados alguns exemplos de conhecimento perfeito positivo para instituições, serviços, médicos, utentes e atos médicos.

3.1.1 Instituição

Cada instituição da rede de clínicas SaúdeMais é representada pelo seu nome e respetiva cidade. Como se trata de conhecimento positivo, sabemos que existe a instituição **csmviana** que se situa em **viana**. Aqui está representado parte do conhecimento positivo acerca das instituições:

```
% Extensao do predicado instituicao: NomeInst, Cidade -> {V,F,D}
instituicao( csmviana, viana ).
instituicao( csmporto, porto ).
instituicao( csmguimaraes, guimaraes ).
```

3.1.2 Serviço

Cada instituição da rede de clínicas tem ao seu dispor um conjunto de serviços variado. Cada serviço é representado pelo seu id, descrição, instituição e cidade. Existe, portanto, um serviço **s1 : ginecologia** na instituição **csmbraga**. Aqui está representado parte do conhecimento positivo acerca dos serviços:

```
% Extensao do predicado servico: IdServ, Descricao, Instituicao, Cidade -> {V,F,D}
servico( s1, ginecologia, csmbraga, braga ).
servico( s2, oftalmologia, csmviana, viana ).
servico( s3, ortopedia, csmporto, porto ).
```

3.1.3 Médico

Cada instituição da rede de clínicas SaúdeMais possui uma equipa de médicos, sendo que cada médico possui um id único, o seu nome, e o id do serviço que realiza na instituição. O médico **m1 : barros** possui o serviço **s1**. Aqui está representado parte do conhecimento positivo acerca dos médicos:

```
% Extensao do predicado medico: IdMed, Nome, IdServ -> {V,F,D}
medico( m1 , barros, s1 ).
medico( m2 , videira, s2 ).
medico( m3 , brandao, s3 ).
```

3.1.4 Utente

Os utentes que procuram os serviços da rede de clínicas podem ser identificados pelo seu id único, e também pelo seu nome, idade e morada. A rede de clínicas possui então uma utente **u1 : maria** com 20 anos que mora em **galegos**. Aqui está representado parte do conhecimento positivo acerca dos utentes:

```
% Extensao do predicado utente: IdUt, Nome, Idade, Morada -> {V,F,D}
utente( u1, maria, 20, galegos ).
utente( u2, rodrigo, 28, torre ).
utente( u3, joana, 59, barcelinhos ).
```

3.1.5 Ato Médico

Cada ato médico é caracterizado pela data em que aconteceu: **Ano**, **Mês**, **Dia** e **Hora**, e também pelo **id** do utente, **id** do serviço, **custo** do ato médico e **id** do médico. Assim sendo, no dia 16 de março de 2017 pelas 8h00 o utente **u5** compareceu a uma consulta do serviço **s4**, com um custo de 45 euros, e foi atendido pelo médico **m4**. Aqui está representado parte do conhecimento positivo acerca dos atos médicos:

```
% Extensao do predicado consulta: Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed -> {V,F,D}
consulta( 2017, 3, 16, 8, u5, s4, 45, m4 ).
consulta( 2017, 2, 23, 10, u5, s9, 25, m9 ).
consulta( 2017, 1, 5, 17, u2, s7, 50, m17 ).
```

3.2 Conhecimento Perfeito Negativo

Como, ao se estender a Programação em Lógica, se torna possível representar informação negativa explicitamente, a extensão de um programa em lógica passa a contar com dois tipos de negação:

- **Negação por falha** - negação utilizada nos programas em lógica tradicional, sendo representada pelo termo (não)
- **Negação forte** - negação utilizada como forma de identificar informação negativa, ou falsa, representada pela conectiva (\neg).

A distinção entre estes dois tipos de negação, $\text{nao}(p)$ e $(\neg p)$ justifica-se pelo facto de passarmos a aceitar o pressuposto do mundo aberto quando estamos a representar o conhecimento imperfeito, pois podemos não possuir toda a informação que existe acerca de p .

É, então, necessário distinguir:

- **$\text{nao}(p)$** - aceitamos que p é falso por falta de prova (*Conhecimento Perfeito*)
- **$\neg p$** - conseguimos provar que é falso através de negação forte (*Conhecimento Imperfeito*)

Uma vez que é necessário representar o **conhecimento negativo** de uma forma explícita, foi necessário definir alguns predicados auxiliares para representar este tipo de informação.

3.2.1 Instituição

Para representar a informação negativa relativa às instituições recorreu-se à negação de factos: a instituição `csmlisboa` não existe em Lisboa.

```
% Conhecimento perfeito negativo
-instituicao( NomeInst, Cidade ) :-
    nao( instituicao( NomeInst, Cidade ) ),
    nao( excecacao( instituicao( NomeInst, Cidade ) ) ).

% Não existe a instituição csmlisboa em Lisboa
-instituicao( csmlisboa, lisboa ).
```

3.2.2 Serviço

Para representar a informação negativa relativa aos serviços recorreu-se à negação de factos: no Porto não existe o serviço `s1` : Ginecologia.

```
% Conhecimento perfeito negativo
-servico( IdServ, Descricao, Instituicao, Cidade ) :-
    nao( servico( IdServ, Descricao, Instituicao, Cidade )),
    nao( excecacao( servico( IdServ, Descricao, Instituicao, Cidade )))

% No Porto não existe o serviço s1: Ginecologia
-servico( s1, genecologia, csmporto, porto ).
```

3.2.3 Médico

Para representar a informação negativa relativa aos médicos recorreu-se à negação de factos: o médico **m1** : **barros** não possui o serviço **s10**.

```
% Conhecimento perfeito negativo
-medico( IdMed, Nome, IdServ ) :-
    nao( medico( IdMed, Nome, IdServ )),
    nao( excecacao( medico( IdMed, Nome, IdServ )))

% O médico m1: barros não tem o serviço 10
-medico( m1, barros, s10 ).
```

3.2.4 Utente

Para representar a informação negativa relativa aos utentes recorreu-se à negação de factos: a utente **u1** : **maria** não tem 30 anos e o utente **u2** : **rodrigo** não vive em **galegos**.

```
% Conhecimento perfeito negativo
-utente( IdUt, Nome, Idade, Morada ) :-
    nao( utente( IdUt, Nome, Idade, Morada )),
    nao( excecacao( utente( IdUt, Nome, Idade, Morada )))

% A utente u1: maria não tem 30 anos
-utente( u1, maria, 30, galegos ).

% O utente u2: rodrigo não vive em galegos
-utente( u2, rodrigo, 28, galegos ).
```

3.2.5 Ato Médico

Para representar a informação negativa relativa aos atos médicos recorreu-se à negação de factos: a consulta de 16-3-2017 ao utente **u5** do serviço **s4** com o médico **m4** não custou 100 euros.

```
% Conhecimento perfeito negativo
-consulta( Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed ) :-
    nao( consulta( Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed )),
    nao( excecacao( consulta( Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed )))

% A consulta de 16-3-2017 ao utente 5 do serviço s4 com o médico m4 não custou 100 euros.
-consulta( 2017, 3, 16, 8, u5, s4, 100, m4 ).
```

Capítulo 4

Representação de Conhecimento Imperfeito

Como já referido anteriormente, este tipo de representação do conhecimento passamos a aceitar pressupostos diferentes daqueles que eram a base do conhecimento perfeito. Para representar informação que não se encontra ainda no sistema de representação, recorre-se, para além do Verdadeiro e Falso, ao Desconhecido. Para tal, recorreu-se à Programação em Lógica Estendida.

4.1 Conhecimento Imperfeito Incerto

Para representar o conhecimento imperfeito **Incerto**, recorreu-se aos valores nulos do tipo **Incerto** e desconhecido. São valores dos quais não possuímos qualquer informação possível sobre o conjunto de valores.

4.1.1 Serviço

Sabe-se que certo serviço é prestado numa dada cidade, mas desconhece-se qual a instituição, sendo a informação representada da seguinte forma:

```
% Desconhecida a instituição onde se presta este serviço
servico( s11, endocrinologia, xptos11, braga ).

execcao( servico( IdServ, Descricao, Instituicao, Cidade )) :-
    servico( IdServ, Descricao, xptos11, Cidade ).
```

Sabe-se que certo serviço é prestado numa dada instituição, mas desconhece-se qual a cidade onde a mesma se localiza, sendo a informação representada da seguinte forma:

```
% Desconhecida a cidade onde se localiza a instituição que presta este serviço
servico( s12, psiquiatria, csmporto, xptos12 ).

execcao( servico( IdServ, Descricao, Instituicao, Cidade )) :-
    servico( IdServ, Descricao, Instituicao, xptos12 ).
```

4.1.2 Médico

Não se sabe qual o serviço a que o médico `m21` : `gouveia` está associado, sendo a informação representada da seguinte forma:

```
% Não se sabe qual o serviço que o médico gouveia está associado
medico( m21, gouveia, xptom21 ).

execcao( medico( IdMed, Nome, IdServ )) :-
    medico( IdMed, Nome, xptom21 ).
```

4.1.3 Utente

Não se sabe qual a morada do utente `u16` : `jose` de 50 anos de idade, sendo a informação representada da seguinte forma:

```
% Não se sabe a morada do utente jose
utente( u16, jose, 50, xptou16 ).

execcao( utente( IdUt, Nome, Idade, Morada )) :-
    utente( IdUt, Nome, Idade, xptou16 ).
```

Não se sabe qual a idade da utente `u17` : `susana` da localidade de `parada`, sendo a informação representada da seguinte forma:

```
% Não se sabe a idade da utente susana
utente( u17, susana, xptou17, parada ).

execcao( utente( IdUt, Nome, Idade, Morada )) :-
    utente( IdUt, Nome, xptou17, Morada ).
```

4.1.4 Ato Médico

Apesar de se saber que se realizou dia 15/01/2017, não se sabe qual a hora da consulta ao utente `u10` do serviço `s4`, que foi efetuada pelo médico `m4` e cujo custo foi de 40 euros, sendo a informação representada da seguinte forma:

```
% Não se sabe a que horas ocorreu a consulta (apesar de saber a data dia/mes/ano)
consulta( 2017, 1, 15, xptoc1, u10, s4, 40, m4 ).

execcao( consulta( Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed )) :-
    consulta( Ano, Mes, Dia, xptoc1, IdUt, IdServ, Custo, IdMed ).
```


Não se sabe qual o médico que deu a consulta do dia 03/04/2017 pelas 20h, do serviço `s4`, ao utente `u10` e cujo custo foi de 42 euros, sendo a informação representada da seguinte forma:

```
% Não se sabe que médico deu a consulta
consulta( 2017, 4, 3, 20, u10, s4, 42, xptoc2 ).

excecao( consulta( Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed ) ) :-
    consulta( Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, xptoc2 ).
```

4.2 Conhecimento Imperfeito Impreciso

Para representar o conhecimento imperfeito **Impreciso**, recorreu-se aos valores nulos do tipo **Impreciso** e desconhecido. São valores dos quais sabemos que pertencem a um conjunto determinado de valores.

4.2.1 Instituição

Sabe-se que a instituição `csmgualtar` se situa em `vizela` ou `celorico`, sendo a informação representada da seguinte forma:

```
% A instituição csmgualtar fica em vizela ou em celorico
excecao( instituicao( csmgualtar, vizela ) ).
excecao( instituicao( csmgualtar, celorico ) ).
```

4.2.2 Serviço

Não se sabe qual a especialidade do serviço `s13`, mas sabe-se que ou é `oncologia` ou é `podologia`, sendo a informação representada da seguinte forma:

```
% Não se sabe qual a descrição/especialidade do serviço s13
% mas sabe-se que é oncologia ou podologia
excecao( servico( s13, oncologia, csmguimaraes, guimaraes ) ).
excecao( servico( s13, podologia, csmguimaraes, guimaraes ) ).
```

Não se sabe qual a instituição que disponibiliza o serviço `s14`, mas sabe-se que ou é `csmporto` ou é `csmbraga`, sendo a informação representada da seguinte forma:

```
% Não se sabe qual em que instituição se disponibiliza o serviço s14 de pneumologia,
% ou é csmporto ou é csmbraga.
excecao( servico( s14, pneumologia, csmbraga, braga ) ).
excecao( servico( s14, pneumologia, csmporto, porto ) ).
```

4.2.3 Médico

Não se sabe qual serviço o médico **m22** : **peixoto** presta, se o serviço **s5** ou **s6**, sendo a informação representada da seguinte forma:

```
% Não se sabe se o médico m22, Dr. Peixoto, prestada o serviço s5 ou s6
execcao( medico( m22, peixoto, s5 ) ).
execcao( medico( m22, peixoto, s6 ) ).
```

4.2.4 Utente

Não se sabe qual a idade do **xavier**, se 24 ou 25 anos, sendo a informação representada da seguinte forma:

```
% Não se sabe ao certo a idade do xavier mas sabe-se é 24 ou 25 anos
execcao( utente( u18, xavier, 24, gondizalves ) ).
execcao( utente( u18, xavier, 25, gondizalves ) ).
```

Não se sabe ao certo a idade do **sergio**, mas sabe-se que é entre 20 e 25 anos, sendo a informação representada da seguinte forma:

```
% Não se sabe ao certo a idade do sergio mas sabe-se que ou é entre 20 a 25 anos
execcao( utente(u19, sergio, Idade, gondizalves)) :-
    Idade >= 20, Idade <= 25.
```

Não se sabe qual a morada da **lucinda**, se **tondela** ou **vizela**, sendo a informação representada da seguinte forma:

```
% Não se conhece a morada da lucinda, é tondela ou meadela
execcao( utente( u20, lucinda, 45, tondela ) ).
execcao( utente( u20, lucinda, 45, meadela ) ).
```

4.2.5 Ato Médico

Não se sabe qual o valor da **consulta**, se 50 ou 150 euros, sendo a informação representada da seguinte forma:

```
% Custa da consulta é de 50 euros ou de 150 euros
execcao( consulta( 2016, 7, 17, 15, u8, s3, 50, m3 ) ).
execcao( consulta( 2016, 7, 17, 15, u8, s3, 150, m3 ) ).
```

Não se sabe ao certo o valor da **consulta**, mas sabe-se que é entre 20 e 30 euros, sendo a informação representada da seguinte forma:

```
% O valor da consulta está entre 20 a 30 euros
execcao( consulta( 2016, 3, 17, 16, u8, s3, Custo, m3 ) )
:- Custo >= 20 , Custo <= 30.
```

Não se sabe qual o médico que deu a `consulta`, se o médico `m10` ou o `m20`, sendo a informação representada da seguinte forma:

```
% Não se sabe se foi o médico m10 ou m20 que deu a consulta
execcao( consulta( 2017, 2, 10, 18, u14, s10, 50, m10 ) ).
execcao( consulta( 2017, 2, 10, 18, u14, s10, 50, m20 ) ).
```

4.3 Conhecimento Imperfeito Interdito

Para representar o conhecimento imperfeito **Interdito**, recorreu-se aos valores nulos do tipo **Interdito** e desconhecido. São valores dos quais não temos informação nem nunca poderemos vir a ter, não é permitido conhecê-la.

4.3.1 Serviço

Não se poderá vir a saber qual instituição possui o serviço `s15`, sendo a informação representada da seguinte forma:

```
% Nunca se poderá saber qual a instituição que tem o serviço 15
servico( s15, reumatologia, xptos15, porto ).
execcao( servico( IdServ, Descricao, Instituicao, Cidade )) :-
    servico( IdServ, Descricao, xptos15, Cidade ).
nulo( xptos15 ).
```

Para garantir que o conhecimento é interdito, implementou-se um invariante que não permite associar uma instituição ao serviço com id `s15`:

```
+servico( IdServ, Descricao, Instituicao, Cidade ) ::
    ( solucoes(
        ( IdServ, Descricao, InstituicaoS, Cidade ),
        ( servico( s15, reumatologia, InstituicaoS, porto ), nao( nulo( InstituicaoS )) ),
        S ),
        comprimento( S, N ), N==0 ).
```

4.3.2 Médico

Não se poderá vir a saber qual o nome do médico `m23`, sendo a informação representada da seguinte forma:

```
% Nunca se poderá saber o nome do médico m23
medico( m23, xptom23, s10 ).
execcao( medico( IdMed, Nome, IdServ )) :-
    medico( IdMed, xptom23, IdServ ).
nulo( xptom23 ).
```

Para garantir que o conhecimento é interdito, implementou-se um invariante que não permite associar um nome ao médico com id m23:

```
+medico( IdMed, Nome, IdServ ) ::  
  ( solucoes(  
    ( IdMed, NomeS, IdServ ),  
    ( medico( m23, NomeS, s10 ), nao( nulo( NomeS ) ) ),  
    S ),  
    comprimento( S, N ), N==0 ).
```

4.3.3 Utente

Não se poderá vir a saber qual a morada do utente u21, sendo a informação representada da seguinte forma:

```
% Não se sabe a morada do utente u21 nem nunca se virá conhecer  
utente( u21, marta, 35, xptou21 ).  
execcao( utente( IdUt, Nome, Idade, Morada ) ) :-  
  utente( IdUt, Nome, Idade, xptou21 ).  
nulo( xptou21 ).
```

Para garantir que o conhecimento é interdito, implementou-se um invariante que não permite associar uma morada ao utente com id u21:

```
+utente( IdUt, Nome, Idade, Morada ) ::  
  ( solucoes(  
    ( IdUt, Nome, Idade, MoradaS ),  
    ( utente( u21, marta, 35, MoradaS ), nao( nulo( MoradaS ) ) ),  
    S ),  
    comprimento( S, N ), N==0 ).
```

Não se poderá vir a saber qual a idade do utente u22, sendo a informação representada da seguinte forma:

```
% Não se sabe a idade do utente nem nunca se virá conhecer  
utente( u22, matilde, xptou22, tibiaes ).  
execcao( utente( IdUt, Nome, Idade, Morada ) ) :-  
  utente( IdUt, Nome, xptou22, Morada ).  
nulo( xptou22 ).
```

Para garantir que o conhecimento é interdito, implementou-se um invariante que não permite associar uma idade ao utente com id u22:

```
+utente( IdUt, Nome, Idade, Morada ) ::  
  ( solucoes(  
    ( IdUt, Nome, IdadeS, Morada ),  
    ( utente( u22, matilde, IdadeS, tibiaes ), nao( nulo( IdadeS ) ) ),  
    S ),  
    comprimento( S, N ), N==0 ).
```

Não se poderá vir a saber qual o nome do utente u23, sendo a informação representada da seguinte forma:

```
% Não se sabe a nome do utente nem nunca se virá conhecer
utente( u23, xptou23, 40, gualtar ).
execcao( utente(IdUt, Nome, Idade, Morada) ) :-
    utente( IdUt, xptou23, Idade, Morada ).
nulo( xptou23 ).
```

Para garantir que o conhecimento é interdito, implementou-se um invariante que não permite associar um nome ao utente com id u23:

```
+utente(IdUt, Nome, Idade, Morada) ::
    ( solucoes(
        ( IdUt, NomeS, Idade, Morada ),
        ( utente(u23, NomeS, 40, gualtar ), nao( nulo( NomeS ) ) ),
        S ),
    comprimento( S, N ), N==0 ).
```

4.3.4 Ato Médico

Não se poderá vir a saber qual o custo desta consulta, sendo a informação representada da seguinte forma:

```
% Não se sabe nem se virá a conhecer o custo da consulta
consulta( 2017, 3, 10, 10, u5, s5, xptoc3, m5 ).
execcao( consulta( Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed ) ) :-
    consulta( Ano, Mes, Dia, Hora, IdUt, IdServ, xptoc3, IdMed ).
nulo( xptoc3 ).
```

Para garantir que o conhecimento é interdito, implementou-se um invariante que não permite associar um custo à consulta do dia 10/03/2017, ao utente u5, do serviço s5, pelo médico m5:

```
+consulta( Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed ) ::
    ( solucoes(
        ( Ano, Mes, Dia, Hora, IdUt, IdServ, CustoS, IdMed ),
        ( consulta( 2017, 3, 10, 10, u5, s5, CustoS, m5 ), nao( nulo( CustoS ) ) ),
        S ),
    comprimento( S, N ), N==0 ).
```

Capítulo 5

Invariantes

De modo a garantir que o conhecimento perfeito positivo e negativo, assim como como conhecimento imperfeito (incerto, impreciso e interdito) são adicionados e removidos de forma adequada respeitando a consistência e integridade dos dados da base de conhecimento, implementou-se vários invariantes. Para tal, foram implementados dois tipos de invariantes:

- **Invariantes Estruturais:** Garantem que a estrutura da base de conhecimento é mantida após cada inserção e cada remoção. Por exemplo, garantindo que não é adicionado conhecimento repetido
- **Invariantes Referenciais:** Garantem que as regras de domínio lógico do problema são cumpridas

5.1 Invariantes de Inserção

Implementaram-se invariantes de inserção para cada um dos predicados que pertencem à base de conhecimento. Uma vez que deixou-se de aplicar o Pressuposto do Mundo Fechado(PMF), aplicado no Exercício 1, e passou-se a estar no domínio da Programação em Lógica Estendida e do Conhecimento Imperfeito, há que implementar vários invariantes que garantam que a informação é adicionada de forma correta.

5.1.1 Instituição

Uma vez que, uma instituição é identificada pelo seu nome, garantiu-se através deste invariante que não é possível adicionar predicados sobre instituições quando esta já existe, a não ser que seja conhecimento imperfeito em que pode ter mais que um predicado com o mesmo nome da instituição. Para além disso, garante-se que não existe informação contraditória, isto é, apenas pode existir na base do conhecimento ou informação positiva, ou informação negativa ou informação incompleta sobre uma dada instituição.

```
% Invariante Estrutural  
% Permitir apenas a existência de um predicado com um dado identificador  
% de conhecimento perfeito positivo ou perfeito negativo.
```

```

% Em alternativa permite ter vários predicados com o mesmo identificador
% de conhecimento imperfeito.
+instituicao(Instituicao, _) :: (
    solucoes(Instituicao, instituicao(Instituicao, _), Lista1),
    solucoes(Instituicao, -instituicao(Instituicao, _), Lista2),
    solucoesSemRepetidos(Instituicao, excecacao(instituicao(Instituicao, _)), Lista3),
    comprimento(Lista1, N1),
    comprimento(Lista2, N2),
    comprimento(Lista3, N3),
    (N1+N2+N3) == 1).

```

5.1.2 Serviço

Uma vez que, um serviço é identificado pelo seu identificador, garantiu-se através deste invariante que não é possível adicionar predicados sobre serviços quando este já existe, a não ser que seja conhecimento imperfeito em que pode ter mais que um predicado com o mesmo identificador de serviço. Para além disso, garante-se que não existe informação contraditória, isto é, apenas pode existir na base do conhecimento ou informação positiva, ou informação negativa ou informação incompleta sobre um dado serviço.

```

% Invariante Estrutural
% Permitir apenas a existência de um predicado com um dado identificador
% de conhecimento perfeito positivo ou perfeito negativo.
% Em alternativa permite ter vários predicados com o mesmo identificador
% de conhecimento imperfeito.
+servico(IdServ, _, _, _) :: (
    solucoes(IdServ, servico(IdServ, _, _, _), Lista1),
    solucoes(IdServ, -servico(IdServ, _, _, _), Lista2),
    solucoesSemRepetidos(IdServ, excecacao(servico(IdServ, _, _, _)), Lista3),
    comprimento(Lista1, N1),
    comprimento(Lista2, N2),
    comprimento(Lista3, N3),
    (N1+N2+N3) == 1).

```

O invariante seguinte assegura que na inserção de um serviço, este tem que ficar associado a uma instituição que exista na base de conhecimento. Caso contrário, não se permite que seja adicionado.

```

% Invariante Referencial
% O serviço tem que corresponder a uma instituição existente na base de conhecimento
+servico(_, _, Instituicao, Cidade) :: (
    solucoes(Instituicao, instituicao(Instituicao, Cidade), Lista),
    comprimento(Lista, N),
    N > 0).

```

Uma vez que cada instituição tem os seus próprios serviços, garantiu-se que cada serviço apenas fica associado a uma instituição.

```

% Invariante Estrutural
% Não permite repetir serviços numa instituição
+servico(_, Descricao, Instituicao, _) :: (
    solucoes(Descricao, servico(_, Descricao, Instituicao, _), Lista),
    comprimento(Lista, N),

```

```
N == 1).
```

5.1.3 Médico

Uma vez que, um médico é identificado pelo seu identificador e serviço, garantiu-se através deste invariante que não é possível adicionar predicados sobre médicos quando este já existe para aquele serviço, a não ser que seja conhecimento imperfeito em que pode ter mais que um predicado com o mesmo identificador de médico e serviço. Para além disso, garante-se que não existe informação contraditória, isto é, apenas pode existir na base do conhecimento ou informação positiva, ou informação negativa ou informação incompleta sobre um dado médico e serviço.

```
% Invariante Estrutural
% Permitir apenas a existência de um predicado com um dado identificador
% de conhecimento perfeito positivo ou perfeito negativo.
% Em alternativa permite ter vários predicados com o mesmo identificador
% de conhecimento imperfeito.
+medico(IdMed, _, IdServ) :: (
    solucoes(IdMed, medico(IdMed, _, IdServ), Lista1),
    solucoes(IdMed, -medico(IdMed, _, IdServ), Lista2),
    solucoesSemRepetidos(IdMed, excecao(medico(IdMed, _, IdServ)), Lista3),
    comprimento(Lista1, N1),
    comprimento(Lista2, N2),
    comprimento(Lista3, N3),
    (N1+N2+N3) == 1).
```

O invariante seguinte assegura que na inserção de um médico, este tem que ficar associado a um serviço que exista na base de conhecimento. Caso contrário, não se permite que seja adicionado.

```
% Invariante Referencial
% Não permitir inserção de um médico num serviço que não existe
+medico(IdMed, _, IdServ) :: (
    solucoes(IServ, servico(IdServ, _, _, _), Lista),
    comprimento(Lista, N),
    N > 0).
```

Além disso, garante-se que cada médico apenas fica associado a um serviço, conforme a especialidade em que se formou.

```
% Invariante Referencial
% Não permitir inserção de um médico com mais que uma especialidade
+medico(IdMed, _, _) :: (
    solucoesSemRepetidos(Especialidade,
        (medico(IdMed, _, IdServ), servico(IdServ, Especialidade, _, _)), Lista),
    comprimento(Lista, N),
    N == 1).
```

5.1.4 Utente

Uma vez que, um utente é identificado pelo seu identificador, garantiu-se através deste invariante que não é possível adicionar predicados sobre utentes quando este já existe, a não ser que seja conhecimento imperfeito em que pode ter mais que um predicado com o mesmo identificador de utente.

Para além disso, garante-se que não existe informação contraditória, isto é, apenas pode existir na base do conhecimento ou informação positiva, ou informação negativa ou informação incompleta sobre um dado utente.

```
% Invariante Estrutural
% Permitir apenas a existência de um predicado com um dado identificador
% de conhecimento perfeito positivo ou perfeito negativo.
% Em alternativa permite ter vários predicados com o mesmo identificador
% de conhecimento imperfeito.
+utente(IdUt, _, _, _) :: (
    solucoes(IdUt, utente(IdUt, _, _, _), Lista1),
    solucoes(IdUt, -utente(IdUt, _, _, _), Lista2),
    solucoesSemRepetidos(IdUt, execucao(utente(IdUt, _, _, _)), Lista3),
    comprimento(Lista1, N1),
    comprimento(Lista2, N2),
    comprimento(Lista3, N3),
    (N1+N2+N3) == 1).
```

5.1.5 Consulta

Uma vez que, uma consulta é identificado pelo conteúdo dos seus campos, garantiu-se através deste invariante que não é possível adicionar predicados sobre consultas quando esta já existe, a não ser que seja conhecimento imperfeito em que pode ter mais que um predicado com os mesmos campos da consulta. Para além disso, garante-se que não existe informação contraditória, isto é, apenas pode existir na base do conhecimento ou informação positiva, ou informação negativa ou informação incompleta sobre uma dada consulta.

```
% Invariante Estrutural
% Permitir apenas a existência de um predicado com um dado identificador
% de conhecimento perfeito positivo ou perfeito negativo.
% Em alternativa permite ter vários predicados com o mesmo identificador
% de conhecimento imperfeito.
+consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed) :: (
    solucoes(Hora, consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed), Lista1),
    solucoes(Hora, -consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed), Lista2),
    solucoesSemRepetidos(Hora,
        execucao(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed)), Lista3),
    comprimento(Lista1, N1),
    comprimento(Lista2, N2),
    comprimento(Lista3, N3),
    (N1+N2+N3) == 1).
```

Para garantir a correta inserção de consultas no sistema implementou-se o seguinte invariante de forma a garantir que cada consulta apresenta sempre uma referência para um utente existente no sistema.

```
% Invariante Estrutural
% Não permitir a adição de uma consulta com um utente inexistente
+consulta(_, _, _, _, IdUt, _, _, _) :: (
    solucoes(IdUt, utente(IdUt, _, _, _), Lista),
    comprimento(Lista, N),
```

```
N > 0).
```

Assim como, implementou-se o seguinte invariante de forma a garantir que cada consulta apresenta sempre uma referência para um médico existente no sistema.

```
% Invariante Estrutural
% Não permitir a adição de uma consulta com um médico inexistente
+consulta(_, _, _, _, _, _, IdMed) :: (
    solucoes(IdMed, medico(IdMed, _, _), Lista),
    comprimento(Lista, N),
    N > 0).
```

No mesmo sentido, assegura-se que não é possível adicionar uma consulta que referencie um serviço que não existe numa instituição.

```
% Invariante Estrutural
% Não permitir a adição de uma consulta com um serviço inexistente numa Instituição
+consulta(_, _, _, _, IdServ, _, _) :: (
    solucoes(IdServ, servico(IdServ, _, _), Lista),
    comprimento(Lista, N),
    N > 0).
```

O seguinte invariante garante que a consulta só é adicionada se o médico que realizou a consulta fornece o serviço em especificado.

```
% Invariante Referencial
% Não permite inserir uma consulta com um médico que não forneça o serviço pretendido
+consulta(_, _, _, _, IdServ, _, IdMed) :: (
    solucoes(IdMed, medico(IdMed, _, IdServ), Lista),
    comprimento(Lista, N),
    N > 0).
```

Na inserção de uma consulta especificou-se que a data da consulta tem que ser concordante com o calendário e respeitar os dias e meses do ano. Para tal, garantiu-se que nos meses com 31 dias, o dia inserido não pode ser superior a 31.

```
% Invariante Referencial
% Não permitir a adição de uma consulta nos meses que têm 31 dias, com o dia superior a 31.
+consulta(_, _, _, _, _, _, _) :: (
    solucoes(
        Mes,
        (consulta(_, Mes, Dia, _, _, _, _),
         pertence(Mes, [1,3,5,7,8,10,12]),
         Dia > 31),
        [])).
```

Ainda no mesmo sentido, para os meses que apresentam 30 dias garante-se que o valor do dia inserido não pode ser superior a 30.

```

% Invariante Referencial
% Não permitir a adição de uma consulta nos meses que têm 30 dias, com o dia superior a 30.
+consulta(_, _, _, _, _, _, _, _) :: (
    solucoes(
        Mes,
        (consulta(_, Mes, Dia, _, _, _, _, _),
         pertence(Mes, [2,4,6,9,11]),
         Dia > 30),
        []).

```

Garante-se, também, que as consultas no mês de Fevereiro não podem apresentar um dia superior a 29.

```

% Invariante Referencial
% Não permitir a adição de uma consulta no mês de fevereiro com o dia superior a 29.
+consulta(_, _, _, _, _, _, _, _) :: (
    solucoes(
        Dia,
        (consulta(_, 2, Dia, _, _, _, _, _),
         Dia > 29),
        []).

```

Por fim, garante-se que o dia do mês não pode ser inferior a 1.

```

% Invariante Referencial
% Não permitir a adição de uma consulta com um dia do mês menor que 1.
+consulta(_, _, _, _, _, _, _, _) :: (
    solucoes(
        Dia,
        (consulta(_, _, Dia, _, _, _, _, _),
         Dia < 1),
        []).

```

Relativamente aos meses, garante-se que o mês escolhido é válido, ou seja, está compreendido entre 1 (Janeiro) e 12 (Dezembro). Desta forma garante-se que não são escolhidos meses inexistentes.

```

% Invariante Referencial
% Não permitir a adição de uma consulta com um mês inferior a 1 e superior a 12
+consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, _, IdMed) :: (
    solucoes(Mes,
        (consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, _, IdMed), (Mes >= 1, Mes <= 12)), Lista),
    comprimento(Lista, N),
    N == 1).

```

Como descrito no caso de estudo, o horário de funcionamento dos estabelecimento clínico é das 8h às 22h, desta forma garante-se que o registo das consultas terá que respeitar esses limites.

```

% Invariante Referencial
% Não permitir a adição de consultas com a hora inferior a 8 ou superior a 22
+consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, _, IdMed) :: (
    solucoes(Hora,
        (consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, _, IdMed), (Hora >= 8, Hora <= 22)), Lista),

```

```
comprimento(Lista, N),  
N == 1).
```

De forma a traduzir o impedimento de um médico conseguir dar duas consultas ao mesmo tempo, implementou-se o seguinte invariante.

```
% Invariante Referencial  
% Não permitir que um medico tenha mais que uma consulta à mesma hora  
+consulta(Ano, Mes, Dia, Hora, _, _, _, IdMed) :: (  
    solucoes(IdMed, consulta(Ano, Mes, Dia, Hora, _, _, _, IdMed), Lista),  
    comprimento(Lista, N),  
    N == 1).
```

De forma análoga, para impossibilitar que um utente tenha dois registos de consultas ao mesmo tempo, implementou-se o seguinte invariante.

```
% Invariante Referencial  
% Não permitir que um utente tenha mais que uma consulta à mesma hora  
+consulta(Ano, Mes, Dia, Hora, IdUt, _, _, _) :: (  
    solucoes(IdUt, consulta(Ano, Mes, Dia, Hora, IdUt, _, _, _), Lista),  
    comprimento(Lista, N),  
    N == 1).
```

Na inserção de uma consulta, garante-se que o respetivo custo é maior ou igual a zero.

```
% Invariante Referencial  
% Não permitir a adição de consultas com um custo negativo  
+consulta(_, _, _, _, _, _, Custo, _) :: (  
    solucoes(Custo, (consulta(_, _, _, _, _, _, _, Custo, _), Custo < 0), Lista),  
    comprimento(Lista, N),  
    N == 0).
```

Na inserção de um predicado consulta negativo especificou-se que a data da consulta tem que concordar com o calendário. Para tal, garantiu-se que nos meses com 31 dias, o dia inserido não pode ser superior a 31.

```
% Invariante Referencial  
% Não permitir a adição de um predicado consulta negativa nos meses que têm 31 dias,  
% com o dia superior a 31.  
+consulta(_, _, _, _, _, _, _, _) :: (  
    solucoes(  
        Mes,  
        (-consulta(_, Mes, Dia, _, _, _, _, _),  
         pertence(Mes, [1,3,5,7,8,10,12]),  
         Dia > 31),  
        []).
```

Para os meses que apresentam 30 dias garante-se que o valor do dia inserido não pode ser superior a 30.

```

% Invariante Referencial
% Não permitir a adição de um predicado consulta negativa nos meses que têm 30 dias,
% com o dia superior a 30.
+consulta(_, _, _, _, _, _, _, _) :: (
    solucoes(
        Mes,
        (-consulta(_, Mes, Dia, _, _, _, _, _),
         pertence(Mes, [2,4,6,9,11]),
         Dia > 30),
        []).

```

Garante-se, também, que as consultas no mês de Fevereiro não podem apresentar um dia superior a 29.

```

% Invariante Referencial
% Não permitir a adição de um predicado consulta negativa no mês de fevereiro
% com o dia superior a 29.
+consulta(_, _, _, _, _, _, _, _) :: (
    solucoes(
        Dia,
        (-consulta(_, 2, Dia, _, _, _, _, _),
         Dia > 29),
        []).

```

Por fim, garante-se que o dia do mês não pode ser inferior a 1.

```

% Invariante Referencial
% Não permitir a adição de um predicado consulta negativa com um dia do mês menor que 1.
+consulta(_, _, _, _, _, _, _, _) :: (
    solucoes(
        Dia,
        (-consulta(_, _, Dia, _, _, _, _, _),
         Dia < 1),
        []).

```

Relativamente aos meses, garante-se que o mês escolhido é válido, ou seja, está compreendido entre 1 (Janeiro) e 12 (Dezembro). Desta forma garante-se que não são escolhidos meses inexistentes.

```

% Invariante Referencial
% Não permitir a adição de um predicado consulta negativo com um mês inferior
% a 1 e superior a 12
+consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed) :: (
    solucoes(Mes,
        (-consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed), (Mes >= 1, Mes <= 12)), Lista),
    comprimento(Lista, N),
    N == 1).

```

Relativamente ao horário, garante-se que a hora escolhida está entre 0 e as 23 horas, o que corresponde às horas de um dia.

```

% Invariante Referencial
% Não permitir a adição um predicado consulta negativa com a hora inferior
% a 0 ou superior a 23
+consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed) :: (
    solucoes(Hora,
        (-consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed), (Hora >= 0, Hora =< 23)),
        Lista),
    comprimento(Lista, N),
    N == 1).

```

5.1.6 Invariantes Gerais de Inserção

Este invariante garante que não se pode inserir duas exceções repetidas.

```

% Invariante Estrutural
% Invariante que não permite a adição de exceções repetidas
+excecao(E) :: (
    solucoes( E , excecao(E) , Lista), comprimento(Lista,N), N==1).

```

Este invariante garante que não se pode inserir valores nulos repetidos.

```

% Invariante Estrutural
% Invariante que não permite a adição de nulo repetidos
+nulo(V) :: (
    solucoes( V, nulo(V), Lista), comprimento(Lista,N), N==1).

```

5.2 Invariantes de Remoção

De forma a garantir que a informação é removida de forma correta implementaram-se invariantes de remoção para alguns predicados da base de conhecimento. Assim, cada vez que se pretende remover um termo relativo a esses predicados, todos os invariantes de remoção associados terão de ser cumpridos. Apenas foram feitos invariantes para a remoção de predicados que possam ter consequência na integridade da base de conhecimento.

5.2.1 Instituição

Para garantir que a informação de cada serviço se mantém íntegra, implementou-se um invariante que impede que se remova uma instituição que esteja associada a pelo menos um serviço.

```

% Invariante Referencial
% Não deixa remover uma instituição caso ainda existam serviços associados a ela
-instituicao(NomeInst, Cidade) :: (
    solucoes(IdServ, servico(IdServ, _, NomeInst, _), Lista),
    comprimento(Lista, N),
    N == 0).

```

5.2.2 Serviço

Para garantir que a informação de cada consulta se mantém íntegra, implementou-se um invariante que impede que se remova um serviço que esteja associado a pelo menos uma consulta.

```
% Invariante Referencial  
% Não deixa remover um serviço caso este ainda esteja associado a alguma consulta  
-servico(IdServ, Descricao, Instituicao, Cidade) :: (  
    solucoes(Ano, consulta(Ano, _, _, _, IdServ, _, _), Lista),  
    comprimento(Lista, N),  
    N == 0).
```

5.2.3 Médico

Para garantir que a informação de cada consulta se mantém íntegra, implementou-se um invariante que impede que se remova um médico que tenha pelo menos uma consulta associada.

```
% Invariante Referencial  
% Não deixa remover um médico se este ainda estiver associado a alguma consulta  
-medico(IdMed, Nome, IdServ) :: (  
    solucoes(Ano, consulta(Ano, _, _, _, _, IdMed), Lista),  
    comprimento(Lista, N),  
    N == 0).
```

5.2.4 Utente

Para garantir que a informação de cada consulta se mantém íntegra, implementou-se um invariante que impede que se remova um utente que tenha pelo menos uma consulta associada.

```
% Invariante Referencial  
% Não deixa remover um utente se este ainda estiver associado a alguma consulta  
-utente(IdUt, _, _, _) :: (  
    solucoes(Ano, consulta(Ano, _, _, _, IdUt, _, _), Lista),  
    comprimento(Lista, N),  
    N == 0).
```

Capítulo 6

Evolução do Conhecimento

A problemática da evolução do conhecimento relaciona-se com o processo de adicionar ou remover informação da base de conhecimento.

Este processo deve garantir que a base de conhecimento mantém a integridade e a consistência de dados.

Para tal, deve-se assegurar que:

- não se elimina informação dependente de outra
- não se adiciona informação repetida
- não se adiciona informação contraditória

Assim, quando se faz alguma alteração na base de conhecimento é necessário analisar e testar se a modificação corrompe de alguma forma a informação existente.

Os invariantes definidos garantem que, caso não se cumpram as restrições necessárias, a informação não é adicionada ou removida da base de conhecimento.

No entanto, em programação de lógica estendida as decisões não se prendem apenas em inserir/remover a informação.

É necessário analisar todas as possíveis situações que possam ocorrer e decidir como proceder em cada um dos cenários.

6.1 Predicado evolução

Na evolução do conhecimento, ou seja, na inserção de conhecimento analisaram-se as várias situações de adicionar cada tipo de conhecimento.

6.1.1 Evoluir com conhecimento perfeito positivo

Adicionar conhecimento positivo à base de conhecimento consiste em inserir factos com informação positiva.

Quando se insere este tipo de conhecimento é fundamental verificar se já existe alguma informação relacionada na base de conhecimento.

Se não existir, então, a nova informação deve ser adicionada permitindo alimentar a base de conhecimento com novos conhecimentos.

Caso exista, é necessário avaliar de que forma o novo conhecimento pode interferir e definir como proceder em cada situação.

Na Tabela 6.1 encontram-se os possíveis casos que podem ocorrer e como atuar em cada um deles.

Tabela 6.1: Inserção de conhecimento positivo

inserir?	Positivo	Negativo	Tipo I	Tipo II	Tipo III
P (V)	depende Inv				
P (V)	inserir	remover			
P (V)	inserir		remover		
P(V)	inserir			remover	
P (V)	não inserir				manter

Por isso, pela Tabela 6.1, ficou definido que se existir, na BC, conhecimento do tipo:

- **Positivo:** Mantém-se a informação já existente. O novo conhecimento é considerado conhecimento repetido e, por isso, deve ser descartado. Garantido pelos invariantes que não permitem inserir conhecimento
- **Negativo:** Não podem ser mantidos os dois, pois resultaria em contradição. Remover o conhecimento negativo e atualizar a BC com o novo conhecimento verdadeiro (positivo). Optou-se por considerar o novo conhecimento como o mais recente e, portanto, deve ser inserido
- **Imperfeito do tipo I:** Remover da BC o conhecimento imperfeito e adicionar o novo conhecimento positivo.
- **Imperfeito do tipo II:** Remover da BC o conhecimento imperfeito e adicionar o novo conhecimento positivo.
- **Imperfeito do tipo III:** Descartar o novo conhecimento e manter o conhecimento do tipo III (uma vez que este nunca se poderá vir a saber, tal é garantido pelo invariante associado a este tipo de valor nulo).

Implementação em Prolog

Como demonstrado na Tabela 6.1, para existir evolução de conhecimento positivo é preciso tratar o conhecimento existente. A evolução de conhecimento positivo é feita à custa do termo `evolucaoPositivo`.

```

evolucaoPositivo(X) :- removeSeTemConhecimentoNegativo(X),
                        evolucao(X).
evolucaoPositivo(X) :- removeSeTemConhecimentoImperfeitoIncerto(X),
                        evolucao(X).
evolucaoPositivo(X) :- removeSeTemConhecimentoImperfeitoImpreciso(X),
                        evolucao(X).

```

```
evolucaoPositivo(X) :- evolucao(X).
```

Este predicado é composto por quatro definições que tratam o conhecimento existente. Caso nenhum conhecimento exista, o último termo tenta evoluir o conhecimento positivo através do predicado `evolucao`.

Para cada um dos conhecimentos que possam existir, foi criada uma condição no predicado `evolucaoPositivo` que, utilizando um dos predicados indicados de seguida, garante que caso o conhecimento exista, será eliminado. A definição de cada um dos predicados tem de ser específica para os termos a evoluir. Para efeitos de demonstração são apresentadas as definições dos predicados para o termo `consulta`.

- `removeSeTemConhecimentoNegativo`

O predicado `removeSeTemConhecimentoNegativo` recolhe todos os termos negativos. Devido ao predicado que existe para garantir o conhecimento perfeito negativo, é necessário garantir que o conhecimento existente tem duas soluções (conhecimento perfeito negativo, conhecimento negativo explícito). Se existirem, o conhecimento será removido utilizando o predicado `remove`.

```
removeSeTemConhecimentoNegativo(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed)) :-  
    solucoes(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed),  
        -consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed),  
        [NegativoGeral, NegativoExplicito]),  
    remove(-NegativoExpliciUto).
```

- `removeSeTemConhecimentoImperfeitoIncerto`

Para o conhecimento imperfeito incerto, é necessário ter definições do predicado `removeSeTemConhecimentoImperfeitoIncerto` por cada parâmetro desconhecido. Em cada definição, para o parâmetro desconhecido, são recolhidas as soluções existentes desses predicados. No caso de existir será apenas uma solução, já que não existe conhecimento imperfeito incerto repetido e portanto, são removidos os termos associados ao conhecimento imperfeito recorrendo ao predicado `removeTermos`.

```
removeSeTemConhecimentoImperfeitoIncerto(  
    consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, _)) :-  
        solucoes(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed),  
            (consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed),  
            excecacao(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed)),  
            nao(nulo(Custo))),  
            [Desconhecido]),  
    removeTermos([  
        Desconhecido,  
        (excecacao(Desconhecido):-Desconhecido)]).  
  
removeSeTemConhecimentoImperfeitoIncerto(  
    consulta(Ano, Mes, Dia, _, IdUt, IdServ, Custo, IdMed)) :-  
        solucoes(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed),  
            (consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed),  
            excecacao(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed)),  
            nao(nulo(Custo))),  
            [Desconhecido]),  
    removeTermos([
```

```
Desconhecido,
(excecao(Desconhecido):-Desconhecido)]).
```

- `removeSeTemConhecimentoImperfeitoImpreciso`

O predicado `removeSeTemConhecimentoImperfeitoImpreciso` segue a mesma lógica dos termos anteriores. É procurada a existência de conhecimento imperfeito impreciso e caso este exista é removido com o predicado `removeTermos`.

```
removeSeTemConhecimentoImperfeitoImpreciso(
    consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, _, IdMed)) :-
    solucoes(
        excecao(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed)),
        (excecao(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed)),
        nao(nulo(Custo))),
        S),
    comprimento(S, N),
    N>1,
    removeTermos(S).
```

Assim, para facilitar a adição de uma `consulta` foi implementado o predicado `registraConsultaP`.

```
registraConsultaP(Ano, Mes, Dia, Hora, IdUt, IServ, Custo, IdMed) :-
    evolucao(consulta(Ano, Mes, Dia, Hora, IdUt, IServ, Custo, IdMed)).
```

6.1.2 Evoluir com conhecimento perfeito negativo

Adicionar conhecimento negativo à base de conhecimento consiste em inserir factos com informação que se sabe ser falsa.

Da mesma forma que anteriormente, é necessário avaliar de que forma o novo conhecimento pode interferir e definir como proceder em cada situação.

Na Tabela 6.2 encontram-se as possibilidades que podem ocorrer e como atuar em cada um delas.

Tabela 6.2: Inserção de conhecimento negativo

inserir?	Positivo	Negativo	Tipo I	Tipo II	Tipo III
N (F)	remover	inserir			
N (F)		depende Inv			
N (F)		inserir	remover		
N (F)		inserir		remover	
N (F)					manter

Por isso, pela Tabela 6.2, ficou definido que se existir na BC conhecimento do tipo:

- **Positivo:** Não podem ser mantidos os dois pois resultaria em contradição. Remover o conhecimento positivo da BC e atualizar a BC com o novo conhecimento negativo (explicitamente falso). Optou-se por considerar o novo conhecimento como o mais recente e, portanto, deve ser inserido.

- **Negativo:** Garantindo pelos invariantes.
- **Imperfeito do tipo I:** Remover da BC o conhecimento imperfeito e adicionar o novo conhecimento negativo.
- **Imperfeito do tipo II:** Remover da BC o conhecimento imperfeito e adicionar o novo conhecimento negativo.
- **Imperfeito do tipo III:** Descartar o novo conhecimento e manter o conhecimento do tipo III (uma vez que este nunca se poderá vir a saber, tal é garantido pelo invariante associado a este tipo de valor nulo).

Implementação em Prolog

Analogamente à evolução de conhecimento positivo, e como demonstrado na Tabela 6.2 da evolução de conhecimento negativo, para haver evolução é necessário tratar o conhecimento existente. O predicado `evolucaoNegativo` é responsável por efectuar a evolução.

```
evolucaoNegativo(-X) :- removeSeTemConhecimentoPositivo(X),
                        evolucao(-X).
evolucaoNegativo(-X) :- removeSeTemConhecimentoImperfeitoIncerto(X),
                        evolucao(-X).
evolucaoNegativo(-X) :- removeSeTemConhecimentoImperfeitoImpreciso(X),
                        evolucao(-X).
evolucaoNegativo(-X) :- evolucao(-X).
```

Caso nenhum conhecimento exista, o último termo tenta evoluir o conhecimento negativo através do já existente, predicado `evolucao`.

O tratamento do conhecimento existente é feito nas três primeiras definições. O predicado `evolucaoNegativo` auxilia-se de 3 diferentes predicados para garantir o tratamento dos diferentes tipos de conhecimento possivelmente existentes. Uma vez mais, cada um destes predicados tem que ser específico de cada termo que se quer adicionar conhecimento. Para demonstração, uma vez mais, utilizou-se o termo `consulta`.

- `removeSeTemConhecimentoPositivo`

O predicado `removeSeTemConhecimentoPositivo` primeiro recolhe todos os termos positivos que não pertençam a conhecimento imperfeito e caso existam alguns resultados, remove os termos através do predicado `removeTermos`.

```
removeSeTemConhecimentoPositivo(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed)) :-
    solucoes(Ano, (consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed),
                     nao(excecao(consulta(Ano, Mes, Dia, Hora, IdUt, IdServ, Custo, IdMed))),
                     nao(nulo(Custo))),
             S),
    comprimento(S, N),
    N>0,
    removeTermos(S).
```

- `removeSeTemConhecimentoImperfeitoIncerto` (já mostrado anteriormente)
- `removeSeTemConhecimentoImperfeitoImpreciso` (já mostrado anteriormente)

6.1.3 Evoluir com conhecimento imperfeito do tipo I

Adicionar conhecimento imperfeito do tipo I à base de conhecimento consiste em inserir conhecimento incerto.

Tal como anteriormente, é necessário verificar como o novo conhecimento pode interferir e definir como proceder em cada situação.

Na Tabela 6.3 definiram-se as possibilidades que podem ocorrer e como atuar em cada um delas.

Tabela 6.3: Inserção conhecimento do tipo I

inserir?	Positivo	Negativo	Tipo I	Tipo II	Tipo III
Tipo I (D)	manter		não inserir		
Tipo I (D)		manter	não inserir		
Tipo I (D)			inserir		
Tipo I (D)			não inserir	manter	
Tipo I (D)			não inserir		manter

Assim, ficou definido que se existir na BC conhecimento do tipo:

- **Positivo:** Manter o conhecimento positivo e não inserir o conhecimento imperfeito do tipo I.
- **Negativo:** Manter o conhecimento positivo e não inserir o conhecimento imperfeito do tipo I.
- **Imperfeito do tipo I:** O novo conhecimento do tipo I deve ser inserido e deve ser analisada a informação incerta anterior, ou seja, analisar se existe alguma contrariedade em manter os dois.
- **Imperfeito do tipo II:** O novo conhecimento imperfeito do tipo I (incerto) deve ser descartado e deve ser mantido o conhecimento já existente do tipo II (impreciso).
- **Imperfeito do tipo III:** O novo conhecimento imperfeito do tipo I (incerto) deve ser descartado e deve ser mantido o conhecimento já existente do tipo III (Interdito). Pois, o conhecimento do tipo III não deve ser atualizado já que nunca se poderá vir a saber.

6.1.4 Evoluir com conhecimento imperfeito do tipo II

Adicionar conhecimento imperfeito do tipo II à base de conhecimento consiste em inserir conhecimento impreciso, ou seja, que se conhece com alguma imprecisão.

Como nas situações anteriores, é necessário analisar de que forma o novo conhecimento pode interferir e definir como proceder em cada caso. Na Tabela 6.4 definiram-se as possibilidades que podem ocorrer e como atuar em cada um delas.

Tabela 6.4: Inserção conhecimento tipo II

inserir?	Positivo	Negativo	Tipo I	Tipo II	Tipo III
Tipo II (D)	manter			não inserir	
Tipo II (D)		manter		não inserir	
Tipo II (D)			não inserir	manter	
Tipo II (D)				atualizar	
Tipo II (D)					manter

Por isso, pela Tabela 6.4, ficou definido que se existir na BC conhecimento do tipo:

- **Positivo:** Manter o conhecimento positivo e não inserir o conhecimento imperfeito do tipo II.
- **Negativo:** Manter o conhecimento positivo e não inserir o conhecimento imperfeito do tipo II.
- **Imperfeito do tipo I:** O novo conhecimento imperfeito do tipo II (impreciso) deve ser descartado e deve ser mantido o conhecimento já existente do tipo I (incerto).
- **Imperfeito do tipo II:** O novo conhecimento do tipo II deve ser inserido e deve ser analisada a informação incerta anterior, ou seja, analisar se existe alguma contrariedade em manter os dois.
- **Imperfeito do tipo III:** O novo conhecimento imperfeito do tipo II (impreciso) deve ser descartado e deve ser mantido o conhecimento já existente do tipo III (Interdito). O conhecimento do tipo III não deve ser atualizado já que nunca se poderá vir a saber o seu valor.

6.1.5 Evoluir com conhecimento imperfeito do tipo III

Adicionar conhecimento imperfeito do tipo III à base de conhecimento consiste em inserir conhecimento interdito, ou seja, que não se sabe nem se poderá vir a saber.

Tal como anteriormente, é crucial analisar de que forma o novo conhecimento pode interferir e definir como proceder em cada situação. Na Tabela 6.5 definiram-se os possíveis casos que podem ocorrer e como atuar em cada um deles.

Tabela 6.5: Inserção de conhecimento tipo III

inserir?	Positivo	Negativo	Tipo I	Tipo II	Tipo III
Tipo III (D)	manter				não inserir
Tipo III (D)		manter			não inserir
Tipo III (D)			manter		não inserir
Tipo III (D)				manter	não inserir
Tipo III (D)					atualizar

Por isso, pela Tabela 6.5, ficou definido que se existir na BC conhecimento do tipo:

- **Positivo:** Deve ser mantido o conhecimento perfeito positivo. Descartar o conhecimento imperfeito interdito.

- **Negativo:** Deve ser mantido o conhecimento perfeito negativo. Descartar o conhecimento imperfeito interdito.
- **Imperfeito do tipo I:** Deve ser mantido o conhecimento imperfeito incerto. Descartar o conhecimento imperfeito interdito.
- **Imperfeito do tipo II:** Deve ser mantido o conhecimento imperfeito impreciso. Descartar o conhecimento imperfeito interdito.
- **Imperfeito do tipo III:** Deve ser mantido o conhecimento imperfeito interdito já existe. Descartar o novo conhecimento. Não devem ser realizadas alterações ao conhecimento do tipo III já existente.

6.2 Predicado regressão

Na regressão do conhecimento, ou seja, na remoção de conhecimento analisaram-se as várias situações de remover cada tipo de conhecimento.

Na Tabela 6.6 encontram-se os possíveis casos que podem ocorrer e como atuar em cada um deles.

Tabela 6.6: Remoção de conhecimento

remover?	Positivo	Negativo	Tipo I	Tipo II	Tipo III
	inv	inv	remover	remover	não remover

Por isso, pela Tabela 6.6, ficou definido que quando se pretende remover conhecimento na BC do tipo:

- **Positivo:** A remoção da informação positiva depende dos invariantes definidos para a remoção associada ao predicado.
- **Negativo:** A remoção da informação negativa depende dos invariantes definidos para a remoção associada ao predicado.
- **Imperfeito do tipo I:** A informação desconhecida do tipo I pode ser eliminada.
- **Imperfeito do tipo II:** A informação desconhecida do tipo II pode ser eliminada.
- **Imperfeito do tipo III:** A informação desconhecida do tipo III não pode ser eliminada. Esta informação nunca poderá ser alterada.

Portanto, o predicado regressão tem que garantir que os invariantes de remoção avaliam a possibilidade de remover ou não o conhecimento perfeito (positivo e negativo). Além disso, tem que permitir eliminar o conhecimento imperfeito tipo I e II e não consentir a remoção de qualquer conhecimento imperfeito do tipo III.

Implementação em prolog

A implementação em prolog do predicado **regressao** foi feita de acordo com as condições explicadas na Tabela 6.6. Assim, primeiramente são recolhidos todos os invariantes antes da remoção, remove-se o termo e de seguida garante-se que os invariantes de remoção se continuam a verificar. Caso esta verificação falhe, os termos serão novamente inseridos através do predicado **remocao**.

```
regressao( Termo ) :-  
    solucoes( Invariante, -Termo::Invariante, Lista ),  
    remocao( Termo ),  
    teste( Lista ).
```

Assim, para remover informação positiva é necessário que os invariantes de remoção continuem a ser verificados (através do predicado teste), caso contrário, a informação não é removida.

Como não foram definidos invariantes de remoção para remover conhecimento imperfeito do tipo I e do tipo II, estes poderão ser removidos sem problemas.

No caso do conhecimento imperfeito do tipo III é necessário definir um invariante de remoção que impeça eliminar este tipo de conhecimento.

Capítulo 7

Sistema de Inferência

Em programação lógica estendida, com a introdução de conhecimento imperfeito, é necessário definir um predicado que possibilite inferir sobre os três tipos de valores nulos. Este predicado concretiza as situações em que ocorre informação incompleta, avaliando as questões a analisar e inferindo sobre o resultado das mesmas. O resultado deste predicado poderá ser um de três valores:

- **Verdadeiro:** Quando é possível provar a questão a partir da informação perfeita positiva que existe na base de conhecimento.
- **Falso:** Quando é possível provar a questão a partir de informação perfeita negativa, ou seja, informação explicitamente falsa que exista na base de conhecimento.
- **Desconhecido:** Quando não existe informação suficiente para inferir sobre a questão.

Designou-se este predicado por **demo** (demonstração).

demo: Questão, Resposta $\rightarrow \{V, F, D\}$

Definiu-se na Tabela 7.1 a tabela de verdade para a conjunção.

Tabela 7.1: Tabela de verdade da Conjunção

Q1 and Q2	V	F	D
V	V	F	D
F	F	F	F
D	D	F	D

Definiu-se na Tabela 7.2 a tabela de verdade para a disjunção.

Tabela 7.2: Tabela de verdade da Disjunção

Q1 or Q2	V	F	D
V	V	V	V
F	V	F	D
D	V	D	D

7.1 Implementação em Prolog

O predicado `demo` foi estendido para permitir a construção de conjunções e disjunções.

As conjunções e disjunções são representadas à custa de dois operadores: `and` e `or`. Para testar estes predicados, é utilizado o predicado `demo` que tentará instanciar a resposta para as duas questões. Cada uma destas questões pode ser composta por múltiplas conjunções e/ou disjunções. No final, para o `demo` da conjunção é feita a conjunção através do predicado `conjuncao` e para o `demo` da disjunção, a disjunção é feita com o predicado `dsijuncao`.

```
demo(Q1 and Q2, R) :-    demo(Q1, R1),
                          demo(Q2, R2),
                          conjuncao(R1, R2, R).

demo(Q1 or Q2, R) :-     demo(Q1, R1),
                          demo(Q2, R2),
                          disjuncao(R1, R2, R).

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

iguais(A, A).

conjuncao(verdadeiro, verdadeiro, verdadeiro).
conjuncao(falso, _, falso).
conjuncao(_, falso, falso).
conjuncao(_, _, desconhecido).

disjuncao(verdadeiro, _, verdadeiro).
disjuncao(falso, falso, falso).
disjuncao(_, verdadeiro, verdadeiro).
disjuncao(desconhecido, desconhecido, desconhecido).

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

Para facilitar o teste de várias questões em simultâneo, foi também desenvolvido um predicado (`demoListas`) que permite testar uma lista de questões. Este predicado devolve uma lista de respostas, uma por cada questão. Utiliza o predicado `demo` para avaliar cada uma das questões.

```
demoListas([], []).
demoListas([Q|TQ], [R|TR]) :- demo(Q, R), demoListas(TQ, TR).
```

Capítulo 8

Predicados Extra

Decidiu-se definir alguns predicados adicionais de modo a extrair alguma informação sobre a base de conhecimento.

8.1 Determinar a média de idade dos utentes

Considerou-se interessante implementar um predicado que permita determinar a média de idades dos utentes que frequentam a clínica SaúdeMais. Este conhecimento possibilita perceber a faixa etária que recorre mais frequentemente à clínica. Contudo visto que agora se consegue representar conhecimento perfeito negativo e conhecimento imperfeito, teve-se que filtrar a informação dos utentes de modo a obter somente o conhecimento perfeito positivo, através do predicado soluções. De seguida, calculou-se o somatório e o comprimento da lista, o que possibilitou obter a média de idades.

```
% Extensão do predicado que permite determinar a média de idade dos utentes com conhecimento  
% perfeito positivo  
% media_idades_utentes : Media -> {V,F,D}  
media_idades_utentes(Media) :-  
    solucoes(  
        Idade,  
        (utente(IdUt, Nome, Idade, Morada),  
         nao(excecao(utente(IdUt, Nome, Idade, Morada))), nao(nulo(Idade))),  
        ListaIdades),  
    somatorio(ListaIdades, S),  
    comprimento(ListaIdades, C),  
    Media is S/C.
```

8.2 Número de consultas de um serviço num determinado ano

Considerou-se interessante implementar um predicado que calcula o número de consultas de um dado serviço da rede de clínicas SaúdeMais, visto que assim a gestão das clínicas poderá alocar melhor os seus recursos para os serviços mais requeridos. Este predicado usa o soluções para calcular a lista das consultas de um serviço num determinado ano, filtrando pelas consultas que correspondem ao

conhecimento perfeito positivo . Após ter esta lista, é calculado o comprimento para saber o número de consultas.

```
% Extensão do predicado que permite determinar número de consultas de um serviço
% num determinado ano, isto para predicados consulta com conhecimento perfeito positivo
% numero_consultas_servico: Ano, IdServico, NumeroConsultas -> {V,F,D}
numero_consultas_servico(Ano, IdServico, NumeroConsultas) :-
    solucoes( Ano,
        (consulta(Ano, Mes, Dia, Hora, IdUt, IdServico, Custo, IdMed),
            nao(execacao(consulta(Ano, Mes, Dia, Hora, IdUt, IdServico, Custo, IdMed))),
            nao(nulo(Ano))), Lista),
        comprimento(Lista, NumeroConsultas).
```

8.3 Serviço mais consultado de uma instituição num determinado ano

Considerou-se interessante implementar este predicado, uma vez que permite que a equipa que faz a gestão das clínicas realoque melhor os seus recursos, fornecendo assim um melhor serviço aos seus utentes. O predicado `servico_mais_consultado` determina o serviço mais consultado de uma instituição num determinado ano. Para tal, usa o predicado `numero_consultas_servico` definido anteriormente.

```
% Extensão do predicado que permite determinar o serviço mais consultado de uma instituição
% num determinado ano, mas somente para predicados serviço com conhecimento perfeito positivo
% servico_mais_consultado : Ano, NomeInst, Descricao -> {V,F, D}
servico_mais_consultado(Ano, NomeInst, Descricao) :-
    solucoesSemRepetidos((Desc, NumeroConsultas),
        (servico(IdServico, Desc, NomeInst, Cidade),
            nao(execacao(servico(IdServico, Desc, NomeInst, Cidade))),
            nao(nulo(Desc)),
            numero_consultas_servico(Ano, IdServico, NumeroConsultas)),
        Lista),
    maior_par(Lista, (Descricao, _)).
```

Capítulo 9

Testes e Resultados

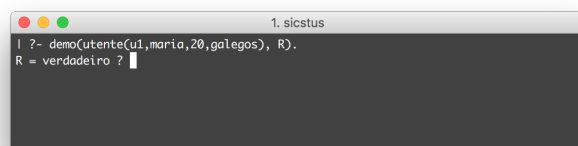
Neste capítulo demonstram-se alguns testes realizados ao sistema implementado e apresentam-se os resultados obtidos.

9.1 Consultas à base de conhecimento

Realizaram-se consultas à base de conhecimento para testar os vários tipos de conhecimentos existentes. Utilizaram-se conhecimentos que se apresentaram no capítulo 3 e 4 (Representação de conhecimento).

- **Consulta de conhecimento perfeito positivo**

Interrogou-se a base de conhecimento sobre a existência da utente maria com $id = 1$. Tal como esperado, uma vez que esta utente existe na base de conhecimento, o resultado foi **verdadeiro**.



```
1. sicstus
| ?- demo(utente(u1,maria,20,galegos), R).
R = verdadeiro ?
```

- **Consulta de conhecimento perfeito negativo**

Na base de conhecimento existe explicitamente informação negativa relativa ao médico $id=m1$, Dr. barros. Interrogou-se a base de conhecimento acerca desse conhecimento, do médico $m1$. Como a informação presente na base de conhecimento é negativa, a resposta obtida foi **falso**.

```
1. sicstus
| ?- demo(medico(m1,barros,s10), R).
R = falso ?
```

- **Consulta de conhecimento imperfeito do tipo I : incerto**

Na base de conhecimento explicitou-se que não se sabe onde mora o utente jose, u16. Não se conhece a morada deste utente, no entanto, sabe-se que o jose existe e existe informação sobre a sua idade. Portanto, questionando através do predicado **demo** se o jose reside em angola ou em faro a resposta foi **desconhecido**. Ao questionar sobre a morada do jose a resposta será sempre **desconhecido**, qualquer que seja o local inserido.

```
1. with-readline
| ?- demo(utente(u16,jose,50,angola), R).
R = desconhecido ?
yes
| ?- demo(utente(u16,jose,50,faro), R).
R = desconhecido ?
```

- **Consulta de conhecimento imperfeito do tipo II : impreciso**

Na base de conhecimento explicitou-se que não se sabe ao certo onde se encontra localizada a clínica **csmgualtar**, sabe-se apenas que ou é em vizela ou em celorico. Ou seja, esta informação é conhecida com alguma imprecisão.

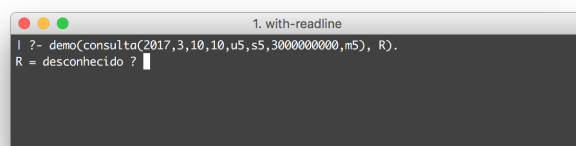
Através do predicado **demo** interrogou-se a base de conhecimento sobre a possibilidade da clínica **csmgualtar** se localizar em vizela e o resultado foi **desconhecido** pois, não se tem a certeza se realmente é em vizela. Interrogando sobre a hipótese de se localizar no porto a resposta foi **falso** dado que, apesar de não saber ao certo onde fica, sabe-se que não é no porto. A clínica **csmgualtar** ou fica em vizela ou em celorico, não se sabe em qual dos dois mas, sabe-se que não se localiza em nenhum outro local.

```
1. with-readline
| ?- demo(instituicao(csmgualtar,vizela), R).
R = desconhecido ?
yes
| ?- demo(instituicao(csmgualtar,porto), R).
R = falso ?
```

- **Consulta de conhecimento imperfeito do tipo III: interdito**

Na base de conhecimento está registado que a consulta do utente id=u5 no serviço id=s5 na data=2017/3/10 pelas 10h dada pelo médico com o id=m5, nunca poderá ter o seu preço

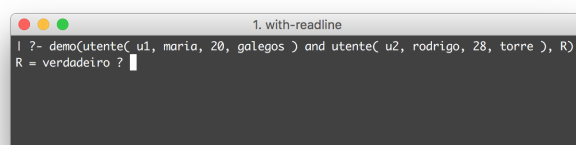
divulgado. Com o predicado **demo** conseguiu-se confirmar que apesar de o conhecimento estar registado, a sua veracidade é desconhecida, tendo a resposta obtida sido: **desconhecido**.



```
1. with-readline
| ?- demo(consulta(2017,3,10,10,u5,s5,3000000000,m5), R).
R = desconhecido ?
```

- **Consulta com a utilização do operador and de disjunção**

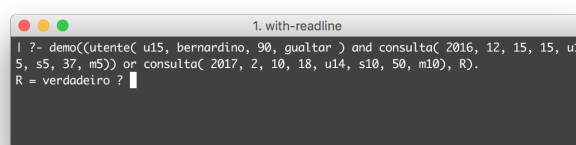
Sabendo de dois utentes que existem na base de conhecimento tentou-se perceber se as duas questões seriam válidas. Tal como esperado, utilizando o predicado **demo**, foi possível confirmar exatamente isso (**verdadeiro**).



```
1. with-readline
| ?- demo(utente(u1, maria, 20, galegos ) and utente(u2, rodrigo, 28, torre ), R).
R = verdadeiro ?
```

- **Consulta com a utilização do operador and de disjunção e or de conjunção**

Tendo um utente e uma consulta sua que existem na base de conhecimento e que se sabem que são verdadeiros, e tendo uma questão que é desconhecida, compôs-se uma disjunção para garantir que caso uma fosse verdade a resposta seria verdade. A resposta obtida pelo predicado **demo** foi a esperada (**verdadeiro**).



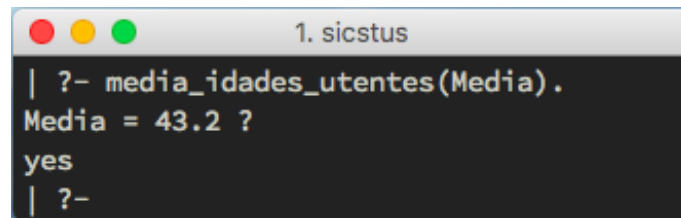
```
1. with-readline
| ?- demo((utente(u15, bernardino, 90, gualtar ) and consulta( 2016, 12, 15, 15, u15, s5, 37, m5)) or consulta( 2017, 2, 10, 18, u14, s10, 50, m10), R).
R = verdadeiro ?
```

9.2 Predicados extra (funcionalidades)

Realizaram-se testes aos predicados extra adicionados ao sistema.

9.2.1 Determinar a média de idade dos utentes

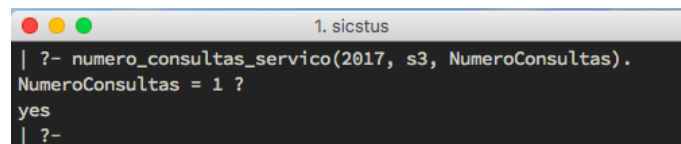
Testou-se o predicado que permite obter a média de idades dos utentes, e obteve-se o seguinte resultado:



```
| ?- media_idades_utentes(Media).  
Media = 43.2 ?  
yes  
| ?-
```

9.2.2 Número de consultas de um serviço num determinado ano

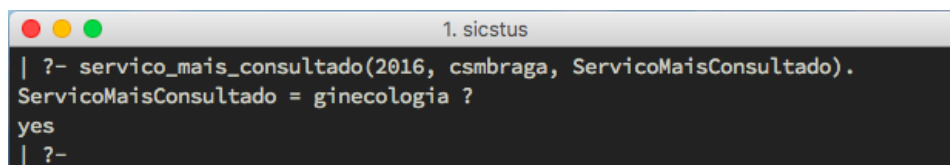
Testou-se o predicado que permite calcular o número de consultas de um serviço num determinado ano, e obteve-se o seguinte resultado:



```
| ?- numero_consultas_servico(2017, s3, NumeroConsultas).  
NumeroConsultas = 1 ?  
yes  
| ?-
```

9.2.3 Serviço mais consultado de uma instituição num determinado ano

Testou-se o predicado que permite determinar o serviço mais consultado de uma instituição num determinado ano, e obteve-se o seguinte resultado:



```
| ?- servico_mais_consultado(2016, csmbraga, ServicoMaisConsultado).  
ServicoMaisConsultado = ginecologia ?  
yes  
| ?-
```

9.3 Evolução da base de conhecimento

Para demonstrar a evolução da base de conhecimento apresentam-se alguns exemplos para o caso de se inserir conhecimento perfeito positivo. Sendo que ao tentar adicionar, na base de conhecimento já se encontrava:

- **Conhecimento inexistente**

Recorrendo ao predicado `demo`, demonstrou-se que o utente com `id = u100`, o pedro, não existia. Após isso, o utente foi registado com auxílio do predicado `registarUtenteP`. Correndo novamente o predicado `demo`, foi possível ver que a informação foi adicionada.


```
1. with-readline
l ?- demo(utente(u100, pedro, 23, lisboa), R).
R = falso ?
yes
l ?- registaUtenteP(u100, pedro, 23, lisboa).
yes
l ?- demo(utente(u100, pedro, 23, lisboa), R).
R = verdadeiro ?
```

- **Conhecimento perfeito positivo**

Recorrendo ao predicado **demo**, começou-se por confirmar que existe na base de conhecimento o utente com id = u100, pedro. Após isso, tentou-se registar o mesmo utente, o que corresponde a informação duplicada. Foi possível verificar que a informação não foi inserida, tal como pretendido. Mantendo-se a informação existente anteriormente.

```
1. sicstus
l ?- demo(utente(u100,pedro,23,lisboa), R).
R = verdadeiro ?
yes
l ?- demo(utente(u100,pedro,23,almada), R).
R = falso ?
yes
l ?- demo(utente(u100,pedro,23,lisboa), R).
R = verdadeiro ?
```

- **Conhecimento perfeito negativo**

Começou-se por confirmar que existia informação negativa sobre o utente pedro (id = u100). Registou-se o utente pedro com informação positiva e essa informação foi adicionada. A informação mais antiga foi removida, ficando a base de conhecimento com a informação mais recente.

```
1. with-readline
l ?- demo(utente(u100, pedro, 23, lisboa), R).
R = falso ?
yes
l ?- registaUtenteP(u100, pedro, 23, lisboa).
yes
l ?- demo(utente(u100, pedro, 23, lisboa), R).
R = verdadeiro ?
```

- **Conhecimento imperfeito do tipo I - incerto**

Na base de conhecimento existe informação imperfeita incerta sobre o utente id=u16 (jose), tal como demonstrado anteriormente. Registou-se a nova informação positiva do utente u16 e confirmou-se que o novo conhecimento positivo foi inserido e o conhecimento imperfeito foi descartado.

```
1. sicstus
I ?- demo(utente( u16, jose, 50, angola), R).
R = desconhecido ?
yes
I ?- registaUtenteP(u16, jose, 50, angola).
yes
I ?- demo(utente( u16, jose, 50, angola), R).
R = verdadeiro ?
```

- **Conhecimento imperfeito do tipo II - impreciso**

O utente id=u18 (xavier), na base de conhecimento, tem informação imperfeita imprecisa, tal como demonstrado anteriormente. A nova informação positiva do utente u18 foi registada e confirmou-se que conhecimento imperfeito foi descartado, tendo sido inserido o novo conhecimento positivo.

```
1. sicstus
I ?- demo(utente( u18, xavier, 24, gondizalves), R).
R = desconhecido ?
yes
I ?- registaUtenteP(u18, xavier, 24, gondizalves).
yes
I ?- demo(utente( u18, xavier, 24, gondizalves), R).
R = verdadeiro ?
```

- **Conhecimento imperfeito do tipo III - interdito**

A informação que existe na base de conhecimento sobre o utente id=u21 (marta), é imperfeito interdito, como dito anteriormente. Tentou-se registar a nova informação positiva do utente u21, contudo, devido às restrições da evolução de conhecimento, confirmou-se que tal não foi possível, mantendo-se a informação imperfeita já existente.

```
1. sicstus
I ?- demo(utente(u21, marta, 35, chaves), R).
R = desconhecido ?
yes
I ?- registaUtenteP(u21, marta, 35, chaves).
no
I ?- demo(utente(u21, marta, 35, chaves), R).
R = desconhecido ?
```

Capítulo 10

Conclusão

Este trabalho visou permitir a utilização da extensão à programação em lógica, com o recurso à linguagem de programação em lógica PROLOG, a fim de permitir a representação de conhecimento imperfeito (incerto, impreciso e interdito), através da utilização de valores nulos e da criação de mecanismos de raciocínio adequados.

Dado isto, adicionou-se funcionalidades e conhecimento ao sistema, incrementando e tornando o caso de estudo do Exercício 1 mais complexo, contudo mais realista, uma vez que suporta agora todos os tipos de conhecimento do universo do discurso.

Para além da representação do conhecimento perfeito positivo, teve-se que adicionar à base do conhecimento exemplos de conhecimento perfeito negativo e exemplos de conhecimento imperfeito - incerto, impreciso e interdito - tornando a base de conhecimento mais rica e complexa.

Uma vez que a base de conhecimento, neste trabalho, contém cinco tipos de conhecimento distinto, houve a necessidade de criar novos invariantes de modo a validar as situações de evolução e regressão dos diversos tipos de conhecimento. Consequentemente, já que o sistema não representa somente aquilo que é verdadeiro, foi necessário criar procedimentos adequados de modo a permitir a evolução e regressão de conhecimento na base de conhecimento. Principalmente a evolução, sofreu de uma profunda análise para se compreender como deve ser evoluída a informação no caso de o conhecimento a ser inserido colidir de alguma forma com o conhecimento já existente na base de conhecimento.

Por fim, desenvolve-se um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas, capaz de determinar se uma questão ou um conjunto de questões é verdadeira, falsa ou desconhecida.

Adicionalmente, implementaram-se três predicados de modo a extrair informação da base de conhecimento, e de forma a demonstrar como se pode extrair informação de sistemas que permitem a representação de informação negativa e desconhecida.

Deste modo, conclui-se que se conseguiu atingir os objetivos propostos com a realização deste trabalho, suportado por um relatório bem estruturado e conciso.

Bibliografia

- [1] Analide, C. and Neves, J. 1996. *Representação de Informação Incompleta* Universidade do Minho
- [2] Nilsson, U. A. and Maluszynski, J. 2000. *Logic, Programming and Prolog*
- [3] Bratko, I. 1986. *Prolog Programming for Artificial Intelligence* British Library Cataloguing in Publication Data
- [4] Sterling, L. - Shapiro, E. 1994. *The Art of Prolog* MIT Press Cambridge, MA, USA
- [5] Frade, M. J. 2006. *Lógica Computacional: PROLOG* Universidade do Minho