UNIVERSIDADE DO MINHO

Trabalho Prático Nº1 - *Programação em lógica estendida e Conhecimento imperfeito*

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(3º ano, 2º semestre, 2019/2020)
Relatório de Desenvolvimento
Grupo 30

Ana Almeida	André Figueiredo	Luís Ferreira	Rafael Lourenço
(a83916)	(a84807)	(a86265)	(a86266)

Braga, 3 de Maio de 2020

Resumo

A programação em lógica tem por objetivo a implementação de um programa cujo conteúdo se prende em factos, ou seja, em registos que se sabem ser verdadeiros e predicados, associados aos factos e regras. Esta baseia-se em dois princípios para a descoberta de respostas a essas questões:

- **lógica**, usada para representar informação e os conhecimentos;
- **inferência**, regras aplicadas à lógica para manipular o conhecimento.

De forma a demonstrar o poder desta linguagem, o primeiro trabalho prático no âmbito da unidade curricular Sistemas de Representação de Conhecimento e Raciocínio consistiu na elaboração de um projeto usando a extensão à programação em lógica, com recurso à linguagem de programação em lógica *PROLOG*, que consiste no desenvolvimento de um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da contratação pública para a realização contratos e para a prestação de serviços.

Deste modo, no presente relatório, explicámos a forma como desenvolvemos um exemplo prático elucidativo deste panorama, cumprindo os requisitos/funcionalidades mínimos solicitados. Usando a nossa criatividade decidimos, ainda, adicionar um conjunto de funcionalidades extra que considerámos relevantes para exemplo prático.

Os principais objetivos deste trabalho prático consistiram em construir uma base de conhecimento que se espera ser capaz de representar conhecimento perfeito, casos de conhecimento imperfeito (*incerto*, *impreciso* e *interdito*), manipular invariantes que designem restrições à inserção e à remoção de conhecimento do sistema, garantir a correta evolução e involução do conhecimento e desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Enquadramento e Contexto

Este relatório é o resultado do primeiro trabalho prático proposto e elaborado na unidade curricular Sistemas de Representação de Conhecimento e Raciocínio. O trabalho teve por base a utilização do *PROLOG* que se trata de uma linguagem de programação enquadrada no paradigma Lógico, fortemente associada à inteligência artificial e linguística computacional (mais direcionada ao conhecimento do que a algoritmos). No mundo do conhecimento imperfeito são considerados três valores de verdade: *verdadeiro*, *falso* e *desconhecido*. Para além disto, é também introduzido o conceito de *negação forte*, salvaguardando, contudo, a existência do predicado *nao* (*negação fraca* - por falha na prova). Assim e ao longo deste relatório, será apresentada a base de conhecimento construída e tudo a ela associado - representação de conhecimento imperfeito (*incerto*, *impreciso* e *interdito*) e invariantes *estruturais* e *referenciais* (garantia de uma correta evolução e involução da base de conhecimento).

1.2 Problema e Objetivo

O problema apresentado consiste em, principalmente, consolidar conceitos apreendidos nas aulas lecionadas sobre programação em lógica estendida e conhecimento imperfeito. O objetivo deste exercício em particular é a criação de um programa capaz de armazenar conhecimento sobre a área contratação pública para a realização contratos e para a prestação de serviços.

1.3 Estrutura do Relatório

O relatório encontra-se dividido nos seguintes capítulos:

1. **Introdução**, onde é feito um enquadramento e contextualização do trabalho prático e uma breve descrição do problema;
2. **Preliminares**, onde é feito um apanhado geral, em forma de resumo, sobre os conceitos necessários para a melhor compreensão do trabalho proposto;
3. **Descrição do Trabalho**, onde é feito um breve resumo de tudo o que foi elaborado durante a realização deste trabalho, desde a base de conhecimento, invariantes, evolução, involução, sistema de inferência, entre outros.

4. **Análise de Resultados**, onde são comparados os resultados obtidos com os resultados esperados.

5. **Conclusões, Decisões e Sugestões**, onde é feito um balanço geral com uma síntese e análise do trabalho realizado.

Capítulo 2

Preliminares

No presente capítulo serão apresentadas distinções, em forma de *bullet points*, de alguns conceitos necessários para a compreensão do desenvolvimento do presente projeto.

2.1 Da programação em lógica à programação em lógica estendida

Como referido anteriormente, um dos principais objetivos do trabalho é a programação em lógica **estendida**, mas porque se chama assim? Teremos de recuar um pouco para entender:

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

Estes pressupostos apresentados limitam-nos o desenvolvimento de bases de conhecimento, pois, aplicados ao contexto do mundo real, não são 100% "corretos". Tomemos o exemplo seguinte, um avião necessita de aterrar numa pista, tendo de ter em atenção se nenhum outro avião vai aterrar primeiro e a ocupar. Recorrendo à programação em lógica e caso não exista informação relativa a aviões a aterrarem na pista ao mesmo tempo, o nosso motor de inferência indicará que o nosso avião pode aterrar. Contudo, isto não prova que não existia outro avião a aterrar primeiro, apenas prova que não existe nenhuma prova em como ele se aproxima, todavia, este pode não ser o cenário, podendo levar, neste exemplo, a um choque entre aviões.

Posto isto, conclui-se que alguns destes pressupostos têm de ser revistos, já que, nem sempre se pretende assumir que a informação representada é a única válida e que as entidades representadas sejam as únicas existentes no mundo exterior. Assim,

- *PNU* é mantido → não interfere negativamente na representação de conhecimento;
- *PMF* não é mantido → é adotado o **Pressuposto do Mundo Aberto** (*PMA*) → podem existir outros factos ou conclusões verdadeiras para além daqueles representados na base de conhecimento;

- *PDF* não é mantido → é adotado o **Pressuposto do Domínio Aberto** (*PDA*) → podem existir mais objetos do universo de discurso para além daqueles designados pelas constantes da base de conhecimento.

Desta forma, foi feita uma **Extensão** à Programação em Lógica. Contudo, não ficámos por aqui.

Para além da alteração dos pressupostos referidos acima, foi também necessário alterar os valores de verdade e o tipo de conhecimento. O **conhecimento perfeito** apenas assume como *verdadeiro* aquilo que é conhecido, ou seja, o que a base de conhecimento contempla e assume que não existem mais objetos do que aqueles presentes na base de conhecimento (tudo o resto é *falso*).
No entanto, como visto no exemplo dos aviões, podemos e devemos ser mais detalhados, identificando quais são os valores mais comuns que podem surgir numa situação de informação *incompleta* (os chamados valores **nulos**). O **conhecimento imperfeito** depende das possíveis respostas a uma questão, mais concretamente no caso de a mesma ser desconhecida. Teremos, então, uma forma de representar a informação incompleta. Assim, são considerados três valores de verdade:

- *verdadeiro* → existe uma prova explícita de que se trata de conhecimento verdadeiro;
- *falso* → existe uma prova explícita de que se trata de conhecimento falso;
- *desconhecido* → não existe informação que permita inferir uma ou outra das conclusões anteriores;

São também considerados três tipos de valores nulos:

- **Incerto** → não é sabido algum dos campos, trata-se de conhecimento *desconhecido* dentro de um conjunto *indeterminado* de hipóteses;
- **Impreciso** → não existe certeza sobre qual das informações é a correta, isto é, é conhecimento *desconhecido* dentro de um conjunto *finito* de hipóteses;
- **Interdito** → não é permitido conhecê-lo.

Um outro objetivo de estender a Programação em Lógica é passar a permitir a representação de informação negativa **explicitamente** - *negação forte*. Deste modo, a extensão de um programa em lógica passa agora a contar com dois tipos de negação:

Negação forte - conectiva *"-"*:

- Possibilita a representação de conhecimento negativo ou falso;
- Existe uma prova na base de conhecimento de que a questão é falsa.

Negação fraca - termo *nao*:

- Parte do *PMF* (se este não existe, então o seu valor é falso) → não existe uma prova na base de conhecimento que responda à questão;

Em suma, temos todos os pontos necessários para a caracterização da **Extensão** à Programação em Lógica e que nos permitirá uma melhor compreensão do projeto apresentado em seguida.

Capítulo 3

Descrição do Trabalho

No desenvolvimento deste sistema de representação de conhecimento e raciocínio para contratos públicos, começámos por definir uma base de conhecimento, com o objetivo de representar conhecimento perfeito.
De forma a garantir que a informação que seria introduzida na base de conhecimento fosse validada, criámos invariantes *estruturais* e *referenciais*. Estes servem para garantir a consistência do conhecimento quando inserimos ou removemos informação.
Para uma melhor experiência de pesquisa na base de conhecimento, desenvolvemos predicados de pesquisa e funções auxiliares dos mesmos. Por exemplo, devido à necessidade do processamento de datas, criámos um conjunto de regras/predicados a fim de conseguir realizar cálculos necessários para lidar com os invariantes e, do mesmo modo, ser uma representação acessível ao utilizador comum.
Visto existir conhecimento imperfeito no quotidiano, apresentámos uma possível representação deste, com o objetivo de lidar com esta questão. Para isso, utilizámos predicados que representam valores nulos: *interditos*, *incertos* e *imprecisos*. É evidente que este tipo de conhecimento levou à reestruturação/inclusão de novos predicados. Um exemplo disso, foi a implementação do sistema de inferência, onde inicialmente, todo o conhecimento imperfeito era *desconhecido*, ou seja, o sistema apenas inferia valores do tipo *verdadeiro*, *falso* ou *desconhecido*. De forma a tornar mais explícito o conhecimento imperfeito, tivemos, então, de restruturar o sistema de inferência, de modo que fosse possível inferir se o conhecimento era *verdadeiro*, *falso*, *incerto*, *impreciso*, *interdito* ou *desconhecido*.
Por último, como sabemos, a informação que hoje é imperfeita um dia pode ser perfeita ou aperfeiçoada, por isso e após criar predicados de inserção/remoção de conhecimento imperfeito, adicionámos uma funcionalidade de substituição de conhecimento. Esta pode substituir conhecimento imperfeito para perfeito ou melhorar o conhecimento imperfeito.

3.1 Base de conhecimento

Esta base de conhecimento, como foi dito anteriormente, está apta para representar conhecimento **perfeito** e **imperfeito**. Esta é constituída por 2 entidades e um contrato, uma vez que tem como objetivo representar conhecimento relacionado com a contratação pública.
Como pretendemos que a base de conhecimento seja alterada, temos de declarar os factos como dinâmicos com a aridade respetiva. Desta forma, vamos ter o seguinte panorama:

```
:- dynamic e_ad/3.
:- dynamic e_ada/3.
:- dynamic contrato/10.
:- dynamic excecao/2.
```

Figura 3.1: Declaração dos factos dinâmicos

De seguida, mostra-se, então, as definições dos factos para as entidades e os contratos na representação do conhecimento perfeito.

3.1.1 Conhecimento Perfeito

3.1.1.1 Entidade Adjudicante

É uma entidade **pública**, que pretende apresentar uma proposta de contrato para a prestação dos serviços.

Facto que representa o conhecimento perfeito: `e_ad`$(Nome, \#Nif, Morada) - \{V,F\}$

Uma entidade adjudicante é identificada por um nome, número identificador fiscal único (NIF) e a sua morada. Decidimos retirar o *id* da entidade, pois o NIF das entidades é um valor acessível ao público e único. Sendo assim, o identificador das entidades é o NIF.
Deste modo, tendo bem definido o facto para a entidade adjudicante, será agora apresentado um excerto da nossa base de conhecimento inicial, relativamente ao conhecimento perfeito:

```
e_ad('CP - Comboios de Portugal',700000000,'Cc Do Duque 14/20, 1249-109,
Misericordia Lisboa, Lisboa').
e_ad('BRISA - Autoestradas de Portugal, S.A.',700000001,'Qta Torre Da
Aguilha Edifício Brisa, 2785-599, Sao Domingos Rana Cascais, Lisboa').
e_ad('NAV Portugal, EPE',700000003,'R D Do Aeroporto De Lisboa Edifício 121
3º, 1700-008, Santa Maria Olivais Lisboa, Lisboa').
e_ad('ULSAM - Unidade Local de Saúde do Alto Minho, EPE',700000004,'Estr. De
Santa Luzia 50, Viana Do Castelo').
e_ad('CGD - Caixa Geral de Depósitos, SA',700000005,'Av João Xxi 63,
1000-300, Areeiro Lisboa, Lisboa').
e_ad('PME Investimentos - Sociedade de Investimentos, SA',700000006,'Rua
Pedro Homem De Melo, 55, S.3.09, 4150-599 Porto').
```

Figura 3.2: Exemplo de conhecimento perfeito para as entidades Adjudicantes

3.1.1.2 Entidade Adjudicatária

É uma entidade **pública** ou **privada** e corresponde à entidade com quem a entidade adjudicante irá celebrar um contrato público.

Facto que representa o conhecimento perfeito: `e_ada`$(Nome, \#Nif, Morada) - \{V,F\}$

Uma entidade adjudicatária possui um nome, número identificador fiscal único (NIF) e a sua morada. Aqui, também decidimos retirar o *id* pelo mesmo motivo, visto que as entidades adjudicantes são semelhantes às entidades adjudicatárias.
Seguidamente, tendo bem definido o facto para a entidade adjudicatária, será agora apresentado um excerto da nossa base de conhecimento inicial, relativamente ao conhecimento perfeito:

```
e_ada('PETROGAL, S.A',300000000,'R Tomás Da Fonseca Torre C, 1600-209, Sao
Domingos Benfica Lisboa, Lisboa').
e_ada('EDP',300000001,'Av 24 De Julho 12, 1249-300, Misericordia Lisboa,
Lisboa').
e_ada('Pingo Doce',300000002,'R Actor António Silva 7, 1649-033, Lumiar
Lisboa, Lisboa').
e_ada('TAP',300000003,'Aeroporto De Lisboa Efifício 27-8 Sala 1, 1700-008,
Santa Maria Olivais Lisboa, Lisboa').
e_ada('GALP',300000004,'R Tomás Da Fonseca Torre C, 1600-209, Sao Domingos
Benfica Lisboa').
e_ada('CTT',300000005,'Av Dom João Ii 13, 1999-001, Parque Nacoes Lisboa,
Lisboa').
e_ada('PRIO SUPPLY, S.A.',300000006,'Porto De Pesca De Aveiro Lote B,
3830-565, Gafanha Nazare Ilhavo, Aveiro').
```

Figura 3.3: Exemplo de conhecimento perfeito para as entidades Adjudicatárias

3.1.1.3 Contrato

É o instrumento dado à administração pública, utilizado para adquirir bens ou serviços a particulares.

Facto que representa o conhecimento perfeito: `contrato`$(Id, Nif\_ad, Nif\_ada, TipoC, TipoP, Descricao, Custo, Prazo, Local, Data) - \{V,F\}$

Um contrato é caracterizado por um identificador (*id*), o NIF da entidade adjudicante, o NIF da entidade adjudicatária, o tipo de contrato, o tipo de procedimento, uma breve descrição, o valor (em euros), o prazo de vigência (em dias), o local onde se efetuou o contrato e, por último, a data de realização do mesmo.
Aqui podemos observar uma vantagem em usar o *id* em relação a usar o NIF como identificador das entidades, dado que como o NIF é usado para identificar a entidade em muitos outros sistemas, evita-se, assim, mais uma pesquisa na base de conhecimento.
Por fim, tendo bem definido o facto para o contrato, será agora apresentado um excerto da nossa base de conhecimento inicial, relativamente ao conhecimento perfeito:

```
contrato(1,700000000,300000000,'Aquisicao de Servicos','Consulta
Previa','Fornecimento de petrólio',50000,365,'Lisboa',"01-01-2018").
contrato(2,700000000,700000005,'Aquisicao de Servicos','Ajuste
Direto','contrato de intermediação financeira',25000,366,
'Lisboa',"01-04-2020").
contrato(3,700000001,300000006,'Aquisicao de Servicos','Concurso
Publico','Controlo das portagens ',100000,365,'Lisboa',"06-01-2010").
contrato(4,700000004,300000001,'Aquisicao de Servicos','Consulta
Previa','Fornecimento de energia',100000,365,'Viana do Castelo',
"06-01-2013").
contrato(5,700000004,300000004,'Aquisicao de Servicos','Concurso
Publico','Fornecimento de Gas Natural ',30000,365,'Viana do
Castelo',"03-01-2010").
contrato(6,700000003,300000003,'Aquisicao de Servicos','Concurso
Publico','Controlo do trafego aerio ',5000000,1096,'Lisboa',"06-01-2017").
contrato(7,700000003,300000003,'Locacao de bens moveis','Ajuste
Direto','Combustivel para automeveis',5000,30,'Porto',"06-01-2018").
contrato(9,700000006,700000004,'Aquisicao de Servicos','Consulta
Previa','Compra de protecoes contra o Covid19',50000,180,'Lisboa',
"01-02-2019").
contrato(10,700000005,700000004,'Empreitadas de obras públicas','Consulta
Previa','Investimento publico para a construção de um novo hospital',
5000000,180,'Viana do Castelo',"01-01-2020").
```

Figura 3.4: Exemplo de conhecimento perfeito para os contratos

Conclui-se, assim, a demonstração do conhecimento perfeito.

3.1.2 Conhecimento imperfeito

Por outo lado, temos o conhecimento imperfeito, que é representado pelo uso de valores *nulos*. Existem 3 tipos de valores *nulos*:

- Em primeiro lugar, temos o valor nulo definido como *incerto*, que carateriza todos os valores nulos que não pertencem a nenhum conjunto determinado de valores, ou seja, é completamente desconhecido;
- Em segundo lugar, temos o valor nulo definido como *impreciso*, que pertence a um conjunto determinado de valores, isto é, existe uma noção do que é falso, mas não se conhece o valor verdadeiro;
- Em terceiro lugar, temos o valor nulo definido como *interdito*, que são aqueles que não são permitidos conhecer.

Em seguida, explicaremos como implementámos cada uma destas formas de representação de conhecimento.

Para inserir um valor *nulo* temos de usar predicados auxiliares da seguinte forma:

- Para introduzir valores nulos *incertos*, basta escrever **incerto** no parâmetro associado.
- Para introduzir valores nulos *imprecisos*, usámos o predicado auxiliar **impreciso(X)**, onde o pârametro **X** é um conjunto de valores.
- Para introduzir valores nulos *interditos*, utilizámos o predicado auxiliar **interdito(X)**, onde o pârametro **X** é o valor que queremos ocultar.

Para permitir a evolução do conhecimento imperfeito no nosso sistema, criámos o facto chamado **excecao(X,Y)** com aridade 2, que nos permite escolher o tipo de conhecimento imperfeito que queremos inserir (pârametro **Y**) e, no pârametro **X**, temos o facto com o conhecimento imperfeito.
Para além disso, sendo que o pârametro **X** é um facto, temos de especificar os argumentos onde irá ter os valores *nulos*, com os predicados mostrados anteriormente.
Os factos podem ter vários valores nulos diferentes, por exemplo, num contrato podemos ter pârametros *interditos* e *incertos*, porém para garantir isso, temos de seguir as seguintes regras:

- Se existir um atributo *interdito*, obrigatoriamente temos de colocar a **excecao** como *interdito*.
- Se não existir um atributo *interdito*, a escolha do tipo de exceção é indiferente, quer seja *incerto* ou *impreciso*.

É de salientar que, para estes casos mencionados acima, assumimos que estamos a inserir vários valores nulos distintos, isto é, existem pelo menos 2 valores nulos distintos no facto que queremos inserir. É evidente que se só existir um tipo de valor nulo, esse é o que tem de ser especificado no facto **excecao**.
Foi usado o predicado **processa** para substituir certos campos por uma *tag* correspondente. No caso do conhecimento imperfeito *interdito*, fazemos a evolução de dois predicados: um com a informação visível para que o utilizador de maior responsabilidade tenha acesso a estes (representado por **interdito**) e outro para o utilizador comum, de modo que não consiga observar esses valores (representado por **info_interdito**). No caso do conhecimento imperfeito *impreciso* armazenámos o predicado tal como recebemos no parâmetro. No caso do conhecimento imperfeito *incerto*, substituímos o valor incerto por **'N/A'**.
Poderá observar-se estes casos ao pormenor, no capítulo seguinte.

3.1.2.1 Entidade Adjudicante

Neste caso, decidimos que a única informação que poderia ser desconhecida é a morada, ou seja, só é permitido inserir uma entidade se tanto o nome como o NIF forem conhecidos.

Facto que representa o conhecimento imperfeito: `excecao(e_ad(`$Nome, \#Nif, Morada$`),` $ValorNuloCorrespondente$**)**

Quando inserimos este tipo de conhecimento, este é armazenado da seguinte forma:

```
excecao(e_ad('EDM - Empresa de Desenvolvimento Mineiro',700000008,'N/A'),
incerto)
excecao(e_ad('EDM - Empresa de Desenvolvimento Mineiro',700000008,
impreciso('Porto','Lisboa')),incerto)
excecao(e_ad(('I','EDM - Empresa de Desenvolvimento Mineiro'),
('I',700000008),interdito('Porto')),interdito)
excecao(e_ad('EDM - Empresa de Desenvolvimento Mineiro', 700000008,'-'),
info_interdito)
```

Figura 3.5: Exemplo de conhecimento perfeito para as entidades Adjudicantes

3.1.2.2 Entidade Adjudicatária

Este caso, é semelhante ao anterior, ou seja, apenas a morada pode adquirir valores nulos.

Facto que representa o conhecimento imperfeito: `excecao(e_ada(`$Nome, \#Nif, Morada$`),` $ValorNuloCorrespondente$`)`

Quando inserimos este tipo de conhecimento, este é armazenado da seguinte forma:

```
excecao(e_ada('TAP',700000008,'N/A'),incerto)
excecao(e_ada('TAP',700000008,impreciso('Braga','Lisboa')),incerto)
excecao(e_ada(('I','TAP'),('I',700000008),interdito('Lisboa')),interdito)
excecao(e_ada('TAP',700000008,'-'),info_interdito)
```

Figura 3.6: Exemplo de conhecimento perfeito para as entidades Adjudicatárias

3.1.2.3 Contrato

No caso dos contratos, decidimos que todos os pârametros do mesmo poderiam ter valores *nulos*, exceto o *id* do contrato. Para além disso, o *id* do contrato é gerado internamente, como poderá ser observado na secção 3.8.

Facto que representa o conhecimento imperfeito: `excecao(contrato(`$Id, Nif\_ad, Nif\_ada, TipoC, TipoP, Descricao, Custo, Prazo, Local, Data$`),` $ValorNuloCorrespondente$`)`

Quando inserimos este tipo de conhecimento, este é armazenado da seguinte forma:

```
excecao(contrato(('I',8), 700000006, 300000005, 'Aquisicao de Servicos',
incerto, 'Distribuicao de correio', interdito(3000), 365,
interdito('Porto'), "06-01-2018"), interdito).
excecao(contrato(8, 700000006, 300000005, 'Aquisicao de Servicos', 'N/A',
'Distribuicao de correio', '-', 365, '-', "06-01-2020"), info_interdito).
excecao(contrato(11, 700000003, 300000005, 'Aquisicao de Servicos', 'Ajuste
Direto', 'Distribuicao de correio', 3000, 365, 'N/A', 'N/A'),incerto).
excecao(contrato(21, 700000004, 700000005, 'Aquisicao de Servicos',
'Consulta previa', 'contrato de intermediação financeira',
impreciso(25000,35000), 'N/A', 'Lisboa', "01-04-2020"),impreciso).
excecao(contrato(22, 700000004, 300000001, 'Aquisicao de Servicos',
'Consulta previa','Fornecimento de Energia Eletrica',
impreciso(100290,110290),365, 'N/A', "01-01-2019"),incerto).
```

Figura 3.7: Exemplo de conhecimento perfeito para as contratos

3.2 Invariantes

Tal como referimos anteriormente, os invariantes podem ser *estruturais*, garantem que não é introduzida informação repetida, ou *referenciais*, garantem que a informação existe, ou seja, não é errada. Garantem, também, que não é retirada informação necessária da base de conhecimento.
Dividimos os invariantes em várias categorias: invariantes universais; invariantes relacionados com a adição e remoção de conhecimento (relacionado com as entidades adjudicante e adjudicatária, como por exemplo, não poder ser removida uma entidade que tenha contratos celebrados) e invariantes relacionados com a celebração de contratos.

3.2.1 Invariantes universais

Alguns invariantes podem ser generalizados, ou seja, existem regras que todos os predicados que se encontram na nossa base de conhecimento respeitam.
Estes invariantes são de um destes dois tipos: ou servem para garantir que não é inserido conhecimento repetido, ou para garantir que não é inserido conhecimento contraditório.

Nesse primeiro tipo, os invariantes vão garantir que não há conhecimento repetido. Tivemos, assim, de definir 2 invariantes:

```
+P :: (findall(P, P, L),
       length(L,1)).
```

Figura 3.8: Invariante para conhecimento positivo repetido

```
+(-P) :: (findall(P, -P, L),
          length(L,1)).
```

Figura 3.9: Invariante para conhecimento negativo repetido

No segundo tipo, para garantir que não há conhecimento contraditório, definimos 2 invariantes:

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

Figura 3.10: Invariante para conhecimento positivo contraditório

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

Figura 3.11: Invariante para conhecimento negativo contraditório

3.2.2 Invariantes para entidade adjudicante

Este invariante torna impossível remover uma entidade adjudicante com um dado NIF ou nome, se algum destes já pertencer a outra entidade adjudicante.

```
+e_ad(Nome,Nif,Morada) :: (findall((Nome,Nif) ,(e_ad(Nome,_,_);e_ad(_,Nif,_) ), S),
                           length(S,N), N == 2).
```

Figura 3.12: Invariante para NIFs diferentes

Este invariante torna impossível remover uma entidade adjudicante que tenha contratos celebrados.

```
-e_ad(Nome,Nif,Morada) :: (findall(Id,contrato(Id,_,Nif,_,_,_,_,_,_,_),S),
                           length(S,0)).
```

Figura 3.13: Invariante para remover só se entidade não tem contratos

3.2.3 Invariantes para entidade adjudicatária

Este invariante torna impossível inserir uma entidade adjudicatária com um dado NIF ou nome, se algum destes já pertencer a outra entidade adjudicatária.

```
+e_ada(Nome,Nif,Morada) :: (findall((Nome,Nif), (e_ada(Nome,_,_);e_ada(_,Nif,_)), S),
                           length(S,N), N == 2).
```

Figura 3.14: Invariante para NIFs diferentes

Este invariante torna impossível remover uma entidade adjudicatária que tenha contratos celebrados.

```
-e_ada(Nome,Nif,Morada) :: (findall(Id,contrato(Id,_,Nif,_,_,_,_,_,_,_),S),
                           length(S,0)).
```

Figura 3.15: Invariante para remover só se entidade não tem contratos

3.2.4 Invariantes para contrato com conhecimento perfeito

Este invariante torna impossível inserir um contrato com um *id* que já se encontra em uso.

```
+contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data) ::
                                        (findall(Id,contrato(Id,_,_,_,_,_,_,_,_,_),S),
                                        length(S,N),
                                        N==1).
```

Figura 3.16: Invariante para *id* de contratos diferentes

Este invariante apenas permite inserir contratos com procedimentos válidos.

```
+contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data) ::
                                                        (TipoP= 'Ajuste Direto';
                                                        TipoP= 'Consulta Previa';
                                                        TipoP='Concurso Publico').
```

Figura 3.17: Invariante para procedimentos válidos nos contratos

Este invariante obriga a que um contrato por ajuste direto cumpra certas restrições.

```
+contrato(Id,Nif_ad,Nif_ada,TipoC,'Ajuste Direto',Descricao,Custo,Prazo,Local,Data) ::(
                                                    diasAno(Data,MaxDias),
                                                    Prazo=<MaxDias,
                                                    Custo =< 5000,
                                                    (TipoC='Contrato De Aquisicao';
                                                    TipoC='Locacao de bens moveis';
                                                    TipoC='Aquisicao de Servicos')).
```

Figura 3.18: Invariante para condições impostas pelo ajuste direto

Este invariante impede a inserção de um contrato se o preço contratual acumulado dos contratos já celebrados seja superior a 75.000 euros, nos últimos 3 anos.

```
+contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data) :: (
                                        anosEcoData(Data,Anos),
                                        somarContratos(Nif_ad,Nif_ada,Anos,Total),
                                        !,
                                        Total-Custo < 75000).
```

Figura 3.19: Invariante para a regra dos 3 anos

Este invariante garante que as duas entidades no contrato existem.

```
+contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data) ::
                                            (encontraAdjudicante(Nif_ad,[L]),
                                            encontraAdjudicataria(Nif_ada, [L1])).
```

Figura 3.20: Invariante para contratos com duas entidades existentes

Este invariante restringe a remoção de contratos que não foram realizados no dia da inserção.

```
-contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data) ::
                                                    data_Atual(Data,sim).
```

Figura 3.21: Invariante para apenas remover contratos no dia da inserção

3.2.5 Invariantes para contrato com conhecimento imperfeito

Este invariante garante que não se pode inserir um contrato com conhecimento imperfeito com um *id* que já se encontra em uso.

```
+excecao(contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data),X) ::
  (findall(Id,excecao(contrato(Id,_,_,_,_,_,_,_,_,_),_),S),
  length(S,N),
  N==1).
```

Figura 3.22: Invariante para *id* de contratos diferentes com conhecimento imperfeito

3.3 Evolução da base do conhecimento

De forma a poder evoluir continuadamente o conhecimento presente na base de conhecimento, criámos predicados de forma a auxiliar e encapsular essa inserção.
Fizemos uso do meta-predicado disponibilizado pelos docentes nas aulas.

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

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

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

Figura 3.23: Meta-predicado evolução

3.3.1 Evolução do conhecimento perfeito

Predicado que regista uma entidade adjudicante com um dado Nome, NIF e morada.

```
novoAdjudicante(Nome,Nif,Morada) :- evolucao(e_ad(Nome,Nif,Morada)).
```

Figura 3.24: Predicado para registar entidade adjudicante

Predicado que regista uma entidade adjudicatária com um dado Nome, NIF e morada.

```
novoAdjudicataria(Nome,Nif,Morada) :- evolucao(e_ada(Nome,Nif,Morada)).
```

Figura 3.25: Predicado para registar entidade adjudicatária

Predicado que regista um dado contrato entre 2 entidades, representadas pelos seus NIFs, um dado tipo de contrato e um dado tipo de procedimento, uma descrição, custo, prazo, local e data.
De forma a agilizar o processo, o *id* é gerado pela base conhecimento. Este será igual ao incremento do maior *id*, para não repetir *ids*. No fim da evolução, é escrito na consola o *id* correspondente.

```
novoContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data) :-
encontraMaior(Id1),
evolucao(contrato(Id1,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data)),
write('ID:'),write(Id1).
```

Figura 3.26: Predicado para registar contrato

3.3.2 Evolução do conhecimento imperfeito

3.3.2.1 Conhecimento incerto

Predicado que permite inserir uma entidade adjudicante com conhecimento incerto.

```
novoAdjudicante(Nome,Nif,Morada,incerto):-
  processa(Morada,R),
  evolucao(excecao(e_ad(Nome,Nif,R),incerto)).
```

Figura 3.27: Predicado para inserir adjudicante com conhecimento incerto

Predicado que permite inserir uma entidade adjudicatária com conhecimento incerto.

```
novoAdjudicataria(Nome,Nif,Morada,incerto):-
  processa(Morada,R),
  evolucao(excecao(e_ada(Nome,Nif,R),incerto)).
```

Figura 3.28: Predicado para inserir adjudicatária com conhecimento incerto

Predicado que permite inserir um contrato com conhecimento incerto.

```
novoContrato(Id1,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,incerto):-
  encontraMaior(Id),
  processaContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,R),
  evolucao(excecao(R,incerto)).
```

Figura 3.29: Predicado para inserir contrato com conhecimento incerto

3.3.2.2 Conhecimento impreciso

Predicado que permite inserir uma entidade adjudicante com conhecimento impreciso.

```
novoAdjudicante(Nome,Nif,Morada,impreciso):-
  processa(Morada,R),
  evolucao(excecao(e_ad(Nome,Nif,R),impreciso)).
```

Figura 3.30: Predicado para inserir adjudicante com conhecimento impreciso

Predicado que permite inserir uma entidade adjudicatária com conhecimento impreciso.

```
novoAdjudicataria(Nome,Nif,Morada,impreciso):-
  processa(Morada,R),
  evolucao(excecao(e_ada(Nome,Nif,R),impreciso)).
```

Figura 3.31: Predicado para inserir adjudicatária com conhecimento impreciso

Predicado que permite inserir um contrato com conhecimento impreciso.

```
novoContrato(Id1,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,impreciso):-
  encontraMaior(Id),
  processaContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,R),
  evolucao(excecao(R,impreciso)).
```

Figura 3.32: Predicado para inserir contrato com conhecimento impreciso

3.3.2.3 Conhecimento interdito

Para poder manter o conhecimento interdito na base de conhecimento, isto é, mesmo que uma morada de uma entidade ou um preço de um contrato sejam *interditos* e de forma a manter essa informação original guardada, é criada uma cópia dessa entidade ou contrato em que, no caso das entidades, o NIF passa a ser um par do tipo ('I', NIF) e o nome também passa a ser um par do tipo ('I', NOME).

Predicado que permite inserir uma entidade adjudicante com conhecimento interdito.

```
novoAdjudicante(Nome,Nif,Morada,interdito):-
  evolucao(excecao(e_ad(('I',Nome),('I',Nif),Morada),interdito)),
  processa(Morada,R),
  evolucao(excecao(e_ad(Nome,Nif,R),interdito)).
```

Figura 3.33: Predicado para inserir adjudicante com conhecimento interdito

Predicado que permite inserir uma entidade adjudicatária com conhecimento interdito.

```
novoAdjudicataria(Nome,Nif,Morada,interdito):-
  evolucao(excecao(e_ada(('I',Nome),('I',Nif),Morada),interdito)),
  processa(Morada,R),
  evolucao(excecao(e_ada(Nome,Nif,R),interdito)).
```

Figura 3.34: Predicado para inserir adjudicatária com conhecimento interdito

Predicado que permite inserir um contrato com conhecimento interdito.

```
novoContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,interdito):-
  encontraMaior(Id1),
  evolucao(excecao(contrato(('I',Id1),Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,
  Prazo,Local,Data),interdito)),
  processaContrato(Id1,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,R),
  evolucao(excecao(R,info_interdito)).
```

Figura 3.35: Predicado para inserir contrato com conhecimento interdito

Predicado auxiliar que permite processar todos os campos de um contrato com conhecimento imperfeito (*incerto*, *impreciso*, *interdito*).

```
processaContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,
contrato(Id,R2,R3,R4,R5,R6,R7,R8,R9,R10)) :-
  (processa(Id,R1),
  processa(Nif_ad,R2),
  processa(Nif_ada,R3),
  processa(TipoC,R4),
  processa(TipoP,R5),
  processa(Descricao,R6),
  processa(Custo,R7),
  processa(Prazo,R8),
  processa(Local,R9),
  processa(Data,R10)).
```

Figura 3.36: Predicado para processar campos de contratos com conhecimento imperfeito

Predicado auxiliar que permite processar um campo com conhecimento imperfeito (*incerto*, *impreciso*, *interdito*).

```
processa(interdito(X),'-').
processa(incerto,'N/A').
processa(X,X).
```

Figura 3.37: Predicado para processar um campo com conhecimento imperfeito

3.4 Involução da base do conhecimento

De forma a poder involuir continuadamente o conhecimento presente na base de conhecimento, criámos predicados de forma a auxiliar e encapsular essa remoção.
Fizemos uso do meta-predicado disponibilizado pelos docentes nas aulas.

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

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

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

Figura 3.38: Meta-predicado involução

3.4.1 Involução do conhecimento perfeito

Remove a entidade adjudicante com o dado NIF.

```
removerAdjudicante(Nif) :- encontraAdjudicante(Nif,[L|T]) , involucao(L).
```

Figura 3.39: Predicado para remover entidade adjudicante

Remove a entidade adjudicatária com o dado NIF.

```
removerAdjudicataria(Nif) :- encontraAdjudicataria(Nif,[L|T]) , involucao(L).
```

Figura 3.40: Predicado para remover entidade adjudicatária

Remove o contrato com o dado *id*.

```
removerContrato(Id) :- encontraContrato(Id,[L|T]) , involucao(L).
```

Figura 3.41: Predicado para remover contrato

3.4.2 Involução do conhecimento imperfeito

3.4.2.1 Conhecimento incerto e impreciso

Neste caso, a involução de conhecimento *incerto* e *impreciso* será tratada de maneira igual, ou seja, não necessita de uma distinção.

Predicado que permite remover uma entidade adjudicante com conhecimento incerto.

```
removeAdjudicante(Nif):- encontraAdjudicante2(Nif,[C]),
  involucao(C).
```

Figura 3.42: Predicado para remover adjudicante com conhecimento incerto

Predicado que permite remover uma entidade adjudicatária com conhecimento incerto.

```
removeAdjudicataria(Nif):- encontraAdjudicataria2(Nif,[C]),
  involucao(C).
```

Figura 3.43: Predicado para remover adjudicatária com conhecimento incerto

Predicado que permite remover um contrato com conhecimento incerto.

```
removeContrato(Id):- encontraContrato2(Id,[C]),
  involucao(C).
```

Figura 3.44: Predicado para remover contrato com conhecimento incerto

3.4.2.2 Conhecimento interdito

Tal como foi referido na secção 3.3.2.3, tivemos de ter em conta a informação interdita na base de conhecimento e a forma como esta estava a ser guardada nela, isto porque eram criadas cópias em que o *id* passava a ser um par ou o NIF passava a ser um par.

Predicado que permite remover uma entidade adjudicante com conhecimento interdito.

```
removeAdjudicante(Nif):- removeInterdito_ad(Nif),
  encontraAdjudicante2(Nif,[C]),
  involucao(C).
```

Figura 3.45: Predicado para remover adjudicante com conhecimento interdito

Predicado auxiliar que permite remover uma entidade adjudicante com conhecimento interdito.

```
removeInterdito_ad(Nif):-
findall(evolucao(excecao(e_ad(('I',Nome),('I',Nif),Morada),TX)),
  evolucao(excecao(e_ad(('I',Nome),('I',Nif),Morada),TX)),[S]),
  involucao(S).
```

Figura 3.46: Predicado auxiliar para remover adjudicante com conhecimento interdito

Predicado que permite remover uma entidade adjudicatária com conhecimento interdito.

```
removeAdjudicataria(Nif):- removeInterdito_ada(Nif),
  encontraAdjudicataria2(Nif,[C]),
  involucao(C).
```

Figura 3.47: Predicado para remover adjudicatária com conhecimento interdito

Predicado auxiliar que permite remover uma entidade adjudicatária com conhecimento interdito.

```
removeInterdito_ada(Nif):-
findall(evolucao(excecao(e_ada(('I',Nome),('I',Nif),Morada),TX)),
  evolucao(excecao(e_ada(('I',Nome),('I',Nif),Morada),TX)),[S]),
  involucao(S).
```

Figura 3.48: Predicado auxiliar para remover adjudicatária com conhecimento interdito

Predicado que permite remover um contrato com conhecimento interdito.

```
removeContrato(Id):-removeInterdito(Id),
  encontraContrato2(Id,[C]),
  involucao(C).
```

Figura 3.49: Predicado para remover contrato com conhecimento interdito

Predicado auxiliar que permite remover um contrato com conhecimento interdito.

```
removeInterdito(Id):-
findall(excecao(contrato(('I',Id),Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,
Prazo,Local,Data),TX),
excecao(contrato(('I',Id),Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,
Local,Data),TX),[S]),
    involucao(S).
```

Figura 3.50: Predicado auxiliar para remover contrato com conhecimento interdito

3.5 Substituição de conhecimento imperfeito

Decidimos criar predicados para a substituição de conhecimento de forma a poder evoluir conhecimento imperfeito na base de conhecimento, isto é, caso tenha sido inserido conhecimento imperfeito, mas, passado algum tempo, se queira substituir por um novo conhecimento, perfeito ou imperfeito, têm de ser usados um dos seguintes predicados.

3.5.1 Entidade Adjudicante

Este predicado é usado quando se quer substituir uma entidade adjudicante com conhecimento imperfeito por conhecimento perfeito.

```
subsE_ad(Nome,Nif,Morada):-
  findall(excecao(e_ad(Nome,Nif,M),P),excecao(e_ad(Nome,Nif,M),P),S),
  length(S,1),
  involucao(excecao(e_ad(Nome,Nif,M),P)),
  novoAdjudicante(Nome,Nif,Morada).
```

Figura 3.51: Predicado para substituir adjudicante com conhecimento imperfeito por perfeito

3.5.2 Entidade Adjudicatária

Este predicado é usado quando se quer substituir uma entidade adjudicatária com conhecimento imperfeito por conhecimento perfeito.

```
subsE_ada(Nome,Nif,Morada):-
  findall(excecao(e_ada(Nome,Nif,M),P),excecao(e_ada(Nome,Nif,M),P),S),
  length(S,1),
  involucao(excecao(e_ada(Nome,Nif,M),P)),
  novoAdjudicataria(Nome,Nif,Morada).
```

Figura 3.52: Predicado para substituir adjudicatária com conhecimento imperfeito por perfeito

3.5.3 Contrato

Este predicado é usado quando se quer substituir contratos com conhecimento interdito por conhecimento perfeito.

```
subsContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data):-
  encontraContrato2(Id,[C]),
  involucao(C),
  removeInterdito(Id),
  novoContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data).
```

Figura 3.53: Predicado para substituir contrato com conhecimento interdito por perfeito

Este predicado é usado quando se quer substituir contratos com conhecimento interdito por conhecimento imperfeito.

```
subsContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,TX):-
  encontraContrato2(Id,[C]),
  involucao(C),
  removeInterdito(Id),
  novoContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,TX).
```

Figura 3.54: Predicado para substituir contrato com conhecimento interdito por imperfeito

Este predicado é usado quando se quer substituir contratos com conhecimento imperfeito, incerto ou impreciso, por conhecimento perfeito.

```
subsContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data):-
  encontraContrato2(Id,[C]),
  involucao(C),
  novoContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data).
```

Figura 3.55: Predicado para substituir contrato com conhecimento imperfeito (incerto ou impreciso) por perfeito

Este predicado é usado quando se quer substituir contratos com conhecimento imperfeito, incerto ou impreciso, por conhecimento imperfeito.

```
subsContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,TX):-
  encontraContrato2(Id,[C]),
  involucao(C),
  novoContrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data,TX).
```

Figura 3.56: Predicado para substituir conhecimento contrato com conhecimento imperfeito (incerto ou impreciso) por imperfeito

Este predicado auxilia a remoção de conhecimento interdito nos contratos, pois quando este é inserido, é criada uma cópia onde está a informação interdita de forma a poder ser consultada no futuro.
Se o contrato original tiver o *id* igual a x, então essa cópia terá como *id* o par ('I', x).

```
removeInterdito(Id) :-
findall(excecao(contrato(('I',Id),Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,
Prazo,Local,Data),TX),
excecao(contrato(('I',Id),Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,
Local,Data),TX),[S]),
  involucao(S).
```

Figura 3.57: Predicado auxiliar para remover conhecimento imperfeito interdito

3.6 Sistema de Inferência

Tendo em conta que este sistema será colocado num mundo aberto, decidimos criar um sistema de inferência também ele capaz de inferir num mundo aberto. Assim, criámos um sistema capaz de inferir se uma dada questão é *verdadeira*, *falsa*, *interdita*, *imprecisa*, *incerta* ou *desconhecida*.

```
si( Questao,verdadeiro ) :- Questao.
si( Questao,falso ) :- -Questao.
si( Questao,R) :- excecao(Questao,R).
si( Questao,desconhecido ).
```

Figura 3.58: Sistema de inferência

Para isso, quando existe prova de que uma dada questão é **verdade**, a resposta será *verdadeiro*. Se houver uma prova, por **negação forte** que a questão é falsa, então a resposta será *falso*.
Depois de testar para estes dois primeiros casos, significa que se trata de conhecimento *imperfeito* ou *desconhecido*, portanto será verificado se existe prova de que a questão é uma **exceção**, se sim, a resposta será esse mesmo tipo de *exceção*, caso contrário, a resposta será *desconhecido*.

3.7 Funcionalidades gerais

Nesta secção serão apresentados os predicados que auxiliam os invariantes e algumas funcionalidades para a pesquisa na base de conhecimento.

Predicado usado para verificar e retornar uma entidade adjudicante com o NIF dado.

```
encontraAdjudicante(Nif,Ad) :- findall(e_ad(Nome,Nif,Morada),
                                       (e_ad(Nome,Nif,Morada);
                                       (excecao(e_ad(Nome,Nif,Morada),TX),
                                        TX \= interdito) ), Ad).
```

Figura 3.59: Predicado para encontrar entidade Adjudicante, dado Nif

Predicado usado para verificar e retornar uma entidade adjudicatária com o NIF dado.

```
encontraAdjudicataria(Nif,Ada) :- findall(e_ada(Nome,Nif,Morada),
                                      (e_ada(Nome,Nif,Morada);
                                      (excecao(e_ada(Nome,Nif,Morada),TX)
                                      ,TX \= interdito)),Ada).
```

Figura 3.60: Predicado para encontrar entidade Adjudicatária, dado Nif

Predicado usado para verificar e retornar um contrato, seja este perfeito ou imperfeito, com o *id* dado.

```
encontraContrato(Id,C) :-
    findall(contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,
    Local,Dat),
    ((contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,
    Data);
    (excecao(contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,
    Data),TX),
    TX \= interdito)),atom_codes(Dat,Data)),C).
```

Figura 3.61: Predicado para encontrar um contrato, dado *id*

Predicado usado para verificar e retornar apenas a exceção do contrato, com o *id* dado.

```
encontraContrato2(Id,C) :-
  findall(excecao(contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,
  Prazo,Local,Data),TX),
  excecao(contrato(Id,Nif_ad,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,
  Local,Data),TX),
  C).
```

Figura 3.62: Predicado para encontrar um contrato com conhecimento imperfeito

Predicado usado para verificar e retornar uma entidade, quer adjudicante como adjudicatária, com um dado NIF.

```
encontraEntidade(Nif ,S) :- findall( Nome ,
    (((excecao(e_ada(Nome,Nif,Morada),TX),TX \= interdito);
    (excecao(e_ad(Nome,Nif,Morada),TX),TX \= interdito));
    (e_ad(Nome,Nif,Morada);e_ada(Nome,Nif,Morada)) ),S).
```

Figura 3.63: Predicado para encontrar uma Entidade, dado Nif

Predicado que retorna a soma das quantias de todos os contratos celebrados entre uma dada entidade adjudicante e uma entidade adjudicatária em todos os anos presentes na lista, variável *Anos*.

```
somarContratos(Nif_ad,Nif_ada,Anos,Total) :- findall((Data,Custo),
    (contrato(Id, Nif_ad, Nif_ada, TipoC, TipoP, Descricao, Custo, Prazo,
    Local, Data);
    (excecao(contrato(Id, Nif_ad, Nif_ada, TipoC, TipoP, Descricao, Custo,
    Prazo, Local, Data), TX),
    TX \= interdito )),S), map1(ano, S, Anos, X),sum_list(X, Total).
```

Figura 3.64: Predicado para calcular o valor acumulado dos contratos nos anos económicos

3.8 Funcionalidades Extra

Nesta secção serão apresentados predicados que permitem uma pesquisa mais restritiva à base de conhecimento.

Predicado que retorna todos os contratos de uma dada entidade adjudicante com o NIF dado.

```
encontraTodosContratos(Nif,L) :-

    findall(contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Dat),
    ((contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data);
    (excecao(contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data),TX),
    TX \= interdito)),atom_codes(Dat,Data) ),L).
```

Figura 3.65: Predicado para encontrar todos os contratos de uma entidade Adjudicante

Predicado que retorna todos os contratos em vigor de uma dada entidade adjudicante com o NIF dado.

```
encontraTodosEmVigor(Nif,L) :-

    findall(contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Dat),
    (((contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data);
    (excecao(contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data),TX),
    TX \= interdito)),verificaPrazo(Data,Prazo,em_vigor)),
    atom_codes(Dat,Data)) ,L).
```

Figura 3.66: Predicado para encontrar todos os contratos de uma entidade Adjudicante em vigor

Predicado que permite encontrar todos os contratos acima de um dado valor, *Val*, tendo as opções de procurar por contratos em vigor ou caducados.

```
encontraTodosConAcimaDe(Val,em_vigor,L) :-
findall(contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Dat),
    ((contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data);
    (excecao(contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data),TX),
    TX \= interdito)),Custo >= Val,verificaPrazo(Data,Prazo,em_vigor)
    ,atom_codes(Dat,Data)) ,L).

encontraTodosConAcimaDe(Val,caducado,L) :-
findall(contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Dat),
    ((contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data);
    (excecao(contrato(Id,Nif,Nif_ada,TipoC,TipoP,Descricao,Custo,Prazo,Local,Data),TX),
    TX \= interdito)),Custo >= Val,verificaPrazo(Data,Prazo,caducado),
    atom_codes(Dat,Data)) ,L).
```

Figura 3.67: Predicado para encontrar todos os contratos acima de um certo valor, caducados ou em vigor

Predicado que retorna a soma das quantias de todos os contratos celebrados que se encontrem em vigor, entre uma dada entidade adjudicante e uma entidade adjudicatária em todos os anos presentes na lista, variável *Anos*.

```
somarContratos_Vigor(Nif_ad,Nif_ada,Anos,Total) :-
findall((Data,Custo,Prazo),
    (contrato(Id, Nif_ad, Nif_ada, TipoC, TipoP, Descricao, Custo, Prazo,
    Local, Data);
    (excecao(contrato(Id, Nif_ad, Nif_ada, TipoC, TipoP, Descricao, Custo,
    Prazo, Local, Data), TX),
    TX \= interdito)),S), map2(ano, S, Anos, X), sum_list(X, Total).
```

Figura 3.68: Predicado para calcular o valor acumulado dos contratos que estão em vigor

Predicado que retorna a soma as quantias de todos os contratos celebrados por uma dada entidade adjudicante.

```
totalAcumulado(Nif, Tot) :-
    findall(Custo, (contrato(Id, Nif, Nif_ada, TipoC, TipoP, Descricao,
    Custo, Prazo, Local, Data);
    (excecao(contrato(Id, Nif, Nif_ada, TipoC, TipoP, Descricao, Custo,
    Prazo, Local, Data), TX)
    ,TX \= interdito)) ,S), sum_list(S, Tot).
```

Figura 3.69: Predicado para calcular o valor total acumulado dos seus contratos

Predicado que retorna as entidades por ordem decrescente da soma das quantias dos seus contratos.

```
topEntidadesAd(S1) :- findall((Nif_ad,Custo),
    (contrato(Id, Nif_ad, Nif_ada, TipoC, TipoP, Descricao, Custo, Prazo,
    Local, Data);
    (excecao(contrato(Id, Nif_ad, Nif_ada, TipoC, TipoP, Descricao, Custo,
    Prazo, Local, Data), TX),
    TX \= interdito)), S), juntarpares(S, S, S2), map3(encontraEntidade, S2,
    S1).
```

Figura 3.70: Predicado para retornar as Entidades por ordem decrescente de valor acumulado nos contratos

Predicado que permite ter o próximo *id* disponível para um contrato.

```
encontraMaior(ID) :- findall(Id,
    ((contrato(Id, Nif_ad, Nif_ada, TipoC, TipoP, Descricao, Custo, Prazo,
    Local, Data);
    (excecao(contrato(Id, Nif_ad, Nif_ada, TipoC, TipoP, Descricao, Custo,
    Prazo, Local, Data),TX),
    TX \= interdito))), L), maior(L,ID).

maior([],1).
maior([H],H1) :-  H1 is H + 1.
maior([F,S|T], R) :- F > S, maior([F|T],R).
maior([F,S|T], R) :- F =< S, maior([S|T],R).
```

Figura 3.71: Predicado para automatização do *id* dos contratos

3.9 Processamento de Datas

Nesta secção serão apresentados os predicados que auxiliaram a manipulação de datas.

Predicado que verifica se uma lista de números é uma data e transforma esta mesma num dado dia, mês e ano, uma vez que, em *prolog*, as *strings* delimitadas pelo carater "são tratadas como inteiros.
Por exemplo, na *string* "3/5/2020", cada carater será transformado no seu número ASCII. Aqui estão contemplados os casos em que o dia é só um carater ou dois carateres, assim como o mês, por exemplo, "13/5/2020" e "3/11/2020" ou, em ambos os casos, "31/12/2020".

```
conv_DateToParam([C0,C1,C2,C3,C4,C5,C6,C7,C8,C9],D,M,A):-
    strToNum([C6,C7,C8,C9],4,A),
    strToNum([C3,C4],2,M),
        strToNum([C0,C1],2,D),
        dataEValida(D,M,A),!,
        stringToDelim(C2),
        stringToDelim(C5).

conv_DateToParam([C1,C2,C3,C4,C5,C6,C7,C8,C9],D,M,A):-
    strToNum([C6,C7,C8,C9],4,A),
        strToNum([C3,C4],2,M),
        strToNum([C1],1,D),
        dataEValida(D,M,A),!,
        stringToDelim(C2),
        stringToDelim(C5).

conv_DateToParam([C0,C1,C2,C4,C5,C6,C7,C8,C9],D,M,A):-
    strToNum([C6,C7,C8,C9],4,A),
        strToNum([C4],1,M),
        strToNum([C0,C1],2,D),
        dataEValida(D,M,A),!,
        stringToDelim(C2),
        stringToDelim(C5).

conv_DateToParam([C1,C2,C4,C5,C6,C7,C8,C9],D,M,A):-
    strToNum([C6,C7,C8,C9],4,A),
        strToNum([C4],1,M),
        strToNum([C1],1,D),
        dataEValida(D,M,A),!,
        stringToDelim(C2),
        stringToDelim(C5).
```

Figura 3.72: Predicado que converte um string com uma data em variáveis separadas

Predicado que verifica se uma dada data ainda se encontra em vigor, tendo em conta o prazo.

```
verificaPrazo(Data,Prazo,Res):-
    conv_DateToParam(Data,D,M,A),
        dateToDays(D,M,A,Days),
        R1 is Days +Prazo,
        curdate(D1,M1,S1),
        dateToDays(D1,M1,S1,Days2),!,
        maior(R1,Days2,Res).

maior(X,Y,em_vigor):- X >= Y .
maior(X,Y,caducado):- X < Y .
```

Figura 3.73: Predicados que verificam se uma data está em vigor

Preficado que dado um dia, mês e ano, retorna o número de dias a que essa data corresponde.

```
dateToDays(D,M,A,Days):-
    Bissexto is mod(A,4),
        ytoDay(A,R2),
         countmonth(Bissexto,M,R1),
         Days is D+R1+R2.
```

Figura 3.74: Predicado que converte uma data para dias

Predicado que dado uma data retorna uma lista com os últimos três anos.
Por exemplo, dada a *string* "3/5/2020", retorna uma lista com os seguintes elementos: 2018, 2019 e 2020.

```
anosEcoData([C0,C1,C2,C3,C4,C5,C6,C7,C8,C9],A):-
    strToNum([C6,C7,C8,C9],4,R),
        R1 is R-1,R2 is R-2,
        A =[R2,R1,R].
```

Figura 3.75: Predicado que converte uma string com uma data em variáveis separadas

Capítulo 4

Análise de Resultados

Demonstração da regra dos Ajuste Diretos:

- Criámos uma entidade adjudicante;
- Mostrámos que realmente está na base de conhecimento;
- Tentativa de inserção de um contrato inválido (Ajuste Direto $\leq$ 5.000);
- Tentativa bem sucedida de inserção de um contrato de ajuste direto (podemos verificar o cálculo do *id* automático);
- Mostrámos como é armazenado na base de conhecimento.

```
SICStus 4.3.0 (x86_64-darwin-13.1.0): Thu May  8 05:39:06 PDT 2014
Licensed to studentSP4.3di.uminho.pt
| ?- novoAdjudicante('Municipio de Braga, M.B',700000031,'Praça do Município, 4700-435 Braga').
yes
| ?- encontraAdjudicante(700000031,E).
E = [e_ad('Municipio de Braga, M.B',700000031,'Praça do Município, 4700-435 Braga')] ?
yes
| ?- novoContrato(ID,700000031,300000003,'Locacao de bens moveis','Ajuste Direto','Combustivel para automoveis',5001,30,'Porto',"06-01-2018").
no
| ?- novoContrato(ID,700000031,300000003,'Locacao de bens moveis','Ajuste Direto','Combustivel para automoveis',5000,30,'Porto',"06-05-2018").
ID:12
yes
| ?- encontraContrato(12,C).
C = [contrato(12,700000031,300000003,'Locacao de bens moveis','Ajuste Direto','Combustivel para automoveis',5000,30,'Porto','06-05-2018')] ?
yes
```

Figura 4.1: Resultado da Regra dos contratos por Ajuste Direto

Demonstração da regra dos 3 anos:

- Adiciona-se um contrato no valor de 70.000;
- Como na demonstração anterior, adicionámos um de 5.000, o resultado do predicado *somarContratos* é de 75.000;
- Tentámos inserir um outro contrato e não é permitido, pois viola o invariante.

```
| ?- novoContrato(ID,700000031,300000003,'Contrato De Aquisicao','Consulta Previa','Combustivel para automoveis',70000,365,'Braga',"07-01-2019").
ID:13
yes
| ?- somarContratos(700000031,300000003,[2018,2019,2020],Tot).
Tot = 75000 ?
yes
| ?- novoContrato(ID,700000031,300000003,'Contrato De Aquisicao','Consulta Previa','Combustivel para automoveis',10000,30,'Braga',"07-01-2020").
no
```

Figura 4.2: Resultado da Regra dos 3 anos

Demonstração da funcionalidade extra para calcular top de entidades adjudicantes, relativamente ao valor do contrato realizado

```
| ?- topEntidadesAd(TOP).

TOP = [(['NAV Portugal, EPE'],5008000),(['CGD - Caixa Geral de DepÃ³sitos, SA'],5000000),(['ULSAM - Unidade Local de SaÃºde do Alto Minho, EPE'],
130000),(['BRISA - Autoestradas de Portugal, S.A.'],100000),(['Municipio de Braga, M.B'],75000),(['CP - Comboios de Portugal'],75000),(['PME Inve
stimentos - Sociedade de Investimentos, SA'],53000)] ? yes
| ?-
```

Figura 4.3: Resultado do predicado Top entidades

Demonstração de conhecimento imperfeito:

- Criámos uma entidade adjudicante;
- Associámos um contrato interdito à mesma;
- Mostrámos como encontrar o contrato com a informação *interdita* (este predicado, alegadamente, não deve ser conhecido pelo utilizador comum);
- Mostrámos como encontrar o contrato com a informação *interdita* oculta;
- Substituímos o contrato por conhecimento imperfeito *interdito*;
- Mostrámos que realmente funcionou;
- Substituímos o contrato por informação perfeita;
- Mostrámos novamente essa informação. Como temos um invariante que não permite inserir conhecimento com *ids* iguais, pode concluir-se que removeu a informação.

```
SICStus 4.3.0 (x86_64-darwin-13.1.0): Thu May  8 05:39:06 PDT 2014
Licensed to studentSP4.3di.uminho.pt
| ?- novoAdjudicante('Municipio de Braga, M.B',700000031,'Praça do Município, 4700-435 Braga').
yes
| ?- novoContrato(ID,700000031,300000003,'Aquisicao de Servicos',interdito('Concurso Publico'),interdito('Fornecimento de Luz'),interdito(30000),365,'Porto',"08
-01-2018",interdito).
ID:12
yes
| ?- encontraContrato2(('I',12),C).
C = [excecao(contrato(('I',12),700000031,300000003,'Aquisicao de Servicos',interdito('Concurso Publico'),interdito('Fornecimento de Luz'),interdito(30000),365,'
Porto',[48,56,45,48,49,45,50|...]),interdito)] ?
yes
| ?- encontraContrato2(12,C).
C = [excecao(contrato(12,700000031,300000003,'Aquisicao de Servicos',_,_,_,365,'Porto',[48,56,45,48,49,45,50|...]),info_interdito)] ?
yes
| ?- subsContrato(12,700000031,300000001,'Aquisicao de Servicos','Concurso Publico','Fornecimento de Luz',interdito(30000),365,'Porto',"08-01-2018",interdito).
ID:12
yes
| ?- encontraContrato(12,R).
R = [contrato(12,700000031,300000001,'Aquisicao de Servicos','Concurso Publico','Fornecimento de Luz',_,365,'Porto','08-01-2018')] ?
yes
| ?- subsContrato(12,700000031,300000001,'Aquisicao de Servicos','Concurso Publico','Fornecimento de Luz',30000,365,'Porto',"08-01-2018").
ID:12
yes
| ?- encontraContrato(12,R).
R = [contrato(12,700000031,300000001,'Aquisicao de Servicos','Concurso Publico','Fornecimento de Luz',30000,365,'Porto','08-01-2018')] ?
yes
| ?-
```

Figura 4.4: Amostra para conhecimento imperfeito *interdito*

Demonstração de conhecimento incerto e impreciso:

- Inserimos um contrato com conhecimento *impreciso* e *imperfeito*. É de notar que o tipo de exceção é *incerto*, porém também poderia ser *impreciso*, uma vez que, existem os dois tipos de valores nulos;
- Substituímos este por conhecimento perfeito;
- Mostrámos o novo contrato, que substituiu o conhecimento *impreciso*;
- Criámos outro contrato apenas com conhecimento *impreciso*;
- Substituímos este por conhecimento perfeito;
- Mostrámos o contrato;
- Por fim, mostrámos que o contrato com a informação *imperfeita* foi removido, pois o predicado *encontracontrato2* apenas procura exceções.

```
| ?-
novoContrato(ID,700000031,300000001,'Aquisicao de Servicos',impreciso('Ajuste Direto','Consulta Previa'),incerto,3500,30,'Porto',"09-01-2018",incerto).
ID:13
yes
| ?- subsContrato(13,700000031,300000001,'Aquisicao de Servicos','Consulta Previa','Aumento nivel Tensão',3500,30,'Porto',"09-01-2018").
ID:13
yes
| ?- encontraContrato(13,R).
R = [contrato(13,700000031,300000001,'Aquisicao de Servicos','Consulta Previa','Aumento nivel Tensão',3500,30,'Porto','09-01-2018')] ?
yes
| ?- novoContrato(ID,700000031,300000001,'Contrato De Aquisicao',impreciso('Ajuste Direto','Consulta Previa'),impreciso('Paineis solares','Disjuntores'),3500,36
5,'Porto',"09-03-2020",impreciso).
ID:14
yes
| ?- subsContrato(14,700000031,300000001,'Contrato De Aquisicao','Ajuste Direto','Paineis solares',4000,365,'Porto',"09-03-2020").
ID:14
yes
| ?- encontraContrato(14,R).
R = [contrato(14,700000031,300000001,'Contrato De Aquisicao','Ajuste Direto','Paineis solares',4000,365,'Porto','09-03-2020')] ?
yes
| ?- encontraContrato2(14,R).
R = [] ?
yes
```

Figura 4.5: Amostra para conhecimento imperfeito incerto e impreciso

Demonstração da funcionalidade extra que nos permite obter todos os contratos em vigor, de uma entidade adjudicante, acima de um valor específico

```
| ?- encontraTodosConAcimaDe(1000,em_vigor,R).
R = [contrato(2,700000000,700000005,'Aquisicao de Servicos','Ajuste Direto','contrato de intermediaÃ§Ã£o financeira',25000,366,'Lisboa','01-04-2020'),contrato(1
0,700000005,700000004,'Empreitadas de obras pÃºblicas','Consulta Previa','Investimento publico para a construÃ§Ã£o de um novo hospital ',5000000,180,'Viana do C
astelo','01-01-2020'),contrato(14,700000031,300000001,'Contrato De Aquisicao','Ajuste Direto','Paineis solares',4000,365,'Porto','09-03-2020'),contrato(8,700000
006,300000005,'Aquisicao de Servicos','Ajuste Direto',-,3000,365,'Porto','06-01-2020')] ?
yes
```

Figura 4.6: Resultado do Predicado *encontraTodosConAcimaDe*

Capítulo 5

Conclusões, Decisões e Sugestões

Devido à existência de informação não consistente na área dos contratos públicos, criámos este sistema proposto pelos docentes, capaz de representar conhecimento perfeito positivo ou conhecimento imperfeito positivo.
Optámos por não usar conhecimento negativo, pois considerámos que, no mundo dos contratos públicos, essa faceta não era muito relevante, nem prática, nem iria trazer grandes vantagens.
Dado que este trabalho foi um processo iterativo, explicaremos, de seguida, as diversas decisões que tivemos de tomar ao longo do mesmo.
Uma vez que o utilizador comum destes sistemas não é um indivíduo da área da informática, decidimos criar um sistema capaz de processar as datas em strings (Dia-Mês-Ano). Decidimos também criar um invariante que só permite remover contratos realizados no dia da inserção na base de conhecimento. Esta implementação pode ser melhorada, visto que um dia poderia ser insuficiente para a remoção, devido à entidade reguladora não conseguir processar todos os contratos recebidos nesse mesmo dia, ou até mesmo não receber os contratos no dia da realização destes. Porém este sistema permite uma maior segurança contra utilizadores mal intencionados. Uma melhoria neste sistema poderia ser a criação de uma data de inserção para cada contrato, garantindo, assim, que apenas dependia da data de inserção no sistema.
Criámos também, um conjunto de predicados que permitem uma pesquisa específica na base de conhecimento, o que facilita a manipulação deste sistema.
Todos os invariantes, tanto os referenciais como estruturais, foram pensados pormenorizadamente para o funcionamento adequado da base de conhecimento, impondo regras na evolução e involução do mesmo.
Em suma, acreditámos que o trabalho foi bem desenvolvido, tendo sido bastante vantajoso para o aperfeiçoamento dos nossos conhecimentos adquiridos ao longo da Unidade Curricular e que conseguimos demonstrar grande parte do poder desta nova ferramenta nos sistemas de representação de conhecimento e raciocínio aplicando programação em lógica estendida.

Apêndice A

Código dos Testes usados no Capítulo 4

```
% Testes usados na análise de resultados
%-------------------------------- - - - - - - - - - - - - - - - - -

% Criação de uma entidade
novoAdjudicante('Municipio de Braga, M.B',700000031,'Praça do Município,
4700-435 Braga').
encontraAdjudicante(700000031,E).

% Invariante do Ajuste Direto
novoContrato(ID,700000031,300000003,'Locacao de bens moveis','Ajuste
Direto','Combustivel para automoveis',5001,30,'Porto',"06-01-2018").
novoContrato(ID,700000031,300000003,'Locacao de bens moveis','Ajuste
Direto','Combustivel para automoveis',5000,30,'Porto',"06-05-2018").

% Invariante dos 3 anos
novoContrato(ID,700000031,300000003,'Contrato De Aquisicao','Consulta
Previa','Combustivel para automoveis',70000,365,'Braga',"07-01-2019").
somarContratos(700000031,300000003,[2018,2019,2020],Tot).
novoContrato(ID,700000031,300000003,'Contrato De Aquisicao','Consulta
Previa','Combustivel para automoveis',10000,30,'Braga',"07-01-2020").

% Extra
topEntidadesAd(TOP).

% Imperfeito
novoContrato(ID,700000031,300000003,'Aquisicao de
Servicos',interdito('Concurso Publico'),interdito('Fornecimento de
Luz'),interdito(30000),365,'Porto',"08-01-2018",interdito).
encontraContrato2(('I',12),C).
encontraContrato2(12,C).
subsContrato(12,700000031,300000001,'Aquisicao de Servicos','Concurso
Publico','Fornecimento de
Luz',interdito(30000),365,'Porto',"08-01-2018",interdito).
subsContrato(12,700000031,300000001,'Aquisicao de Servicos','Concurso
Publico','Fornecimento de Luz',30000,365,'Porto',"08-01-2018").
```

```
encontraContrato(12,R).

% Incerto e Impreciso
novoContrato(ID,700000031,300000001,'Aquisicao de
Servicos',impreciso('Ajuste Direto','Consulta
Previa'),incerto,3500,30,'Porto',"09-01-2018",incerto).
subsContrato(13,700000031,300000001,'Aquisicao de Servicos','Consulta
Previa','Aumento nivel Tensão',3500,30,'Porto',"09-01-2018").
novoContrato(ID,700000031,300000001,'Contrato De
Aquisicao',impreciso('Ajuste Direto','Consulta Previa'),impreciso('Paineis
solares','Disjuntores'),3500,365,'Porto',"09-03-2020",impreciso).
subsContrato(14,700000031,300000001,'Contrato De Aquisicao','Ajuste
Direto','Paineis solares',4000,365,'Porto',"09-03-2020").
encontraContrato2(14,R).

% Extra
encontraTodosConAcimaDe(1000,em\_vigor,R).
```