Universidade do Minho
Escola de Engenharia - Departamento de Informática

Relatório do Trabalho Prático

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Grupo 23

A76936 – Luís Ferreira

A84610 – Gonçalo Almeida

A84776 – Emanuel Rodrigues

A86788 – Lázaro Pinheiro

Braga, abril 2020

Resumo

Este relatório tem como objetivo demonstrar os procedimentos utilizados na construção de um sistema de representação de raciocínio, utilizando o pressuposto de mundo aberto para a representação dos diferentes tipos de conhecimento.

Começamos por apresentar uma pequena fundamentação teórica dos conceitos nos quais as funcionalidades propostas se baseiam.

Em seguida, é apresentada a abordagem da resolução de cada uma das funcionalidades, recorrendo a imagens ilustrativas dos predicados utilizados.

Por fim, na secção Conclusões e Sugestões, é feita uma pequena reflexão sobre o trabalho realizado, de uma forma geral, assim como as dificuldades sentidas ao longo da sua realização.

É importante referir também que na secção Anexos podemos encontrar todos os predicados auxiliares utilizados no desenvolvimento das funcionalidades.

Tabela de Conteúdos

Índice de figuras

Introdução

Este trabalho tem como objetivo aprofundar o conhecimento da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, através do desenvolvimento de um sistema de representação de conhecimento perfeito e imperfeito. Este trabalho serve-se da linguagem de programação em lógica PROLOG para representar o conhecimento e raciocínio na área de contratação pública, recorrendo a contratos para prestação de serviços.

Partindo do enunciado começamos com os predicados: adjudicante, adjudicatária e contrato. O objetivo será introduzir no sistema os diferentes tipos de conhecimento através das mecânicas estudadas na disciplina de SRCR.

Preliminares

Para a realização deste trabalho prático é necessário conhecer a linguagem de programação PROLOG, competência esta adquirida pelos elementos do grupo no decorrer das aulas de Sistemas de Representação de Conhecimento e Raciocínio, bem como os seus conhecimentos teóricos consolidados, para que sejam aprofundados com a realização do mesmo. Ainda é fundamental possuir conhecimentos sobre o pressuposto de mundo aberto, conhecimento negativo, conhecimento imperfeito (incerto, impreciso e interdito) e valores nulos.

De modo a perceber o problema proposto pelo enunciado do trabalho, fez-se uma análise detalhada e minuciosa do mesmo, de modo a ter uma correta interpretação do que era pretendido e com a finalidade de obter o conhecimento necessário para se conseguir enveredar pela melhor resolução. Foi ainda consultada a bibliografia disponibilizada pelos docentes de modo a aprimorar o conhecimento da temática proposta.

Após a identificação de uma solução adequada, foi implementada utilizando a linguagem de programação em lógica PROLOG.

Os fundamentos teóricos estudados para realizar o presente trabalho prático foram:

- Facto;
- Regra;
- Pressuposto do Mundo Fechado;
- Inserção e remoção da base de conhecimento;
- Invariante e sua estrutura;
- Pressuposto do Mundo aberto;
- Extensão à Programação em Lógica:
    - Conhecimento Imperfeito;
    - Sistema de Inferência;

Descrição do Trabalho e Análise de resultados

Numa primeira instância, optou-se por realizar uma pequena fundamentação teórica das temáticas abordadas no tópico anterior, para que melhor se possa compreender o que foi definido na elaboração do trabalho prático.

Fundamentação teórica

Facto:

Um facto, define-se como sendo um predicado verdadeiro que é inserido na base de conhecimento.

Regra:

Uma regra, carateriza-se como sendo um predicado que usa variáveis, com o intuito de provar a veracidade de um facto.

Pressuposto do Mundo Fechado:

Este princípio, refere que numa consulta à base de conhecimento, todo o predicado é avaliado como falso no caso não existir nenhum facto que suporte o termo proposto ou nenhuma regra positiva. Por outras palavras, infere-se que toda a informação que não consta na base de conhecimento é considerada falsa.

Inserção e remoção da base de conhecimento:

A inserção e remoção de conhecimento da base de conhecimento, é feita utilizando os predicados *assert* e *retract*, respetivamente. Apesar disso, estes predicados não dão garantia de consistência e preservação de algumas restrições que se pretendam implementar, tais como, a inserção de factos repetidos ou a remoção de factos que não se encontram presentes na base de conhecimento. Como forma de contornar esta limitação, recorreu-se ao uso de invariantes.

Invariante e sua estrutura:

Um invariante é algo que não apresenta alterações sempre que são aplicadas um conjunto de transformações (operações de inserção e remoção) existentes na base de conhecimento.

Um invariante pode ser:

- estrutural: condiciona a base de conhecimento ao nível estrutural.
- referencial: condiciona a base de conhecimento tendo como base o significado do predicado.

```
+adjudicante(IdAd,N,NIF,M) :: ( solucoes( (IdAd, NIF),
                                  (adjudicante(IdAd,_,_,_),
                                  adjudicante(_,_,NIF,_)),
                                  S ),
                               comprimento(S,C),
                               C == 1).
```

Figura 1 - Invariante Referencial: não existir mais do que uma ocorrência da mesma evidência na relação adjudicante/4.

Legenda:

- ***+ =>*** símbolo que representa a inserção de conhecimento;
- ***- =>*** símbolo que representa a remoção de conhecimento;
- ***:: =>*** símbolo que representa um predicado;
- ***adjudicante(IdAd,N,NIF,M)*:** predicado a ser aplicado ao invariante(*Termo*);
- A vermelho encontra-se o invariante a respeitante à transformação.

Pressuposto do mundo aberto:

O pressuposto de mundo aberto admite que podem existir outros factos ou conclusões verdadeiras para além daqueles representados na base de conhecimento.

Generalizando, permite distinguir o que é realmente falso de algo que não está presente na base de conhecimento.

A programação em lógica mostra-se um recurso limitado, para representar este tipo de conhecimento, uma vez que as respostas às questões colocadas são do tipo **verdadeiro** ou **falso**. Face a esta limitação, surge como uma solução a criação de uma extensão do mesmo que permite a inclusão de respostas do tipo **desconhecido**.

Tomando este pressuposto como ponto de partida, qualquer questão colocada à base de conhecimento pode ter um de 3 valores diferentes:

- **Verdadeiro:** Existe conhecimento que comprova a sua veracidade.
- **Falso ou negação forte:** Existe conhecimento explícito para negar a questão.
- **Desconhecido ou negação por falha**: Não existe conhecimento, isto é, não existe informação que permita retirar uma conclusão.

Extensão à Programação em Lógica

Uma extensão de um programa em lógica envolve 2 componentes:

- Representação de conhecimento, completo e imperfeito;
- Sistema de inferência que permite a interpretação de conhecimento tanto perfeito como imperfeito.

Conhecimento Imperfeito:

Numa extensão de um programa em lógica o conhecimento imperfeito é representado sob a forma de valores nulos.

Existem 3 tipos de valores nulos:

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

Sistema de Inferência:

O sistema de inferência utilizado corresponde a uma extensão de um meta-predicado, representado na seguinte figura.

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

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

Figura 2 - Extensão do meta-predicado demo.

É de salientar que para representar a negação forte, neste meta-predicado, é utilizado o "-" e a negação por falha é representada através do predicado "não".

Após a análise e consolidação dos constructos teóricos apresentados, estão reunidas as condições para avançar na implementação e resolução dos objetivos propostos.

Resolução das funcionalidades propostas

Resolução das funcionalidades propostas

1. Representação de conhecimento positivo

De modo a inserir conhecimento positivo para a resolução deste trabalho, elaboramos os seguintes factos apresentados:

Adjudicantes:

```
% Extensao do predicado adjudicante: IdAd, N, NIF, M -> {V,F}

adjudicante(1, 'Eduardo Ferreira', 700048781, 'Cernache do Bonjardim').
adjudicante(2, 'Orlando Amorim', 154705563, 'Alhandra').
adjudicante(3, 'Valentina Henriques', 220621364, 'Boticas').
adjudicante(21,'Teodora Fernandes',293498947,x001).
adjudicante(25, x008, 938274920, 'Moreira').
```

Figura 3 - Inserção de conhecimento relativo a adjudicantes

Adjudicatárias:

```
% Extensao do predicado adjudicataria: IdAda, N, NIF, M -> {V,F}

adjudicataria(1, 'COFINA MEDIA, S.A.', 288881114, 'Ferrel').
adjudicataria(2, 'LISBOAGAS GDL - SOCIEDADE DISTRIBUIDORA DE GAS NATURAL DE LISBOA, S.A.', 416173057, 'Baiao').
adjudicataria(3, 'AGRUPALTO - AGRUPAMENTO DE PRODUTORES AGROPECUARIOS, S.A.', 133136500, 'Carvalho de Espingarda as Costas').
adjudicataria(11,'Pica Pau',826502849,x004).
adjudicataria(13,x006,923857274,'Fajoses').
```

Figura 4 - Inserção de conhecimento relativo a adjudicatárias

Contratos:

```
% Extensao do predicado contrato: IdC, IdAd, IdAda, TC, TP, D, V, P, L, Dia, Mes, Ano -> {V,F}

contrato(1,  4 , 3,'contrato de aquisicao', 'ajuste direto', 'contrato de aquisicao de um bem', 2761.13, 246, 'Marinha das Ondas', 11, 1, 2018).
contrato(2,  11, 6,'locacao de bens moveis', 'ajuste direto', 'contrato de locacao de um ou mais bens moveis', 390.51, 218, 'Canical', 26, 2, 2011).
contrato(3,  9 , 4,'contrato de arrendamento', 'concurso publico', 'contrato de arrendamento de imovel', 91017.60, 466, 'Aveiras de Cima', 13, 8, 2013).
contrato(51, 5, 1,'contrato de aquisicao', 'ajuste direto', 'contrato de aquisicao de um bem', 335.54, 74, x007, 24, 8, 2017).
contrato(54, 14, 6, 'compra de carro', 'concurso publico', x014, 1339.54, 356, 'Alvalade', 27, 7, 2015).
```

Figura 5 - Inserção de conhecimento relativo a contratos

2. Representação de conhecimento negativo

De modo a representar conhecimento negativo para a resolução deste trabalho, definimos a seguinte representação:

```
-contrato(IdC,IdAd,IdAda,TipoDeContrato,TipoDeProcedimento,Descricao,Valor,Prazo,Local,Dia,Mes,Ano) :-
    nao(contrato(IdC,IdAd,IdAda,TipoDeContrato,TipoDeProcedimento,Descricao,Valor,Prazo,Local,Dia,Mes,Ano)),
    nao(excecao(contrato(IdC,IdAd,IdAda,TipoDeContrato,TipoDeProcedimento,Descricao,Valor,Prazo,Local,Dia,Mes,Ano))).

-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))).

-cancelado(IdC) :-
    nao(cancelado(IdC)),
    nao(excecao(cancelado(IdC))).

-arquivado(IdC) :-
    nao(arquivado(IdC)),
    nao(excecao(arquivado(IdC))).
```

Figura 6 - Representação de conhecimento negativo

3. Representação de casos de conhecimento imperfeito

De modo a representar casos de conhecimento imperfeito, recorremos aos valores nulos dos tipos estudados, incerto, impreciso e interdito.

Conhecimento Incerto:

Para inserir conhecimento incerto utilizamos factos com valores pré-definidos.

Com isto, observando a figura abaixo, o adjudicante com a morada x001 não a tem na base de conhecimento, apesar de esta existir.

```
excecao(adjudicante(IdAd,Nome,Nif,_)) :- adjudicante(IdAd,Nome,Nif,x001).
excecao(adjudicante(IdAd,Nome,_,Morada)) :- adjudicante(IdAd,Nome,x002,Morada).
excecao(adjudicante(IdAd,_,NIF,Morada)) :- adjudicante(IdAd,x003,NIF,Morada).

excecao(adjudicataria(IdAda,Nome,Nif,_)) :- adjudicataria(IdAda,Nome,Nif,x004).
excecao(adjudicataria(IdAda,Nome,_,Morada)) :- adjudicataria(IdAda,Nome,x005,Morada).
excecao(adjudicataria(IdAda,_,NIF,Morada)) :- adjudicataria(IdAda,x006,NIF,Morada).

excecao(contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,_,Dia,Mes,Ano)) :- contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,x007,Dia,Mes,Ano).
```

Figura 7 - Representação de conhecimento incerto para adjudicantes, adjudicatárias e contratos

Conhecimento Impreciso:

A inserção de conhecimento impreciso ocorre quando um adjudicante, adjudicatária ou contrato está registado múltiplas vezes com algum dos campos diferentes.

Com isto, observando a figura abaixo, não se sabe se a morada do Manuel é Braga ou Povoa de Lanhoso, mas sabe-se que é uma destas.

```
excecao(adjudicante(24,'Manuel Aragao',285612438,'Braga')).
excecao(adjudicante(24,'Manuel Aragao',285612438,'Povoa de Lanhoso')).

excecao(adjudicataria(14,'Maquinaria Pesada Unipessoal',765231987,'Montalegre')).
excecao(adjudicataria(14,'Maquinaria Pesada Unipessoal',865231987,'Montalegre')).

excecao(contrato(52, 15, 2, 'contrato de trabalho', 'concurso publico', 'contrato de trabalho a termo certo', 345.1, 10, 'Tavira', D, 12, 2019)) :- D >= 1, D =< 17.
excecao(contrato(53, 1 , 10,'contrato de aquisicao', 'ajuste direto', 'contrato de aquisicao de um bem', V, 716, 'Bairrada', 23, 3, 2013)) :- V > 1000, V < 2500.
```

Figura 8 - Representação de conhecimento impreciso para adjudicantes, adjudicatárias e contratos

Conhecimento Interdito:

A inserção de conhecimento interdito ocorre quando um dos campos de um adjudicante, adjudicatária ou contrato nunca poderá ser sabido pela base de conhecimento.

Com isto, observando a figura abaixo, nunca se poderá saber o nome do adjudicante registado com nome x011.

```
adjudicataria(15, x011, 472947264, 'Caxinas').
excecao(adjudicataria(IdAda,Nome,Nif,Morada)) :- adjudicataria(IdAda, x011, Nif, Morada).
nulo(x011).

+adjudicataria(IdAda,Nome,Nif,Morada) :: ( solucoes( Nome,
                                                     (adjudicataria(15, Nome, 472947264, 'Caxinas'),
                                                     nao(nulo(Nome))),
                                                     S ),
                                          comprimento(S,0) ).
```

Figura 9 - Representação de conhecimento interdito para um adjudicante

```
adjudicante(26, 'Deolinda Matavento', x009, 'Maia').
excecao(adjudicante(IdAd, Nome, Nif, Morada)) :- adjudicante(IdAd, Nome, x009, Morada).
nulo(x009).

+adjudicante(IdAd,Nome,Nif,Morada) :: ( solucoes( Nif,
                                                  (adjudicante(26, 'Deolinda Matavento', Nif, 'Maia'),
                                                  nao(nulo(Nif))),
                                                  S ),
                                       comprimento(S,0) ).
```

Figura 10 - Representação de conhecimento interdito para uma adjudicatária

```
contrato(54, 14, 6, 'compra de carro', 'concurso publico', x014, 1339.54, 356, 'Alvalade', 27, 7, 2015).
excecao(contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano)) :- contrato(IdC,IdAd,IdAda,TC,TP,x014,V,P,L,Dia,Mes,Ano).
nulo(x014).

+contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano) :: ( solucoes( D,
                                                                   (contrato(54, 14, 6, 'compra de carro', 'concurso publico', D, 1339.54, 356, 'Alvalade', 27, 7, 2015),
                                                                   nao(nulo(D))),
                                                                   S ),
                                                          comprimento(S,0) ).
```

Figura 11 - Representação de conhecimento interdito para um contrato

4. Restrições à inserção e remoção de conhecimento

Para implementar restrições à inserção e remoção de conhecimento definimos os seguintes invariantes:

```
+adjudicante(IdAd,N,NIF,M) :: ( solucoes( (IdAd, NIF),
                                          (adjudicante(IdAd,_,_,_),
                                           adjudicante(_,_,NIF,_)),
                                          S ),
                                comprimento(S,C),
                                C == 1).
```

Figura 12 - não existir mais do que uma ocorrência da mesma evidência na relação adjudicante/4

```
+adjudicante(IdAd,N,NIF,M) :: ( solucoes( NIF,
                                          adjudicataria(_,_,NIF,_),
                                          S ),
                                comprimento(S,C),
                                C == 0 ).
```

Figura 13 - não existir mais do que uma ocorrência do NIF nas relações adjudicante/4 e adjudicataria/4

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

Figura 14 - não permitir a involução de um adjudicante caso este já tenha realizado um contrato

```
+adjudicante(IdAd,N,NIF,M) :: ( integer(NIF), NIF > 0, numToList(NIF,R), comprimento(R,9) ).
```

Figura 15 - não permitir a inserção de um adjudicante com nif inválido

```
-adjudicante(IdAd,Nome,Nif,Morada) :: ( solucoes( IdAd,
                                                  adjudicante(IdAd,Nome,Nif,Morada),
                                                  S ),
                                        comprimento( S,N ),
                                        N == 0 ).
```

Figura 16 - não remover adjudicantes que não estejam na base de conhecimento

```
+adjudicataria(IdAda,N,NIF,M) :: ( solucoes( (IdAda,NIF),
                                             (adjudicataria(IdAda,_,_,_),
                                             adjudicataria(_,_,NIF,_)),
                                             S ),
                                   comprimento(S,C),
                                   C == 1 ).
```

Figura 17 - não existir mais do que uma ocorrência da mesma evidência na relação adjudicataria/4

```
+adjudicataria(IdAda,N,NIF,M) :: ( solucoes( NIF,
                                             adjudicante(_,_,NIF,_),
                                             S ),
                                   comprimento(S,C),
                                   C == 0 ).
```

Figura 18 - não existir mais do que uma ocorrência do NIF nas relações adjudicante/4 e adjudicataria/4

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

Figura 19 - não permitir a involução de um adjudicante caso este já tenha realizado um contrato

```
+adjudicataria(IdAd,N,NIF,M) :: ( integer(NIF), NIF > 0, numToList(X,R), comprimento(R,9) ).
```

Figura 20 - não permitir a inserção de uma adjudicataria com nif inválido

```
-adjudicataria(IdAda,N,NIF,M)  :: ( solucoes( IdAda,
                                             adjudicataria(IdAda,N,NIF,M),
                                             S ),
                                   comprimento( S,N ),
                                   N == 0 ).
```

Figura 21 - não remover adjudicatarias que não estejam na base de conhecimento

```
+contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano) :: ( solucoes( IdC,
                                                                   contrato(IdC,_,_,_,_,_,_,_,_,_,_,_),
                                                                   S ),
                                                         comprimento(S,N),
                                                         N == 1 ).
```

Figura 22 - não existir mais do que uma ocorrência do identificador de contrato na relação contrato/12

```
+contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano) :: ( solucoes( IdAd,
                                                                   adjudicante(IdAd,_,_,_),
                                                                   S ),
                                                         comprimento(S,N),
                                                         N == 1 ).
```

Figura 23 - existir uma ocorrência do identificador do adjudicante/4

```
+contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano) :: ( solucoes( IdAda,
                                                                   adjudicataria(IdAda,_,_,_),
                                                                   S ),
                                                         comprimento(S,N),
                                                         N == 1 ).
```

Figura 24 - existir uma ocorrência do identificador do adjudicataria/4

```
+contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano) :: pertence( TP,['ajuste direto', 'consulta previa', 'concurso publico'] ).
```

Figura 25 - não permitir a inserção de tipos de procedimento inválidos

```
+contrato(IdC,IdAd,IdAda,TC,'ajuste direto',D,V,P,L,Dia,Mes,Ano) :: (V =< 5000, V >= 0).
```

Figura 26 - não permitir a inserção de contratos com tipo de procedimento ajuste direto e valor =< a 5000€

```
+contrato(IdC,IdAd,IdAda,TC,'ajuste direto',D,V,P,L,Dia,Mes,Ano) :: pertence( TC,['contrato de aquisicao', 'locacao de bens moveis', 'aquisicao de servicos'] ).
```

Figura 27 - não permitir a inserção de contratos com tipo de procedimento ajuste direto e tipos de contrato inválidos

```
+contrato(IdC,IdAd,IdAda,TC,'ajuste direto',D,V,P,L,Dia,Mes,Ano) :: (P =< 365, P > 0).
```

Figura 28 - não permitir a inserção de contratos com tipo de procedimento ajuste direto e prazo de vigência superior a 1 ano

```
+contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano) :: ( Ano1 is Ano - 1,
                                                         Ano2 is Ano - 2,
                                                         find(IdAd,IdAda,TC,[Ano,Ano1,Ano2],R),
                                                         X is R - V,
                                                         X < 75000 ).
```

Figura 29 - 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

```
+contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano) :: valor_valido(V).
```

Figura 30 - não permitir a inserção de um valor inválido

```
+cancelado(IdC) :: ( solucoes( IdC,
                               cancelado(IdC),
                               S ),
                     comprimento(S,N),
                     N == 1 ).
```

Figura 31 - não existir mais do que uma ocorrência do identificador na relação cancelado/1

```
+arquivado(IdC) :: ( solucoes( IdC,
                               arquivado(IdC),
                               S ),
                     comprimento(S,N),
                     N == 1 ).
```

Figura 32 - não existir mais do que uma ocorrência do identificador na relação arquivado/1

5. Criação de uma API de modo a manipular o conhecimento

Para a criação de uma API que é chamada pelo utilizador ao interagir com a base de conhecimento definimos os seguintes predicados:

```
add_adjudicante(N,NIF,M) :- id_adjudicante(IdAd),                % get próximo id
                            evolucao(adjudicante(IdAd,N,NIF,M)).  % inserir conhecimento
```

Figura 33 - Adiciona um adjudicante, calculando primeiro o seu id

```
remove_adjudicante(IdAd) :- involucao(adjudicante(IdAd,_,_,_)).
```

Figura 34 - Remove um adjudicante através do seu id

```
get_contratos_adjudicante(IdAd,R) :- findall(IdC, contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano), R).
```

Figura 35 - Devolve os ids dos contratos de um adjudicante, dado o seu id

```
get_valor_ano_adjudicante(IdAd,Ano,R) :- findall(V, contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano), S),
                                         soma(S,R1),
                                         R2 is R1 - integer(R1),
                                         R3 is R2 * 100,
                                         R4 is integer(R3) / 100,
                                         R is integer(R1) + R4.
```

Figura 36 - Devolve o valor total dos contratos de um adjudicante realizados num dado ano

```
add_adjudicataria(N,NIF,M) :- id_adjudicataria(IdAda),                % get próximo id
                              evolucao(adjudicataria(IdAda,N,NIF,M)).  % inserir conhecimento
```

Figura 37 - Adiciona uma adjudicataria, calculando primeiro o seu id

```
remove_adjudicataria(IdAda) :- involucao(adjudicataria(IdAda,_,_,_)).
```

Figura 38 - Remove uma adjudicataria através do seu id

```
get_contratos_adjudicataria(IdAda,R) :- findall(IdC, contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano), R).
```

Figura 39 - Devolve os ids dos contratos de uma adjudicataria, dado o seu id

```
get_valor_ano_adjudicataria(IdAda,Ano,R) :- findall(V, contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano), S),
                                            soma(S,R1),
                                            R2 is R1 - integer(R1),
                                            R3 is R2 * 100,
                                            R4 is integer(R3) / 100,
                                            R is integer(R1) + R4.
```

Figura 40 - Devolve o valor total dos contratos de uma adjudicataria realizados num dado ano

```
cancelar(IdC) :- ( solucoes( IdC,
                             contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),
                             S ),
                   comprimento(S,N),
                   N == 1 ),
                 findall(P,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),R1),        % get R1 = [prazo]
                 head(R1,R2),                                                             % get R2 = prazo
                 findall(Dia,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),R3),      % get R3 = [dia]
                 head(R3,R4),                                                             % get R4 = dia
                 findall(Mes,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),R5),      % get R5 = [mes]
                 head(R5,R6),                                                             % get R6 = mes
                 findall(Ano,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),R7),      % get R7 = [ano]
                 head(R7,R8),                                                             % get R8 = ano
                 cmp_datas(R4,R6,R8,R2),                                                  % validar prazo
                 evolucao(cancelado(IdC)).                                                % inserir conhecimento
```

Figura 41 - Cancela um contrato que ainda esteja dentro do prazo através do seu id

```
arquivar(IdC) :- ( solucoes( IdC,
                             contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),
                             S ),
                   comprimento(S,N),
                   N == 1 ),
                 findall(P,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),R1),        % get R1 = [prazo]
                 head(R1,R2),                                                             % get R2 = prazo
                 findall(Dia,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),R3),      % get R3 = [dia]
                 head(R3,R4),                                                             % get R4 = dia
                 findall(Mes,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),R5),      % get R5 = [mes]
                 head(R5,R6),                                                             % get R6 = mes
                 findall(Ano,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),R7),      % get R7 = [ano]
                 head(R7,R8),                                                             % get R8 = ano
                 nao(cmp_datas(R4,R6,R8,R2)),                                             % validar prazo
                 evolucao(arquivado(IdC)).                                                % inserir conhecimento
```

Figura 42 - Arquiva um contrato que já tenha expirad, através do seu id

```
estado(IdC) :- findall(IdC,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),S),
               length(S,N),
               N == 0,
               write('contrato inexistente').

estado(IdC) :- findall(IdC,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),S1),
               length(S1,N1),
               N1 == 1,
               findall(IdC,cancelado(IdC),S2),
               length(S2,N2),
               N2 == 1,
               write('contrato cancelado').

estado(IdC) :- findall(IdC,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),S1),
               length(S1,N1),
               N1 == 1,
               findall(IdC,arquivado(IdC),S2),
               length(S2,N2),
               N2 == 1,
               write('contrato arquivado').

estado(IdC) :- findall(IdC,contrato(IdC,IdAd,IdAda,TC,TP,D,V,P,L,Dia,Mes,Ano),S1),
               length(S1,N1),
               N1 == 1,
               findall(IdC,cancelado(IdC),S2),
               length(S2,N2),
               N2 == 0,
               findall(IdC,arquivado(IdC),S3),
               length(S3,N3),
               N3 == 0,
               write('contrato em vigor').
```

Figura 43 - Devolve o estado de um contrato através do seu id

6. Predicados auxiliares

Para a implementação dos invariantes e da API acima representados, foi necessário desenvolver alguns predicados auxiliares, sendo estes os seguintes:

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

Figura 44 - devolve a soma de todos os elementos de uma determinada lista

```
maior([], R, R).
maior([X|Xs], WK, R):- X >  WK, maior(Xs, X, R).
maior([X|Xs], WK, R):- X =< WK, maior(Xs, WK, R).
maior([X|Xs], R):- maior(Xs, X, R).
```

Figura 45 - devolve o maior elemento de uma determinada lista

```
cmp_datas(Day,Month,Year,P) :- datime(T1,datime(Year,Month,Day,24,00,00)),   % timestamp do dia do contrato
                               T2 is P * 86400,                              % prazo em timestamp
                               T3 is T1 + T2,                                % timestamp do dia final de contrato
                               now(T4),                                      % timestamp atual
                               T3 > T4.                                      % validação
```

Figura 46 - valida se a data se encontra dentro de um prazo

```
head([],_) :- !,fail.
head([H|_], H).
```

Figura 47 - devolve a cabeça de uma determinada lista não vazia

```
timestamp(0,R) :- now(R).
timestamp(Dias,R) :- timestamp(0,T), R1 is Dias * 86400, R is R1 + T.
```

Figura 48 - devolve o timestamp do dia atual somado com um determinado numero de dias

```
valor_valido(N) :- (integer(N)) -> N >= 0 ; N >= 0, Rest is N*(10^2), Rest - integer(Rest) =:= 0.
```

Figura 49 - valida um valor de um contrato

```
id_contrato(R) :- solucoes( IdC,
                            contrato(IdC,_,_,_,_,_,_,_,_,_,_,_),
                            S ),
                  maior(S,R1),
                  R is R1 + 1.
```

Figura 50 - devolve o próximo id de contrato a ser inserido

```
id_adjudicante(R) :- solucoes( IdAd,
                               adjudicante(IdAd,_,_,_),
                               S ),
                     maior(S,R1),
                     R is R1 + 1.
```

Figura 51 - devolve o próximo id de adjudicante a ser inserido

```
id_adjudicataria(R) :- solucoes( IdAd,
                                 adjudicataria(IdAd,_,_,_),
                                 S ),
                       maior(S,R1),
                       R is R1 + 1.
```

Figura 52 - devolve o próximo id de adjudicataria a ser inserido

```
find(IdAd,IdAda,TC,Anos,R) :- solucoes((V),
                              (contrato(_,IdAd,IdAda,TC,_,_,V,_,_,_,_,X),
                              pertence(X,Anos)), S),
                              soma(S,R).
```

Figura 53 - devolve o valor total dos contratos de um determinado tipo entre um adjudicante e uma adjudicataria num conjunto de anos

```
get_dia(datime(Ano,Mes,Dia,Horas,Min,Sec),Dia).
```

Figura 54 - devolve o dia de um datime/6

```
get_mes(datime(Ano,Mes,Dia,Horas,Min,Sec),Mes).
```

Figura 55 - devolve o mes de um datime/6

```
get_ano(datime(Ano,Mes,Dia,Horas,Min,Sec),Ano).
```

Figura 56 - devolve o ano de um datime/6

```
numToList(NUM,[LIST|[]]):- NUM < 10, LIST is NUM, !.
numToList(NUM,LIST):- P is NUM // 10, numToList(P,LIST1), END is (NUM mod 10), append(LIST1,[END] ,LIST).
```

Figura 57 - converte um número numa lista de numeros inteiros

Conclusões e sugestões

O desenvolvimento deste trabalho ajudou-nos a aprofundar o conhecimento sobre a linguagem de programação PROLOG e a sua utilização na construção de sistemas que tiram partido do pressuposto de mundo aberto para a representação de conhecimento perfeito e conhecimento imperfeito.

Os desafios encontrados ao longo do desenvolvimento do projeto vão de encontro aos exercícios realizados nas aulas práticas, sendo possível aplicar os conhecimentos adquiridos nas mesmas neste trabalho.

Inicialmente, houve uma certa dificuldade em interpretar o enunciado e conseguir aplicar o nosso conhecimento da linguagem ao contexto proposto, no entanto, recorrendo à bibliografia recomendada conseguimos prosseguir com o desenvolvimento.

Por fim, considerámos que todos os objetivos foram alcançados com sucesso, de acordo com as especificações do enunciado.

Referências

Contratos Públicos Online. Disponível em:

http://www.base.gov.pt/Base/pt/PerguntasFrequentes

[Analide, 2001] ANALIDE, Cesar, NOVAIS, Paulo, NEVES, José,

"Sugestões para a Elaboração de Relatórios", Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2001.

[Bratko, 2000] BRATKO, Ivan,

"PROLOG: Programming for Artificial Intelligence", 3rd Edition, Addison-Wesley Longman Publishing Co., Inc., 2000

Anexos

Meta-predicados fornecidos

```
% Extensao do meta-predicado nao: Questao -> {V,F}

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

Figura 58 - Meta-predicado 'nao'

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

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

Figura 59 - Meta-predicado 'demo'

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

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

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

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

Figura 60 - Meta-predicados 'evolucao', 'insercao' e 'teste'

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

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

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

Figura 61 - Meta-predicados 'involucao' e 'remocao'

Predicados fornecidos

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

Figura 62 - Predicado 'solucoes'

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

Figura 63 - Predicado 'comprimento'

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

Figura 64 - Predicado 'pertence'