Programação em Lógica Estendida e Conhecimento Imperfeito

Sistemas de Representação de Conhecimento e Raciocínio

Grupo 2

Ivo Baixo (A86579), Luís Ramos (A83930)
Luís Vila (A84439), Ricardo Cunha (A84302)

Abril 2020

Universidade do Minho
Mestrado Integrado em Engenharia Informática

Resumo

Neste exercício prático foi desenvolvido um sistema de representação de conhecimento e raciocínio referente ao universo de discurso da contratação pública, sendo o sistema capaz de representar conhecimento imperfeito.

Conteúdo

1 Introdução

No âmbito da unidade curricular **Sistemas de Representação de Conhecimento e Raciocínio**, resolveu-se o proposto exercício, cujo tema assenta na *Programação lógica extendida e representação de conhecimento imperfeito*. Este panorama permite ao sistema uma maior flexibilidade ao representar conhecimento (uma vez que é possível representar conhecimento desconhecido), aproximando-o mais do mundo real.

2 Preliminares

3 Descrição do trabalho e análise de resultados

Tal como foi referido anteriormente, a temática deste trabalho assenta na Programação Lógica Extendida e na representação de conhecimento Imperfeito. O trabalho desenvolvido teve como principais focus as seguintes questões:

- Representar conhecimento positivo e negativo.
- Representar conhecimento imperfeito através dos três tipos de valores nulos (incerto, impreciso e interdito).
- Criar mecanismos capazes de lidar com a evolução e involução do conhecimento.
- Criar invariantes que restringem a inserção/remoção de conhecimento no sistema.
- Criar um sistema de inferência capaz de implementar mecanismos de raciocínio adequados.

3.1 Base de conhecimento

Estando no universo de discurso de contratação pública, o conhecimento é fornecido do seguinte modo:

- **adjudicante**: IdAd, Nome, NIF, Morada –> {V,F}.
- **adjudicatária**: IdAda, Nome, NIF, Morada –> {V,F}.
- **contrato**: IdC, IdAd, IdAda, Tipo de contrato, Tipo de procedimento, Descrição, Custo, Prazo, Local, Data –> {V,F}.

3.1.1 Adjudicante

Um adjudicante é caracterizado por um *id* e *NIF* que lhe são únicos, assim como um *nome* e uma *morada*.

3.1.2 Adjudicataria

Uma adjudicataria é caracterizada por um *id* e *NIF* que lhe são únicos, assim como um *nome* e uma *morada*.

3.1.3 Contrato

Um contrato é caracterizado por um *id* único, os *ids* do adjudicante e adjudicataria envolvidos, um *tipo de contratro*, um *tipo de procedimento*, um *custo*, uma *descrição*, um *prazo*, um *local* e uma *data*.

É importante salientar que de forma a melhorar a nossa base de conhecimento decidimos utilizar o *Pressuposto do Mundo Fechado* para o predicado *adjudicante* e para o predicado *adjudicataria*, assumindo assim que um adjudicante ou uma adjudicataria que não estejam definidos na nossa base de conhecimento se traduz numa informação **falsa**. Com isto, podemos representar com exatidão conhecimento *imperfeito*, sendo que nos é possível saber se, por exemplo, existe um adjudicante com um *NIF* compreendido num dado intervalo. Para o predicado *contrato*, assumimos o *Pressuposto do Mundo Aberto*, garantindo assim que apenas são consideradas falsas as informações explicitamente negativas na base de conhecimento.

3.2 Conhecimento perfeito

Utilizamos conhecimento perfeito para representar conhecimento que é **verdadeiro** ou **falso**.

3.2.1 Representação de conhecimento positivo

Para que posteriormente seja possível verificar os predicados elaborados, é necessário primeiro adicionar conhecimento positivo referente a cada uma das fontes de conhecimento.

```
adjudicante(1, 'Municipio de Basto', 702675112, 'Portugal, Cabeceiras de Basto').
adjudicante(2, 'Municipio de Braga', 712675112, 'Portugal, Braga').
adjudicante(3, 'Municipio de Guimaraes', 722675112, 'Portugal, Guimaraes').
adjudicante(4, 'Municipio de Paris', 732675112, 'França, Paris').
adjudicante(5, 'Municipio de Rio', 742675112, 'Brasil, Rio de Janeiro').
adjudicante(6, 'Municipio de Lisboa', 752675112, 'Portugal, Lisboa').
adjudicante(7, 'Municipio de Madrid', 762675112, 'Espanha, Madrid').
adjudicante(8, 'Municipio de Barcelona', 772675112, 'Espanha, Barcelona').
```

Figura 1: Conhecimento positivo de adjudicantes

```
adjudicataria(2, 'Sociedad de los Abogados, SP,RL', 712675112, 'Espanha').
adjudicataria(3, 'Societa il Avvocati, SP,RL', 732675112, 'Italia').
adjudicataria(4, 'Sociedade advocacia, SP,RL', 742675112, 'França').
adjudicataria(5, 'Lawyer Society, SP,RL', 752675112, 'Inglaterra').
adjudicataria(6, 'Sociedade Outros Advogados, SP,RL', 762675112, 'Portugal').
adjudicataria(7, 'Sociedade de los otros abogados, SP,RL', 772675112, 'Espanha').
adjudicataria(8, 'Sociedade Advogados alemaes, SP,RL', 782675112, 'Alemanha').
```

Figura 2: Conhecimento positivo de adjudicatarias

```
contrato(1, 1, 1, 'Aquisicao de servicos', 'Consulta previa', 'Assessoria juridica', 15000, 600, 'Cabeceiras de Basto', '11-02-2020').
contrato(2, 2, 2, 'Aquisicao de servicos', 'Consulta previa', 'Assessoria juridica', 15000, 600, 'Braga', '10-02-2020').
contrato(3, 3, 3, 'Aquisicao de servicos', 'Consulta previa', 'Assessoria juridica', 15000, 600, 'Guimaraes', '12-02-2020').
contrato(4, 4, 4, 'Aquisicao de servicos', 'Consulta previa', 'Assessoria juridica', 15000, 600, 'Paris', '11-02-2020').
contrato(5, 5, 5, 'Aquisicao de servicos', 'Consulta previa', 'Assessoria juridica', 15000, 600, 'Rio de Janeiro', '11-12-2020').
contrato(6, 6, 6, 'Aquisicao de servicos', 'Consulta previa', 'Assessoria juridica', 15000, 600, 'Lisboa', '11-05-2020').
contrato(7, 7, 7, 'Aquisicao de servicos', 'Consulta previa', 'Assessoria juridica', 15000, 600, 'Madrid', '24-08-2020').
contrato(8, 8, 8, 'Aquisicao de servicos', 'Consulta previa', 'Assessoria juridica', 15000, 600, 'Barcelona', '12-03-2020').
contrato(9, 1, 6, 'Aquisicao de servicos', 'Consulta previa', 'Assessoria juridica', 15000, 600, 'Cabeceiras de Basto', '07-09-2020').
```

Figura 3: Conhecimento positivo de contratos

3.2.2 Representação de conhecimento negativo

É também necessário representar conhecimento negativo, para além do positivo. Em programação lógica extendida, este conhecimento pode ser de dois tipos diferentes: *Negação por falha na prova* ou *Negação forte*. Assim, foram definidos os seguintes conjuntos de informação:

```
-adjudicante( 9, 'Municipio Alegre', 702675102, 'Portugal, Vinhais' ).
-adjudicante( 10, 'Municipio Triste', 702675113, 'Portugal, Beja' ).
```

Figura 4: Conhecimento negativo de adjudicantes

Através deste conhecimento pode-se concluir que: *É **falso** que o indivíduo com o IdAd=9 do Município Alegra residente em 'Portugal, Vinhais' e cujo NIF=702675102 é um adjudicante.*

```
-adjudicataria( 9, 'Sociedade society, SP,RL', 712685112, 'Bulgaria' ).
-adjudicataria( 10, 'Advocacia de advogados, SP,RL', 712675110, 'Espanha' ).
```

Figura 5: Conhecimento negativo de adjudicatarias

```
-contrato( 10, 1, 2, 'Aquisicao de serviços', 'Consulta prévia', 'Assessoria juridica', 10200, 500, 'Guimaraes', '07-07-2020' ).
-contrato( 11, 2, 3, 'Aquisicao de serviços', 'Consulta prévia', 'Assessoria juridica', 20100, 400, 'Vizela', '05-05-2020' ).
```

Figura 6: Conhecimento negativo de contratos

3.3 Conhecimento Imperfeito

Em *Programação Lógica Extendida* a informação pode ser verdadeira, falsa ou desconhecida. Assim, para representar a informação **desconhecida** utilizamos o *conhecimento imperfeito* que é divisível em três grupos.

3.3.1 Representação de conhecimento incerto

Este tipo de conhecimento diz respeito a conhecimento desconhecido, de um conjunto ilimitado de hipóteses.

Por exemplo, com o uso de *conhecimento incerto* é possível representar um adjudicante cuja morada seja desconhecida (e toda a outra informação conhecida), criando um adjudicante com *'incerto'* no campo da morada e criando também um exceção que retorne **desconhecido** quando se questionar a base de conhecimento em relação àquele adjudicante.

Para todos os predicados, foram então criadas exceções para representar conhecimento incerto em qualquer um dos argumentos do predicado.

```
adjudicante( 11, 'Municipio Triste', 702675113, incerto ).
excecao( adjudicante( IdAd, Nome, NIF, _ ) ) :- adjudicante( IdAd, Nome, NIF, incerto ).
```

Figura 7: Exemplo de conhecimento incerto

3.3.2 Representação de conhecimento impreciso

Este tipo de conhecimento diz respeito a conhecimento desconhecido, de um conjunto limitado de hipóteses.

Usando *conhecimento impreciso* é possível representar, por exemplo, uma adjudicataria cujo NIF esteja compreendido entre 732675112 e 732675312, embora não se sabendo exatamente qual é o NIF. Pode-se então representar este cenário através do uso de uma exceção, como se pode observar de seguida.

```
excecao(adjudicataria(id, a, B, c)) :- B>=732675112, B=<732675312.
```

Figura 8: Exemplo de conhecimento impreciso

3.3.3 Representação de conhecimento interdito

O conhecimento interdito diz respeito a conhecimento desconhecido mas que não é possível de conhecer.

Por exemplo, com o uso de *conhecimento interdito* é possível representar um adjudicante cuja morada seja impossível de saber. Para representar este tipo de conhecimento utiliza-se um processo semelhante ao utilizado para representar *conhecimento incerto* adicionado apenas uma nova instância de **nulo**. Esta instância torna possível a construção de um *invariante* que impeça a *evolução* deste tipo de conhecimento.

```
adjudicante( 13, 'Municipio Vitória', 123456789, morada_imp ).
excecao( adjudicante( IdAd, Nome, NIF, _ ) ) :- adjudicante( IdAd, Nome, NIF, morada_imp ).
nulo( morada_imp ).
+adjudicante(IdAd, Nome, NIF, _ ) :: (solucoes((IdAd, Nome, NIF, _ ),
    (adjudicante(13, coisa, 123456789, morada_imp ),nao(nulo(morada_imp))),R),comprimento(R,0)).
```

Figura 9: Exemplo de conhecimento interdito

3.4 Adição e remoção de conhecimento

Para permitir a adição e remoção de informação na nossa base de conhecimento, foram criados dois meta-predicados (**evolução** e **involução**), assim como, procedimentos próprios para *evoluír/involuír* cada um dos tipos de conhecimento previamente referidos.

3.4.1 Evolução do Sistema

O meta-predicado *evolução* é o responsável por inserir novo conhecimento na nossa base de conhecimento, verificando sempre se esta adição respeita todos os invariantes estabelecidos.

- **Conhecimento positivo e negativo**

```
% EVOLUCAO CONHECIMENTO POSITIVO
evolucao( Termo , p) :-
    solucoes( Invariante,+Termo::Invariante,Lista ),
    insercao( Termo ),
    teste( Lista ).

% EVOLUCAO CONHECIMENTO NEGATIVO
evolucao( Termo , n) :-
    solucoes( Invariante,+(-Termo)::Invariante,Lista ),
    insercao( -Termo ),
    teste( Lista ).
```

Figura 10: Evolução de conhecimento positivo e negativo

Para *evoluír* um termo escolhemos assim **n** caso este seja conhecimento negativo e **p** caso contrário.

Figura 11: Exemplo de evolução de conhecimento positivo e negativo

- **Conhecimento incerto**

  Para este tipo de conhecimento, não basta adicionar o termo respetivo, é também preciso criar uma nova excessão. Assim, como temos três predicados distintos (cada um com vários campos que podem ser incertos), foi preciso criar um predicado de *evolução* para cada um deles. Os argumentos deste predicado de evolução são assim o termo a adicionare o parâmetro que é desconhecido.

```
%ADJUDICANTE
%Evolucao de adjudicante com nome desconhecido
evolucao(adjudicante(Id,Nome,Nif,Morada),nome_desconhecido) :-
    evolucao(adjudicante(Id,Nome,Nif,Morada),p),
    insercao((excecao(adjudicante(I,_,Ni,M)):-adjudicante(I,Nome,Ni,M))).
```

Figura 12: Evolução de conhecimento incerto (ajudicante com nome desconhecido)

```
| ?- listing(adjudicante).
adjudicante(1, 'Municipio de Basto', 702675112, 'Portugal, Cabeceiras de Basto'
).
adjudicante(2, 'Municipio de Braga', 712675112, 'Portugal, Braga').
adjudicante(3, 'Municipio de Guimaraes', 722675112, 'Portugal, Guimaraes').
adjudicante(4, 'Municipio de Paris', 732675112, 'FranÃSa, Paris').
adjudicante(5, 'Municipio de Rio', 742675112, 'Brasil, Rio de Janeiro').
adjudicante(6, 'Municipio de Lisboa', 752675112, 'Portugal, Lisboa').
adjudicante(7, 'Municipio de Madrid', 762675112, 'Espanha, Madrid').
adjudicante(8, 'Municipio de Barcelona', 772675112, 'Espanha, Barcelona').
adjudicante(11, incerto, 6, d).
adjudicante(13, coisa, 123456789, morada_imp).

yes
| ?- evolucao(adjudicante(14,desconhecido,703675112,'China'),nome_desconhecido)
.
yes
| ?- listing(adjudicante).
adjudicante(1, 'Municipio de Basto', 702675112, 'Portugal, Cabeceiras de Basto'
).
adjudicante(2, 'Municipio de Braga', 712675112, 'Portugal, Braga').
adjudicante(3, 'Municipio de Guimaraes', 722675112, 'Portugal, Guimaraes').
adjudicante(4, 'Municipio de Paris', 732675112, 'FranÃSa, Paris').
adjudicante(5, 'Municipio de Rio', 742675112, 'Brasil, Rio de Janeiro').
adjudicante(6, 'Municipio de Lisboa', 752675112, 'Portugal, Lisboa').
adjudicante(7, 'Municipio de Madrid', 762675112, 'Espanha, Madrid').
adjudicante(8, 'Municipio de Barcelona', 772675112, 'Espanha, Barcelona').
adjudicante(11, incerto, 6, d).
adjudicante(13, coisa, 123456789, morada_imp).
adjudicante(14, desconhecido, 703675112, 'China').

yes
| ?-
```

Figura 13: Exemplo de evolução de adjudicante com nome incerto

- **Conhecimento impreciso**

  Para a adição de conhecimento impreciso começamos por criar um predicado que adiciona exceções.

```
% EVOLUCAO CONHECIMENTO IMPRECISO
%VALOR IMPRECISO
evolucao( Termo , i) :-
    solucoes( Invariante,+(excecao(Termo))::Invariante,Lista ),
    insercao( excecao(Termo) ),
    teste( Lista ).
```

Figura 14: Evolução de conhecimento impreciso

Para que seja possível adicionar informação desconhecida entre um intervalo de valores, construímos ainda outros predicados de evolução que além de receberem o termo em questão, recebem a *tag* **i** e os limites inferior e superior, como se pode observar na seguinte figura:

```
%ADJUDICANTE (nif impreciso)
evolucao( adjudicante(Id,Nome,Nif,Morada) , i, Inf, Sup) :-
    insercao((excecao(adjudicante(Id,Nome,Nif,Morada)) :- Nif >= Inf,Nif =< Sup)).
```

Figura 15: Evolução de conhecimento impreciso (sabe-se que o NIF do ajudicante está num certo intervalo)

```
excecao(adjudicante(A,B,C,_)) :-
        adjudicante(A, B, C, morada_imp).
excecao(adjudicataria(A,B,_,C)) :-
        adjudicataria(A, B, nif_imp, C).
excecao(contrato(A,B,C,D,E,F,G,H,_,I)) :-
        contrato(A, B, C, D, E, F, G, H, local_imp, I).
excecao(adjudicante(A,_,B,C)) :-
        adjudicante(A, desconhecido, B, C).

yes
| ?- evolucao(adjudicante(21,'Sintra', A,'Portugal'),i,700111000,700111222).
yes
| ?-
```

```
excecao(adjudicante(A,B,C,_)) :-
        adjudicante(A, B, C, morada_imp).
excecao(adjudicataria(A,B,_,C)) :-
        adjudicataria(A, B, nif_imp, C).
excecao(contrato(A,B,C,D,E,F,G,H,_,I)) :-
        contrato(A, B, C, D, E, F, G, H, local_imp, I).
excecao(adjudicante(A,_,B,C)) :-
        adjudicante(A, desconhecido, B, C).
excecao(adjudicante(21,'Sintra',A,'Portugal')) :-
        A>=700111000,
        A=<700111222.

yes
| ?-
```

```
excecao(adjudicante(A,B,C,_)) :-
        adjudicante(A, B, C, morada_imp).
excecao(adjudicataria(A,B,_,C)) :-
        adjudicataria(A, B, nif_imp, C).
excecao(contrato(A,B,C,D,E,F,G,H,_,I)) :-
        contrato(A, B, C, D, E, F, G, H, local_imp, I).
excecao(adjudicante(A,B,_,C)) :-
        adjudicante(A, B, desconhecido, C).
```

Figura 16: Exemplo de evolução de conhecimento impreciso (Nif varia dentro de um intervalo)

- **Conhecimento interdito**

  Para este tipo de conhecimento, desenvolveu-se um predicado semelhante ao do tipo *incerto*, acrescentando o parâmetro impossível de se saber ao predicado **nulo**. Assim, para cada predicado e para cada um dos parâmetros que possam ser interditos, foi necessário criar um predicado específico. Segue-se um exemplo do predicado desenvolvido para a evolução de um adjudicante cujo nome é impossível de conhecer.

```
%Evolucao de adjudicante com nome impossivel
evolucao(adjudicante(Id,Nome,Nif,Morada),nome_impossivel) :-
    evolucao(adjudicante(Id,Nome,Nif,Morada),p),
    insercao((excecao(adjudicante(I,_,Ni,M)):-adjudicante(I,Nome,Ni,M))),
    insercao((nulo(Nome))),
    insercao(( +adjudicante(I,_,Ni,M) :: (solucoes((I, _, Ni, M ),
    (adjudicante(Id, Nome, Nif, Morada ),nao(nulo(Nome))),R),comprimento(R,0)) )).
```

Figura 17: Exemplo de evolução de conhecimento interdito

3.4.2 Involução do Sistema

O meta-predicado *involução* é o responsável por retirar conhecimento da nossa base de conhecimento, verificando sempre se esta subtração respeita todos os invariantes estabelecidos.

- **Conhecimento positivo e negativo**

```
% INVOLUCAO CONHECIMENTO POSITIVO (funciona)
involucao( Termo , p) :-
    solucoes( Invariante,-Termo::Invariante,Lista ),
    teste( Lista ),
    remocao( Termo ).


% INVOLUCAO CONHECIMENTO NEGATIVO (funciona)
involucao( Termo , n) :-
    solucoes( Invariante,-(-Termo)::Invariante,Lista ),
    remocao( -Termo ),
    teste( Lista ).
```

Figura 18: Involucao de conhecimento positivo e negativo

Para *involuír* um termo escolhemos assim **n** caso este seja conhecimento negativo e **p** caso contrário.

```
| ?- listing(adjudicante).
adjudicante(1, 'Municipio de Basto', 702675112, 'Portugal, Cabeceiras de Basto').
adjudicante(2, 'Municipio de Braga', 712675112, 'Portugal, Braga').
adjudicante(3, 'Municipio de Guimaraes', 722675112, 'Portugal, Guimaraes').
adjudicante(4, 'Municipio de Paris', 732675112, 'FranÃ§a, Paris').
adjudicante(5, 'Municipio de Rio', 742675112, 'Brasil, Rio de Janeiro').
adjudicante(6, 'Municipio de Lisboa', 752675112, 'Portugal, Lisboa').
adjudicante(7, 'Municipio de Madrid', 762675112, 'Espanha, Madrid').
adjudicante(8, 'Municipio de Barcelona', 772675112, 'Espanha, Barcelona').
adjudicante(9, a, 9, b).
adjudicante(11, incerto, 6, d).
adjudicante(13, coisa, 123456789, morada_imp).
adjudicante(14, 'Shangai', 772675113, 'China').

yes
| ?- involucao(adjudicante(14,'Shangai', 772675113, 'China'),p).
yes
| ?- listing(adjudicante).
adjudicante(1, 'Municipio de Basto', 702675112, 'Portugal, Cabeceiras de Basto').
adjudicante(2, 'Municipio de Braga', 712675112, 'Portugal, Braga').
adjudicante(3, 'Municipio de Guimaraes', 722675112, 'Portugal, Guimaraes').
adjudicante(4, 'Municipio de Paris', 732675112, 'FranÃ§a, Paris').
adjudicante(5, 'Municipio de Rio', 742675112, 'Brasil, Rio de Janeiro').
adjudicante(6, 'Municipio de Lisboa', 752675112, 'Portugal, Lisboa').
adjudicante(7, 'Municipio de Madrid', 762675112, 'Espanha, Madrid').
adjudicante(8, 'Municipio de Barcelona', 772675112, 'Espanha, Barcelona').
adjudicante(9, a, 9, b).
adjudicante(11, incerto, 6, d).
adjudicante(13, coisa, 123456789, morada_imp).

yes
| ?-
```

Figura 19: Exemplo de involução de conhecimento positivo

```
| ?- listing(-).
-adjudicante(9,'Municipio Alegre',702675102,'Portugal, Vinhais').
-adjudicante(10,'Municipio Triste',702675113,'Portugal, Beja').
-adjudicataria(9,'Sociedade society, SP,RL',712685112,'Bulgaria').
-adjudicataria(10,'Advocacia de advogados, SP,RL',712675110,'Espanha').
-contrato(10,1,2,'Aquisicao de serviÃ§os','Consulta prÃ©via','Assessoria juridica'
,10200,500,'Guimaraes','07-07-2020').
-contrato(11,2,3,'Aquisicao de serviÃ§os','Consulta prÃ©via','Assessoria juridica'
,20100,400,'Vizela','05-05-2020').
-adjudicante(A,B,C,D) :-
        nao(adjudicante(A,B,C,D)),
        nao(excecao(adjudicante(A,B,C,D))).
-adjudicataria(A,B,C,D) :-
        nao(adjudicataria(A,B,C,D)),
        nao(excecao(adjudicataria(A,B,C,D))).
-adjudicante(18,'Alto Tejo',123456789,'Iberia').

yes
| ?- involucao(adjudicante(18,'Alto Tejo', 123456789, 'Iberia'),n)
     .
yes
| ?- listing(-).
-adjudicante(9,'Municipio Alegre',702675102,'Portugal, Vinhais').
-adjudicante(10,'Municipio Triste',702675113,'Portugal, Beja').
-adjudicataria(9,'Sociedade society, SP,RL',712685112,'Bulgaria').
-adjudicataria(10,'Advocacia de advogados, SP,RL',712675110,'Espanha').
-contrato(10,1,2,'Aquisicao de serviÃ§os','Consulta prÃ©via','Assessoria juridica'
,10200,500,'Guimaraes','07-07-2020').
-contrato(11,2,3,'Aquisicao de serviÃ§os','Consulta prÃ©via','Assessoria juridica'
,20100,400,'Vizela','05-05-2020').
-adjudicante(A,B,C,D) :-
        nao(adjudicante(A,B,C,D)),
        nao(excecao(adjudicante(A,B,C,D))).
-adjudicataria(A,B,C,D) :-
        nao(adjudicataria(A,B,C,D)),
        nao(excecao(adjudicataria(A,B,C,D))).

yes
| ?-
```

Figura 20: Exemplo de involução de conhecimento negativo

- **Conhecimento incerto**

Para tratar da involução de conhecimento incerto, não basta remover o respetivo termo, é também preciso retirar a exceção associada a este. Assim, como temos três predicados distintos (cada um com vários campos que podem ser incertos), foi preciso criar um predicado de *involução* para cada um deles. Os argumentos deste predicado são assim o termo a adicionare o parâmetro que é desconhecido.

```
%Involucao de adjudicataria com nome desconhecido
involucao(adjudicataria(Id,Nome,Nif,Morada),nome_desconhecido) :-
    involucao(adjudicataria(Id,Nome,Nif,Morada),p),
    remocao((excecao(adjudicataria(I,_,Ni,M)) :- adjudicataria(I,Nome,Ni,M))).
```

Figura 21: Involução de conhecimento incerto (ajudicante com nome desconhecido)

```
yes
| ?- listing(adjudicante).
adjudicante(1, 'Municipio de Basto', 702675112, 'Portugal, Cabeceiras de Basto').
adjudicante(2, 'Municipio de Braga', 712675112, 'Portugal, Braga').
adjudicante(3, 'Municipio de Guimaraes', 722675112, 'Portugal, Guimaraes').
adjudicante(4, 'Municipio de Paris', 732675112, 'FranÃ§a, Paris').
adjudicante(5, 'Municipio de Rio', 742675112, 'Brasil, Rio de Janeiro').
adjudicante(6, 'Municipio de Lisboa', 752675112, 'Portugal, Lisboa').
adjudicante(7, 'Municipio de Madrid', 762675112, 'Espanha, Madrid').
adjudicante(8, 'Municipio de Barcelona', 772675112, 'Espanha, Barcelona').
adjudicante(11, incerto, 6, d).
adjudicante(13, coisa, 123456789, morada_imp).
adjudicante(15, 'Municipio Couve', desconhecido, 'Espanha').

yes
| ?- involucao(adjudicante(15,'Municipio Couve', desconhecido,'Espanha'),nif_desc
onhecido).
yes
| ?- listing(adjudicante).                                                ad
judicante(1, 'Municipio de Basto', 702675112, 'Portugal, Cabeceiras de Basto').
adjudicante(2, 'Municipio de Braga', 712675112, 'Portugal, Braga').
adjudicante(3, 'Municipio de Guimaraes', 722675112, 'Portugal, Guimaraes').
adjudicante(4, 'Municipio de Paris', 732675112, 'FranÃ§a, Paris').
adjudicante(5, 'Municipio de Rio', 742675112, 'Brasil, Rio de Janeiro').
adjudicante(6, 'Municipio de Lisboa', 752675112, 'Portugal, Lisboa').
adjudicante(7, 'Municipio de Madrid', 762675112, 'Espanha, Madrid').
adjudicante(8, 'Municipio de Barcelona', 772675112, 'Espanha, Barcelona').
adjudicante(11, incerto, 6, d).
adjudicante(13, coisa, 123456789, morada_imp).

yes
| ?-
```

Figura 22: Aplicação do predicado desenvolvido

- **Conhecimento impreciso**

  Para a remoção de conhecimento impreciso, criamos um predicado que remove exceções.

```
% EVOLUCAO CONHECIMENTO IMPRECISO
%VALOR IMPRECISO
evolucao( Termo , i) :-
    solucoes( Invariante,+(excecao(Termo))::Invariante,Lista ),
    insercao( excecao(Termo) ),
    teste( Lista ).
```

Figura 23: Evolução de conhecimento impreciso

Para que seja possível retirar informação desconhecida de entre um intervalo de valores, construímos ainda outros predicados de involução que além de receberem o termo em questão, recebem a *tag* **i** e os limites inferior e superior, como se pode observar na seguinte figura:

```
%ADJUCATARIA (nif impreciso)
involucao( adjudicataria(Id,Nome,Nif,Morada) , i, Inf, Sup) :-
    remocao((excecao(adjudicataria(Id,Nome,Nif,Morada)) :- Nif >= Inf,Nif =< Sup)).
```

Figura 24: Involução de conhecimento impreciso (sabe-se que o NIF do ajudicante está num certo intervalo)

```
excecao(adjudicante(A,B,C,_)) :-
        adjudicante(A, B, C, morada_imp).
excecao(adjudicataria(A,B,_,C)) :-
        adjudicataria(A, B, nif_imp, C).
excecao(contrato(A,B,C,D,E,F,G,H,_,I)) :-
        contrato(A, B, C, D, E, F, G, H, local_imp, I).
excecao(adjudicante(A,B,_,C)) :-
        adjudicante(A, B, desconhecido, C).
excecao(adjudicante(21,'Sintra',A,'Portugal')) :-
        A>=700111000,
        A=<700111222.

yes
| ?-

| ?- involucao(adjudicante(21,'Sintra', A,'Portugal'),i,700111000,700111222).
yes
```

Figura 25: Aplicação do predicado em questão

- **Conhecimento interdito**

Para este tipo de conhecimento, desenvolveu-se um predicado semelhante ao do tipo *incerto*, acrescentando o parâmetro impossível de se saber ao predicado **nulo**. Assim, para cada predicado e para cada um dos parâmetros que possam ser interditos, foi necessário criar um predicado específico. Segue-se um exemplo do predicado desenvolvido para a involução de um adjudicante cujo nome é impossível de conhecer.

```
%Involucao de adjudicante com nome impossivel
involucao(adjudicante(Id,Nome,Nif,Morada),nome_impossivel) :-
    involucao(adjudicante(Id,Nome,Nif,Morada),p),
    remocao((excecao(adjudicante(I,_,Ni,M)):-adjudicante(I,Nome,Ni,M))),
    remocao((nulo(Nome))),
    remocao(( +adjudicante(I,_,Ni,M) :: (solucoes((I, _, Ni, M ),
    (adjudicante(Id, Nome, Nif, Morada ),nao(nulo(Nome))),R),comprimento(R,0)) )).
```

Figura 26: Exemplo de involução de conhecimento interdito

```
| ?-
| ?- listing(adjudicante).
adjudicante(1, 'Municipio de Basto', 702675112, 'Portugal, Cabeceiras de Basto').

adjudicante(2, 'Municipio de Braga', 712675112, 'Portugal, Braga').
adjudicante(3, 'Municipio de Guimaraes', 722675112, 'Portugal, Guimaraes').
adjudicante(4, 'Municipio de Paris', 732675112, 'FranÃ§a, Paris').
adjudicante(5, 'Municipio de Rio', 742675112, 'Brasil, Rio de Janeiro').
adjudicante(6, 'Municipio de Lisboa', 752675112, 'Portugal, Lisboa').
adjudicante(7, 'Municipio de Madrid', 762675112, 'Espanha, Madrid').
adjudicante(8, 'Municipio de Barcelona', 772675112, 'Espanha, Barcelona').
adjudicante(11, incerto, 6, d).
adjudicante(13, coisa, 123456789, impossivel).

yes
| ?- involucao(adjudicante(13,coisa,123456789,impossivel),morada_impossivel).
yes
| ?- listing(adjudicante).
adjudicante(1, 'Municipio de Basto', 702675112, 'Portugal, Cabeceiras de Basto').

adjudicante(2, 'Municipio de Braga', 712675112, 'Portugal, Braga').
adjudicante(3, 'Municipio de Guimaraes', 722675112, 'Portugal, Guimaraes').
adjudicante(4, 'Municipio de Paris', 732675112, 'FranÃ§a, Paris').
adjudicante(5, 'Municipio de Rio', 742675112, 'Brasil, Rio de Janeiro').
adjudicante(6, 'Municipio de Lisboa', 752675112, 'Portugal, Lisboa').
adjudicante(7, 'Municipio de Madrid', 762675112, 'Espanha, Madrid').
adjudicante(8, 'Municipio de Barcelona', 772675112, 'Espanha, Barcelona').
adjudicante(11, incerto, 6, d).

yes
| ?-
```

Figura 27: Exemplo de aplicação

4 Funcionalidades

4.1 Invariantes Gerais

Estes invariantes têm como objetivo gerir a informação da nossa base de conhecimento. Alguns deles impedem a existência de conhecimento e/ou exceções

repetidos enquanto que outros não permitem que existe informação contraditória.

```
% Invariante para não existir conhecimento repetido
% conhecimento positivo
+GERAL :: (solucoes(GERAL, GERAL, N),
           comprimento(N, 1)).

% Invariante para não existir conhecimento repetido
% conhecimento negativo
+(-GERAL) :: (solucoes(GERAL, -GERAL, N),
           comprimento(N, 1)).

% Invariante para não adicionar conhecimento contraditório
% conhecimento positivo que contradiz conhecimento negativo
+GERAL :: nao(-GERAL).

% Invariante para não adicionar conhecimento contraditório
% conhecimento negativo que contradiz conhecimento positivo
+(-GERAL) :: nao(GERAL).

% Invariante que garante que não existem excecoes repetidas
+(excecao(GERAL)) :: (solucoes(GERAL, excecao(GERAL), N),
                      comprimento(N, 1)).
```

Figura 28: Invariantes gerais

4.2 Invariantes pedidos

4.2.1 Tipos de Procedimento

Como só existem três procedimentos válidos (*Ajuste direto, Consulta prévia, Concurso público*), criamos dois invariantes para garantir este requisito.

```
% Invariante para garantir que o tipo de procedimento é valido
% conhecimento positivo
+contrato(_,_,_,_,Tipo,_,_,_,_,_) :: procedimentoValido(Tipo).

% Invariante para garantir que o tipo de procedimento é valido
% conhecimento negativo
+(-contrato(_,_,_,_,Tipo,_,_,_,_,_)) :: procedimentoValido(Tipo).
```

Figura 29: Invariante do tipo de procedimento

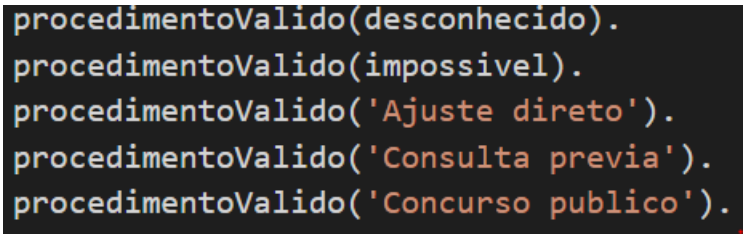

Figura 30: Procedimentos Válidos

4.2.2 Condições do 'Ajuste Direto'

Invariante que garante que quando o *contrato* é do tipo 'Ajuste direto' o valor só poderá atingir 5.000 euros. O tipo de contrato só poderá ser *'Contrato de aquisição'*, *'Locação de bens móveis'* e *'Aquisição de bens'*. O prazo é obrigatoriamente menor ou igual a 1 ano.

```
% Invariante para garantir que quando o tipo é "Ajuste direto" as condições obrigatórias sao cumpridas
% conhecimento positivo
+contrato(_,_,_,TipoC,TipoP,_,Valor,Prazo,_,_) :: condicoesValidas(TipoC,TipoP,Valor,Prazo).

% Invariante para garantir que quando o tipo é "Ajuste direto" as condições obrigatórias sao cumpridas
% conhecimento negativo
+(-contrato(_,_,_,TipoC,TipoP,_,Valor,Prazo,_,_)) :: condicoesValidas(TipoC,TipoP,Valor,Prazo).
```

Figura 31: Invariante de condição de ajuste direto

```
condicoesValidas(TipoC,TipoP,Valor,Prazo) :-
TipoC='Aquisicao de servicos',TipoP='Ajuste direto',Valor =< 5000,Prazo=<365;
TipoC='Contrato de aquisicao',TipoP='Ajuste direto',Valor =< 5000,Prazo=<365;
TipoC='Locacao de bens moveis',TipoP='Ajuste direto',Valor =< 5000,Prazo=<365;
TipoP\='Ajuste direto'.
```

Figura 32: Predicado auxiliar

4.2.3 Regra dos 3 anos válido para contratos

Invariante que não permite a evolução de um contrato caso o adjudicante e a adjudicataria tenham um valor de contratos acumulado, superior ou igual a 75.000 euros, não incluindo o valor do contrato que pretende adicionar. De notar que o contratos só são considerados se estiverem no mesmo ano do pretendido, ou 2 anos anteriores.

```
+contrato(_,IdAd,IdAda,_,TipoProcedimento,_,V,_,_,Data) ::
    (solucoes((Valor),(contrato(_,IdAd,IdAda,_,TipoProcedimento,_,Valor,_,_,Da),dataDentro(Data,Da)), L),
     somaLista(L,R),
     sub(R,V,F),
     F<75000).
```

Figura 33: Invariante da regra dos 3 anos

```
dataDentro([Da,Ma,Aa],[D,M,A]):- Aa=A; sum(A,1,R),Aa=R; sum(A,2,R),Aa=R,Ma=<M,Da=<D.
sum(X,Y,R) :- R is X+Y.
sub(X,Y,R) :- R is X-Y.
mul(X,Y,R) :- R is X*Y.
```

Figura 34: Predicado auxiliar

4.3 Invariantes Extras

4.3.1 Id/Nif não repetivel

Foi criado um invariante para cada predicado (Adjudicante,Adjudicataria,Contrato), de modo a garantir que não exista dois do mesmo tipo com ids iguais, tanto em conhecimento positivo como negativo.

```
% Invariante para garantir que o id não é repetido
% conhecimento positivo
+adjudicataria(IdAd,_,_,_) :: (solucoes(IdAd,adjudicataria(IdAd,_,_,_),L),
                               comprimento(L,1)).

% Invariante para garantir que o id não é repetido
% conhecimento negativo
+(-adjudicataria(IdAd,_,_,_)):: (solucoes(IdAd,-adjudicataria(IdAd,_,_,_),L),
                                 comprimento(L,1)).
```

Também foi garantido que não existia dois nif iguais.

```
% Invariante para garantir que o nif não é repetido
% conhecimento positivo
+adjudicataria(_,_,Nif,_) :: (solucoes(Nif,adjudicataria(_,_,Nif,_),L),
                              comprimento(L,1)).

% Invariante para garantir que o nif não é repetido
% conhecimento negativo
+(-adjudicataria(_,_,Nif,_)):: (solucoes(Nif,-adjudicataria(_,_,Nif,_),L),
                                comprimento(L,1)).
```

4.3.2 Validade dos argumentos

Nesta secção abordou-se a validade dos argumentos.

Criou-se invariantes que garantissem que o valor do **Id** era positivo. No caso do *contrato*, para além dos ids, também se verificou se o **Valor** e **Prazo** eram positivos. Ainda dentro do *contrato*, valida-se também a **Data**, sendo esta mais complexa, verificando assim os anos bissextos e os dias de cada mês.

No adjudicante e adjucataria, optou-se pela criação de um invariante para garantir que o **Nif** é um valor de 9 digitos, tornando assim o trabalho mais realista e completo.

```
% Invariante para garantir que os ids sao validos
% conhecimento positivo
+contrato(IdC,IdAd,IdAda,_,_,_,_,_,_,_) :: idsValidos(IdC,IdAd,IdAda).

% Invariante para garantir que os ids sao validos
% conhecimento negativo
+(-contrato(IdC,IdAd,IdAda,_,_,_,_,_,_,_)) :: idsValidos(IdC,IdAd,IdAda).
```

Figura 35: Invariante ids validos

```
% Invariante para garantir que o valor é valido
% conhecimento positivo
+contrato(_,_,_,_,_,_,Valor,_,_,_) :: valPositivo(Valor).

% Invariante para garantir que o valor é valido
% conhecimento negativo
+(-contrato(_,_,_,_,_,_,Valor,_,_,_)) :: valPositivo(Valor).
```

Figura 36: Invariante valor valido

```
% Invariante para garantir que o prazo é valido
% conhecimento positivo
+contrato(_,_,_,_,_,_,_,Prazo,_,_) :: valPositivo(Prazo).

% Invariante para garantir que o prazo é valido
% conhecimento negativo
+(-contrato(_,_,_,_,_,_,_,Prazo,_,_)) :: valPositivo(Prazo).
```

Figura 37: Invariante prazo valido

```
% Invariante para garantir que a data é valida
% conhecimento positivo
+contrato(_,_,_,_,_,_,_,_,_,Data) :: dataValida(Data).

% Invariante para garantir que a data é valida
% conhecimento negativo
+(-contrato(_,_,_,_,_,_,_,_,_,Data)) :: dataValida(Data).
```

Figura 38: Invariante data valida

```
% PREDICADOS AUXILIARES PARA OS SEGUINTES INVARIANTES
nifValido(desconhecido).
nifValido(impossivel).
nifValido(NIF) :- NIF >= 100000000,NIF =< 999999999.
valPositivo(desconhecido).
valPositivo(impossivel).
valPositivo(N) :- N >= 0.
idsValidos(ID1,ID2,ID3) :- ID1 >= 0, ID2 >= 0, ID3 >= 0.
```

Figura 39: Predicados auxiliares

```
% TODAS ESTES PREDICADOS SERVEM PARA VALIDAR A DATA
bissexto(Ano) :- R is mod(Ano,4),R==0.
dataValida([D,02,A]) :- A>=0, bissexto(A), D>=1, D=<29; A>=0, D>=1, D=<28.
dataValida([D,M,A]) :- A>=0, pertence(M,[01,03,05,07,08,10,12]), D>=1, D=<31.
dataValida([D,M,A]) :- A>=0, pertence(M,[04,06,09,11]), D>=1, D=<30.

% MESES COM 31 -> 01,03,05,07,08,10,12
% MESES COM 30 -> 04,06,09,11
% MESES COM 28/29 -> 02
```

Figura 40: Predicado auxiliar 'dataValida'

4.3.3 Remoção de adjudicante

Foi criado um invariante referente à remoção. Este não permite a remoção de um adjudicante que contenha mais de 2 contratos.

```
% Invariante para garantir que não é possível remover um adjudicante que tem mais de 2 contratos
% Conhecimento positivo
-adjudicante(Id,_,_,_) :: (solucoes(Id, contrato(_,Id,_,_,_,_,_,_,_,_), R),comprimento(R, N),N=<2).
```

Figura 41: Invariante de remoção de ajudicante

4.4 Querie extra

Para enriquecer o trabalho, criou-se um predicado extra denominado de **contratosAtivos** que como o nome indica, este apresentará a lista de contratos que se ecncontram ativos. Este recebe como argumento a data atual.

```
contratosAtivos(Data) :- solucoes((contrato(Id,Id1,Id2,TC,TP,D,V,Prazo,Local,Da)),
(contrato(Id,Id1,Id2,TC,TP,D,V,Prazo,Local,Da),dataNaoExpirada(Data,Da,Prazo)),L),
showLista(L).
```

Figura 42: Predicado contratoAtivos

Para a concretização desta querie foram criados dois predicados auxiliares, um com um algoritmo para determinar se o contrato já expirou ou não, e outro que imprime o resultado de forma limpa e percetível.

```
dataNaoExpirada([Da,Ma,Aa],[D,M,A],Prazo) :- sub(Aa,A,R),mul(R,365,R1),
                                              sub(Ma,M,R2),mul(R2,30,R3),
                                              sub(Da,D,R4),
                                              sum(R1,R3,R5),sum(R5,R4,F),
                                              F < Prazo.

showLista([]).
showLista([X|Y]) :- write(X),nl,showLista(Y).
```

Figura 43: Predicados auxiliares

5 Conclusões e sugestões

Com o auxílio da *programação lógica estendida* e com o uso de *conhecimento imperfeito*, foi-nos possível então criar uma base de conhecimento capaz de satisfazer as necessidades de um sistema de contratação pública, implementando ainda mecanismos para gerir este conhecimento. Porém, a elaboração deste trabalho não veio sem dificuldades, por exemplo na involução de conhecimento imperfeito, onde só após alguns retrocessos e discussão em grupo é que se chegou a um bom resultado. Desta forma, este projeto mostrou ser uma excelente maneira de consolidar os conhecimentos a nós transmitidos ao longo do semestre.

Em futuras iterações deste trabalho, e de forma a melhorá-lo, poder-se-ia acrescentar novos predicados que permitissem pesquisas mais específicas da informação na nossa base de conhecimento, fornecendo assim uma melhor experiência para o utilizador.

6 Referências Bibliográficas

Referências

[1] Cesar Analide, José Neves,
"Representação de Informação Incompleta",
Texto Pedagógico, Grupo de Inteligência Artificial, Centro de Ciências e Tecnologias da Computação, Portugal, 1996.

[2] Paulo Novais, Cesar Analide, José Neves,
"Sugestões para a Redação de Relatórios Técnicos ",
Relatório técnico, Departamento de Informática da Universidade do Minho, Portugal, 2011.