Relatório do Exercício 1

Carolina Cunha (a80142) Hugo Faria (a81283)
João Diogo Mota (a80791) Rodolfo Silva (a81716)

Braga, abril de 2020

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Sistemas de Representação de Conhecimento e Raciocínio
(2º Semestre/2019-2020)
Grupo 13

Resumo

Este projeto teve como objetivo o aperfeiçoamento da utilização da 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.

Através da programação em lógica, foi possível criar uma base de conhecimento relativa a uma área de contratação pública para a realização de contratos para a prestação de serviços.

Conteúdo

Lista de Figuras

1 Introdução

O principal objetivo deste projeto consistiu no aperfeiçoamento da utilização da 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.

O tema em estudo é a área da contratação pública para a realização de contratos para a prestação de serviços.

Assim, espera-se que o grupo construa um caso prático de aplicação dos conhecimentos, que seja capaz de demonstrar as funcionalidades subjacentes à programação em lógica estendida e à representação de conhecimento imperfeito, recorrendo à temática dos valores nulos.

2 Preliminares

Previamente à realização deste trabalho, foram estudados dois princípios imprescindíveis ao entendimento da programação em lógica.

Assim sendo, apresentamos o Pressuposto do Mundo Fechado (PMF), onde se alicerçam as linguagens de manipulação de informação numa Base de Dados. Deste modo, assumimos os seguintes pressupostos:

- **Pressuposto do Mundo Fechado** - toda a informação que não existe mencionada na base de dados é considerada falsa;
- **Pressuposto dos Nomes Únicos** - duas constantes diferentes 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.

No entanto, pretende-se que seja possível assumir que existe informação não representada que seja válida, bem como a existência de entidades no mundo exterior não representadas. Por este motivo, um sistema de representação de conhecimento pretende basear-se nos pressupostos que se seguem.

- **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 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.

Uma extensão de um programa em lógica pode incluir informação negativa explicitamente, bem como explicitar directamente o PMF para alguns predicados. Consequentemente, torna-se possível distinguir três tipos de conclusões para uma questão: esta pode ser **verdadeira**, **falsa** ou, quando não existe informação que permita inferir uma ou outra das conclusões anteriores, a resposta à questão será **desconhecida**. [AN]

3 Breve Descrição do Enunciado e Análise de Resultados

De seguida, será discutido todo o processo de resolução do projeto proposto pelos docentes da unidade curricular em questão.

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

- adjudicante: $\#IdAd, Nome, NIF, Morada \rightarrow \{V, F, D\}$
- adjudicataria: $\#IdAda, Nome, NIF, Morada \rightarrow \{V, F, D\}$
- contrato: $\#IdC, \#IdAd, \#IdAda, TipoDeContrato,$ $TipoDeProcedimento, Descricao, Valor, Prazo, Local, Data$ $\rightarrow \{V, F, D\}$

3.1 Inserção de Conhecimento

De modo a inserir o conhecimento, foram construídos três tipos de predicados distintos, que contribuíram para a obtenção de respostas às questões realizadas.

```
adjudicante(1,maria,123,rua_1_porto).
adjudicante(2,edp,987,rua_26_lisboa).
adjudicante(3,telecomunicacoes_X,135,travessa_45_braga).
adjudicante(4,empresa_Y,246,praca_X_lisboa).
excecao(adjudicante(11,romulus_gama,777,praceta_Z)).
excecao(adjudicante(11,romulus_gama,777,praceta_K)).
-adjudicante(5,ana,144, rua_yyz_caminha).
-adjudicante(6,trefl,222, rua_yyz_lisboa).
```

Inserção de conhecimento relativo a Adjudicantes.

```
adjudicataria(5,amelia,764,rua_992_porto).
adjudicataria(6,universidade_minho,967,rua_25_braga).
adjudicataria(7,hospital_X,762,rua_37_porto).
adjudicataria(8,manuel,016,travessa_123_lisboa).
adjudicataria(9,nobunaga,222,x666).
excecao(adjudicataria(IdAda,Nome,Nif,Morada)) :- adjudicataria(IdAda,Nome,Nif,x666).
-adjudicataria(10,antonio,922,rua_2340_viana).
-adjudicataria(11,morillo,113,travessa_9_lisboa).
```

Inserção de conhecimento relativo a Adjudicatárias.

```
contrato(1,1,5,aquisicao_servicos,ajuste_direto,arrendamento_de_habitacao,4500,730,porto,02-03-2019).
contrato(2,2,6,lococao_bens_moveis,consulta_previa,aquisicao_paineis_solares,5000,200,braga,02-03-2018).
contrato(3,3,7,aquisicao_bens_moveis,ajuste_direto,estabelecimento_de_comunicacoes,1500,150,porto,02-03-2022).
contrato(4,4,8,lococao_bens_moveis,consulta_previa,contratacao_de_servicos,500,257,lisboa,02-03-2021).
contrato(5,1,8,aquisicao_servicos,ajuste_direto,prestacao_de_servicos,2690,360,lisboa,02-03-2020).
contrato(6,3,6,aquisicao_bens_moveis,ajuste_direto,estabelecimento_de_comunicacoes,3200,355,braga,02-03-2021).
contrato(7,2,8,aquisicao_servicos,concurso_publico,fornecimento_de_energia,5000,730,lisboa,02-03-2018).
contrato(8,3,5,aquisicao_bens_moveis,concurso_publico,fornecimento_de_energia,10000,365,porto,02-03-2019).
contrato(9,4,5,aquisicao_servicos,consulta_previa,contratato_de_trabalho,9100,200,porto,02-03-2017).
contrato(10,2,8,lococao_bens_moveis,ajuste_direto,fornecimento_de_energia,1140,100,lisboa,02-03-2020).
contrato(11,1,5,lococao_bens_moveis,concurso_publico,arrendamento_de_habitacao,2250,365,porto,02-03-2021).
contrato(12,11,9,aquisicao_servicos,concurso_publico,null1,8920,846,minho,03-04-2019).
nulo(null1).
```

Inserção de conhecimento relativo a Contratos.

Nesta inserção, é necessário tomar atenção ao registo de cada entidade, de modo a garantir a coerência de dados na base de conhecimento. Para este fim, foram criados invariantes.

Os invariantes funcionam como uma prova sobre o predicado em questão, sendo utilizados tanto no momento de inserção como de remoção, garantindo assim a consistência dos dados.

Estes invariantes, utilizados posteriormente, acompanhados pelos predicados de evolução e involução, permitem a correta inserção/remoção dos dados.

```
adiciona(T) :- solucoes(Invariante, +T::Invariante, S),
               insere(T),
               testa(S).

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

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

Predicados auxiliares à inserção de conhecimento.

```
remove(T) :- solucoes( Invariante, -T::Invariante, S),
             remocao(T),
             testa(S).

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

Predicados auxiliares à remoção de conhecimento.

3.2 Invariantes

3.2.1 Invariantes Estruturais

Os invariantes que se seguem foram desenvolvidos com o objetivo de impedir a inserção de conhecimento repetido.

```
+adjudicante(_,Nome,Nif,Morada) :: (solucoes( (Id,Nome,Nif,Morada),(adjudicante(Id,Nome,Nif,Morada)),S ),
                                    comprimento( S,N ),
                                    N == 1).

+adjudicataria(_,Nome,Nif,Morada) :: (solucoes( (Id,Nome,Nif,Morada),(adjudicataria(Id,Nome,Nif,Morada)),S),
                                      comprimento( S,N ),
                                      N == 1).
```

Invariantes que não permitem a inserção de conhecimento repetido

3.2.2 Invariantes Referenciais

Após garantido que não serão adicionados dados redundantes, é crucial garantir a unicidade de cada conhecimento inserido. Deste modo, os invariantes que se seguem impedem a adição de novos adjudicantes/adjudicatárias com uma identificação idêntica a outra já atribuída.

```
+adjudicante(Id,_,_,_) :: (solucoes( (Id,Nome,Nif,Morada),(adjudicante(Id,Nome,Nif,Morada)),S),
                           comprimento( S,N ),
                           N == 1).

+adjudicataria(Id,_,_,_) :: (solucoes( (Id,Nome,Nif,Morada),(adjudicataria(Id,Nome,Nif,Morada)),S),
                             comprimento( S,N ),
                             N == 1).
```

Invariantes que garantem a unicidade do conhecimento: adjudicante e adjudicatária

No que diz respeito aos contratos celebrados, é também crucial a garantia de que estes possuam um meio de identificação único entre si. Desta forma, foi desenvolvido um invariante que impedisse a adição de um contrato cuja identificação já se encontra registada na base de conhecimento.

```
+contrato(IdC,_,_,_,_,_,_,_,_,_) :: (solucoes( (IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                      (contrato(IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data)),S),
                                      comprimento( S, N),
                                      N == 1).
```

Invariante que garante a unicidade do conhecimento: contratos

De seguida, confirmou-se ser necessária a garantia de que as entidades que celebram o contrato (adjudicante e adjudicatária) existem na base de conhecimento.

```
+contrato(IdC,IdAd,IdAda,_,_,_,_,_,_,_) :: (solucoes( (IdAd),(adjudicante(IdAd,Nome,Nif,Morada)),S1 ),
                                             solucoes( (IdAda),(adjudicataria(IdAda,Nom,Nf,Rua)),S2 ),
                                             comprimento( S1,N1 ),comprimento( S2,N2 ),
                                             N1 == 1, N2 == 1).
```

Invariante que garante a validade dos contratos

Estes invariantes recorrem ao auxílio dos predicados *solucoes* e *comprimento*, que colocam numa lista os dados obtidos através de uma filtragem por parâmetros indicados e o comprimento dessa lista, respetivamente.

```
% Predicado solucoes:
% Termo, Predicado, Lista -> { V, F }
solucoes( T,P,L ) :- findall( T,P,L ).

% Predicado comprimento:
% Lista, Comprimento -> { V, F }
comprimento([],0).
comprimento([H|T],N) :-
  comprimento(T,S),
  N is S+1.
```

Predicados soluções e comprimento

De notar que os contratos não podem ter um invariante que garanta a inexistência de conhecimento repetido, dado que duas entidades podem celebrar diversos contratos entre si.

De acordo com as indicações fornecidas no enunciado do projeto, é necessário ter em atenção as seguintes normas:

1. Tipos de Procedimento
   - Ajuste Direto;
   - Consulta Prévia;
   - Concurso Público.
2. 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.
3. Regra dos 3 anos válida para todos os contratos

- Uma entidade adjudicante não pode convidar a mesma empresa para celebrar um contrato com prestações de serviço do 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.

Assim, por forma a cumprir as normas indicadas, foram desenvolvidos os invariantes que se seguem.

Com o auxílio do predicado *pertence*, este primeiro invariante permite garantir que apenas os tipos de procedimentos indicados são válidos aquando da inserção de um contrato.

```
pertence( X,[X|L] ).
pertence( X,[Y|L] ) :- X \= Y, pertence( X,L ).
```

Predicado Pertence

```
+contrato(_,_,_,_,Procedimento,_,_,_,_,_) :: pertence(Procedimento,[ajuste_direto,consulta_previa,concurso_publico]).
```

Invariantes que garantem a norma dos tipos de procedimento

A segunda norma é garantida através do invariante

```
+contrato(_,_,_,TipoDeCt,ajuste_direto,_,Valor,Prazo,_,_) :: (Valor =< 5000,
                                                              Prazo =< 365,
                                                              pertence(TipoDeCt,[aquisicao_bens_moveis,lococao_bens_moveis,aquisicao_servicos])).
```

Invariantes que garantem a norma dos contratos por ajuste direto

Este invariante permite assegurar que, para um procedimento de ajuste direto, apenas os tipos de contrato aquisicão de bens móveis, locoção de bens móveis e aquisição de serviços são válidos; o valor contratual máximo é de 5000€ e o prazo do contrato é até um ano.

Por fim, a terceira norma é garantida com o auxílio dos predicados *preco*, *custoTotal*, *withinTime*, *within2Years* e *solucoes*.

```
% Predicado para obter os preços contratuais de contratos celebrados
custoTotal(IdAd,IdAda, S) :- solucoes((Valor,Data),(contrato(_,IdAd,IdAda,_,_,_,Valor,_,_,Data)), S).


within2Years((D1-M1-Y1),(D2-M2-Y2)) :- (Y1 =:= (Y2 + 2), M2 =:= M1, D2 >= D1 ;
                                        Y1 =:= (Y2 + 2), M2 >= M1 ;
                                        Y1 =:= (Y2+1) ;
                                        Y1 =:= Y2, M2 =< M1 ;
                                        Y1 =:= Y2, M2 =:= M1, D2 =< D1).

% Predicado que verifica se um contrato foi realizado num período máximo de 3 anos
withinTime(Data,[],S) :- S is 0.
withinTime(Data,[(Val,Dt)|T],S) :- (within2Years(Data,Dt) -> withinTime(Data,T,R), S is R + Val;
                                    withinTime(Data,T,S)).

% Predicado para obter os preços contratuais acumulados de contratos celebrados num período de 3 anos
preco( IdAd, IdAda, R, Data) :- custoTotal(IdAd,IdAda, S), withinTime(Data,S,R).
```

Predicados auxiliares ao invariante que garante a terceira norma

Analisando os predicados apresentados, tem-se que:

preco: Este predicado será responsável pela obtenção dos valores contratuais acumulados de contratos celebrados num período de três anos. Para que tal seja possível, é necessário recorrer a dois outros predicados.

cutoTotal: Predicado que obtém uma lista com todos valores e datas dos contratos realizados pelo adjudicante IdAd e pela adjudicatária IdAda.

withinTime: Após obtenção da lista com os valores e datas dos contratos, é crucial verificar se os contratos foram celebrados num período de três anos antes da celebração do contrato atual.

```
+contrato(_,IdAd,IdAda,_,_,_,Valor,_,_,Data) :: (preco(IdAd,IdAda,P,Data), R is P - Valor, R =< 74999).
```

Invariante que garante a terceira norma

No que diz respeito à remoção de conhecimento, foi considerada a impossibilidade de remoção de contratos cujas entidades contratuais pertençam à base de conhecimento, bem como a remoção de entidades com contratos previamente celebrados.

```
-contrato(_,IdAd,IdAda,_,_,_,_,_,_) :: (solucoes( (IdAd),(adjudicante(IdAd,Nome,Nif,Morada)),S1 ),
                                         solucoes( (IdAda),(adjudicataria(IdAda,Nom,Nf,Rua)),S2 ),
                                         comprimento( S1,N1 ),comprimento( S2,N2 ),
                                         N1 == 0, N2 == 0).
```

Invariante que garante correta remoção de contratos

```
-adjudicante(ID,_,_,_) :: (allContratos_Adjudicante( ID, S ),
                           comprimento( S,N ),
                           N == 0).

-adjudicataria(ID,_,_,_) :: (allContratos_Adjudicataria( ID, S ),
                             comprimento( S,N ),
                              N == 0).
```

Invariante que garante a correta remoção de entidades

3.3 Representação de Conhecimento Imperfeito

De forma a interpretar os resultados obtidos pela representação de conhecimento imperfeito, isto é, interpretar uma questão colocada a um sistema capaz de raciocinar em termos da Programação em Lógica Estendida, recorre-se à noção de que uma questão pode ter como resposta

- verdadeiro, se $\exists X : q(X)$
- falso, se $\exists X : \neg q(X)$
- desconhecido, se $\neg\exists X : q(X) \vee \neg q(X)$

Para que tal fosse possível, utilizou-se o predicado *demo*, como meta-interpretador deste tipo de conhecimento.

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

Extensão do meta-predicado demo

Por sua vez, este predicado é auxiliado pelo meta-predicado *nao*, que permite a negação por falha na prova de um conhecimento. Neste tipo de negação, é utilizado o *cut* (!), que oferece uma maneira de controlar o retrocesso, juntamente com a falha de predicado interno (fail).

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

Extensão do meta-predicado nao

Em adição, o meta-interpretador *demo* ainda faz uso da negação forte -. Deste modo, tem-se:

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

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

-contrato(IdC,ID,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data) :-
        nao(contrato(IdC,ID,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data)),
        nao(excecao(contrato(IdC,ID,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data))).
```

Predicados negação forte

Com o desenvolvimento das extensões dos predicados previamente apresentados, torna-se possível a noção de informação incompleta, podendo esta ser representada por três tipos de valores nulos:

- **Tipo I - Incerto**: Desconhecido, de um conjunto indeterminado de hipóteses;
- **Tipo II - Impreciso**: Desconhecido, mas de um conjunto determinado de hipóteses;
- **Tipo III - Interdito**: Desconhecido e não permitido conhecer.

Como exemplo de representação de conhecimento imperfeito pelo uso de valores nulos do tipo incerto, tem-se:

```
adjudicataria(9,nobunaga,222,x666).
excecao(adjudicataria(IdAda,Nome,Nif,Morada)) :- adjudicataria(IdAda,Nome,Nif,x666).
```

Representação de Conhecimento Imperfeito por Valores Nulos Tipo I

Analisando o exemplo dado, são conhecidos os dados da entidade adjudicataria com identificação 9, no entanto, desconhece-se qual a sua morada.

No que diz respeito à representação de conhecimento imperfeito pelo uso de valores nulos do tipo impreciso, tem-se:

```
excecao(adjudicante(11,romulus_gama,777,praceta_Z)).
excecao(adjudicante(11,romulus_gama,777,praceta_K)).
```

Representação de Conhecimento Imperfeito por Valores Nulos Tipo II

De acordo com os predicados apresentados, sabe-se que a morada do adjudicante é praceta_Z ou praceta_K, no entanto, desconhece-se qual destas é efetivamente a sua morada.

Por fim, relativamente à representação de conhecimento imperfeito pelo uso de valores nulos do tipo interdito, tem-se:

```
contrato(12,11,9,aquisicao_servicos,concurso_publico,null1,8920,846,minho,03-04-2019).
nulo(null1).
+contrato(IdC,_,_,_,_,_,_,_,_,_) :: (solucoes(
                    (Descricao),
                    (contrato(IdC,X1,X2,X3,X4,X5,X6,X7,X8,X9), nao(nulo(X5))),
                    S),
                    comprimento(S,N),
                    N == 0).
```

Representação de Conhecimento Imperfeito por Valores Nulos Tipo III

Pelo exemplo apresentado, confirma-se a impossibilidade de conhecer qual a descrição do contrato celebrado.

3.4 Resposta a Questões Colocadas

3.4.1 Identificar adjudicantes através de critérios de seleção

Para que seja identificado um adjudicante, a sua pesquisa pode ser realizada com base nos seguintes critérios: identificador, nome, NIF ou morada. O resultado de cada um dos predicados é obtido com auxílio do predicado *solucoes*.

```
% Predicado para identificar um adjudicante por ID;
adjudicante_ID( ID, S ) :- solucoes(
                                ( Nome, NIF, Morada ),
                                (adjudicante(ID, Nome, NIF, Morada)),
                                S
                                ).

% Predicado para identificar um adjudicante por Nome;
adjudicante_NOME( Nome, S ) :- solucoes(
                                    ( ID, NIF, Morada ),
                                    (adjudicante(ID, Nome, NIF, Morada)),
                                    S
                                    ).

% Predicado para identificar um adjudicante por NIF;
adjudicante_NIF( NIF, S ) :- solucoes(
                                ( ID, Nome, Morada ),
                                (adjudicante(ID, Nome, NIF, Morada)),
                                S
                                ).

% Predicado para identificar um adjudicante por Morada;
adjudicante_MORADA( Morada, S ) :- solucoes(
                                        ( ID, Nome, NIF ),
                                        (adjudicante(ID, Nome, NIF, Morada)),
                                        S
                                        ).
```

Identificar adjudicante por critérios de seleção

3.4.2 Identificar adjudicatária através de critérios de seleção

Do mesmo modo, pode recorrer-se à pesquisa de uma entidade adjudicatária através dos mesmos critérios de seleção.

```
% Predicado para identificar uma adjudicataria por ID;
adjudicataria_ID( ID, S ) :- solucoes(
                                ( Nome, NIF, Morada ),
                                (adjudicataria(ID, Nome, NIF, Morada)),
                                S
                                ).

% Predicado para identificar um adjudicante por Nome;
adjudicataria_NOME( Nome, S ) :- solucoes(
                                    ( ID, NIF, Morada ),
                                    (adjudicataria(ID, Nome, NIF, Morada)),
                                    S
                                    ).

% Predicado para identificar um adjudicante por NIF;
adjudicataria_NIF( NIF, S ) :- solucoes(
                                  ( ID, Nome, Morada ),
                                  (adjudicataria(ID, Nome, NIF, Morada)),
                                  S
                                  ).

% Predicado para identificar um adjudicante por Morada;
adjudicataria_MORADA( Morada, S ) :- solucoes(
                                        ( ID, Nome, NIF ),
                                        (adjudicataria(ID, Nome, NIF, Morada)),
                                        S
                                        ).
```

Identificar adjudicataria por critérios de seleção

3.4.3 Identificar contratos através de critérios de seleção

Por outro lado, a procura de um contrato pode ser apenas realizada através do seu identificador, uma vez que todos os restantes parâmetros podem ser idênticos em contratos distintos.

```
contrato_ID( ID, S ) :- solucoes(
                           (IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                           contrato(ID,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                           S
                           ).
```

Identificar contratos por identificador

3.4.4 Obtenção de todas as entidades existentes na base de conhecimento

Deve ser possível a obtenção de todas as entidades existentes na base de conhecimento, de modo a ter uma melhor percepção relativamente ao conhecimento existente.

```
% Predicado para obter todos os adjudicantes
allAdjudicantes( S ) :- solucoes( (Id,Nome,Nif,Morada),(adjudicante(Id,Nome,Nif,Morada)),S ).

% Predicado para obter todas as adjudicatarias
allAdjudicatarias( S ) :- solucoes( (Id,Nome,Nif,Morada),(adjudicataria(Id,Nome,Nif,Morada)),S ).

% Predicado para obter todos os contratos
allContratos( S ) :- solucoes(
                            (IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                            (contrato(IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data)),
                            S
                            ).
```

Obtenção das entidades existentes na base de conhecimento

3.4.5 Identificar contratos celebrados por adjudicantes/adjudicatarias

Os contratos celebrados por determinado adjudicante, adjudicataria ou ambos podem ser apresentados através dos seguintes predicados.

```
% Predicado para obter todos os contratos celebrados pelo adjudicante com id ID.
allContratos_Adjudicante( ID, S ) :- solucoes(
                                        (IdC,ID,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                        contrato(IdC,ID,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                        S
                                        ).

% Predicado para obter todos os contratos celebrados pela adjudicatária com id ID.
allContratos_Adjudicataria( ID, S ) :- solucoes(
                                          (IdC,IdAd,ID,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                          contrato(IdC,IdAd,ID,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                          S
                                          ).
% Predicado para obter todos os contratos celebrados entre o adjudicante IdAd e a adjudicataria IdAda.
contratos_partilhados( IdAd,IdAda,S ) :- solucoes(
                                            (IdC,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                            contrato(IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                            S
                                            ).
```

Identificação de contratos celebrados

3.4.6 Identificar contratos celebrados por tipo

Para cada adjudicante, adjudicataria ou na totalidade, é possível apresentar os contratos celebrados pelos mesmos, de acordo com o tipo de contrato celebrado.

```
% Predicado para obter todos os contratos de um determinado tipo celebrados por um adjudicante.
contratos_tipo_adjudicante( IdAd, TipoDeCt, S) :- solucoes(
                                                    (IdC,IdAda,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                                    contrato(IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                                    S
                                                    ).

% Predicado para obter todos os contratos de um determinado tipo celebrados por uma determinada adjudicataria.
contratos_tipo_adjudicataria( IdAda, TipoDeCt, S) :- solucoes(
                                                        (IdC,IdAd,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                                        contrato(IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                                        S
                                                        ).

% Predicado para obter todos os contratos de um determinado tipo celebrados.
contratos_tipo(TipoDeCt, S) :-  solucoes(
                                    (IdC,IdAd,IdAda,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                    contrato(IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                    S
                                    ).
```

Identificação de contratos celebrados por tipo

3.4.7 Identificar contratos celebrados por procedimento

É, também, possível apresentar todos os contratos celebrados de um certo procedimento.

```
contratos_procedimento(Procedimento, S) :- solucoes(
                                                (IdC,IdAd,IdAda,TipoDeCt,Descricao,Valor,Prazo,Data),
                                                contrato(IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Morada,Data),
                                                S
                                                ).
```

Identificação de contratos celebrados por procedimento

3.4.8 Identificar contratos celebrados por morada

De igual modo, consegue-se ainda apresentar todos os contratos celebrados numa certa morada.

```
contratos_morada(Morada, S) :- solucoes(
                                    (IdC,IdAd,IdAda,Procedimento,Descricao,Valor,Prazo,Data),
                                    contrato(IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Morada,Data),
                                    S
                                    ).
```

Identificação de contratos celebrados por morada

3.4.9 Identificar contratos celebrados por valor

Por fim, é possível apresentar todos os contratos celebrados por valor.

```
contratos_valor(Valor, S) :- solucoes(
                                (IdC,IdAd,IdAda,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                contrato(IdC,IdAd,IdAda,TipoDeCt,Procedimento,Descricao,Valor,Prazo,Local,Data),
                                S
                                ).
```

Identificação de contratos celebrados por valor

3.4.10 Obter valores contratuais

Através do predicado *soma*, torna-se possível obter os valores totais dos contratos celebrados ou por uma determinada entidade.

```
soma([],0).
soma([H|T], R) :- soma(T,P),
                  R is H + P.
```

Predicado soma

```
%Predicado para obter o valor total em contratos celebrados
valor_contratos(R) :- solucoes(
                            (Valor),
                            (contrato(_,_,_,_,_,_,Valor,_,_,_)),
                            S),
                            soma(S,R).


% Predicado para obter o valor total em contratos celebrados por um adjudicante.
valor_contratos_adjudicante(IdAd, R) :- solucoes(
                                            (Valor),
                                            (contrato(_,IdAd,_,_,_,_,Valor,_,_,_)),
                                            S),
                                        soma(S,R).

% Predicado para obter o valor total em contratos celebrados por uma adjudicataria.
valor_contratos_adjudicataria(IdAda, R) :- solucoes(
                                              (Valor),
                                              (contrato(_,_,IdAda,_,_,_,Valor,_,_,_)),
                                              S),
                                           soma(S,R).
```

Predicados para obtenção dos valores contratuais

3.4.11 Obter valores contratuais por tipo

No seguimento dos predicados anteriormente apresentados, é ainda possível obter os valores totais ou por uma determinada entidade dos contratos celebrados, por tipo de contrato. Assim, seguem-se os predicados, que recorrem a predicados auxiliares, *custoTotalTipo*, *custoTotalAdjudicante* e *custoTotalAdjudicante*, que armazenam numa lista todos os contratos de um determinado tipo, seja na sua totalidade, ou de uma determinada entidade.

```
% Predicado para obter o valor total em contratos de um certo tipo celebrados.
valor_contratos_tipo(TipoDeCt, S) :- custoTotalTipo(TipoDeCt, R), soma(R,S).

% Predicado para obter os valores contratuais para contratos de um certo tipo celebrados.
custoTotalTipo(TipoDeCt, S) :- solucoes(
                                    (Valor),
                                    (contrato(_,_,_,TipoDeCt,_,_,Valor,_,_,_)),
                                    S
                                    ).

% Predicado para obter o valor total em contratos de um certo tipo celebrados por um adjudicante.
valor_contratos_adjudicante_tipo(IdAd, TipoDeCt, S) :- custoTotalAdjudicante(IdAd, TipoDeCt, R), soma(R,S).

% Predicado para obter os valores contratuais para contratos de um certo tipo celebrados por um adjudicante.
custoTotalAdjudicante(IdAd, TipoDeCt, S) :- solucoes(
                                                (Valor),
                                                (contrato(_,IdAd,_,TipoDeCt,_,_,Valor,_,_,_)),
                                                S
                                                ).

% Predicado para obter o valor total em contratos de um certo tipo celebrados por uma adjudicataria.
valor_contratos_adjudicataria_tipo(IdAda, TipoDeCt, S) :- custoTotalAdjudicataria(IdAda, TipoDeCt, R), soma(R,S).

% Predicado para obter os valores contratuais para contratos de um certo tipo celebrados por uma adjudicataria.
custoTotalAdjudicataria(IdAda, TipoDeCt, S) :- solucoes(
                                                    (Valor),
                                                    (contrato(_,_,IdAda,TipoDeCt,_,_,Valor,_,_,_)),
                                                    S
                                                    ).
```

Predicados para obtenção dos valores contratuais por tipo

3.4.12 Obter média de valores contratuais

De modo a calcular o preço médio dos contratos celebrados (gerais ou por entidade), e recorrendo uma vez mais à função *solucoes*, fornecendo o identificador da entidade, quando assim desejado, procuram-se todos os contratos e armazenam-se os vários preços numa lista, para ser possível calcular a sua média posteriormente. Estes predicados recorrem, assim, ao auxílio do predicado *media*.

```
media([],0).
media(Lista,Media) :-
    soma(Lista,X),
    comprimento(Lista,L),
    Media is (div(X,L)).
```

Predicados média auxiliar

```
% Predicado para obter o valor médio dos contratos celebrados
valor_contratos_media(S) :- solucoes(
                                (Valor),
                                (contrato(_,_,_,_,_,_,Valor,_,_,_)),
                                R),
                            media(R,S).

% Predicado para obter o valor médio dos contratos celebrados por um adjudicante
valor_contratos_media_adjudicante(ID, S) :- solucoes(
                                                (Valor),
                                                (contrato(_,ID,_,_,_,_,Valor,_,_,_)),
                                                R),
                                            media(R,S).

% Predicado para obter o valor médio dos contratos celebrados por uma adjudicataria
valor_contratos_media_adjudicataria(ID, S) :- solucoes(
                                                  (Valor),
                                                  (contrato(_,_,ID,_,_,_,Valor,_,_,_)),
                                                  R),
                                              media(R,S).
```

Predicados para obtenção da média dos valores contratuais por tipo

3.4.13 Apresentar contratos celebrados desde X ano

Aproveitando a ideia da terceira norma indicada, o grupo decidiu ainda construir predicados capazes de apresentar todos os contratos celebrados desde um determinado ano. Para que tal fosse possível, recorreu aos predicados *allContratos*, *allContratos_ Adjudicante* e *allContratos_ Adjudicataria*, que armazenam numa lista todos os contratos celebrados, celebrados por adjudicante e celebrados por adjudicataria, respetivamente.

```
% Predicado que permite obter todos os contratos celebrados desde um certo ano
allContratosSince(Ano,R) :- allContratos(Lista), allContratosSinceAux(Ano,Lista,R).

allContratosSinceAux(Ano,[],[]).
allContratosSinceAux(Ano,[H|T],R) :- getYear(H,X), Ano=<X -> allContratosSinceAux(Ano,T,Sub), R = [H|Sub]; allContratosSinceAux(Ano,T,R).


%Predicado que permite obter todos os contratos celebrados por um adjudicante desde um certo ano
allContratosSinceAdj(Ano,IdA,R) :- allContratos_Adjudicante(IdA,Lista), allContratosSinceAux(Ano,Lista,R).

%Predicado que permite obter todos os contratos celebrados por uma adjudicataria desde um certo ano
allContratosSinceAdja(Ano,IdA,R) :- allContratos_Adjudicataria(IdA,Lista), allContratosSinceAux(Ano,Lista,R).
```

Predicados para apresentar os contratos celebrados desde um ano

3.4.14 Apresentar os três contratos com maior valor de um determinado tipo

```
%Predicado que permite obter os três contratos com maior valor de determinado tipo.
maisFaturou3(Tipo,R) :- solucoes(
                                (IdC,Valor),
                                (contrato(IdC,_,_,Tipo,_,_,Valor,_,_,_)),
                                S
                                ),
                                maisFaturou3Aux(S,P),
                                ordena(P,R).

%Predicado auxiliar para a obtenção dos três contratos com maior valor de determinado tipo.
maisFaturou3Aux(L,R) :- comprimento(L,X), X > 3 -> removeMenor(L,P), maisFaturou3Aux(P,R); R = L.

%Predicado que remove o menor duplo de uma lista
removeMenor([H|T],P) :- menorLista([H|T],R), R == H -> P = T; removeMenor(T,Y), P = [H|Y].

%Predicado que devolve o menor duplo de uma lista.
menorLista([A],A).
menorLista([H|T],R) :-
  menorLista(T,P),
  menor(P,H,R).

%Predicado que devolve o menor duplo entre dois duplos.
menor((X,X1),(Y,Y1),(R,R1)) :-
  X1 =< Y1 -> R is X, R1 is X1; R is Y,R1 is Y1.
```

Predicados para apresentar os três contratos com maior valor de um determinado tipo

De modo a permitir ter conhecimento sobre os contratos celebrados com maior valor contratual, foi pensado um predicado capaz de obter os três contratos celebrados com maior valor. Assim, o predicado *maisFaturou3* obterá a lista dos pares (Id,Valor) de todos os contratos celebrados, aplicando sobre esta um predicado auxiliar, seguido da ordenação crescente da lista resultante.

No que diz respeito ao predicado auxiliar *maisFaturou3Aux*, este vai recursivamente remover o menor par da lista, até que o seu comprimento seja menor ou igual a três.

```
%Predicado que ordena uma lista por ordem crescente.
ordena([X],[X]).
ordena([H|T],R) :-
ordena(T,P),
  acrescenta(H,P,R).

%Predicado que acrescenta um duplo a uma lista.
acrescenta(Num,[],[Num]).
acrescenta((A,B),[(X,Y)|T],R) :-
    B =< Y -> R = [(A,B),(X,Y)|T];
    acrescenta((A,B),T,P),
    R=[(X,Y)|P].
```

Predicados para ordenar uma lista por ordem crescente

3.4.15 Apresentar o contrato com maior valor de cada tipo de contratos celebrados

Por fim, e tendo em conta o predicado previamente calculado, o grupo achou por bem encontrar uma forma de obter, para cada tipo de contrato celebrado, aquele com maior valor contratual.

Desta forma, e recorrendo ao auxílio dos predicados auxiliares *getMaxTipo*, *max*, *getAllTipos*, foi possível trabalhar sobre a lista de todos os contratos celebrados, recolhendo os tipos de cada contrato, e recolhendo sobre estes, aqueles cujo valor contratual era o mais elevado.

```
%Predicado que obtém, para cada tipo de contrato, o contrato com maior valor contratual.
getMaioresTipo(R) :- allContratos(S), getAllTipos(S,[],Tipos), getMaioresTipo(Tipos,[],R).
getMaioresTipo([],Aux,R) :- R = Aux.
getMaioresTipo([H|T],[],R) :- getMaxTipo(H,Max), getMaioresTipo(T,[Max],R).
getMaioresTipo([H|T],[Ha|Ta],R) :- getMaxTipo(H,Max), getMaioresTipo(T,[Max,Ha|Ta],R).

%Predicado que obtém o contrato com maior valor contratual, de um determinado tipo.
getMaxTipo(Tipo,Solution) :- contratos_tipo(Tipo,List), max(List,(_,_,_,_,_,_,0,_,_,_),0,Solution).

%Predicado que calcula o maior valor contratual de uma lista de contratos.
max([],ContratoMaximo,Maximo,Solution) :- Solution = ContratoMaximo.
max([(IdC,IdA,IdAc,TipoC,TipoP,Desc,ValorC,Prazo,Local,Data)|T],ContratoMaximo,Maximo,Solution) :-
                                    ( ValorC >= Maximo -> max(T,(IdC,IdA,IdAc,TipoC,TipoP,Desc,ValorC,Prazo,Local,Data),ValorC,Solution);
                                    max(T,ContratoMaximo,Maximo,Solution) ).

%Predicado que obtém todos os tipos de contratos celebrados.
getAllTipos([],Aux,R) :- R = Aux.
getAllTipos([(_,_,_,TipoC,_,_,_,_,_,_)|T],[],R) :- getAllTipos(T,[TipoC],R).
getAllTipos([(_,_,_,TipoC,_,_,_,_,_,_)|T],[H|Tl],R) :- ( pertence(TipoC,[H|Tl]) -> getAllTipos(T,[H|Tl],R);
                                                          getAllTipos(T,[TipoC,H|Tl],R)).
```

Predicados para obter o contrato com maior valor de cada tipo de contratos celebrados

4 Conclusões e Sugestões

O desenvolvimento deste projeto permitiu um aperfeiçoamento do conhecimento quanto ao paradigma da programação em lógica, fortalecendo a noção da sua utilidade na resolução de diversos problemas.

Com base nas indicações fornecidas, foi possível o desenvolvimento de alguns invariantes cruciais à concretização do projeto, permitindo praticar o uso dos mesmos e conhecer a sua necessidade de utilização.

Através de questões que o grupo achou pertinentes colocar, foram desenvolvidos predicados que permitissem a sua resposta da forma mais correta e eficaz.

Por fim, a realização deste exercício possibilitou o melhor entendimento da representação de conhecimento imperfeito, bem como dos três tipos de valores nulos leccionados.

5 Referências

Referências

[AN] Cesar Analide and José Novais. Representação de informação incompleta.

6 Anexos

Anexo I - Demonstração de Predicados

```
| ?- allAdjudicantes(S).
S = [(1,maria,123,rua_1_porto),(2,edp,987,rua_26_lisboa),(3,telecomunicacoes_X,135,travessa_45_braga),(4,empresa_Y,246,praca_X_lisboa)] ?
yes
| ?- allAdjudicatarias(S).
S = [(5,amelia,764,rua_992_porto),(6,universidade_minho,967,rua_25_braga),(7,hospital_X,762,rua_37_porto),(8,manuel,16,travessa_123_lisboa)] ?
yes
| ?- allContratos(S).
S = [(1,1,5,aquisicao_servicos,ajuste_direto,arrendamento_de_habitacao,4500,730,...,...),(2,2,6,lococao_bens_moveis,consulta_previa,aquisicao_pain
eis_solares,5000,200,...,...),(3,3,7,aquisicao_bens_moveis,ajuste_direto,estabelecimento_de_comunicacoes,1500,...,...),(4,4,8,lococao_bens_moveis,
consulta_previa,contratacao_de_servicos,...,...),(5,1,8,aquisicao_servicos,ajuste_direto,...,...),(6,3,6,aquisicao_bens_moveis,...,...),(7,2,8,...
,...),(8,3,...,...),(9,...,...),(...,...)|...] ?
yes
```

```
| ?- adjudicante_ID(1,S).
S = [(maria,123,rua_1_porto)] ?
yes
| ?- adjudicante_NOME(maria,S).
S = [(1,123,rua_1_porto)] ?
yes
| ?- adjudicante_NIF(123,S).
S = [(1,maria,rua_1_porto)] ?
yes
| ?- adjudicante_MORADA(rua_1_porto,S).
S = [(1,maria,123)] ?
yes
```

```
| ?- allContratos_Adjudicante(1,S).
S = [(1,1,5,aquisicao_servicos,ajuste_direto,arrendamento_de_habitacao,4500,730,...,...),(5,1,8,aquisicao_servicos,ajuste_direto,prestacao_de_servicos,26
90,360,...,...),(11,1,5,lococao_bens_moveis,concurso_publico,arrendamento_de_habitacao,2250,...,...)] ?
yes
| ?- allContratos_Adjudicataria(5,S).
S = [(1,1,5,aquisicao_servicos,ajuste_direto,arrendamento_de_habitacao,4500,730,...,...),(8,3,5,aquisicao_bens_moveis,concurso_publico,fornecimento_de_en
ergia,10000,365,...,...),(9,4,5,aquisicao_servicos,consulta_previa,contratato_de_trabalho,9100,...,...),(11,1,5,lococao_bens_moveis,concurso_publico,arre
ndamento_de_habitacao,...,...)] ?
yes
| ?- contratos_partilhados(1,5,S).
S = [(1,aquisicao_servicos,ajuste_direto,arrendamento_de_habitacao,4500,730,porto,... - ... -2019),(11,lococao_bens_moveis,concurso_publico,arrendamento_
de_habitacao,2250,365,porto,... - ... -2021)] ?
yes
```

```
| ?- contrato_ID(1,S).
S = [(1,5,aquisicao_servicos,ajuste_direto,arrendamento_de_habitacao,4500,730,porto,... - ...)] ?
yes
| ?- contratos_tipo_adjudicante(1,aquisicao_servicos,S).
S = [(1,5,ajuste_direto,arrendamento_de_habitacao,4500,730,porto,... - ... -2019),(5,8,ajuste_direto,prestacao_de_servicos,2690,360,lisboa,... - ... -2020)] ?
yes
| ?- contratos_tipo_adjudicataria(6,aquisicao_bens_moveis,S).
S = [(6,3,ajuste_direto,estabelecimento_de_comunicacoes,3200,355,braga,... - ... -2021)] ?
yes
| ?- contratos_tipo(aquisicao_bens_moveis,S).
S = [(3,3,7,ajuste_direto,estabelecimento_de_comunicacoes,1500,150,porto,... - ...),(6,3,6,ajuste_direto,estabelecimento_de_comunicacoes,3200,355,braga,... - ..
.),(8,3,5,concurso_publico,fornecimento_de_energia,10000,365,...,...)] ?
yes
```

```
| ?- contratos_valor(5000,S).
S = [(2,2,6,consulta_previa,aquisicao_paineis_solares,5000,200,braga,... - ...),(7,2,8,concurso_publico,fornecimento_de_energia,5000,730,lisboa,... - ...)] ? c
 yes
| ?- contratos_morada(lisboa,S).
S = [(4,4,8,consulta_previa,contratacao_de_servicos,500,257,... - ... -2021),(5,1,8,ajuste_direto,prestacao_de_servicos,2690,360,... - ... -2020),(7,2,8,concur
so_publico,fornecimento_de_energia,5000,730,... - ...),(10,2,8,ajuste_direto,fornecimento_de_energia,1140,...,...)] ?
yes
| ?- contratos_procedimento(consulta_previa,S).
S = [(2,2,6,lococao_bens_moveis,aquisicao_paineis_solares,5000,200,... - ... -2018),(4,4,8,lococao_bens_moveis,contratacao_de_servicos,500,257,... - ... -2021)
,(9,4,5,aquisicao_servicos,contratato_de_trabalho,9100,200,... - ...)] ?
yes
```

```
| ?- valor_contratos_tipo(aquisicao_servicos,S).
S = 21290 ?
yes
| ?- valor_contratos_adjudicante_tipo(1,aquisicao_servicos,S).
S = 7190 ?
yes
| ?- valor_contratos_adjudicataria_tipo(5,aquisicao_servicos,S).
S = 13600 ?
yes
```

```
| ?- valor_contratos(S).
S = 44880 ?
yes
| ?- valor_contratos_adjudicante(1,S).
S = 9440 ?
yes
| ?- valor_contratos_adjudicataria(5,S).
S = 25850 ?
yes
```

```
| ?- valor_contratos_media(S).
S = 4080 ?
yes
| ?- valor_contratos_media_adjudicante(1,S).
S = 3146 ?
yes
| ?- valor_contratos_media_adjudicataria(5,S).
S = 6462 ?
yes
```

```
| ?- allContratosSince(2020,S).
S = [(3,3,7,aquisicao_bens_moveis,ajuste_direto,estabelecimento_de_comunicacoes,1500,150,...,...),(4,4,8,lococao_bens_moveis,consulta_previa,contratacao_de_servicos,500,257,...,...),(5,1,8,aquisicao_servicos,ajuste_direto,prestacao_de_servicos,2690,...,...),(6,3,6,aquisicao_bens_moveis,ajuste_direto,estabelecimento_de_comunicacoes,...,...),(10,2,8,lococao_bens_moveis,ajuste_direto,...,...),(11,1,5,lococao_bens_moveis,...,...)] ?
yes
| ?- allContratosSinceAdj(2020,1,S).
S = [(5,1,8,aquisicao_servicos,ajuste_direto,prestacao_de_servicos,2690,360,...,...),(11,1,5,lococao_bens_moveis,concurso_publico,arrendamento_de_habitacao,2250,365,...,...)] ?
yes
| ?- allContratosSinceAdja(2020,5,S).
S = [(11,1,5,lococao_bens_moveis,concurso_publico,arrendamento_de_habitacao,2250,365,...,...)] ?
yes
```

```
no
| ?- adiciona(adjudicante(1,maria,123,rua_1_porto)).
no
| ?- adiciona(adjudicante(5,maria,123,rua_1_port)).
yes
| ?- adiciona(adjudicataria(5,amelia,764,rua_992_porto)).
no
| ?- adiciona(adjudicante(1,maria,123,rua_1_porto)).
no
| ?- adiciona(adjudicante(5,maria,123,rua_1_porto)).
no
| ?- adiciona(adjudicataria(9,amelia,764,rua_992_porto)).
no
| ?- remove(adjudicataria(5,amelia,764,rua_992_porto)).
no
| ?- remove(adjudicante(1,maria,123,rua_1_porto)).
no
```

```
| ?- adiciona(contrato(1,1,5,aquisicao_servicos,ajuste_direto,arrendamento_de_habitacao,4500,730,porto,02-03-2019)).
no
| ?- remove(contrato(1,1,5,aquisicao_servicos,ajuste_direto,arrendamento_de_habitacao,4500,730,porto,02-03-2019)).
no
| ?- adiciona(contrato(1,1,5,aquisicao_servicos,procedimento,arrendamento_de_habitacao,4500,730,porto,02-03-2019)).
no
| ?- adiciona(contrato(12,3,7,aquisicao_bens_moveis,ajuste_direto,estabelecimento_de_comunicacoes,6000,400,porto,02-03-2022)).
no
```

```
| ?- maisFaturou3(aquisicao_servicos,S).
S = [(7,5000),(12,8920),(9,9100)] ?
yes
| ?- maisFaturou3(lococao_bens_moveis,S).
S = [(10,1140),(11,2250),(2,5000)] ?
yes
| ?- maisFaturou3(aquisicao_bens_moveis,S).
S = [(3,1500),(6,3200),(8,10000)] ?
yes
```

```
| ?- getMaioresTipo(R).
R = [(9,4,5,aquisicao_servicos,consulta_previa,contratato_de_trabalho,9100,200,...,...),(2,2,6,lococao_bens_moveis,consulta_previa,aquisicao_paineis_solares,5000,200,...,...),(8,3,5,aquisicao_bens_moveis,concurso_publico,fornecimento_de_energia,10000,...,...)] ?
yes
```