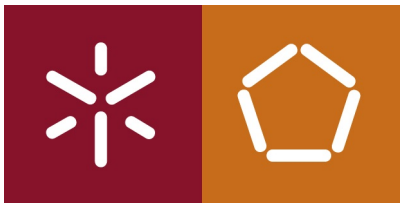
Universidade do Minho



Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

3º Ano, 2º Semestre, 2018/2019

Programação em Lógica Estendida e Conhecimento Imperfeito

A81712 Ana Filipa Pereira
A81368 Inês Alves
A81580 Francisco Freitas
A81611 Maria Dias
A80975 Pedro Freitas

May 25, 2019

1 Resumo

O presente relatório documenta o segundo exercício prático da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio. Este projeto tem como finalidade o 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, recorrendo para isso à extensão à programação em lógica. Pretende-se também mostrar os conhecimentos adquiridos nesta área através da linguagem PROLOG, usando esta para desenvolver um programa que imite o sistema já mencionado.

Índice

Lista de figuras

2 Introdução

No âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, foi-nos proposto que explorássemos as nossas competências ao nível da linguagem de programação lógica *PROLOG*, resolvendo, para isso, um exercício do tema *Programação em Lógica Estendida e Conhecimento Imperfeito*.

Com este exercício, pretende-se que seja desenvolvido um Sistema de Representação de Conhecimento e Raciocínio com capacidade para caracterizar um universo de discurso na área da prestação de cuidados de saúde.

Esclarecido o cenário, facilmente identificamos as três principais fontes de conhecimento deste exercício: o Utente, o Cuidado e o Prestador. Assim, para além de possibilitar dar resposta às questões apresentadas no enunciado, pretende-se também criar um sistema que esteja apto a guardar e processar informações relativamente a estas três fontes.

3 Preliminares

3.1 Representação de Conhecimento Imperfeito

No trabalho prático realizado anteriormente, tínhamos um sistema capaz de representar conhecimento perfeito, assemelhando-se a um sistema de bases de dados, na medida em que assumia que a informação representada seria única e válida, e que as entidades representadas seriam as únicas existentes no mundo exterior. Este sistema era baseado no **pressuposto do mundo fechado**, que enuncia que todo o conhecimento que se encontra na base de conhecimento é verdadeiro, e tudo o que não estiver incluído é obrigatoriamente falso.

Pelo contrário, neste trabalho pretende-se implementar um sistema de representação de conhecimento capaz de representar conhecimento imperfeito, ou seja, pretendemos aplicar o **pressuposto do mundo aberto**, segundo o qual nem sempre podemos afirmar a falsidade de um conhecimento que não conste na base de conhecimento. Assim, poderemos aproximar este sistema à realidade das coisas, em que quando não se sabe algo, apenas se assume esse algo como **desconhecido**. Este termo passará a denotar uma nova forma de catalogar uma resposta, para além dos termos *verdadeiro* e *falso*. Assim, uma questão $q(x)$ poderá ter as seguintes respostas:

- **Verdadeira:** $\exists x : q(x)$
- **Falsa:** $\exists x : \neg q(x)$
- **Desconhecida:** $\neg \exists x : q(x) \vee \neg q(x)$

3.2 Evolução do Conhecimento

A necessidade de representar conhecimento imperfeito faz com que seja também preciso alterar o processo de evolução e retrocesso de conhecimento desenvolvidos na fase anterior. De forma a atender ao pressuposto do mundo aberto, há a necessidade de introduzir conhecimento imperfeito na base de dados, relativo aos diferentes tipos de valores nulos que podemos encontrar no âmbito da lógica estendida. Como teremos oportunidade de ver mais à frente, este conhecimento será representado através de exceções, sendo que, no caso dos valores nulos de terceiro tipo, será também preciso definir invariantes para assegurar que o conhecimento não pode evoluir.

3.3 Conhecimento Negativo

Tradicionalmente, um programa de lógica deduz que a ausência de uma dada informação significa que a mesma é falsa (negação por falha).

Uma extensão de um programa em lógica pode incluir informação negativa explicitamente, bem como explicitar directamente o pressuposto do mundo fechado para alguns predicados, como iremos ver nas próximas secções. A introdução da negação explícita veio trazer uma maior expressividade da linguagem. Esta negação serve para casos em que não basta que um pedaço de conhecimento esteja ausente da base de conhecimento para provar a sua falsidade. Para além disso, em alguns casos, pode acontecer que haja uma ou mais exceções à regra, pelo que estas também devem passar a ser incluídas na expressão da negação. Assim sendo, para negar p(x), passamos a ter:

$$\neg p(x) \leftarrow \neg \exists x : p(x) \vee excecao(p(x)),$$

que significa "$p(x)$ é falso se não ouver nenhuma prova verdadeira de $p(x)$ nem houver nenhuma exceção de $p(x)$" e que, em *PROLOG*, podemos representar da seguinte forma:

```
-p(X) :- nao( p(X) ), nao( excecao( p(X) ) ).
```

Esta regra foi aplicada para representar a negação de todos os predicados presentes na base de conhecimento.

De notar que, de modo a permitir evolução correspondente a conhecimento negativo, o predicado *evolucao* sofreu algumas alterações.

4 Descrição do trabalho

4.1 Conhecimento Perfeito

No que diz respeito ao conhecimento perfeito, propomos de seguida alguns exemplos para cada um dos predicados.

```
% Extensao do predicado utente: Id, Nome, Idade, Sexo, Morada ⇝ {V,F,D}
utente(1,'Ana Santos',34,'Rua São Gonçalo - S.Vicente - Braga').
utente(2,'Bruno Mendonça',23,'Rua de Santa Catarina - Porto').
utente(3,'Carla Martins',29,'Rua de Santa Maria - Brito - Guimarães').
utente(4,'Beatriz Jesus',12,'Rua da Bica - São Paulo - Lisboa').
utente(5,'Júlio Carvalho',36,'Rua Santa Margarida - S.Victor - Braga').
utente(6,'Carlos Silva',19,'Rua das Flores - Sé e Vitória - Porto').
utente(7,'Xavier Teixeira',51,'Rua dos Chãos - Souto - Braga').
utente(8,'Luis Almeida',32,'Rua Padre Feliciano - Fraião - Braga').
utente(9,'Pedro Lima',20,'Rua Prof.Machado Vilela - S.Victor - Braga').
utente(10,'André Campos',42,'Rua da Graça - Graça -Lisboa').

% Extensao do predicado cuidado: Data,  IDUt, IDPrest, Descricao,
Custo ⇝ {V,F,D}
cuidado('10-11-2018',1,5,'Ginecologia',30).
cuidado('18-03-2018',2,10,'Ortopedia',47).
cuidado('09-02-2019',3,5,'Ginecologia',30).
cuidado('28-01-2019',4,2,'Psiquiatria',19).
cuidado('09-02-2019',3,5,'Ginecologia',30).
cuidado('16-09-2018',5,7,'Cardiologia',35).
cuidado('11-04-2018',6,9,'Oftalmologia',22).
cuidado('23-05-2018',7,6,'Oftalmologia',25).
cuidado('29-01-2019',8,7,'Cardiologia',35).
cuidado('11-04-2018',9,8,'Otorrinolaringologia',15).
cuidado('16-09-2018',10,1,'Dermatologia',47).

% Extensao do predicado prestador: IDServ, Descricao, Instituicao,
Cidade ⇝ {V,F,D}
prestador(1,'José Tur',['Dermatologia','Ortopedia'],['Hospital S.José']).
prestador(2,'Ema Félix',['Psiquiatria','Otorrinolaringologia'],
                        ['Hospital S.José','Hospital da Luz']).
prestador(3,'Helena Pereira',['Oftalmologia','Dermatologia'],
            ['Centro Hospitalar e Universitário de Coimbra']).
prestador(4,'Rodrigo Vieira',['Cardiologia'],['Hospital da Senhora
                                      da Oliveira Guimarães']).
prestador(5,'Maria Pinto',['Ginecologia'],['Hospital Senhora da
            Oliveira Guimarães','Centro de Saúde de Maximinos']).
prestador(6,'Mariana Sousa',['Oftalmologia','Ortopedia'],['Centro
                    de Saúde de Maximinos','Hospital de Braga']).
prestador(7,'Susana Costa',['Cardiologia'],['Hospital de Braga']).
prestador(8,'Rui Cruz',['Otorrinolaringologia'],['Hospital de Braga']).
```

```
prestador(9,'Sofia Lopes',['Dermatologia'],['Hospital de Santa Maria']).
prestador(10,'Manuel Marques',['Ortopedia'],['Hospital S.João']).
```

Estes exemplos diferem dos que tínhamos no exercício anterior apenas no contradomínio, uma vez que agora existe um terceiro valor de verdade.

Ainda no âmbito do conhecimento perfeito, mas debruçando-nos agora na capacidade de inserção de conhecimento negativo, apresentamos de seguida alguns exemplos de aplicação deste tipo de conhecimento em cada um dos predicados:

```
% Inserção de conhecimento perfeito negativo:
-utente(30,'Filomena Guimarães', 20, 'Rua do Campo').
-servico(40,'Cardiologia','Clínica da Fonte Nova','Braga').
-cuidado(data(05,05,2005),1,17,87,7).
-prestador(9,'Raul Fernandes', ['Cardiologia'], ['Centro de Saúde de
 Maximinos','Hospital de Braga']).
```

Com estes predicados, estamos a afirmar explicitamente que aqueles utentes, cuidados, serviços e prestadores não constam da base de conhecimento.

Para os predicados utente, prestador e cuidado, recorremos ao Pressuposto do Mundo Fechado para explicitar que um utente/cuidado/prestador que não estejam definidos na base de conhecimento não existem, tirando em casos de exceção, que devem ser definidos.

```
% Pressuposto do mundo fechado para o predicado utente:
-utente( IDUt, Nome, Idade, Morada ) :-
                nao( utente( IDUt, Nome, Idade, Morada ) ),
                nao( excecao( utente( IDUt, Nome, Idade, Morada ) ) ).

% Pressuposto do mundo fechado para o predicado prestador:
-prestador( IDPrest, Nome, Esp, Inst ) :-
                nao( prestador( IDPrest, Nome, Esp, Inst ) ),
                nao( excecao( prestador( IDPrest, Nome, Esp, Inst ) ) ).


% Pressuposto do mundo fechado para o predicado serviço:
-cuidado(Data,IDU,IDPrest,D,C) :-
                        nao( cuidado( Data,IDU,IDPrest,D,C ) ),
                        nao( excecao( Data,IDU,IDPrest,D,C ) ).
```

4.2 Representação de Informação Incompleta

Como já foi referido, este trabalho realiza-se no sentido de dotar um sistema de representação de conhecimento de características capazes de o fazer raciocinar segundo o pressuposto do mundo aberto, de forma a permitir tratar informação incompleta.

Para abordar a representação de informação incompleta, distinguimos três possíveis conclusões para uma questão: verdadeira, falsa ou, quando não se pode concluir nenhuma das anteriores, desconhecida.

Na linguagem *PROLOG*, representamos as possíveis respostas a uma dada questão através do predicado demo (de demonstração), apresentado de seguida:

```
% Extensão do meta-predicado demo: Questao ~> {V,F,D}
demo( Questao, verdadeiro ) :- Questao.
demo( Questao, falso ) :- -Questao.
demo( Questao, desconhecido ) :- nao( Questao ), nao( -Questao ).
```

O predicado anterior consiste num sistema de inferência que aceita como argumento apenas uma questão. De forma a estender este mecanismo de raciocínio, definimos também sistema capaz de dar resposta a um conjunto de questões, apresentando um resultado baseado na conjunção e disjunção dos resultados individuais de cada questão. Para isto, temos de alguma forma conseguir incluir os operadores da disjunção ehttps://www.overleaf.com/project/5cb5d15cee3d2a184a280b10 conjunção, sendo que a solução encontrada foi definir que a lista de entrada fosse uma lista intercalada de questões e operadores, do tipo `[questão1,ou,questão2,e,...]`.

```
% Extensão do meta-predicado demoL: [Questao] ~> {V,F,D}
demoL( [], verdadeiro ).
demoL( [Q], R ) :- demo( Q, R ).
demoL( [Q, OP | T], R ) :- OP == e,
                           demo( Q, R1 ),
                           demoL( T, R2 ),
                           conjuncao( R1, R2, R ).
demoL( [Q, OP | T], R ) :- OP == ou,
                           demo( Q, R1 ),
                           demoL( T, R2 ),
                           disjuncao( R1, R2, R ).
```

De forma a conseguir usar este sistema de inferência, tivemos também de definir, para o novo conjunto de respostas possíveis, os diferentes resultados que a disjunção e conjunção podem ter. Estes foram definidos de acordo com a seguinte tabela:

p	**q**	**conjunção (p∧q)**	**disjunção (p∨q)**
verdadeiro	verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	falso	verdadeiro
verdadeiro	desconhecido	desconhecido	verdadeiro
falso	verdadeiro	falso	verdadeiro
falso	falso	falso	falso
falso	desconhecido	falso	desconhecido
desconhecido	verdadeiro	desconhecido	verdadeiro
desconhecido	falso	falso	desconhecido
desconhecido	desconhecido	desconhecido	desconhecido

4.3 Conhecimento Imperfeito

Na fase anterior, tínhamos definidos regras para a evolução do conhecimento perfeito, permitindo a inserção e remoção de elementos da base de conhecimento. Nesta fase, torna-se necessário possibilitar também a inserção de conhecimento imperfeito. Relativamente ao conhecimento imperfeito, podemos destacar três tipos: incerto, impreciso e interdito.

Para cada um dos tipos de conhecimento imperfeito, é necessário definir regras para a inserção deste tipo de informação na base de conhecimento.

4.3.1 Valores Nulos - Tipo Incerto

Valores nulos do tipo desconhecido são representados, no contexto de um programa em lógica estendido, como situações anómalas a identificar na definição dos casos que são considerados excepções à concretização da informação negativa. Nestes casos, existe um argumento de um predicado que se desconhece, e que não pertence especificamente a um conjunto determinado de valores. Então, este valor deve ser definido como uma exceção ao pressuposto do mundo fechado. Uma exceção para um valor nulo deste tipo segue o formato do exemplo que se segue:

```
% Exceção para um utente com cidade desconhecida:
excecao( utente( ID,N,I,C ) ) :- utente( ID,N,I,cidade_desconhecida ).
```

Como é possível verificar pelas definições do pressuposto do mundo fechado dos predicados utente/cuidado/prestador já expostas na secção anterior, as exceções passam a fazer parte do corpo da cláusula do mesmo.

4.3.2 Valores Nulos - Tipo Impreciso

Este tipo de valores nulos representa os valores de um conjunto de valores bem determinado. Ou seja, existe um dado conjunto de valores admissíveis para um dos atributos de uma instância de um predicado, e apenas se desconhece quais desses valores concretiza a questão.

Para melhor perceber este tipo de valores nulos, tomemos como exemplo o caso em que existe um utente, do qual se conhece apenas o ano de nascimento, 1980. Nesse caso, existem dois valores possíveis que a idade dessa utente pode tomar, 38 ou 39 anos, dependendo de o seu aniversário já ter ocorrido ou não. Assim sendo, não podemos afirmar com certeza que o utente tem qualquer uma das duas idades. O facto de ser possível este utente ter 38 anos é uma exceção à negação de que ele tenha essa idade, tal é o facto de poder ter 39 anos.

Para tratar este tipo de situações, definimos exceções que não permitam negar a informação à qual temos acesso. Neste caso em particular, definimos exceções que não permitam negar que o utente tem a idade de 38 ou 39 anos.

```
% Exceção para um utente com idade imprecisa:
excecao( utente( 17,'Ricardo Campos',38,'Rua do Raio - Braga' ) ).
excecao( utente( 17,'Ricardo Campos',39,'Rua do Raio - Braga' ) ).
```

4.3.3 Valores Nulos - Tipo Interdito

O terceiro tipo de valores nulos caracteriza um tipo de dados que, para além de serem desconhecidos, não se pretende que surjam na base de conhecimento.

Um valor representado como nulo, significa que nunca será possível especificá-lo ou conhecê-lo, e qualquer tentativa futura de concretizar esse valor é rejeitada, pois traria inconsistência à informação constante da base de conhecimento.

Para melhor perceber o conceito de valores desconhecidos do tipo interdito, analisemos um exemplo do trabalho prático. Suponhamos que não se sabe nem é possível saber a morada de um dado utente. O valor da morada deve ser definida numa variável como nula. No exemplo que se segue, por exemplo, a morada é definida como "desconhecido" e esse valor é declarado como nulo.

```
% Exceção para um utente com morada interdita:
utente( 20,'Alberto Joaquim',58,desconhecido ).
nulo( desconhecido ).
excecao( utente( ID,N,I,C ) ) :- utente( ID,N,I,desconhecido ).
```

Para além destas regras, temos também de garantir que não possa haver nenhuma inclusão de informação positiva ou negativa na base de conhecimento que viole a condição imposta pelo valor nulo não permitido. Para isso, basta garantir que nunca se verifica a existência de um utente com morada se essa morada tiver definida como valor nulo. Em programação lógica, podemos atingir este objetivo através de um invariante.

```
% Invariante que impede a inserção e remoção de conhecimento acerca de
conhecimento interdito:
+utente( ID,N,I,M ) :: (solucoes( (ID,N,I,M), (utente(ID,N,I,morada),
                        nulo( morada)),S), comprimento(S,N),N==0).
+(-utente( ID,N,I,M )) :: (solucoes( (ID,N,I,M), (utente(ID,N,I,morada),
                        nulo( morada)),S), comprimento(S,N),N==0).
```

4.4 Evolução de Conhecimento

Para estender o nosso programa em lógica, é agora necessário permitir a inserção de conhecimento imperfeito. Como já pudemos ver, este pode tomar três formas distintas. Nas secções que se seguem passamos a descrever o processo de inserção e remoção de cada uma dessas três formas de conhecimento na base de conhecimento.

4.4.1 Conhecimento Incerto

O conhecimento incerto trata-se de um tipo de conhecimento que, para além de desconhecido, pertence a um conjunto indeterminado de hipóteses. Vejamos o seguinte exemplo em que não se conhece a morada do utente.

```
evolucao(utente(Id, Nome, Idade, Morada), Type, Incerto) :-
    Type == incerto,
    Incerto == idade,
    evolucao(utente(Id, Nome, Idade, Morada), positivo),
    assert((excecao(utente(I, N, Idd, M)) :-
        utente(I, N, Idade, M))).

involucao(utente(Id, Nome, Idade, Morada), Type, Incerto) :-
    Type == incerto,
    Incerto == idade,
    involucao(utente(Id, Nome, Idade, Morada), positivo),
    assert((excecao(utente(I, N, Idd, M)) :-
        utente(I, N, Idade, M))).
```

4.4.2 Conhecimento Impreciso

O conhecimento impreciso refere-se a um tipo de conhecimento que, apesar de desconhecido, pertence a um conjunto determinado de hipóteses. A diferença entre este

e o conhecimento incerto dá-se no valor nulo, isto é, neste caso, o valor nulo representará um ou mais valores de um conjunto de valores bem determinado, embora não seja conhecimento com rigor qual o valor do conjunto considerado que concretizará a questão.

Para este, foram adicionados os seguintes predicados:

```
evolucao([OPT1 | R], Type) :-
    Type == impreciso,
    solucoes(I, +(excecao(OPT1))::I, L),
    teste(L),
    insercao((excecao(OPT1))),
    evolucao(R, impreciso).

evolucao([], impreciso).
```

Representa-se assim o conhecimento relativo a todas as possibilidades na lista.

Posto isto, vejamos, a título de exemplo, a seguinte inserção de conhecimento impreciso sobre a idade dos utentes:

```
% permitir inserir conhecimento impreciso sobre a idade dos utentes
evolucao(utente(Id, Nome, Idade, Morada), Type, Impreciso,
ValorInicio, ValorFim) :-
    Type == impreciso,
    Impreciso == idade,
    solucoes(I, +(excecao(utente(Id, Nome, Idade, Morada)))::I, L),
    teste(L),
    insercao((excecao(utente(Id, Nome, Idade, Morada)) :-
                Idade >= ValorInicio,
                Idade =< ValorFim)).
```

Como podemos ver, o predicado acima permite a evolução relativa a um utente cujo conhecimento acerca da sua idade é impreciso, encontrando-se entre dois valores. Aquando da realização deste trabalho, fez-se também um caso para a inserção de conhecimento impreciso sobre o custo de um cuidado médico, sendo que neste é o custo que se encontra entre dois valores.

Também para a remoção de conhecimento foi necessário adicionar os seguintes predicados:

```
involucao( [OPT1 | R], Type ) :-
    Type == impreciso,
    solucoes(I, -OPT1::I, Li),
    teste(Li),
    remocao((excecao(OPT1))),
    involucao(R, impreciso).

involucao( [], impreciso ).
```

Desta forma, é-nos permitido remover várias possibilidades numa lista ou todas entre dois valores, como é o caso da idade de um utente e do custo de um cuidado médico.

Analogamente, vejamos o exemplo que se refere à remoção de conhecimento impreciso sobre a idade dos utentes:

```
involucao(utente(Id, Nome, Idade, Morada), Type, Impreciso,
ValorInicio, ValorFim) :-
    Type == impreciso,
    Impreciso == idade,
    solucoes(I, -(excecao(utente(Id, Nome, Idade, Morada)))::I, L),
    teste(L),
    remocao((excecao(utente(Id, Nome, Idade, Morada)) :-
                Idade >= ValorInicio,
                Idade =< ValorFim)).
```

4.4.3 Conhecimento Interdito

Por fim, o conhecimento interdito refere-se a um tipo de conhecimento desconhecido e não permitido conhecer.

Como exemplo, mais uma vez utilizando o utente, vejamos como se procede à inserção e remoção de conhecimento no caso em que não se sabe qual a sua morada, nem é possível saber.

```
evolucao(utente(Id, Nome, Idade, Morada), Type, Interdito) :-
    Type == interdito,
    Interdito == morada,
    evolucao(utente(Id, Nome, Idade, Morada), positivo),
    assert((excecao(utente(Id, Nome, Idade, Morada)) :-
        utente(Id, Nome, Idade, Morada))),
        assert(nulo(Morada)).

involucao(utente(Id, Nome, Idade, Morada), Type, Interdito) :-
    Type == interdito,
    Interdito == morada,
    retract(nulo(Morada)),
    involucao(utente(Id, Nome, Idade, Morada), incerto, morada).
```

5 Resultados

```
5 ?- evolucao(cuidado(data(3,2,2019),3,3,'Ortopedia',x),impreciso,custo,10,15).
true.
```

Figura 1: Exemplo de predicado de evolução de conhecimento imperfeito (1)

```
7 ?- evolucao(cuidado(data(3,2,2019),3,3,'Ortopedia',x),interdito,custo).
true .
```

Figura 2: Exemplo de predicado de evolução de conhecimento imperfeito (2)

```
12 ?- evolucao(utente(40,'Jose Dias',30,'Amares'),negativo).
true .

13 ?- evolucao(utente(42,'Maria Vieira',28,'Braga'),positivo).
true .
```

Figura 3: Exemplos de predicado de evolução de conhecimento perfeito

```
17 ?- listing(utente).
:- dynamic utente/4.

utente(1, 'Ana Santos', 34, 'Rua São Gonçalo - S.Vicente - Braga').
utente(2, 'Bruno Mendonça', 23, 'Rua de Santa Catarina - Porto').
utente(3, 'Carla Martins', 29, 'Rua de Santa Maria - Brito - Guimarães').
utente(4, 'Beatriz Jesus', 12, 'Rua da Bica - São Paulo - Lisboa').
utente(5, 'Júlio Carvalho', 36, 'Rua Santa Margarida - S.Victor - Braga').
utente(6, 'Carlos Silva', 19, 'Rua das Flores - Sé e Vitória - Porto').
utente(7, 'Xavier Teixeira', 51, 'Rua dos Chãos - Souto - Braga').
utente(8, 'Luis Almeida', 32, 'Rua Padre Feliciano - Fraião - Braga').
utente(9, 'Pedro Lima', 20, 'Rua Prof.Machado Vilela - S.Victor - Braga').
utente(10, 'André Campos', 42, 'Rua da Graça - Graça -Lisboa').
utente(15, 'André Campos', 42, cidade_desconhecida).
utente(23, 'Carla Martins', idade_desconhecida, 'Rua de Santa Maria - Oliveira do Castelo - Guimarãe
s').
utente(20, 'Alberto Joaquim', 58, desconhecido).
utente(40, 'Jose Dias', 30, 'Amares').
utente(42, 'Maria Vieira', 28, 'Braga').

true.

18 ?- demo(utente(20, 'Alberto Joaquim', 58,'Braga'),desconhecido).
true.

19 ?- demo(utente(20, 'Alberto Joaquim', 58,'Braga'),verdadeiro).
false.

20 ?- demo(utente(20, 'Alberto Joaquim', 58,'Braga'),falso).
false.
```

Figura 4: Exemplos de resultados de diferentes queries

```
22 ?- demoL([utente(20, 'Alberto Joaquim', 58,'Braga'),e,utente(40, 'Jose Dias', 30, 'Amares')],falso).
false.

23 ?- demoL([utente(20, 'Alberto Joaquim', 58,'Braga'),e,utente(40, 'Jose Dias', 30, 'Amares')],verdadeiro).
false.

24 ?- demoL([utente(20, 'Alberto Joaquim', 58,'Braga'),ou,utente(40, 'Jose Dias', 30, 'Amares')],falso).
false.

25 ?- demoL([utente(20, 'Alberto Joaquim', 58,'Braga'),ou,utente(40, 'Jose Dias', 30, 'Amares')],desconhecido).
true .

26 ?- demoL([utente(20, 'Alberto Joaquim', 58,'Braga'),e,utente(40, 'Jose Dias', 30, 'Amares')],desconhecido).
true .
```

Figura 5: Exemplo de resultado para conjunções/disjunções de queries

6 Conclusões e Sugestões

Contrariamente ao que foi feito no exercício anterior, desta feita o principal foco do grupo foi, não apostar tanto na diversidade de predicados capazes de explorar o sistema em questão, mas garantir que representávamos o conhecimento de forma completa, exemplificando explicitamente, para os três predicados base do enunciado do problema, todos os tipos de conhecimento estudados.

Assim, consolidamos os conhecimentos previamente adquiridos aulas da unidade curricular e exploramos a nossa autonomia aquando da necessidade de resolver novos problemas. Em suma, estamos satisfeitos com o resultado final deste exercício, tendo implementado todas as funcionalidades propostas no enunciado.

Referências

[1] Analide, Cesar; Neves, José
Representação de Informação Incompleta.

[2] Bratko, Ivan
PROLOG: Programming for Artificial Intelligence.

[3] Analide, Cesar; Neves, José; Novais, Paulo
Sugestões para a Redacção de Relatórios Técnicos.

Anexos

A Predicados Auxiliares

Com o objetivo de simplificar a abordagem a tomar para responder às questões propostas, foram criados alguns predicados auxiliares.

```
% novo predicado evolucao: F, Type -> V,F
evolucao(T, Type) :-
    Type == positivo,
    solucoes(I, +T::I, L),
    teste(L),
    insercao(T).

evolucao(T, Type) :-
    Type == negativo,
    solucoes(I, +(-T)::I, L),
    teste(L),
    insercao(-T).

% novo predicado involucao: F, Type -> V,F
involucao(T, Type) :-
    Type == positivo,
    solucoes(I, -T::I, L),
    teste(L),
    remocao(T).

involucao(T, Type) :-
    Type == negativo,
    solucoes(I, -(-T)::I, L),
    teste(L),
    remocao(-T).
```