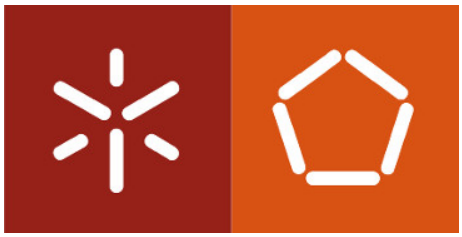
Sistemas de Representação de Conhecimento e Raciocínio

3 de Maio de 2020

Grupo nr.	15
a83899	André Morais
a84577	José Pedro Silva
a85954	Luís Ribeiro
a84783	Pedro Rodrigues



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1 Introdução

1.1 Objetivos

Com a realização deste exercício pretendemos utilizar a extensão à programação em lógica, usando a linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento imperfeito, recorrendo à utilização de valores nulos e da criação de mecanismos de raciocínio adequados.

1.2 Programação em Lógica e Prolog

Uma linguagem de programação lógica utiliza a lógica para representar conhecimento e inferências para manipular informações. Como tal, um programa deste tipo de programação possui os seguintes parâmetros:

- **Factos** - algo que se conhece como e se sabe ser verdadeiro (ex: cor(preto)).
- **Predicados** - implementações de relações, por exemplo, pai(pai,filho) é a implementação da relação de descendência (ser pai de).
- **Regras** - as regras PROLOG são descrições de predicados por meio de condicionais.

2 Descrição do Trabalho

2.1 Descrição

Pretende-se que seja desenvolvido um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da contratação pública para a realização contratos para a prestação de serviços.

Para o efeito, considere-se que o panorama poderá ser caracterizado por conhecimento, por exemplo, dado na forma que se segue:

- **adjudicante**: #IdAd, Nome, NIF, Morada $\rightsquigarrow \{\mathbb{V}, \mathbb{F}, \mathbb{D}\}$
- **adjudicatária**: #IdAda, Nome, NIF, Morada $\rightsquigarrow \{\mathbb{V}, \mathbb{F}, \mathbb{D}\}$
- **contrato**: #IdC, #IdAd, #IdAda, TipoDeContrato, TipoDeProcedimento, Descrição, Valor, Prazo, Local, Data $\rightsquigarrow \{\mathbb{V}, \mathbb{F}, \mathbb{D}\}$

Sendo assim, os contratos públicos devem seguir as seguintes regras:

- **Tipo de Procedimento**
  - Ajuste Direto;
  - Consulta prévia;
  - Concurso público.
- **Condições obrigatórias do contrato por ajuste direto**
  - Valor igual ou inferior a 5000 euros;
  - Contrato de aquisição ou locação de bens móveis ou aquisição de serviços;
  - Prazo de vigência até 1 ano, inclusive, a contar da decisão de adjudicação.

- **Regra dos 3 anos válida para todos os contratos**
  - Uma entidade adjudicante não pode convidar a mesma empresa para celebrar um contrato com prestaçõesde serviçodo mesmo tipo ou idênticas às de contratos que já lhe foram atribuídos, no ano económico em curso e nos dois anos económicos anteriores, sempre que:
    - O preço contratual acumulado dos contratos já celebrados (não incluindo o contrato que se pretende celebrar) seja igual ou superior a 75.000 euros

2.2 Adjudicante

Foi necessário criar uma base de conhecimento com adjudicantes já existentes, sendo assim, foi utilizado método **adjudicante(IdAd, Nome, NIF, Morada**:

```
adjudicante(1,"Pedro Rodrigues",12345,"Barcelos").
adjudicante(2,"Ribeiro",9876,"Braga").
adjudicante(3,"Bruno",89976,"Taipas").
adjudicante(5,"Mário",243647596,"Nazaré").
adjudicante(6,"Nanda",275436742,"Taipas").
adjudicante(7,"Tiago",210437961,"Cascais").
adjudicante(8,"Davide",224160310,"Lisboa").
adjudicante(9,"Nuno",243764597,"Fão").
adjudicante(10,"Bruno",214657342,"Viseu").
```

2.3 Adjudicatária

Foi necessário criar uma base de conhecimento com adjudicatárias já existentes, sendo assim, foi utilizado método **adjudicatária(IdAda, Nome, NIF, Morada**:

```
adjudicataria(1,"André Morais",98765,"Esposende").
adjudicataria(2,"Luís Ribeiro",45454,"Braga").
adjudicataria(3,"José Silva",913213,"Guimarães").
adjudicataria(5,"Fatrix Dipetas",214256289,"Taipas").
adjudicataria(6,"Ana Moura",220213200,"Barcelos").
adjudicataria(7,"Tiago Malavita",240123123,"Coimbra").
adjudicataria(8,"João Carlos",263236210,"Viana do Castelo").
adjudicataria(9,"Dinis Teleti",278259256,"Guimarães").
adjudicataria(10,"Alfredo Vski",222986643,"Porto").
```

2.4 Contratos

Foi necessário criar uma base de conhecimento com contratos já existentes, sendo assim, foi utilizado método **contrato(IdC, IdAd, IdAda, TipoDeContrato, TipoDeProcedimento, Descrição, Valor, Prazo, Local, Data)**. Para melhor compreensão e verificação, foi criado o predicado data.

contrato(1, 1, 1, "Aquisição de Serviços", "Concurso Público", "Descrição", 70000, 365, "Braga", data(10, 02, 2020)).

contrato(2, 1, 2, "Aquisição de Serviços", "Ajuste Direto", "Descrição", 300, 100, "Braga", data(10, 05, 2020)).

contrato(3, 2, 3, "Contrato de Aquisição", "Consulta Prévia", "Descrição", 70000, 90, "Braga", data(15, 05, 2021)).

contrato(4, 2, 1, "Locação de Bens Móveis", "Ajuste Direto", "Descrição", 500, 90, "Braga", data(10, 11, 2022)).

contrato(5, 1, 3, "Aquisição de Serviços", "Consulta Prévia", "Descrição", 70000, 90, "Braga", data(24, 09, 2022)).

3 Adição de conhecimento

Para que fosse possível a inserção e remoção de conhecimento na base, desenvolvemos os predicados **evolução** e **involução**. Estes predicados obrigam a que a inserção ou remoção cumpra certas regras definidas pelos invariantes.

```
evolucao(T) :- solucoes(Inv, +T::Inv, S),
               insercao(T),
               teste(S).
```

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

```
involucao(T) :- T,
                solucoes(Inv, -T::Inv, S),
                remocao(T),
                teste(S).
```

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

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

Como podemos ver no código apresentado, os predicados evolução e involução dependem do predicado teste, que testa se o elemento a inserir ou remover cumpre com todos os invariantes.

3.1 Invariantes

Como concluímos acima, é necessária a implementação de certos invariantes para definir regras de adição e remoção de conhecimento à base. Então, para cada predicado (adjudicante, adjudicatária e contrato) foram criados invariantes de inserção e remoção.

Adjudicante:

Um adjudicante não pode ser inserido se o seu id e nif não forem números naturais e caso não sejam únicos na base de conhecimento.

```
+adjudicante(Id, _, NIF, _) :: (
                integer(Id),
                integer(NIF),
                Id >= 0,
                NIF > 0,
                solucoes(Id, adjudicante(Id, _, _, _), S),
                comprimento(S, N), N == 1,
                solucoes(NIF, adjudicante(_, _, NIF, _), NS),
                comprimento(NS, C), C == 1
            ).
```

No caso da remoção, um adjudicante não pode ser removido se tiver algum contrato associado a ele mesmo.

```
-adjudicante(IdAd, _, _ , _) :: (
     solucoes(IdAd, contrato(_ , IdAd, _, _, _, _, _, _ ,_ , _),S),
     comprimento(S, N), N == 0
).
```

Adjudicatária:

Uma adjudicatária não pode ser inserida se o seu id e nif não forem números naturais e caso não sejam únicos na base de conhecimento.

```
+adjudicataria(Id, _, NIF, _) :: (
              integer(Id),
              integer(NIF),
              Id >= 0,
              NIF > 0,
              solucoes(Id, adjudicataria(Id, _, _, _), S),
              comprimento(S, N), N == 1,
              solucoes(NIF, adjudicataria(_, _, NIF, _), NS),
              comprimento(NS, C), C == 1
        ).
```

No caso da remoção, uma adjudicatároa não pode ser removida se tiver algum contrato associado à mesma.

```
-adjudicataria(IdAda, _, _ , _) :: (
    solucoes(IdAda, contrato(_ , _, IdAda, _, _, _, _, _ ,_ , _),S),
    comprimento(S, N), N == 0
       ).
```

Contrato:

Um contrato não pode ser inserido se o id do mesmo, do adjudicante e da adjudicatária não forem números naturais e únicos, o id do adjudicante e adjudicatária devem existir e o restante do contrato deve seguir as regras descritas acima.

```
+contrato(IdC, IdAd, IdAda, TC, TP, _, C, P, _, Data) :: (
          integer(IdC), IdC >= 0,
          integer(IdAd), IdAd >= 0,
          integer(IdAda), IdAda >= 0,
          solucoes(IdC,contrato(IdC,_,_,_,_,_,_,_,_,_),LC),
          comprimento(LC, NC), NC == 1,
          solucoes(IdAd,adjudicante(IdAd,_,_,_),LAd),
          comprimento(LAd, NAd), NAd == 1,
          solucoes(IdAda,adjudicataria(IdAda,_,_,_),LAda),
          comprimento(LAda, NAda), NAda == 1,
          verificaTipo(TP,TC,P,C),
```

```
        verificaData(Data),
        ano(Data,A),
        regraAnos(IdAd,IdAda,TC,C,A)
).
```

Neste invariante são utilizados predicados criados com intuito auxiliar. O predicado verificaTipo, tal como o nome indica, verifica o tipo do contrato, de forma a que o contrato cumpra as regras descritas acima (**Tipo de Procedimento** e **Condições obrigatórias do contrato por ajuste direto**), o predicado verificaData verifica se a data é válida e o predicado regraAnos cumpre a regra **Regra dos 3 anos válida para todos os contratos**

4 Conhecimento Imperfeito

4.1 Definição

Este tipo de conhecimento depende das possíveis respostas a uma questão, mais concretamente no caso de a mesma ser desconhecida. Logo, existem 3 tipos de conhecimento imperfeito:

- **Incerto** - representa valores nulos que não pertencem a nenhum conjunto determinado de valores, sendo portanto completamente desconhecidos.
- **Impreciso** - representa valores nulos que pertencem a um conjunto determinado de valores, i.e, existe uma noção do que é falso mas não se conhece qual a verdadeira resposta.
- **Interdito** - representa valores nulos desconhecidos e que não serão permitidos conhecer.

Assim, foi desenvolvido, com base na lógica clássica um interpretador de questões. Este interpretador de questões trata-se de um predicado que se baseia nos diferentes tipos de conhecimento imperfeito:

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

4.2 Implementação dos diferentes tipos de conhecimento imperfeito

4.2.1 Incerto

Para a declaração de conhecimento incerto, primeiramente declaramos p(x) e, de seguida, é necessário declarar a exceção de p(x). Então, foram já criadas as declarações dos diferentes tipos de exceção possíveis, por exemplo:

Adjudicante - Nome Desconhecido

```
excecao(adjudicante(Id,Nome,NIF,Morada)) :-
                adjudicante(Id,nomeDesconhecido,NIF,Morada).
```

Adjudicante - Morada Desconhecida

```
excecao(adjudicante(Id,Nome,NIF,Morada)) :-
                adjudicante(Id,Nome,NIF,moradaDesconhecida).
```

Adjudicatária - Nome e Morada Desconhecidos

```
excecao(adjudicataria(Id,Nome,NIF,Morada)) :-
            adjudicataria(Id,nomeDesconhecido,NIF,moradaDesconhecida).
```

Contrato - Valor Desconhecido

```
excecao(contrato(Id,IdAd,IdAda,TC,TP,Desc,V,P,L,D)) :-
            contrato(Id,IdAd,IdAda,TC,TP,Desc,valorDesconhecido,P,L,D).
```

Contrato - Local Desconhecido

```
excecao(contrato(Id,IdAd,IdAda,TC,TP,Desc,V,P,L,D)) :-
            contrato(Id,IdAd,IdAda,TC,TP,Desc,V,P,localDesconhecido,D).
```

Contrato - Data Desconhecida

```
excecao(contrato(Id,IdAd,IdAda,TC,TP,Desc,V,P,L,D)) :-
            contrato(Id,IdAd,IdAda,TC,TP,Desc,V,P,L,dataDesconhecida).
```

Assim, com estas excecções já definidas (mais exceções no programa), para representar o conhecimento imperfeito, é apenas necessário representar p(x), caso excecao(p(x)), esteja definida no programa:

Exemplo 1

Um contrato de aquisição, com o procedimento de concurso público, contendo a seguinte descrição: "Descrição", com id 6, foi realizado entre o adjudicante 3 e a adjudicatária 2, em Braga, no dia 17/07/2020, com um valor desconhecido e um prazo de um ano.

```
contrato(6,3,2,"Contrato de Aquisição",
        "Concurso Público","Descrição",
        valorDesconhecido,365,"Braga",data(17,07,2020)).
```

Exemplo 2

Um contrato de aquisição de serviços, com o procedimento de ajuste direto, contendo a seguinte descrição: "Descrição", com id 7, foi realizado entre o adjudicante 3 e a adjudicatária 2, em Viseu, numa data desconhecida, com um valor de 20 unidades monetárias e um prazo de 10 dias.

```
contrato(7,3,2,"Aquisição de Serviços",
        "Ajuste Direto","Descrição",20,10,
        "Viseu",dataDesconhecida).
```

4.2.2 Impreciso

Mais uma vez, é tirado partido do predicado excecao.

Exemplo 1

Um contrato de aquisição de serviços, com o procedimento de ajuste direto, contendo a seguinte descrição: "Descrição", com id 9, foi realizado entre o adjudicante 1 e a adjudicatária 3, em Barcelos, no dia 9/10/2020, com um valor de 40 unidades monetárias e um prazo entre 20 a 30 dias.

```
excecao(contrato(9,1,3,"Aquisição de Serviços",
        "Ajuste Direto","Descricao",40,P,
        "Barcelos",data(9,10,2020))) :- P>=20, P=<30.
```

Exemplo 2

Um contrato de locação de bens móveis, com o procedimento de concurso público, contendo a seguinte descrição: "Descrição", com id 10, foi realizado entre o adjudicante 2 e a adjudicatária 1, em Lisboa, no dia 9/10/2020, com um prazo de um ano e um valor que não se sabe se é 7000 ou 70000 unidades monetárias.

```
excecao(contrato(10,2,1,"Locação de Bens Móveis",
        "Concurso Público","Descricao",7000,365,
        "Lisboa",data(12,12,2020))).
excecao(contrato(10,2,1,"Locação de Bens Móveis",
        "Concurso Público","Descricao",70000,365,
        "Lisboa",data(12,12,2020))).
```

Exemplo 3

Um contrato de locação de bens móveis, com o procedimento de consulta prévia, contendo a seguinte descrição: "Descrição", com id 8, foi realizado entre o adjudicante 2 e a adjudicatária 3, no dia 9/10/2020, com um prazo de 20 dias e um valor de 400 unidades monetárias. Quanto ao local, apenas sabe-se que não foi em Braga.

```
contrato(8,2,3,"Locação de Bens Móveis",
        "Consulta Prévia","Descrição",400,20,
        localDesconhecido,data(15,08,2021)).

-contrato(8,2,3,"Locação de Bens Móveis",
        "Consulta Prévia","Descrição",400,20,
        "Braga",data(15,08,2021)).
```

4.2.3 Interdito

Como sabemos, conhecimento interdito é o que não se sabe qualquer tipo de informação e está destinado a nunca se saber, por isso é necessário garantir que este nunca evolui. Para proibir a expansão desse conhecimento é utilizado o predicado nulo em conjunto com a sua utilização em invariantes.

Exemplo

Um adjudicante nomeado de João Silva, regista-se com o id 4, NIF 775873 mas ninguém pode saber/é impossível saber a sua morada.

```
adjudicante(4,"João Silva",775873,moradaInterdita).
excecao(adjudicante(Id,N,NIF,M)) :- adjudicante(Id,N,NIF,moradaInterdita).
nulo(moradaInterdita).
+adjudicante(Id,NA,NIF,Morada) :: (
            solucoes((Id,NA,NIF,Morada),(adjudicante(4,"João Silva",
            775873,M),nao(nulo(M))),S),
            comprimento(S,N),
            N==0
        ).
```

4.3 Demonstração do Conhecimento Imperfeito

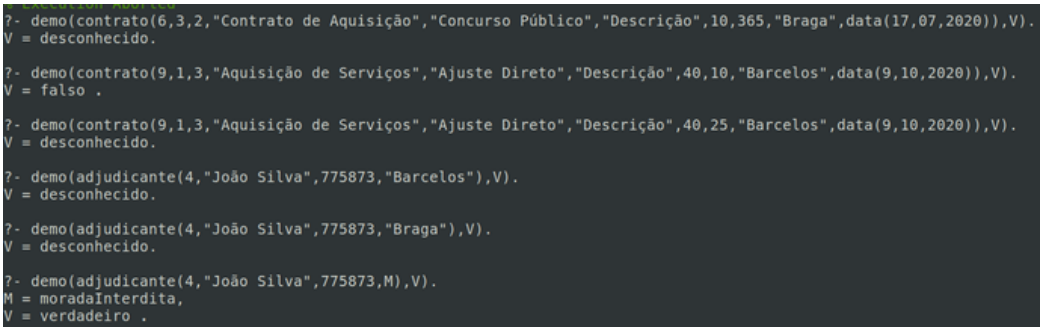


Figura 1: Demonstração do conhecimento imperfeito representado pelo exemplos acima

5 Funcionalidades Extras do Sistema

Nesta secção apresentam-se todas as funcionalidades extras presentes no sistema.

5.1 Inserção

- **registaAdjudicante(Id,Nome,NIF,Morada)** - adiciona adjudicante à base de conhecimento através do método evolução.
- **registaAdjudicataria(Id,Nome,NIF,Morada)** - adiciona adjudicataria à base de conhecimento através do método evolução.
- **realizaContrato(Id,IdAd,IdAda,TC,TP,D,C,P,L,Data)** - adiciona contrato à base de conhecimento através do método evolução.

5.2 Procura

- **adjudicantes(S)** - fornece a lista de todos os adjudicantes da base de conhecimento
- **adjudicatarias(S)** - fornece a lista de todas as adjudicatarias da base de conhecimento
- **contratos(S)** - fornece a lista de todos os contratos da base de conhecimento
- **contratosAdjudicante(IdAd,S)** - fornece a lista de todos os contratos da base de conhecimento do adjudicante com id = IdAd
- **contratosAdjudicatarias(IdAda,S)** - fornece a lista de todos os contratos da base de conhecimento da adjudicataria com id = IdAda
- **getIdAdjudicante(NIF,Id)** - fornece o Id do adjudicante com o nif = NIF
- **getIdAdjudicataria(NIF,Id)** - fornece o Id do adjudicante com o nif = NIF
- **contratosNIFAdjudicante(NIF,S)** - fornece a lista de todos os contratos do adjudicante com nif = NIF

- **contratosNIFAdjudicataria(NIF,S)** - fornece lista de todos os contratos da adjudicataria com nif = NIF
- **contratosLocal(L,S)** - fornece lista de todos os contratos feitos no local L
- **contratosAno(A,S)** - fornece lista de todos os contratos feitos no ano A
- **custoTotalAdjudicante(NIF,N)** - fornece o custo total dos contratos feitos pelo adjudicante com nif = NIF
- **custoTotalAdjudicanteAno(NIF,A,N)** - forncece o custo total dos contratos feitos pelo adjudicante com nif = NIF no ano A
- **ganhoTotalAdjudicataria(NIF,N)** - fornece ganho total dos contratos feitos pela adjudicataria com nif = NIF
- **ganhoTotalAdjudicatariaAno(NIF,A,N)** - fornece ganho total dos contratos feitos pela adjudicataria com nif = NIF no ano A
- **getAdjudicante(Id,Nome,NIF,Morada)** - devolve as restantes informações do adjudicante, ao ser apresentado o Id do mesmo.
- **getAdjudicataria(Id,Nome,NIF,Morada)** - devolve as restantes informações da adjudicataria, ao ser apresentado o Id da mesma.
- **getContrato(Id,IdAd,IdAda,TC,TP,D,C,P,L,Data)** - devolve as restantes informações do contrato, ao ser apresentado o Id do mesmo.

6 Conclusão

Depois de terminado o trabalho prático da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio, concluímos que a realização deste permitiu-nos consolidar os conhecimentos estudados nas aulas da disciplina. Houve assim, uma aprendizagem notória relativamente à representação do conhecimento perfeito e conhecimento imperfeito.

Estado o trabalho como terminado, o grupo reconhece que foi desenvolvido com sucesso um sistema de caraterização de um universo na área dos Contratos Públicos. Este sistema inclui diversos tipos de conhecimento como adjudicantes, adjudicatários, contratos, entre outros.

Foram ainda implementadas funções que permitem a manipulação da base de conhecimento, quer a adição (evolução) e remoção (involução) de conhecimento e a aplicação de uma série de queries sobre ela.

Relativamente ao Conhecimento Imperfeito, abordamos os 3 tipos (Incerto, Impreciso e Interdito) através de exemplos que transmitem possíveis realidades de imperfeição na realização de um contrato.

A boa preparação do grupo e o estudo prévio, quer do motor de inferência quer do caso de estudo, possibilitaram que todo o processo decorresse dentro da normalidade, sem que tenham surgido problemas de dimensão considerável

7 Predicados Auxiliares

Extensão do meta-predicado **nao**

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

Extensão do meta-predicado **comprimento** através do predicado predefinido **length**

```
comprimento(S, N) :- length(S, N).
```

Extensão do meta-predicado **sum**, que soma os elementos da lista

```
sum([], 0).
sum([H|T], N):-
    sum(T, X),
    N is X + H.
```

Extensão do meta-predicado **solucoes** através do predicado predefinido **findall**

```
solucoes(T,Q,S) :- findall(T,Q,S).
```

Extensao do meta-predicado **verificaTipo**
verificaTipo("Consulta Prévia",_,_,_).
verificaTipo("Concurso Público",_,_,_).
verificaTipo("Ajuste Direto","Contrato de Aquisição",P,C) :- P =< 365, C =< 5000.
verificaTipo("Ajuste Direto","Locação de Bens Móveis",P,C) :- P =< 365, C =< 5000.
verificaTipo("Ajuste Direto","Aquisição de Serviços",P,C) :- P =< 365, C =< 5000.

Extensão do meta-predicado **regraAnos**

regraAnos(IdAd, IdAda, TC, C, A) :-
solucoes(V,(contrato(_,IdAd,IdAda,TC,_,_,V,_,_,data(_,_,AS)),A-AS < 3),S),
sum(S,N), nao(N-C >= 75000).

Extensão do meta-predicado **verificaData**

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