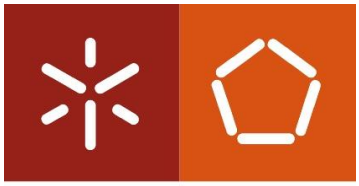

Universidade do Minho

Mestrado Integrado de Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2ºSemestre - 2019/20)

Relatório SRCR - TP1

Grupo Nº 33

A85813 António Alexandre Carvalho Lindo

A84414 João Pedro Gonçalves Correia

A85400 Nuno Azevedo Alves da Cunha

A85919 Pedro Dias Parente

Braga

3 de Maio de 2020

Resumo

Este relatório tem como objetivo descrever a solução criada pelo grupo para a problemática proposta pelos docentes da cadeira de Sistemas de Representação de Conhecimento e Raciocínio.

A meta do trabalho consistiu na implementação de um sistema que fosse capaz de descrever e caracterizar o universo da contratação pública, tendo como objetivo a utilização do conhecimento adquirido na linguagem declarativa Prolog para a modelação correta e consistente das funcionalidades associadas ao nosso universo.

Foi utilizada lógica extendida, criando um terceiro valor lógico para além do verdadeiro e falso: o desconhecido. Isto levará à existência de conhecimento imperfeito, que deverá ser manipulado de forma cautelosa, lidando com os vários tipos de conhecimento incompleto.

Como tal, foi desenvolvido um sistema de inserção e remoção de conhecimento que será regido por um conjunto extenso de invariantes que ditarão as regras de manipulação do conhecimento, lidando também com a evolução e involução de conhecimento imperfeito para perfeito.

Esta mecânica culminará num sistema de inferência completo capaz de responder às inquisições do utilizador acerca dos registos públicos de contratação estatal.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Este trabalho teve como objetivo a representação do universo de contratação pública, recorrendo, para esse fim, à programação em lógica extendida.

Um contrato público é um meio ao dispor da administração pública quando surge a necessidade de adquirir bens ou serviços fornecidos por outras entidades..

Este universo consistirá, assim, na existência de Adjudicantes, Adjudicatários e os Contratos em si. Iniciando a partir de um conhecimento inicial, que foi representado através de factos, pretendeu-se que o sistema permitisse a evolução e involução desse conhecimento, sem alterar a consistência da base de conhecimento.

Para assegurar a consistência da base de conhecimento foram definidos invariantes, regras base que teriam de ser respeitados, quer na inserção como na remoção de conhecimento.

Capítulo 2

Descrição do Trabalho e Análise de Resultados

O trabalho está divido em cinco partes que representarão cada uma uma parte do processo de manipulação do conhecimento relativo ao nosso universo.

O primeiro componente será a **base de informação**, onde será guardado o conhecimento pré-existente em relação às entidades contratuais e contratos, tanto conhecimento perfeito como imperfeito. Esta será depois atualizada através da mecânica de evolução e involução de conhecimento.

O componente dos **invariantes** ditará as regras que limitarão a inserção e remoção de conhecimento na base de informação. Os invariantes poderão ser de dois tipos: *estrutural* se apenas depender do próprio predicado que está a ser inserido ou *referencial* caso a inserção de um predicado dependa de outros predicados também.

O componente da **evolução e involução** implementará a mecânica de inserção e remoção de conhecimento. Este conhecimento tanto poderá ser perfeito como imperfeito. Como tal, será necessário ter um sistema complexo que tratará de transicionar conhecimento imperfeito para perfeito, removendo o conhecimento imperfeito da base de informação, de forma a não gerar informação contraditória e incoerente.

O componente do **sistema de inferência** será responsável por responder às inquisições do utilizador sobre informação acerca, tanto das entidades contratuais, como do próprio contrato. Para além de um predicado capaz de indicar a existência de entidades e contratos, existirão outros predicados capazes de responder a questões mais específicas como, por exemplo, quais contratos estão associados a uma específica entidade.

O componente dos **predicados auxiliares** irá conter a implementação de certas mecânicas transversais aos anteriores módulos, como a mecânica de validação de datas e NIFs.

2.1 Base de Informação

A base de conhecimento terá como fontes de conhecimento as seguintes entidades: Adjudicantes, Adjudicatárias e Contratos. Estas encontram-se definidas com o seguinte formato:

A entidade **adjudicante** representa uma entidade que, quando celebra um contrato, está a contratar serviços a uma outra entidade.

Esta será caraterizada pelo seu ID, nome, NIF, o tipo de entidade (por exemplo "Organismo de Administração pública"), e pela sua morada. O *tipo de de entidade* será uma informação relativa à atividade pública de entidade adjudicante. Esta será crucial na validação dos contratos públicos, como é explicado na secção referente ao invariante de tipo de entidade, em 2.2.2.

Adjudicantes : Id, Nome, NIF, TipoEntidade, Morada $\rightarrow \{V, F, D\}$

A entidade **adjudicatária** representa a entidade que, na celebração de um contrato, irá prestar serviços.

A sua estrutura será bastante semelhante à entidade adjudicante. É também caraterizada pelo seu ID, nome, NIF, o tipo de entidade, e pela sua morada. A diferença principal em relação à entidade adjudicante será no campo de *tipo de entidade* que será mais amplo e permissivo, de acordo com a explicação em a 2.2.3.

Adjudicatárias : Id, Nome, NIF, TipoEntidade, Morada $\rightarrow \{V, F, D\}$

O **contrato** será caraterizado pelo seu ID, o ID da entidade adjudicante associada, o ID da entidade adjudicatária associada, o tipo de contrato, o tipo de procedimento, a sua descrição, o valor associado, o prazo, o local e se o contrato é subsidiado em pelo menos 50% ou não.

Contratos : IdContrato, IdAdjudicante, IdAdjudicatária, TipoDeContrato, TipoDeProcedimento, Descrição, Valor, Prazo, Local, Subsidiado $\rightarrow \{V, F, D\}$

Devido à extensão da programação em lógica, para todas estas fontes de conhecimento os valores lógicos obtidos de inquisições feitas em relação a elas poderão ser: **Verdadeiro**, **Falso** ou **Desconhecido**.

O valor lógico *desconhecido* deve-se à existência de conhecimento imperfeito, que tornará impossível verificar a veracidade de certas questões.

É importante realçar também que por causa da programação em lógica estendida e do nosso sistema de inferência apenas predicados explicitamente falsos (conhecimento perfeito negativo) ou que não se encontrem na nossa base de conhecimento com exceções são considerados **falsos**. Apenas os predicados explicitamente verdadeiros (conhecimento perfeito positivo) são efetivamente **verdadeiros** e por último, predicados que não se encontrem nestas duas áreas (conhecimento imperfeito com exceções) são considerados **desconhecidos**. De seguida entraremos em mais detalhe sobre estes tipos diferentes de conhecimento.

2.1.1 Conhecimento Perfeito

Conhecimento Perfeito refere-se ao tipo de conhecimento em que todos os campos do predicado têm que ter valores exatos completamente dentro das regras que inferimos. Porém, este conhecimento pode ser **positivo** ou **negativo**, sendo que a diferença é que o positivo indica explicitamente que o predicado **existe** e o negativo indica explicitamente que esse predicado **não existe**.

Figura 2.1: Inserção de Conhecimento Perfeito Positivo

```
% Conhecimento Perfeito Positivo
contrato(0,1,2,'Aquisicão de serviços','Consulta previa','Assessoria jurdica','Concerto', 400, 50,'Alto de Basto',data(14,02,2020),false).
perfeito(contrato(0)).
contrato(1,2,1,'Aquisicão de serviços','Consulta previa','Assessoria juridica','Concerto' ,500, 60,'Alto de Basto',data(15,02,2020),false).
perfeito(contrato(1)).
contrato(2,2,2,'Aquisição de serviços','Consulta previa','Assessoria juridica','Concerto' ,600, 70,'Alto de Basto',data(16,02,2020),false).
perfeito(contrato(2)).
contrato(3,1,3,'Aquisição de serviços','Consulta previa','Assessoria juridica', 'Concerto',700, 80,'Alto de Basto',data(17,02,2020),false).
perfeito(contrato(3)).
contrato(4,1,3,'Aquisicao de serviços','Procedimento de negociacao','Concerto', 800, 90,'Alto de Basto',data(18,02,2020),false).
perfeito(contrato(4)).
contrato(5,7,5,'Aquisicao de servicos', 'Consulta previa', 'Acabamentos', 1000, 100, 'Adega Trincao', data(12,05,2020),false).
perfeito(contrato(5)).
```

Figura 2.2: Inserção de Conhecimento Perfeito Negativo

```
% Conhecimento Perfeito Negativo
-contrato(1000,1,2,'Locacao de bens moveis', 'Ajuste direto','Arrendamento de espaço', 100, 30,'Cruzeiro de Felgueiras',data(2,02,2020),false).
perfeito(contrato(4)).
```

2.1.2 Conhecimento Imperfeito

Ao contrário do Conhecimento Perfeito, o Conhecimento Imperfeito refere-se ao tipo de conhecimento em que não temos a certeza que valores têm alguns dos campos. Porém, por causa da forma como decidimos implementar a modulação do problema, apenas em alguns campos podem ser aceites como tendo informação imcompleta. No caso do Adjudicante e Adjudicatária apenas podem ter uma morada desconhecida, e no contrato apenas podem ter uma descrição desconhecida, pois achamos que os outros campos não faziam sentido terem a possibilidade de estarem imcompletos.

Também é de realçar que o Conhecimento Imperfeito pode ser de 3 tipos: **Impreciso**, **Incerto** ou **Interdito**.

Conhecimento Imperfeito Impreciso:

Conhecimento Imperfeito Impreciso diz respeito a conhecimento em que se tem vários valores para uma mesma entidade, mas não se têm a certeza qual desses corresponde aos valores reais, e por isso é desconhecido.

Figura 2.3: Inserção de Conhecimento Imperfeito Impreciso

```
% Conhecimento Imperfeito Impreciso
% Não se sabe qual das moradas é
excecao(adjudicataria(200,'Antonio Reparações LDA',500000000,'Pessoa coletiva','Rua do Crescente')).
excecao(adjudicataria(200,'Antonio Reparações LDA',500000000,'Pessoa coletiva','Rua da Nascente')).
impreciso(adjudicataria(200)).
excecao(adjudicataria(201,'Padaria Armando',500123454,'Pessoa coletiva','Rua de Barros')).
excecao(adjudicataria(201,'Padaria Armando',500123454,'Pessoa coletiva','Rua do Fontelo')).
impreciso(adjudicataria(201)).
```

Conhecimento Imperfeito Incerto:

Conhecimento Imperfeito Incerto diz respeito a conhecimento em que se sabe que um valor existe (por exemplo, a morada de um adjudicante) mas não se sabe qual é.

Figura 2.4: Inserção de Conhecimento Imperfeito Incerto

```
% Conhecimento Imperfeito Incerto
% Não se sabe a morada
adjudicataria(100,'Tasquinha Bracarense',568744322,'Pessoa coletiva',morada_desconhecida).
excecao(adjudicataria(IdAda,Nome,NIF,TipoEntidade,_)):-
    adjudicataria(IdAda,Nome,NIF,TipoEntidade,morada_desconhecida).
incertoMorada(adjudicataria(100),_).
adjudicataria(101,'Alberto Sá',568744365,'Pessoa singular',morada_desconhecida).
excecao(adjudicataria(IdAda,Nome,NIF,TipoEntidade,_)):-
    adjudicataria(IdAda,Nome,NIF,TipoEntidade,morada_desconhecida).
incertoMorada(adjudicataria(101),_).
```

Conhecimento Imperfeito Interdito:

Por último, Conhecimento Imperfeito Interdito, diz respeito a conhecimento em que se sabe que um valor existe (por exemplo, a morada de um adjudicante) mas que **não é possivel ser conhecido**.

Figura 2.5: Inserção de Conhecimento Imperfeito Interdito

```
% Conhecimento Imperfeito Interdito
% Interdito saber o nome do adjudicante
adjudicataria(300,nome_interdito,511111118,'Pessoa coletiva','Rua Nova').
excecao(adjudicataria(IdAd,_,NIF,TipoEntidade,Morada)):-
    adjudicataria(IdAd,nome_interdito,NIF,TipoEntidade,Morada).
nulo(nome_interdito).
interdito(adjudicataria(300)).
```

2.2 Invariantes sobre conhecimento

A existência de invariantes que ditem as regras de inserção e remoção de conhecimento é crucial para a preservação da coerência e veracidade da base de informação. Desta forma, foram desenvolvidos invariantes que tanto explicitarão a forma como se pode manusear e manipular o conhecimento, tanto de forma geral como de forma específica para cada uma das entidades do universo da contratação pública. A motivação da existência dos invariantes será então tanto por razões de lógica de funcionamento da base de informação, como de respeito ao Código de Contratos Públicos.

2.2.1 Invariantes gerais

De forma geral, pretende-se manter a coerência do conhecimento inserido na base de informação. Sendo assim, pretende-se evitar que se insira conhecimento positivo quando esse exato termo já existe sobre a forma de conhecimento negativo. A mesma mecânica deve também ser aplicada à inserção de conhecimento negativo aquando da existência desse conhecimento sob a forma positiva.

Figura 2.6: Invariante sobre conhecimento contraditório

```
%Invariante estrutural: Não permitir  a inserção de conhecimento contraditório
+Termo :: (
    nao(-Termo)
).

%Invariante estrutural: Não permitir  a inserção de conhecimento contraditório
+(-Termo) :: (
    nao(Termo)
).
```

Foi também criado um invariante que impeça que a mesma exceção seja inserida novamente, o que facilitará posteriormente o manuseamente do conhecimento imperfeito.

Figura 2.7: Invariante sobre exceções repetidas

```
% Invariante estrutural: Não permitir excecoes repetidas
+(excecao(Termo)) :: (
    solucoes(Termo, excecao(Termo), R),
    comprimento(R, 1)
).
```

2.2.2 Invariantes sobre entidades adjudicantes

No que diz respeito ao manuseamento de conhecimento relativo a entidades adjudicantes, serão estabelecidos invariantes que dizem respeito à inserção de conhecimento, tanto positivo como negativo, assim como invariantes que dizem respeito à remoção de entidades adjudicantes.

No que diz respeito à inserção de entidades adjudicante, o **primeiro** invariante apenas verificará que cada um dos campos do termo a inserir correspondem ao tipo de dados esperado. Como se pode ver pelo exemplo de código, o ID deverá ser um inteiro, o nome um átomo, o nif um inteiro, o tipo de entidade um átomo e a morada outro átomo.

Figura 2.8: Invariante sobre os tipos de dados do adjudicante(conhecimento positivo)

```
+adjudicante(IdAd,Nome,Nif,TipoEntidado,Morada) :: (
    integer(IdAd),
    atom(Nome),
    integer(Nif),
    atom(TipoEntidado),
    atom(Morada)
).
```

O mesmo invariante é também aplicado a conhecimento negativo.

Figura 2.9: Invariante sobre os tipos de dados do adjudicante(conhecimento negativo)

```
+(-adjudicante(IdAd,Nome,Nif,TipoEntidado,Morada)) :: (
    integer(IdAd),
    atom(Nome),
    integer(Nif),
    atom(TipoEntidado),
    atom(Morada)
).
```

O **segundo** invariante impede que seja inserido um adjudicante repetido. Visto que existem certos campos de um adjudicante que podem não ser únicos (o tipo de entidade e a morada), os três campos que são usados para verificar se já uma entidade adjudicante já existe serão o id, o nome e o nif. Estes campos serão *obrigatoriamente* únicos em cada entidade adjudicante.

Para permitir a inserção de conhecimento impreciso, visto que o termo adjudicante nunca é verdadeiramente inserido (apenas exceção(adjudicante())), mas deve-se na mesma testar na mesma os invariantes de inserção para garantir coerência, cria-se uma cláusula alternativa em que o adjudicante não precisa de existir caso se queira inserir o conhecimento impreciso.

Figura 2.10: Invariante sobre entidades adjudicantes repetidas(conhecimento positivo)

```
+adjudicante(IdAd,Nome,Nif,_,_) :: (
    solucoes( (IdAd),(adjudicante( IdAd,_,_,_,_)),S1 ),
    solucoes( (Nome),(adjudicante( _,Nome,_,_,_)),S2 ),
    solucoes( (Nif),(adjudicante( _,_,Nif,_,_)),S3 ),
    comprimento( S1,N1 ),
    comprimento( S2,N2 ),
    comprimento( S3,N3 ),
    ((N1 = 1,
    N2 = 1,
    N3 = 1);
    impreciso(adjudicante(IdAd)))
).
```

A mesma lógica deve também ser aplicada ao conhecimento negativo, porém, visto que se trata da negação de factos, o campo do nome e nif não precisam de ser únicos, visto que se pode negar que um ajdudicante tenha um id x e negar também que esse adjudicante tem um id y.

Devido ao **princípio do mundo fechado**, visto que o nosso predicado de negação forte depende em parte da negação por falha, o sistema indicará, sempre que se tenta inserir uma negação, que essa negação já existe devido ao facto de não conseguir encontrar o facto na base de informação e, como tal, ocorrer a prova por falha.

Como tal, é preciso tratar esta negação que se pretende inserir como uma segunda negação, a própria *negação forte*. Isto explica o facto de o comprimento da lista a verificar ser 2 em vez do habitual 1.

Figura 2.11: Invariante sobre entidades adjudicantes repetidas(conhecimento positivo)

```
+(-adjudicante(IdAd,_,_,_,_)) :: (
    solucoes( (IdAd),(-adjudicante( IdAd,_,_,_)),S1 ),
    comprimento(S1,R1),
    R1==2 % pois há sempre uma por negação por falha
).
```

O **terceiro** invariante refere-se à validação do NIF. A especificação deste ficou estipulado no *Decreto-lei n.º 14/2013, de 28 de Janeiro*. O nono dígito do nif, serve como dígito de validação. Este dígito é obtido através de uma versão alterada do algoritmo *módulo 11*.

Resumidamente, este algoritmo consiste em multiplicar cada um dos 8 algarismos do nif(pois o 9º será o dígito de validação) por um peso descendente a partir de 9. O valor destes algarismos é depois somado e dívidido por 11. Susbtrai-se a 11 o valor do resto da operação. Se o resto for 1 ou 0, o dígito de validação será 0, caso contrário será o próprio valor resultante. Segue um exemplo da obtenção do dígito de validação para o nif cujos primeiros oito dígitos são 50690117 :

Dígitos: 5 0 6 9 0 1 1 7
Peso: 9 8 7 6 5 4 3 2
Resultados: 45 + 0 + 42 + 54 + 0 + 4 + 3 + 14 = 162
Soma dos valores: 152 / 11 = 14
Resto: 8
Dígito de validação: 11 - 8 = 3
NIF válido: 506901173

Assim, o invariante verificará se o NIF inserido é um válido. Devido à ligeira complexidade matemática da verificação, são usados predicados auxiliares definidos no ficheiro `predicados_auxiliares` e apresentado no apêndice A.1.

Figura 2.12: Invariante sobre o NIF(conhecimento positivo)

```
+adjudicante(_,_,Nif,_,_) :: (
    Nif>=100000000, Nif=<999999999,
    ultimo_digito_valido(Nif)
).
```

O mesmo predicado foi também definido para a inserção de conhecimento negativo.

Figura 2.13: Invariante sobre o NIF(conhecimento negativo)

```
+(-adjudicante(_,_,Nif,_,_)) :: (
    Nif>=100000000, Nif=<999999999,
    ultimo_digito_valido(Nif)
).
```

O **quarto** invariante refere-se ao tipo da entidade adjudicante e ao seu NIF correspondente.

As entidades que podem ser adjudicantes em contratos públicos estão definidas no *artigo 2.º nº1 e n.º 2, alíneas a), b) e d)* e *artigo 7.º n.º 1.º* do código dos contratos públicos. Estas entidades são representadas no trabalho como **Organismo de administração pública**, seguindo a definição no *Decreto-Lei n.º 4/2015, Artigo 20.º*.

No entanto, o *artigo 275.º* explicíta que, no caso de contratos públicos subsidiados em pelo menos 50%, outras entidades para além das supramencionadas podem também ser consideradas entidades adjudicantes. Como tal, foram consideradas também como possíveis entidades adjudicantes as **pessoas coletivas** e **socieades civis**.

Está por lei definido que cada uma das categorias de entidades acima mencionadas têm para si reservada uma gama de valores do NIF. Esta costuma ser definida pelo primeiro dígito do nif, por exemplo, o NIF das pessoas coletivas começará sempre pelo algarismo 5. Por vezes, é usado também o segundo algarismo em conjunto com o primeiro, como no caso das socieades civis, em que o primeiro algarismo deve ser 9 e o segundo 9 também.

A especificação completa foi explicitada no predicado *nifCorrespondeTipoEntidadeAdjudicante*.

Figura 2.14: Predicado de validação de tipo de entidade adjudicante

```
nifCorrespondeTipoEntidadeAdjudicante([5|_],'Pessoa coletiva').

nifCorrespondeTipoEntidadeAdjudicante([6|_],'Organismo de administração pública').

nifCorrespondeTipoEntidadeAdjudicante([9,9|_],'Sociedade civil').
```

Este predicado é depois usado no invariante que restringe o tipo de entidades que se podem inserir.

Figura 2.15: Invariante sobre o tipo de entidade adjudicante(conhecimento positivo)

```
+adjudicante(_,_,Nif,TipoEntidade,_) :: (
    nif_para_lista(Nif,NifLista),
    nifCorrespondeTipoEntidadeAdjudicante(NifLista,TipoEntidade)
).
```

O mesmo predicado é aplicado à inserção de conhecimento negativo, pois não faria sentido estar a inserir negações que por si próprias já seriam impossíveis de existir.

Figura 2.16: Invariante sobre o tipo de entidade adjudicante(conhecimento negativo)

```
+(-adjudicante(_,_,Nif,TipoEntidade,_)) :: (
    nif_para_lista(Nif,NifLista),
    nifCorrespondeTipoEntidadeAdjudicante(NifLista,TipoEntidade)
).
```

O **quinto** invariante diz respeito à manutenção da coerência entre entidades adjudicantes e adjudicatárias.

Pela natureza dos contratos públicos, a mesma entidade pode estar registada como entidade adjudicante e como entidade adjudicatária (desde que não participe num contrato como ambas as entidades).

Uma aplicação real desta possibilidade será, por exemplo, o instituto nacional de estatística lançar concursos públicos enquanto que também se candidata a um outro conjunto de contratos emitidos por outras entidades, tendo de estar assim registado tanto como entidade adjudicante e entidade adjudicatária.

Para garantir essa coerência, este invariante verificará que, quando se tenta inserir uma entidade adjudicante, se o NIF dessa entidade já estiver registada como entidade adjudicatária, então o nome

dessa entidade adjudicatária terá de coincidir com o nome da entidade adjudicante que se está a tentar inserir, garantindo, assim, que é a mesma entidade.

Figura 2.17: Invariante sobre a coerência sobre uma entidade adjudicante e adjudicatária(conhecimento positivo)

```
+adjudicante(_,Nome,Nif,_,_) :: (
    solucoes((Nomes),adjudicataria(_,Nomes,Nif,_,_),S1),
    comprimento(S1,N1),
    (N1 == 0; % Se o comprimento for 0 indica que o nif não está registado para nenhuma entidade adjudicataria
     (nth0(0, S1,NomeAdjudicataria),  % %Caso contrário, acede-se ao elemento na lista
     NomeAdjudicataria == Nome)   % %verifica-se que o nome é o mesmo
    )
).
```

Este predicado não está definido para conhecimento negativo pois esta restrição não faria sentido na negação da existência de entidades.

O **sexto** predicado diz respeito à manutenção do conhecimento imperfeito interdito. Se se tentar inserir conhecimento perfeito em relação a um adjudicante que tem o seu nome marcado como interdito, este predicado impedirá que isso aconteça.

Figura 2.18: Invariante sobre a coerência sobre uma entidade adjudicante e adjudicatária(conhecimento positivo)

```
+adjudicante(IdAd,_,Nif,TipoEntidade,Morada) :: (
    solucoes((IdAd,Nome_interdito,Nif,TipoEntidade,Morada),
    (adjudicante(IdAd,Nome_interdito,Nif,TipoEntidade,Morada), nulo(Nome_interdito)), R),
    comprimento(R,0)
).
```

No que diz respeito à remoção de entidades adjudicantes. Criou-se um invariante que impede que seja removida uma entidade adjudicante que esteja envolvida em algum contrato público. Desta forma, torna-se impossível apagar informação pública, garantido a transparência necessária de um sistema democrático.

Figura 2.19: Invariante sobre remoção de entidades adjudicantes

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

2.2.3 Invariantes sobre entidades adjudicatárias

Os invariantes que regem a inserção e remoção de entidades adjudicatárias serão, na sua maioria, os mesmos que foram aplicados às entidades adjudicantes, devido à natureza legal semelhante entre ambes as entidades, com uma exceção notável: o invariante que rege o tipo de entidades que podem ser entidades adjudicatárias.

No que concerne as entidades adjudicantes, os artigos do código de contratos públicos mencionados anteriormente definem um conjunto limitado dentidades podem ser entidades ajdudicantes. No

entanto, um conjunto mais diversificado de entidades podem ser entidades adjudicatárias num contrato. Para além das categorias já existentes como entidade adjudicante adidionam-se agora: **Pessoa singular**, **Fundo de investimento**, **Sujeito passivo** e **Condomínio** .

Devido à existência de novas categorias de entidade, o predicado que associa o nif ao tipo de categoria teve também de ser expandido, através do predicado *nifCorrespondeTipoEntidadeAdjudicataria*.

Figura 2.20: Predicado de validação de tipo de entidade adjudicante

```
nifCorrespondeTipoEntidadeAdjudicataria([Primeiro_digito|_],'Pessoa singular'):-
    Primeiro_digito>= 1,
    Primeiro_digito =< 3.

nifCorrespondeTipoEntidadeAdjudicataria([5|_],'Pessoa coletiva').

nifCorrespondeTipoEntidadeAdjudicataria([6|_],'Organismo de administração pública').

nifCorrespondeTipoEntidadeAdjudicataria([7,2|_],'Fundo de investimento').

nifCorrespondeTipoEntidadeAdjudicataria([7,7|_],'Sujeito passivo').

nifCorrespondeTipoEntidadeAdjudicataria([Primeiro_digito,Segundo_digito|_],'Condominio'):-
    Primeiro_digito == 9 ,
    (Segundo_digito ==0 ; Segundo_digito == 1).

nifCorrespondeTipoEntidadeAdjudicataria([9,9|_],'Sociedade civil').
```

Este predicado é depois aplicado no invariante que rege a evolução do conhecimento e que relaciona o tipo de entidade com o nif

Figura 2.21: Invariante sobre o tipo de entidade adjudicante(conhecimento positivo)

```
+adjudicataria(_,_,Nif,TipoEntidade,_) :: (
    nif_para_lista(Nif,NifLista),
    nifCorrespondeTipoEntidadeAdjudicataria(NifLista,TipoEntidade)
).
```

O mesmo invariante é aplicado à evolução de conhecimento negativo.

Figura 2.22: Invariante sobre o tipo de entidade adjudicante(conhecimento negativo)

```
+adjudicataria(_,_,Nif,TipoEntidade,_) :: (
    nif_para_lista(Nif,NifLista),
    nifCorrespondeTipoEntidadeAdjudicataria(NifLista,TipoEntidade)
).
```

2.2.4 Invariantes sobre contratos

No que diz respeito ao manuseamento de conhecimento relativo a contratos públicos, serão estabelecidos invariantes que dizem respeito à inserção de conhecimento, tanto positivo como negativo. Porém, não serão estabelecidos invariantes sobre a remoção de contratos públicos, isto deve-se ao facto de não permitirmos que sejam removidos contratos públicos, visto ser informação pública que deve ser mantida num esforço de garantir um sistema justo e transparente de contratação pública.

Quanto à inserção de contratos , o **primeiro** invariante apenas verificará que cada um dos campos do termo a inserir correspondem ao tipo de dados esperado. É ainda feita uma verificação à data, através de um predicado auxiliar, de forma a garantir que esta é válida.

Os tipos de dados podem ser inferidos pelo exemplo de código seguinte.

Figura 2.23: Invariante sobre os tipos de dados do contrato

```
+contrato(IdCont,IdAd, IdAda, TipoDeContrato, TipoDeProcedimento, Descricao, Valor, Prazo, Local, Data, Subsidiado) :: (
    integer(IdCont),
    integer(IdAd),
    integer(IdAda),
    atom(TipoDeContrato),
    atom(TipoDeProcedimento),
    atom(Descricao),
    integer(Valor),
    integer(Prazo),
    atom(Local),
    data_valida(Data),
    member(Subsidiado,[true,false])
).
```

O mesmo invariante é aplicado à evolução de conhecimento negativo.

O **terceiro** invariante garante que tanto o custo como o prazo de um contrato são positivos. O mesmo invariante é aplicado à evolução de conhecimento negativo.

Figura 2.24: Invariante sobre o custo e o prazo

```
+contrato(_,_,_,_,_,_,Custo,Prazo,_,_,_) :: (
    Custo >= 0,
    Prazo > 0
).
```

O **quarto** invariante obrigará a que tanto a entidade adjudicante como a entidade adjudicatária indicadas no contrato existam na base de informação. O mesmo invariante é aplicado à evolução de conhecimento negativo.

Figura 2.25: Invariante sobre as entidades contratuais

```
+contrato(_,IdAd,IdAda,_,_,_,_,_,_,_,_) :: (
    solucoes( (IdAd),(adjudicante( IdAd,_,_,_,_)),S1 ),
    solucoes( (IdAda),(adjudicataria( IdAda,_,_,_,_)),S2 ),
    comprimento( S1,N1 ),
    comprimento( S2,N2 ),
    N1 == 1,
    N2 == 1
).
```

O **quinto** invariante certificará que a mesma entidade não está a registar-se no contrato tanto como entidade adjudicante e adjudicatária. Como a mesma entidade pode tem ids diferentes no seu regsisto como entidade adjudicante e entidade adjudicatária, verificar se os ids são iguais não iria resolver o problema.

Porém, visto que o NIF de uma entidade tem de ser igual quer ela seja adjudicante quer seja adjudicatária, pode-se realizar a verificação comparando o NIf da entidade adjudicante no contrato

com a entidade adjudicatária. Este invariante não é aplicado à inserção de conhecimento negativo pois pode surgir este caso como negação forte.

Figura 2.26: Segundo invariante sobre as entidades contratuais

```
+contrato(_,IdAd, IdAda,_,_,_,_,_,_,_) :: (
    solucoes((NifAd), (adjudicante( IdAd,NifAd,_,_,_)),S1 ),
    solucoes((NifAda), (adjudicataria( IdAda,NifAda,_,_,_)),S2),
    S1 \= S2
).
```

O **sexto** invariante limita o tipo de procedimento contratual a um conjunto definido no *artigo 16.º n.º 1* co CCP. O tipo de procedimento estará limitado, por consequente, ao seguinte conjunto: **Ajuste direto**, **Consulta prévia**, **Concurso público**,**Concurso limitado por prévia qualificação**,**Procedimento de negociação**, **Diálogo concorrencial** e **Parceria para a inovação**.

Figura 2.27: Invariante sobre os tipos de procedimentos contratuais

```
+contrato(_,_,_,_,Procedimento,_,_,_,_,_,_) :: (
    member(Procedimento,['Ajuste direto','Consulta prévia','Concurso público',
        'Concurso limitado por prévia qualificação','Procedimento de negociação',
        'Diálogo concorrencial','Parceria para a inovação'])
).
```

A partir daqui, os invariantes passam a ser **seletivos**, ou seja, só serão aplicados em casos específicos de inserção ou remoção.

O **sétimo** invariante é aplicado quando se tenta inserir um contrato **não subsidiado**. Como já referido em 2.2.2, quando o contrato não é subsidiado em pelo menos 50%, apenas organismos de administração pública ou pessoas coletivas (quando estas representam organismos de administração pública) podem ser entidades adjudicantes.

Figura 2.28: Invariante sobre contratos não subsidiados

```
+contrato(_,IdAd,_,_,_,_,_,_,_,_,false) :: (
    solucoes((Tipo),adjudicante(IdAd,_,_,Tipo,_),ListaTipo), %ista terá comprimento 1
    nth0(0, ListaTipo, TipoAdjudicante), %acedemos então ao tipo do adjudicante
    member(TipoAdjudicante,['Pessoa coletiva','Organismo de administração pública']) %CCP
).
```

O **oitavo** invariante concerne apenas aos contratos por *ajuste direto*. Conforme indicado no enunciado, quando se trata de um contrato por ajuste direto, o tipo de contrato apenas pode pecenter ao seguinte conjunto: **Aquisição de bens móveis**, **Locação de bens móveis** e **Aquisição de serviços**.

Figura 2.29: Invariante sobre contratos por ajuste direto

```
% Aplicado a conhecimento perfeito negativo
+(-contrato(_,_,_,TipoDeContrato,'Ajuste direto',_,Custo,Prazo,_,_,_)) :: (
    Custo =< 5000,
    Prazo =< 365,
    member(TipoDeContrato,['Aquisição de bens móveis','Locação de bens móveis','Aquisição de serviços'])
).
```

O **nono** invariante refe-se aos contratos de aquisição de serviços. Quando uma entidade adjudicante convida uma entidade adjudicatária para este tipo de contrato, devem-se verificar que a soma do valor dos contratos realizados no próprio ano fiscal e nos dois anteriores,por aquisição de serviços, não pode ultrapassar o valor de 75,000.00€, excluindo o próprio contrato em questão.

Figura 2.30: Invariante sobre aquisição de serviços

```
+contrato(_,IdAd,IdAda,'Aquisição de serviços',_,_,Valor,_,_,data(_,_,Ano),_) :: (
    solucoes((Data,Custo),
      contrato(_,IdAd,IdAda,'Aquisição de serviços',_,_,Custo,_,_,Data,_),
      ParesDataCusto
    ), %lista de pares data de assinatura x custo do contrato
    %valores dos contratos no ano de assinatura do contrato + nos dois anos anteriores
    somaCustosContratos(ParesDataCusto,Ano,SomaCustos),
    (SomaCustos-Valor) < 75000
).
```

2.2.5 Invariantes sobre a inserção de conhecimento imperfeito

Para além dos invariantes descritos acima para cada tipo de entidade, de forma a garantir coerência na inserção de conhecimento imperfeito, foi criado um conjunto de invariantes que também devem ser satisfeitos quando se insere conhecimento imperfeito.

No que concerne a inserção de conhecimento incerto ou interdito, os invariantes serão marcados com **:-:**. Estes impedirão que se insira conhecimento incerto quando já existe conhecimento perfeito, quando já existe conhecimento impreciso, visto que este revela informação sobre o campo, e impede também quando já existe conhecimento incerto, pois estaria-se a inserir conhecimento repetido.

Figura 2.31: Invariante sobre evolução de conhecimento incerto

```
+adjudicante(IdAd,_,_,_,_) :-: (
        nao(perfeito(adjudicante(IdAd))),
        nao(impreciso(adjudicante(IdAd))),
        nao(incertoMorada(adjudicante(IdAd)))
).

% Adjudicataria incerto/interdito
+adjudicataria(IdAda,_,_,_,_) :-: (
        nao(perfeito(adjudicataria(IdAda))),
        nao(impreciso(adjudicataria(IdAda))),
        nao(incertoMorada(adjudicataria(IdAda)))
).

% Contrato incerto/interdito
+contrato(IdContrato,_,_,_,_,_,_,_,_,_,_) :-: (
        nao(perfeito(contrato(IdContrato))),
        nao(impreciso(contrato(IdContrato))),
        nao(incertoDescricao(contrato(IdContrato)))
).
```

No que é relevante à inserção de conhecimento impreciso, também existem invariante que a regulam, desta vez marcados com **: :**. Estes invariantes irão impedir que se insira conhecimento impreciso quando já existe conhecimento perfeito ou quando já existe conhecimento incerto, visto que, caso contrário,

ao inserir impreciso quando há incerto, se estaria a expandir o conhecimento, o que não pode ser permitido.

Figura 2.32: Invariante sobre evolução de conhecimento incerto

```
+adjudicante(IdAd,_,_,_,_) :~: (
        nao(perfeito(adjudicante(IdAd))),
  nao(incertoMorada(adjudicante(IdAd),_))
).

% Adjudicataria impreciso
+adjudicataria(IdAda,_,_,_,_) :~: (
        nao(perfeito(adjudicataria(IdAda))),
  nao(incertoMorada(adjudicataria(IdAda),_))
).

% Contrato impreciso
+contrato(IdContrato,_,_,_,_,_,_,_,_,_,_) :~: (
        nao(perfeito(contrato(IdContrato))),
  nao(incertoDescricao(contrato(IdContrato),_))
).
```

2.3 Evolução e involução de conhecimento

2.3.1 Introdução

A introdução/remoção de conhecimento na base de dados é regida por diversos predicados de evolução e involução que regulam o seu modo de inserção e efeito na base de conhecimento do trabalho, tendo em especial atenção os testes de invariantes definidos e as relação do conhecimento a ser inserido com o que já se encontra presente.

2.3.2 Predicados relevantes de evolução

A evolução de conhecimento perfeito é tratada pelo predicado **evolucao** e **evolucao_perfeito**

Figura 2.33: Predicado de evolução

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

Figura 2.34: Predicado de evolução perfeito

```
evolucao_perfeito(adjudicante(Id, Nome, NIF, TipoEntidade, Morada)) :-
    remove_imperfeito(adjudicante(Id, Nome, NIF, TipoEntidade, Morada)),
    evolucao(adjudicante(Id, Nome, NIF, TipoEntidade, Morada)),
    insercao(perfeito(adjudicante(Id))).
```

Existem predicados semelhantes para os 2 restantes entidades presentes no nosso modelo que seguem os mesmos passos.

É removido primeiramente todo o conhecimento imperfeito (exceto em caso de existir conhecimento interdito) relativo ao termo a inserir

Figura 2.35: Predicado de remoção de conhecimento imperfeito

```
remove_imperfeito(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)) :-
    retract(excecao(adjudicataria(Id,_,_,_,_))),
    remove_imperfeito(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)).

remove_imperfeito(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)) :-
    retract(impreciso(adjudicataria(Id))),
    remove_imperfeito(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)).

remove_imperfeito(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)) :-
    remove_incerto(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)).
```

```
remove_imperfeito(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)) :-
    remove_incerto(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)).

remove_incerto(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)) :-
    incertoMorada(adjudicataria(Id), Morada_desconhecido),
    remocao((excecao(adjudicataria(IdE, NomeE,NifE,TipoE,MoradaE)) :-
      adjudicataria(IdE, NomeE, NifE, TipoE, Morada_desconhecido))),
    remocao(adjudicataria(Id,_,_,_,_)),
    retract(incertoMorada(adjudicataria(Id), _)).

remove_incerto(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)).
```

Seguidamente todos os inveriantes referentes ao termo a inserir são testados no predicado **evolucao** e caso se verifique que o conhecimento a inserir é consistente com todos este é inserido na base de conhecimento e marcado como **perfeito**

A inserção de conhecimento imperfeito está dividida em 3 partes, 1 para cada tipo de conhecimento incompleto.
Primeiramente a inserção de conhecimento incerto é regida pelos predicados de **evolucao_incerto** e **evolucao**.

Figura 2.36: Predicado de evolução

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

Figura 2.37: Predicado de evolução incerto

```
evolucao_incerto(adjudicataria(IdAda, Nome, NIF, TipoEntidade, Morada_desconhecido), morada) :-
    evolucao_incerto(adjudicataria(IdAda, Nome, NIF, TipoEntidade, Morada_desconhecido)),
    insercao(incertoMorada(adjudicataria(IdAda), Morada_desconhecido)),
    insercao((excecao(adjudicataria(IdAdj, N, Ni, T, M)) :-
            adjudicataria(IdAdj, N, Ni, T, Morada_desconhecido))).
```

Estes predicados tem em grande parte a mesma fucionalidade que os de introdução de conhecimento perfeito com 2 importantes diferenças:

- Devem ser testados também os invariantes respeitantes ao conhecimento incerto (**:-:**).
- Deve ser inserida a exceção que denota que o conhecimento inserido é incerto.

É ainda de notar que o conhecimento inserido deve ser marcado como incerto (neste caso **incertoMorada**).

O conhecimento impreciso difere do perfeito e incerto na medida que se podem inserir diversas entidades com os mesmos parametros diferindo apenas no parametro que se pretende incerto. A inserção deste conhecimento é governada pelo predicado **evolucao_impreciso** e **evolucao_impreciso_morada**.

Figura 2.38: Predicado de evolução impreciso

```
evolucao_impreciso(Termo) :-
  solucoes(Invariante, +Termo::Invariante, Lista),
  solucoes(Invariante, +Termo:~:Invariante, ListaImpreciso),
  insercao(excecao(Termo)),
  teste(Lista),
  teste(ListaImpreciso).
```

Figura 2.39: Predicado de evolução impreciso morada

```
evolucao_impreciso_morada(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)) :-
  remove_incerto(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)),
  insercao(impreciso(adjudicataria(Id))),
  evolucao_impreciso(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)).
```

É de notar que neste conhecimento os nunca é inserida a entidade em si, apenas as exceções a ela realtivas, e que os invariantes que testam a inserção de conhecimento repetido estão preparados para permitir a passagem de conhecimento impreciso que repita ids, nifs e nomes.
Ao inserir o conhecimento impreciso na base de conhecimento é ainda necessário **retirar todo o conhecimento incerto** que lá se encontra.

Figura 2.40: Predicado de evolução impreciso morada

```
remove_incerto(adjudicante(Id, Nome, NIF, TipoEntidade, Morada)) :-
    incertoMorada(adjudicante(Id), Morada_desconhecido),
    remocao((excecao(adjudicante(IdE, NomeE,NifE,TipoE,MoradaE)) :-
      adjudicante(IdE, NomeE, NifE, TipoE, Morada_desconhecido))),
    remocao(adjudicante(Id,_,_,_,_)),
    retract(incertoMorada(adjudicante(Id), _)).


remove_incerto(adjudicante(Id, Nome, NIF, TipoEntidade, Morada)).
```

Quanto ao conhecimento imperfeito do tipo interdito, a sua inserção é também regida inicialmente pelo predicado **evolucao** e **evolucao_interdito**.

Figura 2.41: Predicado de evolução

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

Figura 2.42: Predicado de evolução interdito

```
evolucao_interdito(adjudicataria(Id, Nome_interdito, NIF, TipoEntidade, Morada), nome) :-
    evolucao_incerto(adjudicataria(Id, Nome_interdito, NIF, TipoEntidade, Morada)),
    insercao((excecao(adjudicataria(IdAd, N, Ni, T, M)) :-
        adjudicante(IdAd, Nome_interdito, Ni, T, M))),
    insercao((nulo(Nome_interdito))),
    insercao(interdito(adjudicataria(Id))).
```

Para além dos testes dos invariantes tal como no conhecimento incerto, é também:

- Inserida a exceção que denota que o conhecimento inserido é incerto.
- Marcado o valor interdito como nulo.
- Marcado o conhecimento como interdito.

2.3.3 Predicados relevantes de involução

A involução do conhecimento perfeito é tratada pelos predicados**involucao** e **involucao_perfeito**:

Figura 2.43: Predicado de involução

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

Figura 2.44: Predicado de involução perfeita

```
involucao_perfeito(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)) :-
    involucao(adjudicataria(Id, Nome, NIF, TipoEntidade, Morada)),
    remocao(perfeito(adjudicataria(Id))).
```

Como pode ser observado, é apenas executada a involução do conhecimento, através do predicado involução que testa os respetivos invariantes e posteriormente, o conhecimento é desmarcado de perfeito.

Quanto à involução do conhecimento imperfeito incerto, para além do mesmo predicado **involucao** utilizado anteriormente, é também usado o seguinte predicado **involucao_incerto**:

Figura 2.45: Predicado de involução imperfeita incerta

```
involucao_incerto(adjudicataria(IdAda, Nome, NIF, TipoEntidade, Morada_desconhecido), morada) :-
    involucao(adjudicataria(IdAda, Nome, NIF, TipoEntidade, Morada_desconhecido)),
    remocao((excecao(adjudicataria(IdAd, N, Ni, T, M)) :-
            adjudicataria(IdAdj, Nome, Ni, T, Morada_desconhecido))),
    remocao(incerto(adjudicataria(IdAda))).
```

Como pode ser observado, para além do processo de involução do conhecimento, é removida a exceção para o conhecimento incerto e desmarcado o mesmo conhecimento da categoria incerto.

Já para a involução do conhecimento imperfeito impreciso são utilizados os predicados **involucao_impreciso** e **involucao_im** de modo a retirar de forma eficaz todas as exceções relativas aos predicados inseridos anteriormente

Figura 2.46: Predicado de involução imperfeita im

```
involucao_im(adjudicataria(Id, Nome_interdito, NIF, TipoEntidade, Morada)) :-
    solucoes( Invariante,-(adjudicataria(Id, Nome_interdito, NIF, TipoEntidade, Morada))::Invariante,Lista ),
    remocao(excecao(adjudicataria(Id, Nome_interdito, NIF, TipoEntidade, Morada))),
    teste( Lista ).
```

Figura 2.47: Predicado de involução imperfeita imprecisa

```
involucao_impreciso(adjudicataria(Id, Nome_interdito, NIF, TipoEntidade, Morada)) :-
    involucao_im(adjudicataria(Id, Nome_interdito, NIF, TipoEntidade, Morada)),
    remocao(impreciso(adjudicataria(Id))).
```

Como podemos observar, no predicado **involucao_im** são testados os invariantes respetivos e removidas as exceções do conhecimento em causa e no predicado **involucao_impreciso** o conhecimento é desmarcado de impreciso.

Em relação ao conhecimento interdito, para além do predicado involução é utilizado o predicado **involucao_interdito**:

Figura 2.48: Predicado de involução imperfeita interdita

```
involucao_interdito(adjudicataria(Id, Nome_interdito, NIF, TipoEntidade, Morada), nome) :-
    involucao(adjudicataria(Id, Nome_interdito, NIF, TipoEntidade, Morada)),
    remocao((excecao(adjudicataria(IdAd, N, Ni, T, M)) :-
        adjudicante(IdAd, Nome_interdito, Ni, T, M))),
    remocao((nulo(Nome_interdito))),
    remocao(interdito(adjudicataria(Id))).
```

Para além da involução do conhecimento, é removida a exceção, desmarcado o conhecimento da categoria de nulo e ainda desmarcado o conhecimento da categoria de interdito.

2.4 Sistema de inferência

Devido à programação em lógica extendida, a necessidade de criar um sistema de inferência capaz de lidar com os três valores lógicos, torna-se crucial.

Para tal, foi desenvolvido o predicado *demo*. Este predicado irá responder a uma questão fornecida com um dos seguintes valores:

- **Verdadeiro**, caso o conhecimento exista explicitamente na base de informação;
- **Negativo**, caso a negação da questão exista através do predicado de negação forte;
- **Desconhecido**, caso exista conhecimento imperfeito acerca da questão que está a ser testada;

Figura 2.49: Predicado demo

```
%------------------------------- - - - - - - - - - -  -  -  -  -  -
% 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 ).
```

Para além deste predicado, foi desenvolvido um conjunto de predicados que indicarão informação sobre as entidades contratuais que serão também relevantes para manter a transparência do regime de contratação pública.

O **primeiro** destes predicados indicará, para uma dada entidade, em quantos contratos ela está envolvida.

Figura 2.50: Predicado de inferência do número de contratos

```
% Extensao do predicado numeroContratosAdjudicante: IdAdjudicante,R -> {V,F}
% Devolve em quantos uma entidade adjudicante está envolvida
numeroContratosAdjudicante(IdAd,R) :-
    solucoes((IdAd),contrato(_,IdAd,_,_,_,_,_,_,_,_),Lista),
    comprimento(Lista,R).

%------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensao do predicado numeroContratosAdjudicante: IdAdjudicante,R -> {V,F}
% Devolve em quantos uma entidade adjudicatária está envolvida
numeroContratosAdjudicataria(IdAda,R) :-
    solucoes((IdAda),contrato(_,_,IdAda,_,_,_,_,_,_,_),Lista),
    comprimento(Lista,R).
```

O **terceiro** predicado indicará a soma do valor dos contratos em que uma entidade contratual está envolvida. Este predicado foi desenvolvido tanto para as entidades adjudicantes como adjudicatárias.

Figura 2.51: Predicado de inferência do valor de contratos

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado valorContratosAdjudicante: IdAdjudicante,R -> {V,F}
% Devolve a soma do valor dos contratos em que uma entidade adjudicante está envolvida
valorContratosAdjudicante(IdAd,R) :-
    solucoes((Valores),contrato(_,IdAd,_,_,_,_,Valores,_,_,_,_),Lista),
    sum_list(Lista,R).

%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensao do predicado valorContratosAdjudicante: IdAdjudicante,R -> {V,F}
% Devolve a soma do valor dos contratos em que uma entidade adjudicatária está envolvida
valorContratosAdjudidicataria(IdAda,R) :-
    solucoes((Valores),contrato(_,_,IdAda,_,_,_,Valores,_,_,_,_),Lista),
    sum_list(Lista,R).
```

O **quarto** predicado será bastante mais complexo. O predicado indicará o id de todos os contratos em que uma entidade está envolvida, que ainda estejam em execução numa certa data fornecida pelo utilizador. Isto será possível devido a um conjunto de predicados auxiliares que implementarão um mecânismo de cálculo de diferença, em dias, entre duas datas fornecidas. Este mecânismo está explicitado no anexo B.1. O mesmo predicado foi desenvolvido para entidades adjudicatárias.

Figura 2.52: Predicado de inferência de contratos ativos

```
listaContratosAtivosAdjudicante(IdAd,DataReferencia,R):-
    solucoes((IdsContratos,PrazosContratos,DatasContratos),
        contrato(IdsContratos,IdAd,_,_,_,_,_,PrazosContratos,_,DatasContratos,_),
        ListaTriplosIdPrazoData),
    filtraListaTriplosIdPrazoDataAtivo(ListaTriplosIdPrazoData,DataReferencia,R).
```

O **quinto** predicado será bastante semelhante ao anterior, porém este selecionará o conjunto de contratos que já acabaram. numa certa data. O predicado será cuidadoso também para não selecionar contratos que ainda não foram começados. O mesmo predicado foi desenvolvido para entidades adjudicatárias.

Figura 2.53: Predicado de inferência de contratos acabados

```
listaContratosAcabadosAdjudicante(IdAd,DataReferencia,R):-
    solucoes((IdsContratos,PrazosContratos,DatasContratos),
        contrato(IdsContratos,IdAd,_,_,_,_,_,PrazosContratos,_,DatasContratos,_),
        ListaTriplosIdPrazoData),
    filtraListaTriplosIdPrazoDataAcabado(ListaTriplosIdPrazoData,DataReferencia,R).
```

Capítulo 3

Conclusão

Em conclusão, julgamos que nos consguimos manter fiéis ao universo espitulado por lei e, como tal, o trabalho resultou numa correta representação do problema pretendido. Foram criadas as entidades envolvidas no universo de contratação pública de forma completa, contendo cada uma destas um conjunto de informações que permitiram uma representação completa e transparente da entidade. Foi também implementado com sucesso um sistema completo de inferência assim como um processo de evolução e involução de conhecimento que permitiu a manutenção da veracidade e coerência da base de informação.

Durante a execução deste projeto, as nossas maiores dificuldades consistiram na decisão de quais os predicados mais importantes a implementar, assim como o seu significado e os invariantes relevantes. O trabalho implicou uma mudança na nossa forma de raciocinar e organizar como grupo, visto que o enunciado não apresentava muitas restrições quanto à modulação deste, dando a liberdade de estruturar o problema como o grupo bem entendesse.

Em futuras iterações do trabalho, gostaríamos de implementar também o processo de candidatura e decisão das entidades que irão participar num contrato, sendo assim possível representar a totalidade do universo da contratação pública

Capítulo 4

Referências

4.1 Referências Bibliográficas

[Analide, 2010] ANALIDE, NEVES, José,
"Representação de informação completa",
"Documento pedagógico, Departamento de Informática, Universidade do Minho, Portugal, 2010.

[Analide, 2011] ANALIDE, Cesar, Novais, Paulo, Neves, José,
"Sugestões para a Elaboração de Relatórios",
"Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011.

[Brakto, 2000] BRATKO, Ivan,
"Programming for Artificial Intelligence, 3rd Edition Portugal, 2000.

[RUSSEL,2010], RUSSEL S. J., Norvig, P., & Davis, E.
Artificial intelligence: a modern approach. 3rd ed. Upper Saddle River, NJ, 2010

4.2 Referências Eletrónicas

Constituição e interpretação: NIF. Disponíel em:
https://pt.wikipedia.org/wiki/Número_de_identificação_fiscal. Acesso em: 10 abr. 2020.

Código dos contratos públicos. Disponível em:
https://dre.pt/legislacao-consolidada/-/lc/34455475/view. Acesso em: 10 abr. 2020.

FAQ dos contratos públicos. Disponível em:
http://www.base.gov.pt/Base/pt/PerguntasFrequentes. Acesso em 10 abr. 2020.

Apêndice A

Mecânica de cálculo de validação do NIF

```
% Predicados auxiliares
%
%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado ultimo_digito_valido: Nif ->{V,F}
% Validação do último bit através da técnica módulo 11
ultimo_digito_valido(Nif):-
    Primeiros_oito_digitos is div(Nif,10),
    Ultimo_digito is mod(Nif,10),
    digito_de_controlo(Primeiros_oito_digitos,Digito),
    Ultimo_digito =:= Digito.

% Extensão do predicado que obtém o digito de controlo a partir dos primeiros 8 dígitos através da técnica módulo 11
digito_de_controlo(Primeiros_oito_digitos,Digito_de_controlo):-
    multiplica_lista(Primeiros_oito_digitos,Ultimo_digito),
    (0 is mod(Ultimo_digito,11); 1 is mod(Ultimo_digito,11)),
    Digito_de_controlo is 0.

digito_de_controlo(Primeiros_oito_digitos,Digito_de_controlo):-
    multiplica_lista(Primeiros_oito_digitos,Ultimo_digito),
    Mod_Result is mod(Ultimo_digito,11),
    Digito_de_controlo is 11-Mod_Result.


% Extensão do predicado que, dado um nif, o transforma numa lista de dígitos e obtém o dígito de validação
multiplica_lista(Nif,Ultimo_digito):-
    nif_para_lista(Nif,Lista),
    multiplicacao_decrescente(Lista,Ultimo_digito,9).


% Extensão do predicado que converte uma lista de códigos ascii na lista de dígitos que representam.
nif_para_lista(Nif,L):-
    number_codes(Nif,X),
    maplist(ascii_to_digit,X, L).

% Extensão do predicado que converte um dígito código ascii para o número correspondente ascii_to_digit
ascii_to_digit(Ascii_number,R) :- R is Ascii_number-48.
```

Figura A.1: Mecânica de validação do NIF

Apêndice B

Mecânica de cálculo de diferença entre datas

```
diferenca_entre_datas(Data1,Data2,R):-
    numero_dias(Data1,NumeroDiasData1),
    numero_dias(Data2,NumeroDiasData2),
    R is NumeroDiasData2-NumeroDiasData1.


% Coloca em R o número de dias Desde o Ano 0 até à data data
numero_dias(data(Dia,Mes,Ano),R):-
    AnoEDia is (Ano*365 + Dia),
    MesAnterior is (Mes-1), % pois os dias deste mês já estão contados
    soma_meses(MesAnterior,SomaMeses),
    contaAnosBissextos(data(Dia,Mes,Ano),NumeroDiasBissextos),
    R is (AnoEDia+SomaMeses+NumeroDiasBissextos).


%Devolve o nº de dias que já passaram no dado ano, a partir do mês atual
soma_meses(0,0).
soma_meses(Mes,R):-
    Meses = [31,28,31,30,31,30,31,31,30,31,30,31],
    Indice is (Mes - 1),
    (nth0(Indice,Meses,NumeroDeDias)),
    soma_meses(Indice,SomaMesesAnteriores), % aproveita-se que Indice já é MEs -1
    R is (SomaMesesAnteriores + NumeroDeDias).


% Predicado que conta o número de anos bissextos desde o ano 0 até ao ano dado
contaAnosBissextos(data(_,Mes,Ano),R):-
    Mes =< 2,
    NovoAno is Ano-1,
    R is (NovoAno//4 - NovoAno//100 + NovoAno//400).

contaAnosBissextos(data(_,Mes,Ano),R):-
    Mes > 2,
    R is (Ano//4 - Ano//100 + Ano//400).
```

Figura B.1: Mecânica de diferença de datas