Universidade do Minho

Sistemas de Representação de Conhecimento e Raciocínio

Departamento de Informática

# Trabalho Prático 1


| *Grupo 24:* | *Número:* |
|---|---|
| João Abreu | A84802 |
| Hugo Matias | A85370 |
| Tiago Magalhães | A84485 |
| Pedro Fernandes | A84313 |


3 de Maio de 2020

# Conteúdo

# 1. Resumo

O presente relatório é constituído por uma breve descrição e demonstração do trabalho realizado no âmbito do Exercício 1 da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Assim sendo, ao longo deste documento será apresentada toda a base do conhecimento e predicados que a constituem e, em seguida, os invariantes estruturais e referenciais que permitem garantir uma correta evolução da base de conhecimento, com adição de novo conhecimento, que também será aqui descrito.

Nas secções seguintes serão ainda analisados os tópicos de remoção de conhecimento, identificações de predicados através de alguns critérios.

Finalmente, serão apresentadas todas as funcionalidades extras que implementamos neste exercício.

# 2. Introdução

Este exercício teve como principal objectivo motivar-nos para a utilização da linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas.

Durante o exercício foi desenvolvido um sistema para caracterizar um universo de discurso na área da contratação pública que incluía conhecimento de adjudicantes, adjudicatários e os próprios contratos.

Este sistema tinha como requisitos a implementação de diversas funcionalidades, apresentadas ao longo deste relatório, que permitem uma correta manipulação da base de conhecimento (quer adição quer remoção de conhecimento) e a aplicação de uma série de queries sobre ela.

# 3. Preliminares

Para realização e compreensão deste trabalho é necessário o conhecimento do paradigma da **programação em lógica**, bem como a sua linguagem de programação, o *PROLOG*.

Como qualquer sistema de computacional necessita de armazenar e manipular informação, os dois tipos de armazenamento utilizados por um sistema computacional são: **Bases de dados** e **Sistemas de representação de conhecimento**.
O primeiro baseia-se nos pressupostos de:

- **Pressuposto do Mundo Fechado** - toda a informação que não existe mencionada na base de dados é considerada falsa;
- **Pressuposto dos Nomes Único** - duas constantes diferentes (que definam valores atómicos ou objectos) designam, necessariamente, duas entidades diferentes do universo de discurso;
- **Pressuposto do Domínio Fechado** - não existem mais objectos no universo de discurso para além daqueles designados por constantes na base de dados.

enquanto que o segundo se baseia nos pressupostos de:

- **Pressuposto do Mundo Aberto** – podem existir outros factos ou conclusões verdadeiros para além daqueles representados na base de conhecimento;
- **Pressuposto dos Nomes Únicos** - duas constantes diferentes (que definam valores atómicos ou objectos) designam,necessariamente,duas entidades diferentes do universo de discurso;
- **Pressuposto do Domínio Aberto** - podem existir mais objectos do universo de discurso para além daqueles designados pelas constantes da base de conhecimento.

Os pressuposto de mundo e domínio fechado, comuns na programação em **Lógica clássica** apresentam algumas limitações para tratar de informação incompleta. Sendo assim os Sistemas de Representação de Conhecimento reais não trabalham com estes pressupostos, uma vez que contextualizam situações em que não existem informação completa, sendo necessária a representação de **conhecimento Imperfeito** que vai para além do verdadeiro e falso, permitindo assim o **desconhecido**. Desta forma foi feita uma **extensão à programação em lógica**.

Este tipo de extensão faz-se pela introdução na linguagem de representação, de dois tipos de **negação**: por **falha na prova** representada pelo termo *'não'*, que nos indica que não existe uma prova na base de conhecimento que responda à questão, e a **negação forte**, representado por *'-'*, que nos indica que existe uma prova na base de conhecimento de que a questão é falsa.

Com esta representação de **informação negativa**, ao contrário de se assumir que não se conhece é falso, passamos a representar algo que sabemos que é falso.

Com isto passamos a conseguir representar **conhecimento imperfeito**, ou seja informação incompleta. Esta identificação de valores, designados por **valores nulos**, surge como estratégia para representar respostas que deverão ser concretizadas como **conhecidas** (verdadeiras ou falsas) ou **desconhecidas**.Sendo estes de três tipos:

- **Incerto** - desconhecido de um conjunto indeterminado de hipóteses;
- **Impreciso** - desconhecido, mas de um conjunto determinado de hipóteses
- **Indeterminado** - desconhecido e não permitido conhecer.

# 4. Descrição do Trabalho e Análise de Resultados

Para uma melhor exposição do trabalho realizado, dividimos o mesmo em **6** diferentes partes. A primeira parte é a **Base de Conhecimento**, onde descrevemos os predicados que constituem a base de conhecimento do nosso sistema de representação de conhecimento e raciocínio.

Em seguida, referimos os **Invariantes estruturais e referenciais** que controlam a inserção e remoção de conhecimento, a **Adição de Conhecimento** e a **Remoção de Conhecimento**, onde apresentamos os procedimentos construídos para o efeito, devidamente ilustrados com simples exemplos práticos de aplicação.

Por fim, os vários **Extras** que foram incluídos no nosso sistema, quer ao nível das capacidades de representação de conhecimento, quer ao nível das faculdades de raciocínio.

Em **Anexo** apresentamos as Funções Auxiliares, onde se encontram as funções que nos auxiliaram na construção dos predicados pedidos.

Ao longo da presente secção, fomos descrevendo o trabalho e simultaneamente analisando os resultados obtidos.

## 4.1 Sistema de inferência

Ao adicionar-se a capacidade para representar informação incompleta, por conseguinte foi necessário o desenvolvimento de um sistema de inferência capaz de representar o desconhecido.

```
% Extensao do meta-predicado demo: Questao,Resposta -> {V,F,D}
%                           Resposta = { verdadeiro,falso,desconhecido }
% Sistema de Inferência

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

Através do código acima podemos ver que sempre que uma questão $Q$ for colocada ela pode ser definida da seguinte maneira: Verdadeira quando for possível provar a questão $Q$ na base de conhecimento, Falsa quando for possível provar a falsidade de uma questão -Q na base de conhecimento e Desconhecida quando não for possível provar a questão $Q$ nem a questão -$Q$.

## 4.2 Base de Conhecimento

O nosso Sistema de Representação de Conhecimento é relativo ao universo de discurso na área da contratação pública para a realização de contratos para a prestação de serviços. Assim a nossa base de conhecimento é constituída por um conjunto de factos acerca de contratos públicos, adjudicantes e adjudicatárias.

### 4.2.1 Conhecimento Perfeito Positivo

#### 4.2.1.1 Adjudicante

O predicado adjudicante é é definido da seguinte forma:

```
adjudicante: #IdAd, Nome, NIF, Morada -> {V,F,D}
```

Base de conhecimento conhecimento perfeito

```
adjudicante(500745471,'Santa Casa de Lisboa',500745471,'Portugal,Lisboa').
adjudicante(506415082,'Municipio de Coimbra',506415082,'Portugal,Lisboa').
adjudicante(501102752,'Municipio de Amarante',501102752,'Portugal,Amarante').
adjudicante(506770664,'Municipio de Vouzela',506770664,'Portugal,Viseu').
adjudicante(600020339,'Procuradoria Geral da Republica',600020339,'Portugal,Lisboa').
adjudicante(500051070,'Municipio de Lisboa',500051070,'Portugal,Lisboa').
```

#### 4.2.1.2 Adjudicatária

O predicado adjudicataria é é definido da seguinte forma:

```
Extensão do predicado adjudicatária: #IdAda, Nome, NIF, Morada -> {V,F,D}
```

Base de conhecimento conhecimento perfeito

```
adjudicataria(514023708,'Arcos combinados-Arquietetos Associados LDA.',514023708,'Portugal').
adjudicataria(513826602,'VANITYFORMULA - PECAS AUTO, UNIPESSOAL, LDA',513826602,'Portugal').
adjudicataria(508559871,'EDILAGES,S.A.',508559871,'Portugal').
adjudicataria(514495790,'Dream2Fly,Lda',514495790,'Portugal').
adjudicataria(506155676,'Xamane,S.A.',506155676,'Portugal').
adjudicataria(508190495,'Alugal,Lda',508190495,'Portugal').
adjudicataria(505002892,'INESC',505002892,'Portugal,Lisboa').
adjudicataria(500489297,'A.da Costa, Lda.',500489297,'Portugal').
```

#### 4.2.1.3 Contrato

O predicado contrato é definido da seguinte forma:

```
contrato: #IdAd, #IdAda, TipoDeContrato, TipoDeProcedimento, Descrição, Custo, Prazo,
Local, Data -> {V,F,D}
```

Base de conhecimento conhecimento perfeito

```
contrato(500745471,514023708,'Aquisicao de servicos','Concurso Publico',
        'Levantamento Topografico',31500,90,'Portugal',data(24,4,2020)).
contrato(506415082,513826602,'Aquisicao de servicos','Concurso Publico',
        'Pecas para viaturas,maquinas e equipamentos',31250,1080,
        'Portugal,Coimbra',data(27,12,2019)).
contrato(501102752,508559871,'Empreitadas de obras publicas',
        'Concurso Publico','Pavilhao Desportivo da EB2
        3',987853,80,'Portugal,Porto,Amarante',data(30,3,2020)).
contrato(506770664,514495790,'Aquisicao de bens moveis','Ajuste Direto',
        'Aquisicao de Computadores Portateis',1080,8,'Portugal,Viseu',data(30,4,2020)).
contrato(506770664,506155676,'Aquisicao de bens moveis','Ajuste Direto',
        'Equipamento de protecao individual',3241,10,'Portugal,Viseu',data(27,4,2020)).
contrato(501102752,508190495,'Lococao de bens moveis','Ajuste Direto',
        'Lococao de monoblocos',300,100,'Portugal,Porto,Amarante',data(1,5,2020)).
```

#### 4.2.1.4 Tipo Procedimento

O predicado tipo de procedimento é definido da seguinte forma:

```
% Extensão do predicado contrato: TP -> {V,F,D}
```

Base de conhecimento conhecimento perfeito

```
tipoProcedimento('Ajuste Direto').
tipoProcedimento('Consulta Previa').
tipoProcedimento('Concurso Publico').
```

#### 4.2.1.5 Tipo Contrato

O predicado tipo de contrato é definido da seguinte forma:

```
% Extensão do predicado contrato: TC -> {V,F,D}
```

Base de conhecimento conhecimento perfeito

```
tipoContrato('Aquisicao de bens moveis').
tipoContrato('Locacao de bens moveis').
tipoContrato('Aquisicao de servicos').
tipoContrato('Empreitadas de obras publicas').
```

### 4.2.2 Conhecimento Negativo

Como por definição de um contrato público este para o ser é necessário que esteja identificado e com acesso público, bem como as entidades envolvidas, desta forma mantivemos o pressuposto do mundo fechado para o predicado adjudicante,adjudicataria e contrato. Assim contratos, adjudicantes, adjudicatarias que não estejam definidos na base de conhecimento nem que contenham alguma exceção são considerados falsos, pois não são públicos. Deste modo definimos por negação na falha os seguintes predicados e regras:

```
-adjudicante(ID,N,NIF,M) :- nao(adjudicante(ID,N,NIF,M)), nao(excecao(adjudicante(ID,N,NIF,M))).
-adjudicataria(ID,N,NIF,M) :- nao(adjudicataria(ID,N,NIF,M)), nao(excecao(adjudicataria(ID,N,NIF,M))).
-contrato(AN,AT,TC,TP,D,V,P,L,DT) :- nao(contrato(AN,AT,TC,TP,D,V,P,L,DT)),
                                     nao(excecao(contrato(AN,AT,TC,TP,D,V,P,L,DT))).
```

Alguns contratos, adjudicantes e adjudicatárias podem ser inválidos por isso definimos os seguintes predicados e regras por negação forte:

```
% Nif da entidade adjudicataria não pode ser uma pessoa singular
-adjudicataria(203931467,'Patricia',203931467,'Evora').

% Conhecimento Perfeito Negativo
% Entidades publicas não têm NIF de privados,pessoas singulares
-adjudicante(276836642,'Pedro' ,276836642,'Braga').

% Conhecimento Perfeito Negativo
% Contrato com um tipo de contrato pre reforma não está previsto no
% código de contrato públicos mas sim de trabalho
-contrato(501102752,508190495,'Pre-reforma','Ajuste direto',
          'Lococao de monoblocos',6.300,181,'Portugal,Porto,Amarante',data(19,12,2019)).
```

### 4.2.3 Conhecimento Imperfeito

O conhecimento imperfeito caracteriza-se pela utilização de valores nulos.

#### 4.2.3.1 Conhecimento Imperfeito Incerto

Exemplo de valor nulo de tipo incerto para quando não se sabe o valor num conjunto indeterminado de hipoteses:

```
% Conhecimento Imperfeito Incerto
% Não se sabe a morada do adjudicante Municipio de Vinhais
excecao(adjudicante(ID,N,NIF,M)) :- adjudicante(ID,N,NIF,moradaDesconhecida).
adjudicante(223184241,'Municipio de Vinhais',223184241,moradaDesconhecida).
```

#### 4.2.3.2 Conhecimento Imperfeito Impreciso

Exemplo de valor nulo de tipo impreciso para quando não se sabe o valor num conjunto determinado de hipóteses tanto contido num intervalo ou entre hipóteses:

```
% Conhecimento Imperfeito Impreciso(morada do adjudicante é desconhecido
%porém sabe-se é uma de entre um intervalo de respostas)
excecao(adjudicante(264306422,'Municipio de Braga' ,264306422,'Braga,Pacos de Concelho')).
excecao(adjudicante(264306422,'Municipio de Amarante' ,264306422,'Braga,Se')).

% Conhecimento Imperfeito Impreciso
% Só se sabe que o valor está entre 7000 e 10000
excecao(contrato(500051070,500489297,'Aquisicao de bens moveis',
                'Concurso publico','Aquisicao de artigos de fardamento',V,90,'Portugal,Lisboa'
                ,data(2,4,2020))) :- V >= 7000,V =< 10000.
```

#### 4.2.3.3 Conhecimento Imperfeito Interdito

Exemplo de valor nulo de tipo interdito. Certos contratos nos termos da lei podem ser declarado secretos, assim apresentamos o seguinte exemplos:

```
% É impossivel saber o tipo de contrato,descricao e localizacao  cuja
% execução deve ser acompanhada de medidas especiais de segurança
% bem como  os interesses essenciais de defesa e
% segurança do Estado o exigirem

contrato(600020339,505002892,contratoSecreto,'Ajuste Direto',descricaoSecreta,75000,60,localizacaoSecreta,data(29,4,2020)).
excecao(contrato(AN,AT,TC,TP,D,V,P,L,DT)) :- contrato(AN,AT,contratoSecreto,TP,descricaoSecreta,V,P,localizacaoSecreta,DT).

nuloInterdito(contratoSecreto).
nuloInterdito(descricaoSecreta).
nuloInterdito(localizacaoSecreta).

% Garantir que não é adicionado valor conhecido a um contrato cujo valor deve permanecer desconhecido
+contrato(AN,AT,TC,TP,D,V,P,L,DT) :: (solucoes((AN,AT,TC,TP,D,V,P,L,DT),
                                        (contrato(600020339,505002892,contratoSecreto,'Ajuste Direto'
                                        ,descricaoSecreta,75000,60,localizacaoSecreta,data(29,4,2020))
                                        ,nao(nuloInterdito(contratoSecreto))
                                        ,nao(nuloInterdito(descricaoSecreta))
                                        ,nao(nuloInterdito(localizacaoSecreta))),S),
                                    comprimento( S,0 )).
```

Figura 4.1: Base de conhecimento conhecimento imperfeito interdito

```
% É impossivel saber o valor do contrato cuja execução deve ser acompanhada de medidas especiais de segurança
contrato(202344142,244059039,'Aquisicao de bens moveis','Ajuste Direto','Assessoria juridica',valorSecreto,364,'Alto de Basto',data(29,4,2020)).
excecao(contrato(AN,AT,TC,TP,D,V,P,L,DT)) :- contrato(AN,AT,TC,TP,D,valorSecreto,P,L,DT).
nuloInterdito(valorSecreto).

% Garantir que não é adicionado valor conhecido a um contrato cujo valor deve permanecer desconhecido
+contrato(AN,AT,TC,TP,D,V,P,L,DT) :: (solucoes((AN,AT,TC,TP,D,V,P,L,DT),
                                        (contrato(202344142,244059039,'Aquisicao de bens moveis','Ajuste Direto','Assessoria juridica',
                                         valorSecreto,364,'Alto de Basto',data(29,4,2020))
                                        ,nao(nuloInterdito(valorSecreto))),S),
                                    comprimento( S,0 )).
```

Figura 4.2: Base de conhecimento conhecimento imperfeito interdito

## 4.3 Invariantes

Antes de construirmos os procedimentos que permitem a adição e a remoção de conhecimento na base de conhecimento do nosso sistema, construímos invariantes estruturais e referenciais para todos os predicados anteriormente mencionados.

### 4.3.1 Universais

1. Invariante que garante que não existe conhecimento perfeito positivo repetido.

```
+T :: (solucoes(T, T, S), comprimento(S, 1)).
```

2. Invariante que garante que não existe conhecimento perfeito negativo repetido.

```
+(-T) :: (solucoes(T, -T, S), comprimento(S, 1)).
```

3. Invariante que não permite adicionar conhecimento perfeito positivo que contradiz conhecimento perfeito negativo.

```
+T :: nao(-T).
```

4. Invariante que não permite adicionar conhecimento perfeito negativo que contradiz conhecimento perfeito positivo.

```
+(-T) :: nao(T).
```

5. Invariante que garante que não existem excecoes repetidas.

```
+(excecao(T)) :: (solucoes(T, excecao(T), S),
                  comprimento(S, 1)).
```

### 4.3.2 Estruturais

1. Garantir que cada adjudicante é único.

```
+adjudicante(ID,N,NIF,M)::
    (solucoes( (ID,N,NIF,M),( adjudicante( ID,N,NIF,M ) ),S ),
    comprimento( S,Num ),
    Num == 1  ).
```

2. Garantir que a não inserção de um conhecimento negativo de uma adjudicante que já exista.

```
+(-adjudicante(ID,N,NIF,M))::
    (solucoes( (ID,N,NIF,M),( (-adjudicante( ID,N,NIF,M ) )),S ),
    comprimento( S,Num ),
    Num == 1  ).
```

3. Garantir que cada adjudicataria é única.

```
+adjudicataria(ID,N,NIF,M)::
    (solucoes( (ID,N,NIF,M),(adjudicataria( ID,N,NIF,M )),S ),
    comprimento( S,Num ),
    Num == 1  ).
```

4. Garantir que a não inserção de um conhecimento negativo de uma adjudicataria que já exista.

```
+(-adjudicataria(ID,N,NIF,M))::
    (solucoes( (ID,N,NIF,M),((-adjudicataria( ID,N,NIF,M ))),S ),
    comprimento( S,Num ),
    Num == 1  ).
```

5. Garantir que cada contrato é único.

```
+contrato(AN,AT,TC,TP,D,V,P,L,DT)::
    (solucoes((AN,AT,TC,TP,D,V,P,L,DT),(contrato(AN,AT,TC,TP,D,V,P,L,DT)),S),
    comprimento( S,Num ),
    Num == 1  ).
```

6. Garantir que hánão inserção de um conhecimento negativo de um contrato que já exista.

```
+(-contrato(AN,AT,TC,TP,D,V,P,L,DT))::
    (solucoes((AN,AT,TC,TP,D,V,P,L,DT),((-contrato(AN,AT,TC,TP,D,V,P,L,DT))),S),
    comprimento( S,Num ),
    Num == 1  ).
```

7. Garantir que cada procedimento é único (não sendo possível adicionar um novo tipo).

```
+tipoProcedimento(T)::
    (solucoes(T,(tipoProcedimento(T)),S),
    comprimento( S,Num ),
    Num == 0  ).
```

### 4.3.3 Referenciais

1. Garantir que ID e NIF de adjudicante são únicos.

```
+adjudicante(ID,N,NIF,M)::
    (solucoes((Ns,Ms),(adjudicante(ID,Ns,NIF,Ms)),S),
    comprimento(S,Num),
    Num=<1).
```

2. Garantir que não se pode remover um adjudicante que não exite na base de conhecimento.

```
-adjudicante(ID,N,NIF,M)::
    (solucoes((ID,N,NIF,M),(adjudicante(ID,N,NIF,M)),S),
    comprimento(S,Num),
    Num == 1).
```

3. Garantir que ID e NIF de adjudicatária são únicos.

```
+adjudicataria(ID,N,NIF,M)::
    (solucoes((Ns,Ms),(adjudicataria(ID,Ns,NIF,Ms) ),S),
    comprimento( S,Num ),
    Num=<1).
```

4. Garantir que apenas se pode retirar da base de conhecimento uma adjudicataria que exista.

```
-adjudicataria(ID,N,NIF,M)::
    (solucoes((ID,N,NIF,M),(adjudicataria(ID,N,NIF,M)),S),
    comprimento(S,Num),
    Num == 1).
```

5. Garantir que ID e NIF de adjudicatária são iguais.

```
+adjudicataria(ID,_,NIF,_):: (ID > 0, ID == NIF).
```

6. Garantir que para o tipo de procedimento Ajuste Direto não se podem adicionar outros tipos de contrato para além deAquisição e Locação de bens móveis e Aquisição de Serviços.

```
+contrato(AN,AT,TC,'Ajuste Direto',D,V,P,L,DT) :: (TC == 'Aquisicao de bens moveis';
                                                    TC == 'Locacao de bens moveis';
                                                    TC == 'Aquisicao de servicos').
```

7. Para um contrato entre as mesmas entidades com o mesmo serviço prestado não se pode adicionar dito contrato, caso o valor acumulado de todos os contrato seja maior ou igual que 75000, e a diferença entre o ano económico e o ano do ultimo contrato realizado seja menor ou igual que dois.

```
+contrato(AN,AT,tipoContrato('Aquisição de Serviços'),TP,D,V,P,L,data(_,_,Ano)) ::
    (acumulaContrato(AN,AT,C),
    C >= 75000,
    difAno(Ano,AN,AT,tipoContrato('Aquisição de Serviços'),D,S),
    S =< 2).
```

8. Garantir que se um contrato tiver tipo de procedimento 'Ajuste Direto', não lhe é permitido ter um valor maior que 5000 e prazo maior que 365 dias.

```
+contrato(AN,AT,TC,'Ajuste Direto',D,V,P,L,DT) :: (V =< 5000, P =< 365).
```

9. Garantir que não se pode remover um contrato que não exite na base de conhecimento.

```
-contrato(AN,AT,TC,TP,D,V,P,L,DT) ::
    (solucoes((AN,AT,TC,TP,D,V,P,L,DT),(contrato(AN,AT,TC,TP,D,V,P,L,DT)),S),
    comprimento( S,Num ),
    Num == 0).
```

10. dos dias do prazo final.

```
-contrato(AN,AT,TC,TP,D,V,P,L,DT) ::
    (datimeToData(X),
    avancaDias(DT,P,F),
    isAfter(F, X),
    fidelizacao(P,R),
    avancaDias(DT,R,RD),
    isAfter(X,RD)).
```

11. Garantir que um contrato está associado a um adjudicante e adjudicatária que existam na base de conhecimento.

```
+contrato(AN,AT,_,_,_,_,_,_,_) ::
    (solucoes(AN,(adjudicante(AN,_,_,_)),S),
    solucoes(AT,(adjudicataria(AT,_,_,_)),R),
    comprimento( S,1 ),
    comprimento( R,1 )).
```

12. Garantir que um contrato é válido (campos corretos).

```
+contrato(AN,AT,TC,TP,D,V,P,L,DT) ::
    (countDigits(AN,N1),
    countDigits(AT,N2),
    tipoContrato(TC),
    tipoProcedimento(TP),
    isData(DT),
    N1 == 9,
    N2 == 9).
```

13. Garantir que um o ID e NIF de uma adjudicante tem 9 dígitos.

```
+adjudicante(ID,N,NIF,M) :: (countDigits(ID,N1), N1 == 9).
```

14. Garantir que um o ID e NIF de uma adjudicataria tem 9 dígitos.

```
+adjudicataria(ID,N,NIF,M) :: (countDigits(ID,N1), N1 == 9).
```

## 4.4 Adicionar e Remover conhecimento

Para adicionar conhecimento tivemos de definir teoremas que verificavam teoremas, de modo a que a inserção de conhecimento seguisse certas regras.

```
% Extensão do predicado que permite a evolucao do conhecimento

% Insere novo conhecimento na base de conhecimento positivo
evolucao( Termo ) :-
    solucoes( Invariante,+Termo::Invariante,Lista ),
    insercao( Termo ),
    teste( Lista ).

% Insere novo conhecimento na base de conhecimento negativo
evolucao( -Termo) :-
    solucoes( Invariante,+(-Termo)::Invariante,Lista ),
    insercao( -Termo ),
    teste( Lista ).

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

Figura 4.3: Evolução de conhecimento positivo e negativo.

Para remover conhecimento tivemos de definir teoremas que verificavam teoremas, de modo a que a inserção de conhecimento seguisse certas regras.

```
% Extensão do predicado que permite a involucao do conhecimento

% Retira novo conhecimento na base de conhecimento positivo
involucao( Termo ) :-
    solucoes( Invariante,-Termo::Invariante,Lista ),
    remocao( Termo ),
    teste( Lista ).

% Retira novo conhecimento na base de conhecimento negativo
involucao( -Termo ) :-
    solucoes( Invariante,-(-Termo)::Invariante,Lista ),
    remocao( -Termo ),
    teste( Lista ).

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

Figura 4.4: Involução de conhecimento positivo e negativo.

Também tivemos de definir para o caso de conhecimento imperfeito, segue a seguir o código para tal definição:

```
%Adjudicante

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso de uma entidade adjudicante com morada desconhecida
evolucao(adjudicante(ID,N,NIF,moradaDesconhecida)) :-
    evolucao(adjudicante(ID,N,NIF,moradaDesconhecida)),
    insercao((excecao(adjudicante(IDAd,NAd,NIFAd,MAd)) :-
                adjudicante(IDAd,NAd,NIFAd,moradaDesconhecida))).

% Remove conhecimento imperfeito incerto na base de conhecimento
% no caso de uma entidade adjudicante com morada desconhecida
involucao(adjudicante(ID,N,NIF,moradaDesconhecida)) :-
    involucao(adjudicante(ID,N,NIF,moradaDesconhecida)),
    remocao((excecao(adjudicante(IDAd,NAd,NIFAd,MAd)) :-
                adjudicante(IDAd,NAd,NIFAd,moradaDesconhecida))).

%Adjudicataria

% Insere conhecimento imperfeito impreciso na base de conhecimento
% no caso de uma entidade adjudicante com morada desconhecida entre
% duas moradas
evolucao(adjudicataria(ID,N,NIF,MD),M1,M2) :-
    insercao((excecao(adjudicataria(ID,N,NIF,M1)))),
    insercao((excecao(adjudicataria(ID,N,NIF,M2)))).

% Remove conhecimento imperfeito incerto na base de conhecimento
% no caso de uma entidade adjudicante com morada desconhecida
% duas moradas
involucao(adjudicataria(ID,N,NIF,M)) :-
    remocao((excecao(adjudicataria(ID,N,NIF,M)) :-
                adjudicataria(ID,N,NIF,M))).
```

Figura 4.5: Involução/Evolução de conhecimento imperfeito incerto e indeterminado.

```
%Adjudicante

% Insere conhecimento imperfeito incerto na base de conhecimento
% no caso de uma entidade adjudicante com morada desconhecida
evolucao(adjudicante(ID,N,NIF,moradaDesconhecida)) :-
    evolucao(adjudicante(ID,N,NIF,moradaDesconhecida)),
    insercao((excecao(adjudicante(IDAd,NAd,NIFAd,MAd)) :-
                adjudicante(IDAd,NAd,NIFAd,moradaDesconhecida))).

% Remove conhecimento imperfeito incerto na base de conhecimento
% no caso de uma entidade adjudicante com morada desconhecida
involucao(adjudicante(ID,N,NIF,moradaDesconhecida)) :-
    involucao(adjudicante(ID,N,NIF,moradaDesconhecida)),
    remocao((excecao(adjudicante(IDAd,NAd,NIFAd,MAd)) :-
                adjudicante(IDAd,NAd,NIFAd,moradaDesconhecida))).

%Adjudicataria

% Insere conhecimento imperfeito impreciso na base de conhecimento
% no caso de uma entidade adjudicante com morada desconhecida entre
% duas moradas
evolucao(adjudicataria(ID,N,NIF,MD),M1,M2) :-
    insercao((excecao(adjudicataria(ID,N,NIF,M1)))),
    insercao((excecao(adjudicataria(ID,N,NIF,M2)))).

% Remove conhecimento imperfeito incerto na base de conhecimento
% no caso de uma entidade adjudicante com morada desconhecida
% duas moradas
involucao(adjudicataria(ID,N,NIF,MD),M1,M2) :-
    remocao((excecao(adjudicataria(ID,N,NIF,M1)))),
    remocao((excecao(adjudicataria(ID,N,NIF,M2)))).
```

Figura 4.6: Involução/Evolução de conhecimento imperfeito interdito e impreciso.

```
% Remove conhecimento imperfeito interdio na base de conhecimento
involucao(contrato(AN,AT,contratoSecreto,TP,descricaoSecreta,ValorD,P,localizacaoSecreta,DT)) :-
    involucao(contrato(AN,AT,contratoSecreto,TP,descricaoSecreta,ValorD,P,localizacaoSecreta,DT)),
    remocao((excecao(contrato(ANC,ATC,TCC,TPC,DC,ValorDC,PC,LC,DTC)) :-
    remocao(AN,AT,contratoSecreto,TP,descricaoSecreta,ValorD,P,localizacaoSecreta,DT))),
    remocao((nuloInterdito(contratoSecreto))),
    remocao((nuloInterdito(descricaoSecreta))),
    remocao((nuloInterdito(localizacaoSecreta))).
```

Figura 4.7: Involução de conhecimento imperfeito interdito.

# 5. Análise de Resultados

Aqui vamos demonstrar o funcionamento do projecto no interpretador utilizado (Sicstus).

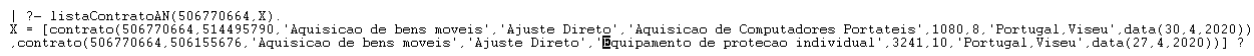


Figura 5.1: Query que mostra a lista de contratos de um adjudicante.

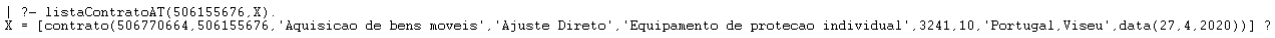


Figura 5.2: Query que mostra a lista de contratos de uma adjudicatária.

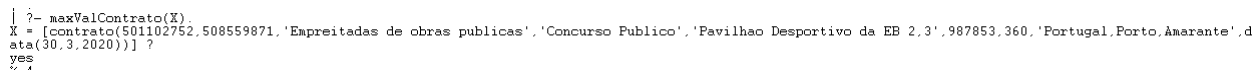


Figura 5.3: Query que calcula o/os contrato/os com o maior valor.

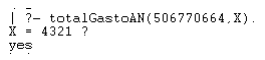


Figura 5.4: Query que calcula a soma dos valores gastos por um adjudicante em todos os seus contratos.

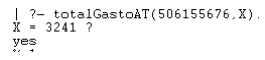


Figura 5.5: Query que calcula a soma dos valores recebidos por uma adjudicataria em todos os seus contratos.

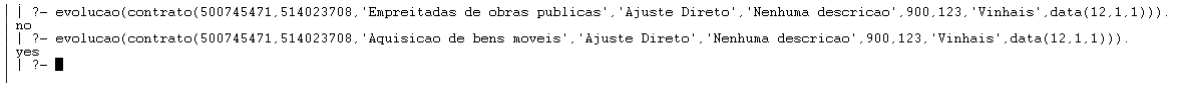


Figura 5.6: Demonstração que contratos por ajusto direto são de um dos 3 tipos de procedimentos .

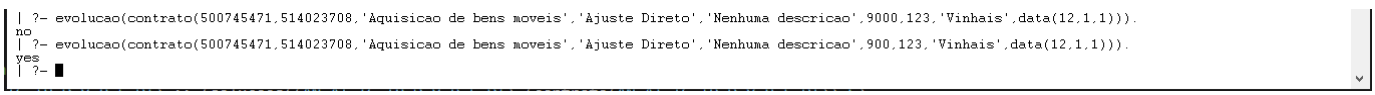


Figura 5.7: Demonstração que contratos por ajusto direto possuem um valor contratual inferior a 5000.

```
| ?- evolucao(contrato(501102752,508559871,'Aquisicao de bens moveis','Ajuste Direto','Nenhuma descricao',987,120,'Portugal,Porto,Amarante',data(30,3,2029))).
yes
| ?- evolucao(contrato(501102752,508559871,'Aquisicao de bens moveis','Ajuste Direto','Nenhuma descricao',987,400,'Portugal,Porto,Amarante',data(30,3,2038))).
no
```

Figura 5.8: Demonstração que contratos por ajusto direto possuem um prazo contratual inferior a 1 ano.

# 6. Conclusões e Sugestões

No final deste exercício, a equipa conclui que foi desenvolvido com sucesso um sistema de caracterização de um universo na área da contratação pública.

Foram ainda implementadas todas as funcionalidades solicitadas que permitem uma correta manipulação da base de conhecimento (quer adição quer remoção de conhecimento) e a aplicação de uma série de queries sobre ela.

A boa preparação da equipa 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. Referências

Cesar Analide, José Neves, "Representação de Informação Incompleta", Texto Pedagógico, 2010.

Contratos Públicos Online disponíveis em:<http://www.base.gov.pt/Base/pt/ResultadosPesquisa?type= contratosquery=tipoAcesso em 30 abr.2020.

Código dos Contratos Públicos disponíveis em:<http://www.base.gov.pt/mediaRep/inci/files/ccp2018/CCP_consolidado_com_LEO_DL_33_2018.pdf>. Acesso em 10 mai.2020.

# 8. Anexos

## 8.1 Funções Auxiliares

Para o desenvolvimento dos procedimentos requeridos pelo nosso sistema de representação de conhecimento e raciocínio, recorremos a várias funções auxiliares que foram extremamente úteis no decorrer de todo o processo.

Em seguida, apresentamos essas funções desenvolvidas, que fomos referindo ao longo do desenvolvimento do presente relatório.

O meta-predicado **soluções** que coloca em Z a lista com as soluções do tipo X para todo o Y que encontrar.

```
solucoes( X,Y,Z ) :- findall( X,Y,Z ).
```

O predicado **comprimento**, que coloca em N o comprimento da lista passada como argumento.

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

O meta-predicado **inserção** que coloca Termo na base de conhecimento no caso de sucesso, retornando yes, e retira Termo no caso de haver retrocesso, retornando no. O metapredicado remoção, que faz o oposto do meta-predicado inserção, ou seja, elimina Termo da base de conhecimento no caso de sucesso e adiciona Termo no caso de haver retrocesso, retornando no.

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

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

O meta-predicado **teste**, que testa se todos os predicados passados como parâmetro são verdadeiros.

```
teste( [] ).
teste( [R|LR] ) :- R, teste( LR ).
```

O meta-predicado **evolução**, responsável por adicionar novo conhecimento na base de conhecimento, verificando se todos os invariantes estruturais e referenciais de adição do conhecimento que se pretende adicionar (Termo) continuam verdadeiros. Só em caso de sucesso é que o conhecimento fica efectivamente registado, em caso contrário, o conhecimento inicialmente inserido é removido. Foi também preciso adicionar uma versão da evolução para a inserção de conhecimento negativo.

```
evolucao( Termo ) :-
    solucoes( Invariante,+Termo::Invariante,Lista ),
    insercao( Termo ),
    teste( Lista ).
```

```
evolucao( -Termo) :-
    solucoes( Invariante,+(-Termo)::Invariante,Lista ),
    insercao( -Termo ),
    teste( Lista ).
```

O meta-predicado **involução**, responsável por remover conhecimento da base de conhecimento, verificando se todos os invariantes estruturais e referenciais de remoção do conhecimento que se pretende remover (Termo) continuam verdadeiros. Só em caso de sucesso é que o conhecimento é permanentemente removido, em caso contrário, o conhecimento é novamente inserido. Foi também preciso adicionar uma versão da involução para a remoção de conhecimento negativo.

```
involucao( Termo ) :-
    solucoes( Invariante,-Termo::Invariante,Lista ),
    remocao( Termo ),
    teste( Lista ).

involucao( -Termo ) :-
    solucoes( Invariante,-(-Termo)::Invariante,Lista ),
    remocao( -Termo ),
    teste( Lista ).
```

1. Testa se um número é par.

```
% Extensao do predicado par: X -> {V,F}
par(0).
par(X) :- NX is X-2,NX >= 0,par(NX).
```

2. Testa se um número é ímpar.

```
% Extensao do predicado impar: X -> {V,F}
impar(1).
impar(X) :- nao(par(X)).
```

3. Testa se A é um ano bissexto.

```
% Extensao do predicado isB6: X -> {V,F}
isB6(0).
isB6(A) :- 0 is mod(A,4).
```

4. Calcula o tempo da fidelização de um contrato. Não podemos remover um contrato cuja fidelização ainda não tenha passado.

```
% Extensao do predicado fidelizacao: X, Y -> {V,F}
fidelizacao(X,R) :- R is 0.3 * X.
```

5. Conta o número de algarismos de um número.

```
% Extensao do predicado conta número de digitos de um número: X -> {V,F}
countDigits(0,0) :- !.
countDigits(X,N) :- NX is div(X,10) ,NX >= 0,countDigits(NX,NN), N is NN + 1.
```

6. Calcula se a primeira data é depois da segunda.

```
% Extensao do predicado isAfter: data(X,Y,Z), data(W,Q,S) -> {V,F}
isAfter(data(_,_,A1),data(_,_,A2)) :- A1 > A2.
isAfter(data(_,M1,A),data(_,M2,A)) :- M1 > M2.
isAfter(data(D1,M,A),data(D2,M,A)) :- D1 > D2.
```

7. Calcula o data atual.

```
% Extensao do predicado datimeToData: data(X,Y,Z) -> {V,F}
datimeToData(data(D,M,A)) :- datime(datime(A,M,D,_,_,_)).
```

8. Novo predicado data.

```
% Extensao do predicado data: X, Y, Z -> {V,F}
data(D,2,A) :- D > 0, 0 < A,(isB6(A)) -> D =< 29; D =< 28.
data(D,M,A) :- M \= 2,M > 0, M =< 12,D > 0, D =< 31, 0 < A.
```

9. Testa a validade de uma data.

```
% Extensao do predicado isData: data(D,M,A) -> {V,F}
isData(data(D,M,A)) :- data(D,M,A).
```

10. Dada uma data inicial e um número de dias, calcula a data resultante da soma da data inicial com o número de dias.

```
% Extensao do predicado avancaDias: X, Y, Z -> {V,F}
avancaDias(_,0,_).
avancaDias(data(D,M,A),Dias,data(Ds,Ms,As)) :-  ((2 is M,isB6(A)) -> ((D + Dias > 29) -> (avancaDias(data(1,3,A),Dias - (29 - D + 1),data(Ds,Ms,As)));
                                                                                          (Ds is (D + Dias), Ms is M, As is A) ) );
                                                 ((2 is M,par(M),nao(isB6(A) )) -> ((D + Dias > 28) -> (avancaDias(data(1,3,A),Dias - (28 - D + 1),data(Ds,Ms,As)));
                                                                                          (Ds is (D + Dias), Ms is M, As is A)) );
                                                ((M =< 7,impar(M)) -> ( (D + Dias > 31) ->  (avancaDias(data(1,M + 1,A),Dias - (31 - D + 1),data(Ds,Ms,As)));
                                                                                  (Ds is (D + Dias), Ms is M, As is A) ) );
                                                ((M > 2,M =< 7,par(M)) ->  ( (D + Dias > 30) -> (avancaDias(data(1,M + 1,A),Dias - (30 - D + 1),data(Ds,Ms,As)));
                                                                                   (Ds is (D + Dias), Ms is M, As is A) ) );
                                                ((M > 12 ) -> avancaDias(data(1,1,A+1),Dias,data(Ds,Ms,As) ) );
                                                ((M >= 8,impar(M)) -> ( (D + Dias > 30 ) ->  (avancaDias(data(1,M + 1,A),Dias - (30 - D + 1),data(Ds,Ms,As)));
                                                                                  (Ds is(D + Dias), Ms is M, As is A) ) );
                                                ((M >= 8,par(M)) ->  ((D + Dias > 31 ) -> (avancaDias(data(1,M + 1,A),Dias - (31 - D + 1),data(Ds,Ms,As)));
                                                                                  (Ds is (D + Dias), Ms is M, As is A) ))  .
```

Figura 8.1: Função avancaDias auxiliar para invariante

11. Apaga todas as ocorrências de X numa lista.

```
% Extensao do predicado apagaT: [H|T] -> {V,F}
apagaT(_,[],[]).
apagaT(X,[X|T],L) :- apagaT(X,T,L).
apagaT(X,[H|T],[H|L]) :- X \= H, apagaT(X,T,L).
```

12. Calcula o máximo de uma lista.

```
% Extensao do predicado maxL: [H|T] -> {V,F}
maxL([H],H) :- !. % Bang serve como paragem de modo a evitar ciclo infinito
maxL([H|T],M) :- maxL(T,M), M >= H. % Continua o M pois M > H
maxL([H|T],H) :- maxL(T,M), H > M. % Substitui H pelo M pois H > M
```

13. Calcula a soma de uma lista.

```
% Extensao do predicado somaList: [H|T] -> {V,F}
somaList([],0).
somaList([Head|Tail],Sum) :- somaList(Tail,NSum),Sum is Head+NSum.
```

14. Calcula a diferença de anos.

```
% Extensao do predicado difAno: X, Y, Z, W, Q, S -> {V,F}
difAno(AnoEconomico,AN,AT,TC,D,C) :-
    solucoes(Ano,contrato(AN,AT,TC,_,D,_,_,_,data(_,_,Ano)),S), maxL(S,J), C is AnoEconomico - J.
```

15. Calcula a acumulação de preços de um contrato

```
% Extensao do predicado acumulaContrato: X, Y, Z -> {V,F}
acumulaContrato(AN,AT,C) :- solucoes(V,contrato(AN,AT,_,_,_,V,_,_,_),S),somaList(S,C).
```