Universidade do Minho
Mestrado Integrado em Engenharia Informática
Departamento de Informática

Sistemas de Representação de Conhecimento e Raciocínio

Exercício 2 - Programação em Lógica Estendida e Conhecimento Imperfeito

---

Isabel Sofia da Costa Pereira A76550
José Francisco Gonçalves Petejo e Igreja Matos A77688
Maria de La Salete Dias Teixeira A75281
Tiago Daniel Amorim Alves A78218

20 de Abril de 2018

Conteúdo

1 Resumo

Ao longo deste exercício, foi elaborado um sistema de prestação de cuidados de sáude, onde representamos utentes, prestadores e cuidados. O centro deste trabalho foi a expressão de conhecimento positivo, negativo e imperfeito e a exploração destes.

2 Introdução

Este trabalho baseia-se na implementação e extensão de um sistema de saúde. Nesta fase procurou-se expressar conhecimento imperfeito, proveniente de necessidades lógicas do mundo real. Desta forma, foi considerado um novo tipo lógico, o *Desconhecido*. Sendo assim, é assumido o *PMA(Pressuposto do Mundo Aberto)*, onde é possível admitir este conhecimento. Para além disto, foi também considerado o *PDA(Pressuposto do Domínio Aberto)*, que admite a existência de objetos no universo para além daqueles declarados na base de dados. Para tal, foi necessária a continuação da elaboração de vários predicados e invariantes na linguagem de progamação em lógica, *PROLOG*.

3 Preliminares

Neste segundo exercício foi utilizada a extensão à programação em lógica, de forma a representar o conhecimento perfeito e imperfeito de um sistema de cuidados de saúde. Desta forma, as entidades permaneceram as mesmas utilizadas no exercício anterior, sendo considerado um atributo extra, *Ano de Atualizacao*, no prestador e no utente, fundamental para a edição de dados da base de conhecimento, tratando assim da sua evolução.

Sendo necessário demonstrar conhecimento perfeito (positivo e negativo) e imperfeito (interdito, incerto e impreciso), foram criados predicados e invariantes capazes de gerir estes tipos de conhecimentos, especificamente conhecimento imperfeito, visto ser este o foco deste exercício.

Como referido anteriormente, neste trabalho assume-se também o *PDA (Pressuposto do Domínio Aberto)*, admitindo-se o valor lógico *Desconhecido*. De forma a implementar este novo valor, foi necessário reformular o Sistema de Inferência.

4 Descrição do Trabalho e Ánalise dos Resultados

Com o objetivo de descrever um sistema de saúde com conhecimento imperfeito, foi essencial estabelecer quais os tipos de conhecimentos necessários e aplicar predicados sobre estes, para assim podermos inferir factos.

4.1 Base de Conhecimento

Tal como no primeiro exercício prático, o objetivo foi construir uma representação de conhecimento relacionado com cuidados de saúde. No entanto, nesta fase, considerou-se que a informação presente não é completa, ou seja, é imperfeita.

Cada utente é identificado por um número, nome, a sua idade, morada e a data da última atualização dos seus dados. Os prestadores têm associados a si um identificador, um nome, a sua especialidade, o identificador da instituição a que pertencem e a data da última atualização dos seus dados. Por sua vez, os cuidados, que representam atos médicos, contêm informações pertinentes tais como a data, a hora, o identificador do utente e do prestador, uma descrição do porquê da consulta e o custo da mesma. O formato escolhido para a representação da data foi *ANO-MÊS-DIA*.

Assim, foi elaborada uma base de conhecimento para se poder usufruir dos predicados desenvolvidos e para testar a eficiência dos mesmos. Os dados inseridos podem ser visualizados nas tabelas que se seguem.

Tabela 1: Tabela Utentes.

IdUtente	**Nome**	**Idade**	**Morada**	**Ano Atualização**
1	Tiago Alves	#1	Vila Verde	2018
2	Isabel Pereira	21	Vila Verde	2017
3	Francisco Matos	16	#7	2010
4	Maria Teixeira	21	Gondizalves	2011
-5	Rita Pereira	45	Maximinos	2016
5	Rita Pereira	45	#7	2016
-6	António Silva	12	#13	2009
-6	António Silva	#12	Horta	2009
6	António Silva	#12	#13	2009
7	Mario Duarte	#6	Gualtar	1987
8	Helena Dias	19	Amares	1997
9	Sara Pires	6	#2	2016
10	Hugo Antunes	83	#2	2008
11	Rita Dias	{15;16}	Ferreiros	2010
12	Luisa Fernandes	#6	{Gondomar;Boavista}	2012
13	Ricardo Pries	{32;33}	{Olhão;Barros}	2013
-14	Maria Felgueiras	25	Gondomar	2010

Tabela 2: Tabela Prestador.

IdPrestador	**Nome**	**Especialidade**	**IdInstituição**	**Ano Atualização**
1	Tania Fernandes	Psiquiatria	#3	2012
2	Mario Oliveira	Clínica Geral	2	2010
3	Filipa Ferreira	Pediatria	#3	2017
4	João Machado	Dermatologia	2	2014
-5	Andre Correia	Psiquiatria	4	2016
5	Andre Correia	Psiquiatria	#8	2016
6	Renato Torres	Pediatria	#8	2018
7	Andreia Silva	Cardiologia	4	2017
8	Andre Fernandes	Neurologia	#8	2016
9	Filipe Alves	Oftamologia	#8	2001
-10	Júlio Gonçalves	Urologia	1	2018
-11	Gonçalo Matos	Ginecologia	1	2010
12	Filipe Alves	Clinica Geral	#8	2017
13	Nuno Oliveira	Oftamologia	{2;3}	2013

Tabela 3: Tabela Cuidado.

Data	Hora	IdUtente	IdPrestador	Descriçao	Custo
-2018-4-10	16:00	1	1	Ansiedade	4
2018-4-10	16:00	#11	1	Ansiedade	4
2018-4-11	17:00	5	7	#4	45
2018-5-16	9:00	2	4	Alergia na pele	#10
2018-5-30	9:00	2	4	#4	60
2018-6-2	15:30	3	3	#9	10
2018-9-30	11:00	3	5	#21	#22
2018-10-20	10:45	1	4	Acne	60
2018-10-22	14:25	#14	10	#15	5
2018-11-5	11:30	#16	11	Ecografia	#17
2018-11-5	11:00	#18	2	#19	#20
2018-11-27	13:45	6	9	Miopia	55
2018-12-4	12:00	8	8	#21	#22
2018-12-8	7:00	10	8	Dores de Cabeça	{35;36;37}
2018-12-21	21:00	11	12	{Checkup;Febre}	5
2018-12-28	15:00	{9;13}	7	Ataque Cardíaco	38
-2019-01-05	17:45	3	5	Esquizofrenia	10
-2019-01-05	17:45	6	8	Epilepsia	60

4.2 Conhecimento Perfeito

4.2.1 Conhecimento Positivo

O conhecimento positivo trata-se do conhecimento onde se afirma com certeza que, por exemplo, o utente com ID 2 tem como nome Isabel Pereira, tem 21 anos de idade e mora em Vila Verde.

Este tipo de conhecimento é o mesmo considerado no primeiro trabalho realizado, pelo que não apresenta grandes mudanças nem decisões.

4.2.2 Evolução do Conhecimento Positivo

Tendo em conta os dados presentes numa instituição, considerou-se que não deve haver conhecimento imperfeito associado a esta entidade e que não

há necessidade atualizar os dados da mesma, pois estes devem manter-se constantes ao longo do tempo. Assim, para evoluir o conhecimento das instituições deve ser utilizado o mesmo predicado evolução que foi considerado no mesmo trabalho, sendo que o único cuidado a ter é o de não permitir a inserção de conhecimento repetido, ou seja, não existir mais do que uma instituição para um determinado ID. Assim, foram utilizados o invariante e predicado que se seguem.

```
+instituicao(ID,D,C)  :: (solucoes(ID, instituicao(ID,X,Y), S),
        comprimento(S,L), L =< 1).

evolucao(Termo) :- solucoes(Inv, +Termo :: Inv, S),
                                    insere(Termo),  teste(S).
```

Sendo o conhecimento positivo perfeito o mais correto de se ter numa base de conhecimento, o grupo considerou que qualquer tipo de conhecimento pode ser substituído por este, exceto no caso de existir conhecimento perfeito negativo ou interdito.

Assim, no caso dos utentes, prestadores e cuidados, quando se tem conhecimento incerto ou impreciso, este é removido e é inserido o conhecimento positivo pretendido.

No caso de se querer inserir conhecimento positivo nos utentes ou prestadores, quando já se tem conhecimento positivo, é verificada a data da última atualização, que está presente no predicado existente na base de conhecimento. No caso dos utentes, é permitida a evolução do conhecimento caso a última atualização tenha ocorrido à um ano ou mais. Esta decisão foi tomada devido ao utente ter como argumento a sua idade, que muda todos os anos. Relativamente aos prestadores, é permitida a evolução caso a última atualização tenha ocorrido à 4 anos ou mais, pois considerou-se que não há necessidade de uma mudança assim tão recorrente. Nos cuidados considerou-se que não faz sentido atualizar conhecimento inicialmente positivo, pois um cuidado é algo momentâneo.

O conhecimento interdito não pode ser alterado por conhecimento positivo, pois o conceito deste é que não pode ser afirmado nada acerca de determinado argumento do respetivo predicado. Assim, seria incorreto substituir este por conhecimento positivo.

Por sua vez, o conhecimento perfeito negativo não pode ser substituído por positivo, pois considerou-se contraditório e errado afirmar algo que foi

inicialmente considerado falso. No entanto, o conhecimento negativo que está associado a conhecimento incerto é também removido ao mesmo tempo que o restante.

Para realizar a ação de inserção/substituição foi criado o predicado *evolucaoPositiva* e os invariantes necessários para garantir a consistência da base de conhecimento aquando da utilização deste predicado. Para facilitar o desenvolvimento dos invariantes, consideramos os predicados *utente_perfeito(ID)*, *prestador_perfeito(ID)* e *cuidado_perfeito(D,H,IDp)*. Sempre que é adicionado um novo utente/prestador/cuidado é também adicionado o predicado respetivo. Como um cuidado não tem identificador, é representado pela data, hora e identificador do prestador, que garantem a sua unicidade.

Para a remoção dos predicados que já estavam na base de conhecimento antes da inserção do predicado atualizado, foram desenvolvidos os predicados *removeImpreciso*, *removeIncerto* e *removePerfeitoDatas*, que testam se os predicados do tipo de conhecimento respetivo existem no sistema e, em caso afirmativo, os removem.

De seguida estão presentes alguns exemplos dos invariantes utilizados e do *evolucaoPositiva* para o caso do utente.

```
+utente(ID,N,I,M,D) :: (solucoes(ID, (utente(ID,_,_,_,Ds),
                        utente_perfeito(ID), abaixo1ano(D,Ds)),S),
                        comprimento(S,L), L=<1).

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

evolucaoPositiva(utente(ID,N,I,M,D)) :-
        solucoes(Inv, +utente(ID,N,I,M,D) :: Inv, S),
        insere(utente(ID,N,I,M,D)), teste(S),
        removeImpreciso(utente(ID,_,_,_,_)),
        removeIncerto(utente(ID,_,_,_,_)),
        removePerfeitoDatas(utente(ID,_,_,_,D)),
        assert(utente_perfeito(ID)).
```

4.2.3 Conhecimento Negativo

O conhecimento negativo é utilizado para representar algo que se sabe ser falso na base de conhecimento, por exemplo, quando se sabe que o utente

com o ID 14, com nome Maria Felgueiras e 25 anos não vive em Gondomar. Para o efeito do trabalho, consideramos que o conhecimento negativo é perfeito quando adicionado sozinho e que é imperfeito quando associado a conhecimento incerto.

Como referido no ponto anterior, não é inserido conhecimento negativo acerca das instituições, pois considerou-se que é necessário saber todas as informações corretas acerca das mesmas.

O conhecimento negativo é representado recorrendo à negação forte da respetiva entidade, predicado que pode ser visualizado de seguida.

```
-utente(ID,N,I,M,A) :- nao(utente(ID,N,I,M,A)),
        nao(excecao(utente(ID,N,I,M,A))).
```

4.2.4 Evolução do Conhecimento Negativo

Relativamente á evolução do conhecimento negativo, considerou-se que este só pode ser inserido no caso de não existir ainda na base de conhecimento informações sobre o respetivo utente/prestador/cuidado ou no caso de existir conhecimento incerto.

Considerou-se que o conhecimento negativo não deve ser substituído por conhecimento positivo pelo mesmo motivo que se impediu o inverso, ou seja, por não se achar correto negar algo que inicialmente foi considerado verdadeiro.

No caso de já existir conhecimento negativo sobre aquela entidade na base de conhecimento, é possível adicionar mais conhecimento negativo, desde qu este não seja repetido.

Quanto ao conhecimento imperfeito, considerou-se que este não deve ser substituído por negativo, pois não faz sentido passar de uma gama de valores conhecidas para um valor que se sabe não ser o correto. Isto é, faz mais sentido, por exemplo, saber que o João tem 15 ou 16 anos, do que saber que o João não tem 40 anos.

O conhecimento interdito, como em todos os outros casos, não deve ser substituído, pois não devem ser tiradas conclusões acerca do mesmo.

Quando é associado conhecimento negativo a conhecimento incerto, ao contrário do que acontece no caso do positivo, não é removido o predicado incerto, pois considera-se que isso não apresenta inconsistência. É correto afirmar que, por exemplo, não sabemos a idade do João, mas sabemos que ele não tem 40 anos.

Tendo em conta os pressupostos considerados, recorreu-se ao predicado *testa_perfeito* para determinar se o conhecimento negativo inserido é único na base de dados, pelo que passa a ser considerado perfeito, ou se está associado a conhecimento incerto.

Tal como no caso positivo, para garantir a eficiência e consistência da base de conhecimento, foi criado o predicado *evolucaoNegativa* e os invariantes necessários para verificar as decisões descritas. Assim, apresenta-se de seguida alguns exemplos destes.

```
+(-utente(ID,N,I,M,D)) :: (solucoes(ID, (utente(ID,_,Is,Ms,_),
                            Is \= xpto6, Is \= xpto12,
                             Ms \= xpto7), S),
                          comprimento(S,L), L==0).

+(-utente(ID,N,I,M,D)) :: (solucoes(ID,
        (utente_impreciso_idade(ID); utente_impreciso_morada(ID)), S),
        comprimento(S,L), L==0).

evolucaoNegativa(utente(ID,N,I,M,D)) :-
        solucoes(Inv, +(-utente(ID,N,I,M,D)) :: Inv, S),
        insere(-utente(ID,N,I,M,D)),
        teste(S),
    solucoes(ID, (utente(ID,N,Is,Ms,D),
        (Is == xpto6; Is == xpto12; Ms == xpto7)), R),
        testaPerfeito(R,utente(ID,_,_,_,_)).
```

4.3 Conhecimnto Imperfeito

4.3.1 Conhecimento Interdito

O conhecimento interdito é utilizado para caraterizar informações que nunca podem ser obtidas, seja por estas serem restritas ou por serem impossíveis de determinar. Por exemplo, é impossível determinar a data exata de nascimento de uma pessoa que não tenha certidão de nascimento e que não conheça os seus pais biológicos nem saiba nada sobre a sua vida. Deste modo, diz-se que a data de nascimento do indivíduo é um nulo interdito.

Para o caso de estudo em questão, decidiu-se que apenas alguns valores podem ser nulos interditos nas entidades, havendo argumentos que devem ser de conhecimento obrigatório.

No caso dos utentes, determinou-se que só podem ser considerados nulos interditos a idade e/ou morada do utente, pois o seu ID é algo específico do sistema de saúde, logo é acessível, o nome deve ser sempre obrigatoriamente disponibilizado pelo próprio e a data de atualização também é algo específico do sistema, sendo sempre também possível de determinar. Assim, a idade e a morada são os únicos argumentos que podem, por algum motivo, ter um valor interdito. Por exemplo, um utente pode sentir a necessidade de não disponibilizar a sua morada.

Para os prestadores apenas se aceita o argumento identificador da instituição como nulo interdito, pois, tal como no utente, considerou-se que o ID, nome e data de atualização são valores necessários e de acesso fácil. Relativamente à especialidade, presumiu-se que esta é essencial num prestador, pois o sistema de saúde tem necessidade de saber a especialidade para poder, em cada cuidado, atribuir um prestador adequado às necessidades do utente. O identificador da instituição foi considerado com um valor possivelmente nulo, porque pode tratar-se, por exemplo, de um médico que apenas aceite fazer consultas por conta própria.

Relativamente aos cuidados, foram aceites como nulos interditos a descrição e o custo do mesmo. A data e hora de um cuidado são essenciais para a correta realização do mesmo, tal como o ID do prestador e do utente em questão, daí se ter impossibilitado a interdição destes. No caso da descrição e do custo, considerou-se que se pode, por exemplo, estar a lidar com utentes que são figuras públicas importantes e que podem necessitar de garantir a máxima discrição nos seus registos.

Mais uma vez, considerou-se que não podem haver anomalias nos argumentos das instituições.

Para a implementação deste tipo de conhecimento foram desenvolvidas as exceções necessárias na base de conhecimento, bem como criados os predicados utilizados para identificar nulos interditos. De seguida, apresenta-se alguns exemplos destes predicados.

```
nulo_utente_idade(xpto1).
excecao(utente(Id,N,I,M,A)) :- utente(Id,N,xpto1,M,A).

nulo_utente_morada(xpto2).
excecao(utente(Id,N,I,M,A)) :- utente(Id,N,I,xpto2,A).
```

4.3.2 Evolução do Conhecimento Interdito

Tendo em conta o conceito de nulo interdito e a delicadeza envolvida ao lidar com estes valores, considerou-se que seria uma boa opção dar prioridade a esta forma de conhecimento acima de todas as restantes. Então, é possível substituir conhecimento positivo, negativo, incerto e impreciso por conhecimento interdito. Por questões de consistência, o único cuidado/restrição tomado para estes valores é a impossibilidade de adicionar predicados repetidos.

Esta decisão foi tomada seguindo a mesma linha de pensamento descrita no tópico anterior, onde, por exemplo, um utente pode ter a necessidade de salvaguardar certas informações pessoais, mesmo que em momentos passados tenha fornecido todos os seus dados. Daí ser possível substituir até o conhecimento perfeito positivo por conhecimento interdito.

Para tal, apenas foi necessário criar invariantes que garantem a unicidade do conhecimento e o predicado *evolucaoInterdita*, tal como é possível verificar de seguida para o caso do utente.

Para além destes, foi também necessário recorrer aos predicados *removeImpreciso*, *removeIncerto* e *removePerfeito*, para remover predicados que poderiam já existir no sistema antes da inserção do nulo interdito, garantindo assim que não há contradições.

```
^(utente(ID,N,I,M,D)) ::
        (solucoes(Is,(utente(ID,_,Is,_,_), nulo_utente_idade(Is),
            I\=xpto1),S),
        comprimento(S,L),
        L==0).


^(utente(ID,N,I,M,D)) ::
        (solucoes(Ms,(utente(ID,_,_,Ms,_), nulo_utente_morada(Ms),
            M\=xpto2),S),
        comprimento(S,L),
        L==0).
```

```
evolucaoInterdita(utente(ID,N,nulo,M,D)) :-
        M \= nulo,
        solucoes(Inv, ^(utente(ID,N,xpto1,M,D)) :: Inv, S),
        insere(utente(ID,N,xpto1,M,D)),
        teste(S),removeImpreciso(utente(ID,_,_,_,_)),
        removeIncerto(utente(ID,_,_,_,_)),
        removePerfeito(utente(ID,_,_,_,_)).

evolucaoInterdita(utente(ID,N,I,nulo,D)) :-
        I \= nulo,
        solucoes(Inv, ^(utente(ID,N,I,xpto2,D)) :: Inv, S),
        insere(utente(ID,N,I,xpto2,D)),
        teste(S),
        removeImpreciso(utente(ID,_,_,_,_)),
        removeIncerto(utente(ID,_,_,_,_)),
        removePerfeito(utente(ID,_,_,_,_)).

evolucaoInterdita(utente(ID,N,nulo,nulo,D)) :-
        solucoes(Inv, ^(utente(ID,N,xpto1,xpto2,D)) :: Inv, S),
        insere(utente(ID,N,xpto1,xpto2,D)),
        teste(S),
        removeInterdito(utente(ID,_,_,_,_)),
        removeImpreciso(utente(ID,_,_,_,_)),
        removeIncerto(utente(ID,_,_,_,_)),
        removePerfeito(utente(ID,_,_,_,_)).
```

4.3.3 Conhecimento Incerto

O conhecimento incerto é caraterizado por ser informação que não é conhecida, mas que, ao contrário de no interdito, pode vir a ser conhecida no futuro. Tratam-se então de dados que, por algum motivo, não estão acessíveis no momento em que o conhecimento foi adicionado ao sistema.

Assim, no caso dos utentes, foram postas em prática as mesmas considerações que no conhecimento interdito, ou seja, que podem ser considerados como valores incertos a idade e/ou a morada do utente.

Os prestadores também se mantêm como na caso do interdito, em que apenas é possível ter como valor incerto o identificador da instituição.

No caso dos cuidados, considerou-se que para além da descrição e preço,

também podemos ter como incerto o identificador do utente. Apesar de poder ser visto como algo controverso, por ser um valor essencial, pôs-se em causa a possibilidade de haver algum problema na marcação do cuidado e ser perdido do ID do utente em questão. O mesmo não se considerou para o identificador do prestador, pois acreditou-se que seria mais fácil descobrir o ID do prestador do que o do utente e por ser necessário garantir pelo menos um destes valores, para se poder verificar a unicidade de cada cuidado. Assim, para se verificar se um cuidado é único recorre-se à data, hora e ID do prestador do mesmo.

Para representar o conhecimento incerto, foi necessário criar exceções adequadas. De seguida estão representadas as exceções desenvolvidas para o utente.

```
excecao(utente(Id,N,I,M,A)) :- utente(Id,N,xpto6,M,A).
excecao(utente(Id,N,I,M,A)) :- utente(Id,N,I,xpto7,A).
excecao(utente(Id,N,I,M,A)) :- utente(Id,N,xpto12,xpto13,A).
```

4.3.4 Evolução do Conhecimento Incerto

Para se inserir conhecimento incerto na base de conhecimento, considerou-se que não pode haver qualquer tipo de conhecimento acerca da entidade em questão presente na mesma. Isto é, para se adicionar conhecimento incerto, por exemplo, sobre o utente com ID 5, não pode haver à partida nenhum predicado referente ao utente com o identificador 5.

Esta decisão foi tomada porque, à partida, é pior não saber nada sobre determinado dado de uma entidade do que saber o valor concreto do dado, saber se não é um certo valor ou saber uma gama de valores possíveis. Então, não faria sentido substituir conhecimento positivo, negativo ou impreciso por conhecimento incerto. Para além destes, o interdito, como em todos os outros casos, não pode ser substituído.

Com base nestes pressupostos, foram criados vários invariantes para garantir a unicidade do predicado, bem como o cumprimento de todas as regras indicadas. Para além dos invariantes, foi também desenvolvido o predicado *evolucaoIncerta*. Neste caso, não houve necessidade de recorrer a predicados de remoção de conhecimento anterior, visto que isto nunca é permitido.

```
*(utente(ID,N,I,M,D)) ::
        (solucoes(ID, utente(ID,_,_,_,_), S),
        comprimento(S,L),
        L==1).

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

*(utente(ID,N,I,M,D)) ::
        (solucoes(ID, (utente_impreciso_idade(ID);
            utente_impreciso_morada(ID)), S),
        comprimento(S,L),
        L==0).

evolucaoIncerta(utente(ID,N,nulo,M,D)) :-
        solucoes(Inv, *(utente(ID,N,xpto6,M,D)) :: Inv, S),
        insere(utente(ID,N,xpto6,M,D)),
        teste(S).

evolucaoIncerta(utente(ID,N,I,nulo,D)) :-
        solucoes(Inv, *(utente(ID,N,I,xpto7,D)) :: Inv, S),
        insere(utente(ID,N,I,xpto7,D)),
        teste(S).

evolucaoIncerta(utente(ID,N,nulo,nulo,D)) :-
        solucoes(Inv, *(utente(ID,N,xpto12,xpto13,D)) :: Inv, S),
        insere(utente(ID,N,xpto12,xpto13,D)),
        teste(S).
```

4.3.5 Conhecimento Impreciso

O conhecimento impreciso representa um dado em que, apesar de não ser conhecido o seu valor em concreto, é conhecido um conjunto ou uma gama de valores onde se sabe estar o correto. Por exemplo, não sabemos qual é a idade do utente com ID 11, mas sabemos que é 15 ou 16 anos. Outro exemplo que se pode demonstrar por conhecimento impreciso é nos cuidados, em que

pode não se saber o custo correto de um cuidado, mas saber que este ficou entre os 40 e os 50 euros.

Os argumentos que se admitiu poderem tomar valores imprecisos correspondem aos mesmos que se assumiu no conhecimento incerto, ou seja, a idade e morada dos utentes, o ID da instituição dos prestadores e o ID do utente, a descrição e o custo dos cuidados.

O conhecimento impreciso, ao contrário dos restantes, não origina predicados de exceções genéricos. Neste caso, cada entidade que é apresentada como imprecisa deve ser inserida como uma exceção. Para além disto, de maneira a facilitar o desenvolvimento dos predicados e invariantes necessários, criou-se também os predicados *utente_impreciso_idade*, *utente_impreciso_morada*, *prestador_impreciso_idInst*, *cuidado_impreciso_idUt*, *cuidado_impreciso_descricao* e *cuidado_impreciso_preco*, que são utilizados sempre que é inserido uma informação imprecisa.

De seguida, apresenta-se alguns exemplos de conhecimento impreciso presente na base de dados.

```
excecao(utente(12, luisa_fernandes, xpto6, gondomar, 2012)).
excecao(utente(12, luisa_fernandes, xpto6, boavista, 2012)).
utente_impreciso_morada(12).
excecao(utente(13, ricardo_pires, 32, barros, 2013)).
excecao(utente(13, ricardo_pires, 32, olhao, 2013)).
excecao(utente(13, ricardo_pires, 33, olhao, 2013)).
excecao(utente(13, ricardo_pires, 33, barros, 2013)).
utente_impreciso_idade(13).
utente_impreciso_morada(13).
```

4.3.6 Evolução do Conhecimento Impreciso

Relativamente à evolução do conhecimento impreciso, pressupôs-se que apenas se pode substituir conhecimento incerto por por impreciso e qye pode-se adicionar mais conhecimento impreciso ao já existente.

Considerou-se que não faria sentido evoluir de conhecimento positivo para impreciso, porque no positivo, para além de ser perfeito, já indica os valores certos dos dados.

Para a conhecimento negativo pressupôs-se que é mais explícito ter um facto negativo do que um facto impreciso. Assim, também não é possível evoluir de conhecimento negativo para impreciso.

Quanto ao interdito, é tido em conta o mesmo conceito que nas restantes evoluções, isto é, que não pode ser substituído conhecimento com nulos interditos.

Por outro lado, faz todo o sentido evoluir de incerto para impreciso, pois é melhor saber, por exemplo, que o Francisco tem 20 ou 21 anos do que não saber qual a idade do Francisco.

Relativamente à existência de conhecimento impreciso à priori, é sempre possível adicionar mais imprecisos, desde que a informação não seja repetida.

Assim, para evoluir o conhecimento e garantir a consistência da base de conhecimento, foram desenvolvidos os invariantes necessários para respeitar os pressupostos considerados e o predicado *evolucaoImprecisa*.

```
~(utente(ID,N,I,M,D)) ::
        (solucoes(ID, (utente(ID,_,Is,Ms,_),
            Is \= xpto6, Is \= xpto12, Ms \= xpto7), S),
        comprimento(S,L),
        L==0).

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

~(utente(ID,N,I,M,D)) ::
        (solucoes(ID, (utente(ID,_,Is,_,_), nulo_utente_idade(Is)), S),
        comprimento(S,L),
        L==0).

~(utente(ID,N,I,M,D)) ::
        (solucoes(ID, (utente(ID,_,_,Ms,_), nulo_utente_morada(Ms)), S),
        comprimento(S,L),
        L==0).

~(utente(ID,N,I,M,D)) ::
        (solucoes(ID, (excecao(utente(ID,N,I,M,Ds)),
            utente_impreciso_idade(ID)), S),
        comprimento(S,L),
        L=<1).
```

```
~(utente(ID,N,I,M,D)) ::
        (solucoes(ID, (excecao(utente(ID,N,I,M,Ds)),
            utente_impreciso_morada(ID)), S),
        comprimento(S,L),
        L=<1).

evolucaoImprecisa(utente(ID,N,I,M,D),T) :-
        solucoes(Inv, ~(utente(ID,N,I,M,D)) :: Inv, S),
        insere(excecao(utente(ID,N,I,M,D))),
        teste(S),
        removeIncerto(utente(ID,_,_,_,_)),
        ((nao(T==i),nao(T==im)); utente_impreciso_idade(ID);
            assert(utente_impreciso_idade(ID))),
        ((nao(T==m),nao(T==im)); utente_impreciso_morada(ID);
            assert(utente_impreciso_morada(ID))).
```

4.4 Involução de Conhecimento

Para a remoção de informação foi desenvolvido o predicado *involucao*. Este predicado deve ser utilizado caso se queira remover qualquer tipo de conhecimento, pois considerou-se que não seria essencial criar diferentes involuções para cada tipo de conhecimento.

Para assegurar a consistência da base de conhecimento, apenas se assumiu que, para remover um utente, este não pode estar relacionado com nenhum cuidado, caso contrário os cuidados a si associados estariam a guardar o ID de um utente não existente. Tendo isto em conta, o mesmo é assumido para os prestadores. Pelo mesmo motivo, considerou-se que, para remover uma instituição, esta não pode estar relacionada com nenhum prestador.

De seguida estão representados os invariantes e o predicado desenvolvidos.

```
-utente(ID,N,I,M,D) ::
        (solucoes(ID, (cuidado(Dt,H,ID,IDp,Ds,C);
            -cuidado(Dt,H,ID,IDp,Ds,C)) , S),
        comprimento(S,L),
        L==0).
```

```
-prestador(ID,N,E,IDi,D) ::
        (solucoes(ID, (cuidado(Dt,H,IDu,ID,Ds,C);
            -cuidado(Dt,H,IDu,ID,Ds,C)) , S),
        comprimento(S,L),
        L==0).

-prestador(ID,N,E,IDi,D) ::
        (solucoes(ID, (excecao(cuidado(Dt,H,IDu,ID,Ds,C)),
            IDu == xpto11; IDu == xpto14; IDu == xpto16;
            IDu == xpto18; Ds == xpto9; Ds == xpto21; C == xpto11) , S),
        comprimento(S,L),
        L==0).

-instituicao(ID,N,M) ::
        (solucoes(ID, (prestador(IDp,Np,E,ID);
            -prestador(IDp,Np,E,ID)), S),
        comprimento(S,L),
        L==0).

involucao(Termo) :-
        solucoes(Inv, -Termo :: Inv, S),
        remove(Termo),
        teste(S).
```

4.5 Sistema de Inferência

Neste exercício consideramos um novo valor lógico, *desconhecido*. Tendo isso em conta, foi necessário restabelecer os resultados lógicos entre os possiveis valores.

No caso da conjunção, consideramos que caso exista a possibilidade do conhecimento não ser totalmente verdadeiro ou totalmente desconhecido então o resultado deverá ser o valor lógico mais "negativo"entre os dois. Ou seja, caso exista *desconhecido* então nunca será possivel obter o valor *V*, e se for uma relação entre este e *Falso*, na nossa opinião, fará sentido obter o valor *F*. Por fim, apenas sobra a relação entre *Desconhecido* consigo próprio, que resulta em si próprio.

No caso da disjunção, foi realizado o raciocínio inverso à conjunção, ou seja, caso exista a possibilidade do conhecimento não ser totalmente verdadeiro ou totalmente desconhecido então o resultado deverá ser o valor lógico

mais "forte". Sendo assim, *Verdadeiro* $\vee$ *Desconhecido* irá originar V, *Falso* $\vee$ *Desconhecido* deverá obter a resposta D. Tal como na conjunção, a relação do *Desconhecido* consigo próprio resultará em si próprio.

Tabela 4: Tabela Lógica - Conjunção.

	Verdadeiro	**Desconhecido**	**Falso**
Verdadeiro	V	D	F
Desconhecido	D	D	F
Falso	F	F	F

Tabela 5: Tabela Lógica - Disjunção.

	Verdadeiro	**Desconhecido**	**Falso**
Verdadeiro	V	V	V
Desconhecido	V	D	D
Falso	V	D	F

```
conjuncao(V1,V2,falso) :- V1 == falso; V2 == falso.
conjuncao(verdadeiro,verdadeiro,verdadeiro).
conjuncao(verdadeiro,desconhecido,desconhecido).
conjuncao(desconhecido,V,desconhecido) :- V == desconhecido; V == verdadeiro.

disjuncao(V1,V2,verdadeiro) :- V1 == verdadeiro; V2 = verdadeiro.
disjuncao(falso,falso,falso).
disjuncao(falso,desconhecido,desconhecido).
disjuncao(desconhecido,V,desconhecido) :- V == desconhecido; V == falso.
```

Tal como referido anteriormente, foi necessário definir os predicados conjuncao e disjuncao, de forma a lidar também com o novo valor lógico introduzido *desconhecido*. Os resultados correspondem às tabelas apresentadas anteriormente.

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

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

demoLista([],[]).
demoLista([Q1|T],[R1|R]) :- demo(Q1,R1), demoLista(T,R).

demoComp(Q1 && CQ, R) :- demo(Q1,R1), demoComp(CQ,R2), conjuncao(R1,R2,R).
demoComp(Q1 ## CQ, R) :- demo(Q1,R1), demoComp(CQ,R2), disjuncao(R1,R2,R).
demoComp(Q1, R) :- demo(Q1,R).
```

Estes quatro predicados permitem ao sistema responder a qualquer questao. Através do *demo*, realiza uma verificação da questão e caso esta se prove, devolve verdadeiro, caso contrário, devolve falso. No caso de não ser possivel verificar se é verdadeiro ou falso, obtemos o novo valor lógico introduzido, *desconhecido*.

Quanto à *demoLista*, como o nome indica, recebe uma lista de questões, e atravês da demo, devolve o resultado da conjunção de todos os resultados.

Por fim, a *demoComp* já é capaz de interpretar sinais de disjunçao e conjunçao (## e &&, respetivamente), atravês das funçoes explicadas anteriormente de forma a obter o resultado esperado

4.6 Outros Predicados

Para uma melhor utilização da base de conhecimento, desenvolveu-se predicados que devolvem determinadas anomalias presentes na mesma. Estes predicados foram criados com o intuito de facilitar o trabalho do administrador do sistema, melhorando o acesso às entidades que apresentam conhecimento imperfeito.

O predicado *utentesAnomalias* devolve uma lista com todos os utentes que apresentam conhecimento interdito, incerto ou impreciso.

Quanto ao predicado *demoLista*, este permite lidar, como o nome indica, com uma lista de questões, devolvendo o valor da conjunção de todas as respostas correspondentes.

Por fim, o predicado *demoComp* realiza o mesmo processo que o anterior, no entanto, permite a utilização de conjunções e disjunçoes (os símbolos && e ## respetivamente), que foram desenvolvidas anteriormente, e obtém a resposta correspondente.

```
utentesAnomalias(S) :-
        solucoes(utente(ID,N,I,M,D), (utente(ID,N,I,M,D),
          (utente_impreciso_idade(ID); utente_impreciso_morada(ID);
          nulo_utente_idade(ID); nulo_utente_morada(M);
          I == xpto6; I == xpto12; M == xpto7)),
        S).
```

O predicado *prestadoresAnomalias* devolve uma lista com todos os prestadores que apresentam conhecimento interdito, incerto ou impreciso.

```
prestadoresAnomalias(S) :-
        solucoes(prestador(ID,N,E,IDi,D),
          (prestador(ID,N,E,IDi,D), (prestador_impreciso_idInst(ID);
          nulo_prestador_idInst(IDi); IDi == xpto8)),
        S).
```

O predicado *cuidadosAnomaliasUtente* devolve uma lista com todos os cuidados que apresentam conhecimento interdito, incerto ou impreciso para um determinado utente, a partir do seu ID.

```
cuidadosAnomaliasUtente(IDu,S) :-
        solucoes(cuidado(D,H,IDu,IDp,Ds,C), (cuidado(D,H,IDu,IDp,Ds,C),
          (cuidado_impreciso_descricao(D,H,IDp);
          cuidado_impreciso_preco(D,H,IDp);
          nulo_cuidado_descricao(Ds); nulo_cuidado_custo(C);
          Ds == xpto9; Ds == xpto21; C == xpto11)),
        S).
```

O predicado *cuidadosAnomaliasPrestador* devolve uma lista com todos os cuidados que apresentam conhecimento interdito, incerto ou impreciso para um determinado prestador, a partir do seu ID.

```
cuidadosAnomaliasPrestador(IDp,S) :-
        solucoes(cuidado(D,H,IDu,IDp,Ds,C), (cuidado(D,H,IDu,IDp,Ds,C),
          (cuidado_impreciso_idUt(D,H,IDp);
          cuidado_impreciso_descricao(D,H,IDp);
          cuidado_impreciso_preco(D,H,IDp);
          nulo_cuidado_descricao(Ds); nulo_cuidado_custo(C);
          IDu == xpto11; IDu == xpto14; IDu == xpto16;
          IDu == xpto18; Ds == xpto9; Ds == xpto21; C == xpto11)),
        S).
```

5 Conclusões e Sugestões

A elaboração deste segundo trabalho permitiu-nos aprofundar os nossos conhecimentos sobre a utilização da extensão à programação em lógica e conhecimento imperfeito, utilizando a linguagem de programacão em lógica, *PROLOG*.

No geral, considera-se que a base de conhecimento desenvolvida é bastante consistente e funcional. Conseguimos implementar todos os tipos de conhecimento estudado e lidar com estes duma maneira eficaz. Para além disso, criamos diferentes predicados *demos* para conseguir fornecer ao utilizador resposta a todo o tido de perguntas que este desejar. Com isto, conseguimos aplicar corretamente o valor lógico desconhecido.

No entanto, é importante referir que, para trabalho futuro, poderia ser melhorada a evolução de conhecimento impreciso, sendo que a implementada não aceita casos em que se pretende inserir um impreciso numa gama de valores. Por exemplo, não é possível inserir um utente com idade compreendida entre 10 e 15 anos, a não ser inserindo todas as idades possíveis.

É também apontado como trabalho futuro aperfeiçoar o caso em que se pretende inserir conhecimento negativo e simultaneamente interdito. Por exemplo, o caso em que sabemos que um utente não tem 20 anos e a sua morada é um nulo interdito. Neste caso específico, a base de conhecimento desenvolvida não consegue lidar corretamente com a informação. Por último, seria também interessante aprofundar os pressupostos tomados para a involução de conhecimento.

Tendo em conta os objetivos do projeto e o trabalho realizado pelo grupo, consideramos que foi cumprido o esperado e vemos o trabalho desenvolvido com sucesso.