

UNIVERSIDADE DO MINHO

Programação em Lógica Estendida e Conhecimento Imperfeito

MESTRADO INTEGRADO EM ENGENHARIA
INFORMÁTICA

SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO

2º SEMESTRE 2019/20

Grupo:

Etienne Costa A76089

Mario Santos A70697

Pedro Costa 85700

Maurício Salgado A71407

Ricardo Lopes A72062

Docente:

Paulo Novais

25 de Abril de 2020

Conteúdo

1	Resumo	2
2	Introdução	3
3	Preliminares	4
3.1	Programação em lógica e PROLOG	4
4	Descrição do Trabalho e Análise dos Resultados	5
5	Base de conhecimento	5
5.1	Entidades	5
5.1.1	Adjudicantes	5
5.1.2	Adjudicatários	6
5.1.3	Contratos	6
5.1.4	Anúncios	6
5.1.5	Concorrentes	7
6	Representação de Conhecimento	8
6.1	Conhecimento positivo	8
6.2	Conhecimento negativo	8
7	Conhecimento Imperfeito	9
7.1	Incerto	9
7.2	Impreciso	10
7.3	Interdito	11
8	Integridade da Base de Conhecimento	11
8.1	Inserção de Conhecimento	12
8.2	Remoção de conhecimento	13
8.3	Conhecimento Incerto	13
8.4	Conhecimento Impreciso	15
8.5	Conhecimento Interdito	15
9	Sistema de Inferência	16
10	Conclusão	18
11	Referências Bibliográficas	19
12	Funções Auxiliares	20

1 Resumo

O presente trabalho tem como principal objetivo aprofundar os conhecimentos na linguagem de programação em lógica PROLOG.

Com base nisso foi desenvolvido o relatório explicando o desenvolvimento do primeiro exercício prático no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Este trabalho desenvolvido consiste na implementação de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da contratação pública tirando partido da programação em lógica estendida e à representação de conhecimento imperfeito, recorrendo a temática dos valores nulos.

2 Introdução

O relatório apresentado diz respeito ao primeiro exercício proposto no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, utilizando a linguagem de programação em lógica PROLOG. O universo de discurso para qual estamos a desenvolver este sistema é o da contratação pública, assim esta base de conhecimento consiste em adjudicantes, adjudicatários, contratos, anúncios e concorrentes.

3 Preliminares

Para o desenvolvimento deste projeto foi necessário alguns conhecimentos previamente adquiridos de programação em lógica, e a utilização da linguagem PROLOG. Este conhecimento foi absorvido durante as aulas de Sistemas de Representação de Conhecimento e Raciocínio, e também com alguma pesquisa nossa. De seguida, apresentamos alguns conceitos fundamentais para a compreensão e realização deste trabalho.

3.1 Programação em lógica e PROLOG

Uma das principais ideias da programação em lógica é de que um algoritmo é constituído por dois elementos disjuntos: a lógica e o controle. O componente lógico corresponde à definição do que deve ser solucionado, enquanto que o componente de controle estabelece como a solução pode ser obtida. O programador precisa somente descrever o componente lógico de um algoritmo, deixando o controle da execução para ser exercido pelo sistema de programação em lógica utilizado. Em outras palavras, a tarefa do programador passa a ser simplesmente a especificação do problema que deve ser solucionado, razão pela qual as linguagens lógicas podem ser vistas simultaneamente como linguagens para especificação formal e linguagens para a programação de computadores. O paradigma fundamental da programação em lógica é o da programação declarativa, em oposição à programação procedimental típica das linguagens convencionais. Um programa em lógica é então a representação de determinado problema ou situação expressa através de um conjunto finito de um tipo especial de sentenças lógicas denominadas cláusulas. Pode-se então expressar conhecimento (programas e/ou dados) em Prolog por meio de cláusulas de dois tipos: fatos e regras. Um fato denota uma verdade incondicional, enquanto que as regras definem as condições que devem ser satisfeitas para que uma certa declaração seja considerada verdadeira. Como fatos e regras podem ser utilizados conjuntamente, nenhum componente dedutivo adicional precisa ser utilizado. Além disso, como regras recursivas e não-determinismo são permitidos, os programadores podem obter descrições muito claras, concisas e não-redundantes da informação que desejam representar. Como não há distinção entre argumentos de entrada e de saída, qualquer combinação de argumentos pode ser empregada. Os termos "programação em lógica" e "programação Prolog" tendem a ser empregados indistintamente. Deve-se, entretanto, destacar que a linguagem Prolog é apenas uma particular abordagem da programação em lógica.

4 Descrição do Trabalho e Análise dos Resultados

De modo a complementar o sistema a ser desenvolvido foi implementado o conhecimento imperfeito e a lógica estendida, sendo assim foi introduzido um valor de verdade novo, i.e, o desconhecido. Para além disso, surge a possibilidade de representar conhecimento negativo, positivo e o imperfeito, sendo que este último pode ser incerto, impreciso ou mesmo interdito. Foram construídas certas regras da negação dos predicados de modo que seja possível obter resposta mesmo quando não existe factos negativos e para tal foi adotado o pressuposto do mundo fechado.

5 Base de conhecimento

Um programa em Prolog é um conjunto de axiomas e de regras de inferência definindo relações entre objectos que descrevem um dado problema. A este conjunto chama-se normalmente base de conhecimento.

A base de conhecimento do sistema desenvolvido é essencial à representação do conhecimento e raciocínio, tendo em conta o sistema foram desenvolvidas as seguintes entidades:

- adjudicante: $\text{IdAd}, \text{Nome}, \text{NIF}, \text{Morada} \rightarrow \{V, F, D\}$
- adjudicatária: $\text{IdAda}, \text{Nome}, \text{NIF}, \text{Morada} \rightarrow \{V, F, D\}$
- contrato: $\text{IdC}, \text{IdAd}, \text{IdAda}, [\text{IdAnuncio}], \text{Tipo de Contrato}, \text{Tipo de Procedimento}, \text{Descrição}, \text{Valor}, \text{Prazo}, \text{Local}, \text{Data} \rightarrow \{V, F, D\}$
- anúncio: $\text{IdAnuncio}, \text{IdAd}, \text{Nome}, \text{Descrição}, \text{Tipo de Contrato}, \text{Preço}, \text{Prazo}, \text{Data} \rightarrow \{V, F, D\}$
- concorrente: $\text{IdAnuncio}, [\text{IdsAda}] \rightarrow \{V, F, D\}$

5.1 Entidades

Nesta secção são apresentadas e caracterizadas as Entidades acima propostas.

5.1.1 Adjudicantes

Num mundo real o adjudicante é caracterizado por diversos factores, que por sua vez no contexto do Prolog esses factores passam a ser denominados por átomos cujo o seu propósito é identificar os objectos. Sendo assim para os adjudicantes optou-se por usar os seguintes átomos:

- IdAd: Identificador do adjudicante, que por sua vez é um valor único.
- Nome: Nome do adjudicante.
- NIF: Número de Identificação Fiscal.
- Morada: Espaço geográfico aonde se encontra o adjudicante.

5.1.2 Adjudicatários

Um adjudicatário por sua vez é caracterizado pelos seguintes átomos:

- IdAda: Identificador do adjudicatário, que por sua vez é um valor único.
- Nome: Nome do adjudicatário.
- NIF: Número de Identificação Fiscal.
- Morada: Espaço geográfico aonde se encontra o adjudicatário.

5.1.3 Contratos

Uma contrato será sempre referente a um adjudicante e um adjudicatário, sendo que esse contrato é celebrado numa determinada data associando o local de execução do mesmo, bem como o seu prazo de execução. Este contrato possui ainda uma breve descrição do serviço ou bem a adquirir, bem como o seu respectivo valor. No âmbito do nosso projecto ficou definido que existiriam 3 tipos de contratos e 3 tipos de procedimentos.

Tipos de Contratos	Tipo de Procedimentos
Aquisição de serviços	Ajuste direto
Locação de bens móveis	Consulta prévia
Contrato de aquisição	Concurso público

Portanto optou-se por representar o conhecimento do seguinte modo:

- IdC: Identificador do contrato, que por sua vez é uma valor único.
- IdAd: Identificador do adjudicante.
- IdAda: Identificador do adjudicatário.
- IdsAnuncio: Lista com o id do anúncio.
- Tipo de Contrato.
- Tipo de Procedimento.
- Descrição: Breve descrição do serviço.
- Valor: Custo associado ao serviço adquirido.
- Prazo: Dias para realização do contrato.
- Local: Espaço geográfico para execução do contrato.
- Data: Data da celebração do contrato.

5.1.4 Anúncios

O anúncio por sua vez surge devido a contratos que são celebrados por via de concursos públicos. Sendo assim foi caracterizado pelos seguintes átomos:

- IdAnuncio: Identificador do anúncio, que por sua vez é uma valor único.
- IdAd: Identificador do adjudicante.
- Nome: Nome da entidade responsável pelo anúncio .

- Descrição: Breve descrição do serviço
- Tipo de Contrato.
- Preço: Custo associado ao contrato.
- Prazo: Dias para realização do contrato.
- Data: Data da publicação do anúncio.

5.1.5 Concorrentes

Os concorrentes surgem no contexto de ser necessário saber que entidades adjudicatárias concorrem num determinado anúncio, para tal foir representado do seguinte modo:

- IdAnuncio: Identificador do anúncio, que por sua vez é uma valor único.
- IdsAda: Identificadores dos adjudicatários.

6 Representação de Conhecimento

Sendo que é necessário apresentar uma distinção entre o conhecimento positivo e negativo, surgiu a necessidade de implementar o operador '-'. Sendo que esse conhecimento negativo é representado por um negação forte, i.e, informa que um dado termo é falso.

Sendo assim eis à seguinte listagem das negações fortes implementadas :

```
-adjudicante (IdAd, Nome, NIF, Morada): -  
nao ( adjudicante (IdAd, Nome, NIF, Morada) ) ,  
nao ( excecao ( adjudicante (IdAd, Nome, NIF, Morada) ) ) .
```

```
-adjudicataria (IdAda, Nome, NIF, Morada): -  
nao ( adjudicataria (IdAda, Nome, NIF, Morada) ) ,  
nao ( excecao ( adjudicataria (IdAda, Nome, NIF, Morada) ) ) .
```

```
-contrato (IdC, IdAd, IdAda, IdsAnu, TC, TP, Des, Valor, Prazo, Local, Data): -  
nao ( contrato (IdC, IdAd, IdAda, IdsAnu, TC, TP, Des, Valor, Prazo, Local, Data) ) ,  
nao ( excecao ( contrato (IdC, IdAd, IdAda, IdsAnu, TC, TP, Des, Valor, Prazo, Local, Data) ) ) .
```

```
-anuncio (IdAnuncio, IdAd, Nome, Des, TC, Preco, Prazo, Data): -  
nao ( anuncio (IdAnuncio, IdAd, Nome, Des, TC, Preco, Prazo, Data) ) ,  
nao ( excecao ( anuncio (IdAnuncio, IdAd, Nome, Des, TC, Preco, Prazo, Data) ) ) .
```

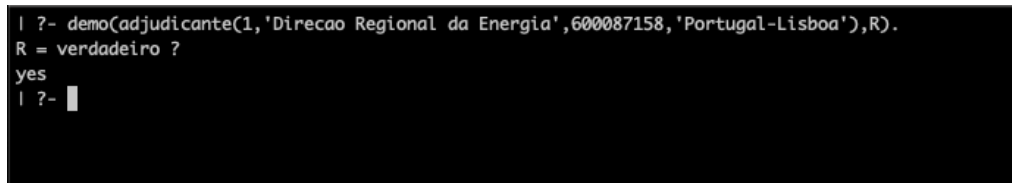
```
-concorrente (IdAnuncio, IdsAd): -  
nao ( concorrente (IdAnuncio, IdsAd) ) ,  
nao ( excecao ( concorrente (IdAnuncio, IdsAd) ) ) .
```

6.1 Conhecimento positivo

Quanto ao conhecimento positivo, este faz referência à informação existente na base de conhecimento. Podemos interpretar o seguinte exemplo:

- adjudicante(1, 'Direcao Regional da Energia', 600087158, 'Portugal-Lisboa').

É verdadeiro que a Direção Regional da Energia é uma entidade adjudicante com o identificador igual à 1 cujo número de identificação fiscal é 600087158 e a mesma é sediada em Portugal-Lisboa.



```
| ?- demo(adjudicante(1,'Direcao Regional da Energia',600087158,'Portugal-Lisboa'),R).  
R = verdadeiro ?  
yes  
| ?- █
```

Figura 1: Representação do conhecimento positivo.

6.2 Conhecimento negativo

Quanto ao conhecimento negativo, este faz referência à informação falsa existente na base de conhecimento. Estes factos são desenvolvidos com recurso a negação forte. Podemos interpretar o seguinte exemplo:

- -adjudicante(9, 'Municipio de Valongo', 501138960, 'Portugal-Porto').

É falso que o Município de Valongo seja uma entidade adjudicante com o identificador igual 9, que possua 501138960 como número de identificação fiscal e que esteja sediada em Portugal-Porto.

```
| ?- demo(adjudicante(9,'Municipio de Valongo',501138960,'Portugal-Porto'),R).
R = falso ?
yes
| ?- █
```

Figura 2: Representação do conhecimento negativo.

7 Conhecimento Imperfeito

Nesta secção pretende-se fazer uma breve explicação e representação do tipo de valores mais comuns que podem surgir numa situação de informação incompleta. Valores esses que são designados por valores nulos, sendo que os mesmos surgem como uma estratégia para a enumeração de caso, para os quais se pretende fazer a distinção entre situações em que as respostas a questões deverão ser concretizadas como verdadeiras, falsas ou desconhecidas.

Serão três os valores nulos aqui representados:

- Incerto: Desconhecido, de um conjunto indeterminado de hipóteses.
- Impreciso: Desconhecido, mas de um conjunto determinado de hipóteses.
- Interdito: Desconhecido e não permitido conhecer.

7.1 Incerto

Para descrever um adjudicante que tenha sido registado no IMPIC embora o seu número de identificação fiscal seja desconhecido adotou-se a seguinte representação:

```
adjudicante(11,'Municipio de Guimaraes',x001,'Portugal-Guimaraes').
excecao(adjudicante(Id,Nome,NIF,Morada)):-
    adjudicante(Id,Nome,x001,Morada).
```

Utilizou-se o átomo x001 para representar o valor incerto relativo ao adjudicante tirando partido da exceção do mesmo para representar o valor incerto.

Como resposta ao exemplo apresentado obtemos a seguinte resposta:

```
| ?- demo(adjudicante(11,'Municipio de Guimaraes',100321345,'Portugal-Guimaraes'),R).
R = desconhecido ?
yes
| ?- █
```

Figura 3: Conhecimento Incerto de um adjudicante.

Dado que não existe conhecimento perfeito em relação ao utente acima mencionado, obtemos como esperado à resposta "desconhecido".

7.2 Impreciso

Como o próprio nome indica, este tipo de conhecimento posiciona-se em situações em que sabe-se que a informação é desconhecida mas pertence a uma gama de hipóteses. Esta imprecisão pode ser dividida em dois grupos:

- Conjunto de hipóteses distinguíveis.
- Intervalo de valores.

Para o conjunto de hipóteses distinguíveis temos casos em que o conhecimento representado é impreciso face aos valores que um determinado facto apresenta. Por exemplo temos a entidade adjudicante com nome impreciso pois não se sabe ao certo se o nome é Freguesia de Caldas das Taipas ou Freguesia de Caldelas.

- `execcao(adjudicante(13,'Freguesia de Caldas das Taipas',507186265,'Portugal-Guimaraes'),R).`
- `execcao(adjudicante(13,'Freguesia de Caldelas',507186265,'Portugal-Guimaraes'),R).`

```
| ?- demo(adjudicante(13,'Freguesia de Caldas das Taipas',507186265,'Portugal-Guimaraes'),R).
R = desconhecido ?
yes
| ?- demo(adjudicante(13,'Freguesia de Caldelas',507186265,'Portugal-Guimaraes'),R).
R = desconhecido ?
yes
| ?- demo(adjudicante(13,'ISLAB',507186265,'Portugal-Guimaraes'),R).
R = falso ?
yes
| ?- █
```

Figura 4: Conhecimento Impreciso de um Adjudicante.

Quanto ao intervalo de valores o exemplo mais prático seria a representação de um intervalo de possíveis valores celebrados no contrato, i.e, não ser preciso ao ponto de afirmar que o contrato tem um custo de x unidades monetárias, mas sim que o seu valor esteja coomprendido num intervalo. Para tal temos o exemplo de um contrato celebrado entre a Ordem dos Engenheiros e a SDL Global Solutions LTD cujo o valor celebrado é impreciso visto que pertence ao intervalo [100,4000].

- `execcao(contrato(9,6,10,[],'Aquisicao de servicos', 'Ajuste direto', 'Prestacao de servicos de sistemas informaticos',Valor,90,'Portugal',(24,04,2020))),- Valor>=100, Valor <=4000.`

```
| ?- demo(contrato(9,6,10,[],'Aquisicao de servicos', 'Ajuste direto', 'Prestacao de servicos de sistemas informaticos',150,90,'Portugal',(24,04,2020)),R).
R = desconhecido ?
yes
| ?- demo(contrato(9,6,10,[],'Aquisicao de servicos', 'Ajuste direto', 'Prestacao de servicos de sistemas informaticos',5000,90,'Portugal',(24,04,2020)),R).
R = falso ?
yes
| ?- █
```

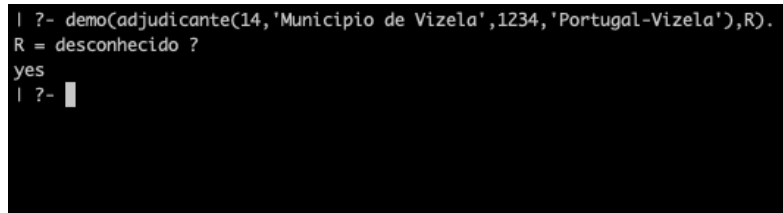
Figura 5: Conhecimento Impreciso de um Contrato.

7.3 Interdito

O conhecimento imperfeito do tipo interdito recai na situação em que o conhecimento relativo à uma entidade é desconhecido e permanecerá assim, não sendo possível saber o seu valor.

Como exemplo temos a entidade adjudicante com o nome de Município de Vizela que por questões de confidencialidade não se sabe o seu número de identificação fiscal, sendo que este conhecimento é definido a custa do predicado nulo.

- `nulo(np001).`
- `adjudicante(14,'Municipio de Vizela',np001,'Portugal-Vizela').`
- `excecao(adjudicante(Ida,Nome,NIF,Morada)):- adjudicante(Ida,Nome,np001,Morada).`
- `+adjudicante(IdAda,Nome,NIF,Morada)::`
`(findall(N,(adjudicante(14,'Municipio de Vizela',N,'Portugal-Vizela'), nao(nulo(N))),S),length(S,0)).`



```

| ?- demo(adjudicante(14,'Municipio de Vizela',1234,'Portugal-Vizela'),R).
R = desconhecido ?
yes
| ?-

```

Figura 6: Conhecimento Interdito de um Adjudicante.

8 Integridade da Base de Conhecimento

De forma a manter a integridade da base do conhecimento, e esta esteja de acordo com a realidade que pretendemos representar, é necessário implementar algum mecanismo que nos garanta isso. Assim, ao longo do trabalho fomos dando uso ao conceito de invariante. Estes permitem-nos controlar em específico a correta inserção e remoção na Base de Conhecimento. Os invariantes em PROLOG são representados da seguinte forma:

- `+Termo :: Premissas.`
- `-Termo :: Premissas.`

Em Prolog, já existem predicados que nos permitem inserir e remover factos da Base de Conhecimento. Estes são o `assert` e o `retract`, respetivamente. No entanto, a utilização destes predicados, exclusivamente, não garante consistência. Sendo assim, é necessário a criação de predicados auxiliares que nos garantam a integridade da Base de Conhecimento. Portanto para a realização da evolução deve ser garantido que não exista conhecimento positivo, negativo e imperfeito repetido sendo que este último é definido a custa de exceções.

Com base nisso os invariantes adotados tornaram-se muito semelhantes e obedecem o seguinte padrão:

```

+adjudicante (IdAd, __, __, __) ::
(integer (IdAd),
findall (IdAd, adjudicante (IdAd, __, __, __), R),
length (R, 1)).

```

```

+adjudicante(____,NIF,_)::
( findall(NIF, adjudicante(____,NIF,_) ,R) ,
length(R,1) ).

+adjudicante(IdAd, Nome, NIF, Morada)::
(( findall(Nome, adjudicante(____,Nome,____) ,R) ,
length(R,1) ) ).

+(-adjudicante(IdAd,____,____))::
(integer(IdAd) ,
findall(IdAd, -adjudicante(IdAd,____,____) ,R) ,
length(R,2) ).

-adjudicante(IdAd,____,____)::
( findall(IdAd, adjudicante(IdAd,____,____) ,R) ,
length(R,0) ).

-adjudicante(IdAd,____,____)::
( findall(IdAd, contrato(____,IdAd,____,____,____,____,____,____) ,R) ,
length(R,0) ).

```

Sendo que para os restantes basta uma alteração para o respectivo predicado e os respectivos átomos. Há situações que tivemos em conta para o sistema ser consistente, não podendo por sua vez evoluir de qualquer maneira, por exemplo:

- Não ser possível registar um contrato com um adjudicante ou adjudicatário que não exista na base de conhecimento.
- Não ser possível registar contratos com tipo de procedimentos ou tipo de contratos diferente do que foi previamente definido.
- Não ser possível registar contratos que não obedecem as condições obrigatórias do contrato por ajuste direto.
- Não ser possível registar contratos que não obedecem a regra dos 3 anos.

8.1 Inserção de Conhecimento

Como já havia sido dito anteriormente o uso exclusivo da função `assert` não garante a integridade, para tal surgiu um conjunto de condições que são verificadas de modo a preservar a consistência da base de conhecimento, e só assim inserir conhecimento. Este processo foi denominado por evolução. É Implementado do seguinte modo:

```

evolucao(T):- findall(I,+T::I, Li) ,
               insercao(T) ,
               teste(Li).

insercao(T):- assert(T).
insercao(T):- retract(T), !, fail.

```

Sendo que agora existe a necessidade de lidarmos com o conhecimento negativo surgiu a necessidade de criar uma função que lidasse com a evolução deste conhecimento, tendo sido ela implementada da seguinte maneira:

```

evolucaoNeg(T):- findall(I,+(~T)::I,Li),
                 insercao(~T),
                 teste(Li).

```

8.2 Remoção de conhecimento

Para a remoção do conhecimento o processo foi análogo ao de inserção, só que para este caso concreto o uso exclusivo do retract não garante a integridade, para tal surgiu um conjunto de condições de modo a preservar a consistência da base de conhecimento, e só assim remover conhecimento. Este processo foi denominado por involução. É implementado do seguinte modo:

```

involucao(T):- findall(I,~T::I,Li),
               remocao(T),
               teste(Li)..

```

```

remocao(T):- retract(T).
remocao(T):- assert(T), !, fail.

```

Sendo que agora existe a necessidade de lidarmos com o conhecimento negativo surgiu a necessidade de criar uma função que lidasse com a involução deste conhecimento, tendo sido ela implementada da seguinte maneira:

```

involucaoNeg(T):- findall(I,+(~T)::I,Li),
                  remocao(~T),
                  teste(Li).

```

8.3 Conhecimento Incerto

Para lidar com o conhecimento incerto optou-se por seguir dois processos distintos:

1º Inserir conhecimento incerto sem ser repetido, tendo sido tirado partido da seguinte função:

```

evolucaoNomeIncerto(adjudicante(IdAd, NomeIncerto, NIF, Morada)) :-
evolucao(adjudicante(IdAd, NomeIncerto, NIF, Morada)),
insercao(((excecao(adjudicante(I, No, Ni, M))) :- adjudicante(I, NomeIncerto, Ni, M))).

```

A imagem a seguir, retrata a tentativa de inserir um conhecimento incerto já existente e um conhecimento incerto novo, sendo o primeiro caso rejeitado face a existência do mesmo.

```

| ?- evolucaoNifIncerto(adjudicante(11,'Municipio de Guimaraes',x001,'Portugal-Guimaraes')).
no
| ?- evolucaoNomeIncerto(adjudicante(100,x99,12345,'Portugal')).
yes
| ?- demo(adjudicante(100,'Haskell',12345,'Portugal'),R).
R = desconhecido ?
yes
| ?- listing(adjudicante).
adjudicante(1, 'Direcao Regional da Energia', 600087158, 'Portugal-Lisboa').
adjudicante(2, 'Ordem dos Contabilistas Certificados', 503692310, 'Portugal-Lisboa').
adjudicante(3, 'Municipio de Vila Nova de Gaia', 505335018, 'Portugal-Porto').
adjudicante(4, 'Instituto Politecnico de Lisboa', 508519713, 'Portugal-Lisboa').
adjudicante(5, 'Policia Judiciaria', 600011712, 'Portugal-Lisboa').
adjudicante(6, 'Ordem dos Engenheiros', 500839166, 'Portugal').
adjudicante(7, 'Direcao Geral da Saude', 600037100, 'Portugal-Lisboa').
adjudicante(8, 'Universidade do Minho', 502011378, 'Portugal-Braga').
adjudicante(12, 'Universidade Aberta', 502110660, x002).
adjudicante(14, 'Municipio de Vizela', np001, 'Portugal-Vizela').
adjudicante(11, 'Municipio de Guimaraes', x001, 'Portugal-Guimaraes').
adjudicante(100, x99, 12345, 'Portugal').

```

Figura 7: Evolução do conhecimento Incerto.

2º Modificação de conhecimento incerto para conhecimento perfeito:

Para este caso em concreto optou-se por corrigir o conhecimento incerto partindo do princípio que o novo valor a ser inserido fosse o correto, e para a realização desta etapa criou-se por exemplo o seguinte predicado:

```

evolucaoNomePerfeito(adjudicante(IdAd, Nome, NIF, Morada)) :-
    involucaoNomeIncerto(adjudicante(IdAd, NomeIncerto, NIF, Morada)),
    evolucao(adjudicante(IdAd, Nome, NIF, Morada)).

```

A imagem a seguir, retrata a evolução de um conhecimento incerto para conhecimento perfeito.

```

| ?- evolucaoNomeIncerto(adjudicante(100,x99,12345,'Portugal')).
yes
| ?- demo(adjudicante(100,'Haskell',12345,'Portugal'),R).
R = desconhecido ?
yes
| ?- evolucaoNomePerfeito(adjudicante(100,'Haskell',12345,'Portugal')).
yes
| ?- demo(adjudicante(100,'Haskell',12345,'Portugal'),R).
R = verdadeiro ?
yes
| ?- listing(adjudicante).
adjudicante(1, 'Direcao Regional da Energia', 600087158, 'Portugal-Lisboa').
adjudicante(2, 'Ordem dos Contabilistas Certificados', 503692310, 'Portugal-Lisboa').
adjudicante(3, 'Municipio de Vila Nova de Gaia', 505335018, 'Portugal-Porto').
adjudicante(4, 'Instituto Politecnico de Lisboa', 508519713, 'Portugal-Lisboa').
adjudicante(5, 'Policia Judiciaria', 600011712, 'Portugal-Lisboa').
adjudicante(6, 'Ordem dos Engenheiros', 500839166, 'Portugal').
adjudicante(7, 'Direcao Geral da Saude', 600037100, 'Portugal-Lisboa').
adjudicante(8, 'Universidade do Minho', 502011378, 'Portugal-Braga').
adjudicante(11, 'Municipio de Guimaraes', x001, 'Portugal-Guimaraes').
adjudicante(12, 'Universidade Aberta', 502110660, x002).
adjudicante(14, 'Municipio de Vizela', np001, 'Portugal-Vizela').
adjudicante(100, 'Haskell', 12345, 'Portugal').

yes
| ?-

```

Figura 8: Evolução do conhecimento Incerto para Perfeito.

8.4 Conhecimento Impreciso

O nosso sistema permite também a inserção de conhecimento imperfeito impreciso, podendo contemplar duas situações distintas de inserção, sendo a 1ª fruto do seguinte predicado:

```
evolucaoImpreciso(T):- findall(I,+(excecao(T))::I, Lint),
                        insercao(excecao(T)),
                        teste(Lint).
```

Tal como abordado em Conhecimento Imperfeito Impreciso, para adicionar conhecimento impreciso é somente necessário adicionar exceções para cada uma dessas imprecisões e é necessariamente isso que é feito no predicado acima. Sendo esse predicado genérico para qualquer termo.

A 2ª segunda situação consiste na evolução de um predicado cujo o seu valor seja impreciso,i.e, é desconhecido embora seja de um conjunto determinado de valores. Para tal foi recorrido ao seguinte predicado:

```
evolucaoPrecoImpreciso(anuncio(IdAnuncio,IdAd,Nome,Descricao,TipoContrato,PrecoImpreciso,Prazo,Data),LimiteInf,LimiteSup):-
    insercao((excecao(anuncio(IdAnuncio,IdAd,Nome,Descricao,TipoContrato,PrecoImpreciso,Prazo,Data)):-
        PrecoImpreciso >=LimiteInf , PrecoImpreciso <= LimiteSup)).
```

Figura 9: Evolução do conhecimento Impreciso.

A imagem a seguir demonstra a utilização do código apresentado e os resultados esperados.

```
| ?- demo(anuncio(4,7,'Direcao Geral da Saude','Prestacao de servicos de seguranca','Aquisicao de servicos',25,60,(01,01,2020)),R).
R = falso ?
yes
| ?- evolucaoPrecoImpreciso(anuncio(4,7,'Direcao Geral da Saude','Prestacao de servicos de seguranca','Aquisicao de servicos',Preco,60,(01,01,2020)),10,50).
yes
| ?- demo(anuncio(4,7,'Direcao Geral da Saude','Prestacao de servicos de seguranca','Aquisicao de servicos',25,60,(01,01,2020)),R).
R = desconhecido ?
yes
| ?- evolucaoPrecoImpreciso(anuncio(4,7,'Direcao Geral da Saude','Prestacao de servicos de seguranca','Aquisicao de servicos',Preco,60,(01,01,2020)),10,50).
yes
| ?- demo(anuncio(4,7,'Direcao Geral da Saude','Prestacao de servicos de seguranca','Aquisicao de servicos',25,60,(01,01,2020)),R).
R = falso ?
yes
| ?- █
```

Figura 10: Evolução do conhecimento Impreciso.

8.5 Conhecimento Interdito

Relativamente ao conhecimento interdito , este só deve ser inserido e nunca modificado sendo que para isso bastou inserir a respectiva exceção bem como o seu invariante.


```

%----- Inserções -----%
%                           %
%      CONHECIMENTO IMPERFEITO INTERDITO      %
%-----%

evolucaoNomeInterdito(adjudicataria(IdAda, NomeInterdito, NIF, Morada)) :-
    evolucao(adjudicataria(IdAda, NomeInterdito, NIF, Morada)),
    insercao((execucao(adjudicataria(I, No, Ni, M)) :-
        adjudicataria(I, NomeInterdito, Ni, M))),
    insercao((nulo(NomeInterdito))),
    insercao(+adjudicataria(ID, NO, NI, MO) :: {findall(Nome, (adjudicataria(IdAda, Nome, NIF, Morada), nao(nulo(Nome)))), S),
        length(S, 0))).

```

Figura 11: Evolução do conhecimento Interdito.

```

| ?- demo(adjudicataria(100, 'Francis', 12345, 'Paris'), R).
R = falso ?
yes
| ?- evolucaoNomeInterdito(adjudicataria(100, np100, 12345, 'Paris')).
yes
| ?- listing(adjudicataria).
adjudicataria(1, 'EVCE Power, Lda', 514385472, 'Portugal').
adjudicataria(2, 'Junior Salgado', 232830185, 'Portugal').
adjudicataria(3, 'Dilicontas, Lda', 504906232, 'Portugal').
adjudicataria(4, 'Etienne Costa', 700000003, 'Franca').
adjudicataria(5, 'X LIGHT, LDA', 515325180, 'Portugal').
adjudicataria(6, 'Pedro Costa', 700000004, 'Portugal').
adjudicataria(7, 'Securitas', 500243719, 'Portugal').
adjudicataria(8, 'Grafica 99. Lda', 503956759, 'Portugal').
adjudicataria(9, 'PETROGAL S.A', 500697370, 'Portugal').
adjudicataria(10, 'SDL Global Solutions LTD', 48733689, 'Países Baixos').
adjudicataria(11, 'BBZ - Publicidade e Marketing, SA', 503453838, 'Espanha').
adjudicataria(14, 'Pichelaria Chaves, Lda', 500125512, x003).
adjudicataria(15, 'Serralharia Martins, Lda', 520426412, x004).
adjudicataria(16, 'Energias Renovaveis, Lda', 551464123, np002).
adjudicataria(100, np100, 12345, 'Paris').

yes
| ?- demo(adjudicataria(100, 'Francis', 12345, 'Paris'), R).
R = desconhecido ?
yes
| ?- evolucao(adjudicataria(100, 'Francis', 12345, 'Paris')).
no
| ?- involucaoNomeInterdito(adjudicataria(100, np100, 12345, 'Paris')).
yes
| ?- demo(adjudicataria(100, 'Francis', 12345, 'Paris'), R).
R = falso ?
yes
| ?-

```

Figura 12: Evolução do conhecimento Interdito.

9 Sistema de Inferência

Foi desenvolvido um sistema que permite implementar mecanismos de inferência sobre três tipos de valores nulos, que concretizam situações de informação incompleta. Este sistema suporta três respostas possíveis:

- verdadeiro.
- falso.

- desconhecido.

Para tal foi desenvolvido o predicado `demo` que por sua vez recebe dois argumentos, sendo um deles a Questão a ser colocada e o segundo a Resposta da questão colocada.

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

Sendo que esse predicado apresenta limitações no que concerne a quantidade de questões decidiu-se abranger as conjunções e disjunções ao nosso demonstrador e por fim gerar uma espécie de map do demonstrador podendo assim operar sobre um conjunto de questões.

De seguida são representadas todas as combinações possíveis das conjunções e disjunções bem como as cláusulas que as representam:

Disjunção:

```
demo(Q1, ou, Q2, F) :- demo(Q1, F1) ,
                        demo(Q2, F2) ,
                        disjuncao(F1, F2, F) .
```

Questão1	Questão2	Resposta
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Verdadeiro	Desconhecido	Verdadeiro
Falso	Falso	Falso
Falso	Verdadeiro	Verdadeiro
Falso	Desconhecido	Desconhecido
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Verdadeiro
Desconhecido	Falso	Desconhecido

Conjunção:

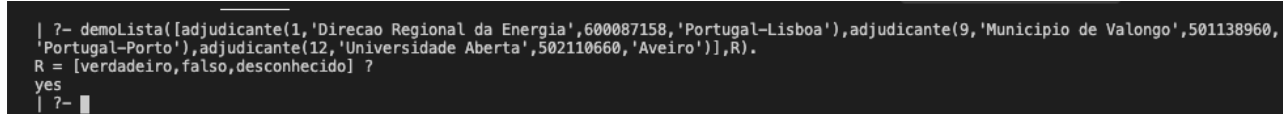
```
demo(Q1, e, Q2, F) :- demo(Q1, F1) ,
                       demo(Q2, F2) ,
                       conjuncao(F1, F2, F) .
```

Questão1	Questão2	Resposta
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Verdadeiro	Desconhecido	Desconhecido
Falso	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Desconhecido	Falso
Desconhecido	Desconhecido	Desconhecido
Desconhecido	Verdadeiro	Desconhecido
Desconhecido	Falso	Falso

DemoLista:

O demoMap recebe dois argumentos, uma lista de questões e uma lista de respostas para as questões passadas como parâmetro, foi implementado de modo a ser possível serem respondidas várias questões em simultâneo.

```
demoLista ([], []).
demoLista ([H|T], [X|Xs]): - demo(H,X),
                             demoLista(T,Xs).
```



```
| ?- demoLista([adjudicante(1,'Direcao Regional da Energia',600087158,'Portugal-Lisboa'),adjudicante(9,'Municipio de Valongo',501138960,
'Portugal-Porto'),adjudicante(12,'Universidade Aberta',502110660,'Aveiro')]),R).
R = [verdadeiro,falso,desconhecido] ?
yes
| ?- █
```

Figura 13: Map Demo.

Extras

Como extra optou-se por definir sistemas de inferência que fossem capazes de inferir o valor lógico da conjunção ou disjunção para uma lista de Questões, obtendo como resposta verdadeiro,falso ou desconhecido.

```
demoConjuncao ([],R).
demoConjuncao ([H],R): - demo(H,R).
demoConjuncao ([H|T],R): - demo(H,R1),
                             demoConjuncao(T,R2),
                             conjuncao(R1,R2,R).
```

```
demoDisjuncao ([],R).
demoDisjuncao ([H],R): - demo(H,R).
demoDisjuncao ([H|T],R): - demo(H,R1),
                             demoDisjuncao(T,R2),
                             disjuncao(R1,R2,R).
```

10 Conclusão

A realização deste trabalho prático permitiu consolidar o conhecimento adquirido ao longo das aulas, no que concerne à programação em lógica estendida e conhecimento imperfeito. Como tal, o suporte utilizado para caracterizar um universo de discurso na área da contratação pública foi o PROLOG. Em forma de conclusão, o grupo considera que conseguiu corresponder as expectativas e consolidar os conceitos relativos ao conhecimento imperfeito e às variantes que este apresenta. Sendo que uma das principais dificuldades encontradas no desenvolvimento deste sistema passou pela forma como era feita a evolução de conhecimentos face aos diferentes tipos de conhecimento, sendo assim no que concerne a melhorias, passa por num futuro próximo inserir outros factos e regras de modo a aproximar o trabalho de um contexto mais real.

11 Referências Bibliográficas

Referências

- [1] Cesar Analide, Jose Neves. *"Representação de Informação Incompleta"*.
- [2] Ivan Bratko. *"PROLOG: Programming for Artificial Intelligence"*.

12 Funções Auxiliares

```
%-----
disjuncao(verdadeiro , X, verdadeiro).
disjuncao(X, verdadeiro , verdadeiro).
disjuncao(desconhecido , Y, desconhecido) :- Y \= verdadeiro.
disjuncao(Y, desconhecido , desconhecido) :- Y \= verdadeiro.
disjuncao(falso , falso , falso).

%-----
- - -

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

%-----
- - -

if p then q else r.

x:-p,! ,q.
x:-r.

%-----
- - -

zip([],[],[]).
zip([X|XS],[Y|YS],[ (X,Y)|Z]) :- zip(XS,YS,Z).

%-----
- - -

concatenar([],L2,L2).
concatenar(L1,[],L1).
concatenar([Head|Tail],X,[Head|R]):-concatenar(Tail,X,R).

%-----
- - -

somatorio([],0).
somatorio([H|T],R):-somatorio(T,R1),
                    R is H + R1.

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

```

data(D, M, A) :-
    A >= 0,
    pertence(M, [1, 3, 5, 7, 8, 10, 12]),
    D >= 1,
    D <= 31.
data(D, M, A) :-
    A >= 0,
    pertence(M, [4, 6, 9, 11]),
    D >= 1,
    D <= 30.
data(D, 2, A) :-
    A >= 0,
    A mod 4 =\= 0,
    D >= 1,
    D <= 28.
data(D, 2, A) :-
    A >= 0,
    A mod 4 =:= 0,
    D >= 1,
    D <= 29.

validaData((D,M,A)) :- data(D,M,A).

%-----
- - -
teste([]).
teste([I|Is]):-I, teste(Is).

```