Universidade do Minho
Escola de Engenharia

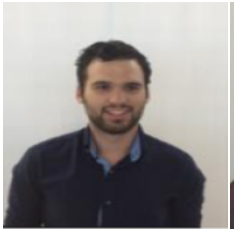
Mestrado Integrado em Engenharia Informática
Sistemas de Representação de Conhecimento e Raciocínio

Programação em lógica estendida e Conhecimento imperfeito

August 19, 2020


Ana Margarida Campos (A85166)


Ana Catarina Gil (A85266)


Roberto Freitas (A67746)


Tânia Rocha (A85176)

1 Resumo

O presente relatório documenta o primeiro exercício prático no âmbito da Unidade Curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Ao longo deste documento serão apresentadas as componentes desenvolvidas para o desenvolvimento deste exercício, mais concretamente a base de conhecimento, com conhecimentos perfeito positivo e negativo, o conhecimento imperfeito, utilizando os três tipos de valores nulos estudados, os invariantes, a problemática da evolução do conhecimento e o sistema de inferência.

Contents

List of Figures

2 Introdução

Este projeto tem como objetivo a utilização da programação em lógica, usando a linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento imperfeito, recorrendo à utilização de valores nulos e da criação de mecanismos de raciocínio adequados, de modo a ser desenvolvido 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 para a prestação de serviço.
Posto isto, neste documento irão ser apresentadas as nossas decisões, tal como excertos de código desenvolvido, casos práticos e seus resultados.

3 Preliminares

Para a realização deste trabalho a equipa colocou em prática tudo aprendido nas aulas praticas e teóricas. No início aprofundou-se o conhecimento sobre o paradigma da promoção lógica e do motor de inferência Prolog. Demos também enfase ao estudo do problema, analisando o Código de contratos públicos, nomeadamente a ala das perguntas frequentes, as quais nos deu um contexto mais real ao problema, e tentando assim colocar o nosso trabalho o mais próximo com a realidade.

Porém, com o estudo mais aprofundado em relação a programação em logica tivemos algumas limitações pois este paradigma baseia-se nos **Prossupostos de Mundo Fechado** e os **Prossupostos de Domínio fechado** em que no primeiro toda a informação não mencionada no base de conhecimento é falsa e o segundo indica que não existem mais objetos no universo para alem daqueles designados por constantes na base de conhecimento.

Dado isto tivemos de adotar pelos **Prossupostos de Mundo Aberto**, em que podemos ter outros factos ou conclusões verdadeiras para além daquelas representadas na base de conhecimento e os **Prossupostos de Domínio aberto** que garantem que podem existir mais objetos do universo de discurso para além dos designados pelas constantes da base de conhecimento. Assim sendo, chegamos a Extensão á programação lógica, a mesma que ainda tem uma forma de representar informação incompleta.

Assim sendo, neste momento temos três tipos de conclusões para cada questão, A **verdadeira**, em que existe prova explicita de um conhecimento verdadeiro, a **falsa**, que existe prova explicita de conhecimento falso e a **desconhecida**, que é quando não existe uma das conclusões anteriores.

Outro dos objetivos de estender a programação lógica é permitir representar dois tipos de negação, a **negação forte** e a **negação por falha**, na primeira é uma forma de representar informação negativa ou falsa, sendo representada pela conectiva '-', e na segunda que indica que não existe qualquer prova na base de conhecimento que responda á questão, sendo esta representa pelo termo *nao*.

Por fim, na representação de conhecimento imperfeito temos três maneira de representar a informação incompleta, a qual é designada por **valores nulos**. Estes podem ser **incerto** quando não sabemos o conhecimento dentro de um conjunto indeterminado de hipóteses, podem ser **imprecisos** quando não sabemos o conhecimento num conjunto finito de hípotes e por fim temos o **interdito** o qual não é possível conhecer.

4 Descrição do Trabalho e Análise de Resultados

De maneira a desenvolver um sistema de representação de conhecimento e raciocínio utilizando os 3 predicados que serão posteriormente apresentados e aplicando conhecimentos obtidos nas aulas, foi criado um ficheiro em *prolog* com todos os dados necessários. Neste relatório vão ser descritos todos os passos que foram dados para a obtenção do resultado final.

O relatório encontra-se dividio em 6 partes. A primeira é a **Base de Conhecimento**, onde são apresentados e descritos todos os predicados que constituem a nossa base de conhecimento, tanto a nível do conhecimento perfeito positivo e negativo como ao nível do conhecimento imperfeito impreciso, incerto e interdito. Numa segunda fase, são apresentados todos os **Invariantes** criados de maneira a controlar a inserção e remoção de conhecimento. De seguida, é mostrado como se abordou a problemática da **Evolução e Involução de conhecimento**, onde são apresentados todos os mecanismos construídos e como se precedeu ao seu desenvolvimento. Posteriormente é apresentado o **Sistema de Inferênica** construído e por último, uma secção de **Extras** que foram desenvolvidos de maneira a tornar o trabalho mais completo.

5 Base de Conhecimentos

Neste trabalho existem 3 tipos de predicados que representam o conhecimento, sendo, eles os **adjudicantes**, **adjudicatarias** e **contratos**. Tanto os adjudicantes, como as adjudicatarias possuem um identificador, um nome, o nif e uma morada. Os contratos possuem um identificador, os Nifs do adjudicante e da adjudicataria, o tipo de contrato e de procedimento, a descrição, o valor, o prazo, o local e a data.

- adjudicante: #IdAd, Nome, NIF, Morada $\rightsquigarrow \{ \mathbb{V}, \mathbb{F}, \mathbb{D} \}$
- adjudicatária: #IdAda, Nome, NIF, Morada $\rightsquigarrow \{ \mathbb{V}, \mathbb{F}, \mathbb{D} \}$
- contrato: #IdAd, #IdAda, TipoDeContrato, TipoDeProcedimento, Descrição, Custo, Prazo, Local, Data $\rightsquigarrow \{ \mathbb{V}, \mathbb{F}, \mathbb{D} \}$

Podemos então observar então que no caso do **adjudicante**, este possuí um id único, o seu nome, NIF e morada.

No caso da **adjudicatária**, tal como no adjudicante, é apresentado um id, nome, NIF, e morada.

Por fim, o **contrato** contém os respetivos ids da adjudicatária e do adjudicante que fazem parte do mesmo, descreve o tipo de contrato, o tipo de procedimento, uma descrição breve, o custo, o prazo, o local e a data.

Como temos como objetivo neste exercício representar informação incompleta, este dominio de soluções terá de ser alterado. Para isto, é definido que para uma dada questão o conhecimento pode ser **verdadeiro**, **falso** ou **desconhecido**. Desta forma, temos as seguintes definições iniciais corresposdentes às fontes de conhecimento anteriores:

```
:- dynamic(adjudicante/4).
:- dynamic(adjudicataria/4).
:- dynamic(contrato/10).
```

Figure 1: Definições iniciais

5.1 Conhecimento Perfeito

Tendo conhecido a nossa base de conhecimento, seguimos então ao povoamento do nosso sistema tanto através de conhecimento positivo como negativo de cada uma das nossas entidades.

5.1.1 Conhecimento Perfieto Positivo

Foram acrescentados os factos perfeitos *positivos* relativos a todos os prefica-dos:

```
% Extenção do predicado adjudicante: Id, Nome, Nif, Morada -> {V,F,D}

%Conhecimento Perfeito Positivo
adjudicante( 1,'Munincipio de Alto de Basto','705330336','Portugal, Braga, Alto de Basto').
adjudicante( 2,'Agrupamento de Escolas do Cerco, Porto','600078965','Portugal, Porto').
adjudicante( 3,'Munincipio de Braga','506901173','Portugal, Braga, Braga').
adjudicante( 4,'Munincipio de Amares','506797627','Portugal, Braga, Amares').
adjudicante( 5,'Cooperativa Agricola de Barcelos , CRL','500967580','Portugal, Braga, Barcelos').
adjudicante( 6,'AgdA - Aguas Publicas do Alentejo, S. A.','509133843','Portugal, Beja, Beja').
adjudicante( 7,'Uniao das Freguesias de Lomar e Arcos','510837581','Portugal').
```

Figure 2: Conhecimento Perfeito positivo do adjudicante

```
% Extensao do predicado adjudicataria: Id, Nome, NIF, Morada -> {V,F,D}

% Conhecimento Perfeito Positivo
adjudicataria( 1,'Agripesca Entreposto Frigorifico Lda','501063013','Portugal').
adjudicataria( 2,'Gonaalo Leite de Campos & Associados - Sociedade de Advogados, SP, RL','513210865','Portugal').
adjudicataria( 3,'Landfound-Levantamentos Cadastrais, Lda.','506206416','Portugal').
adjudicataria( 4,'LIGALOTE, LDA.','509605540','Portugal').
adjudicataria( 5,'Buscardini Communications SRL','738570569','Belgica').
adjudicataria( 6,'PROVILOJAS','505927764','Portugal').
adjudicataria( 7,'OLMAR - Artigos de Papelaria, Lda.','508831989','Portugal').
adjudicataria( 8,'MLL - Construção Civil e Obras Publicas, Lda','502621559','Portugal').
adjudicataria( 9,'XXX - Associados - Sociedade de Advogados, SP, RL','702675112','Portugal').
```

Figure 3: Conhecimento Perfeito positivo do adjudicatário

```
% Extensao do predicado contrato: idAd, idAda, TipoDeContrato, TipoDeProcedimento, Descricao, Custo, Prazo, Local, Data -> {V,F,D}

% Conhecimento Perfeito Positivo
contrato(1,'510837581','509605540','Empreitadas de obras publicas','Ajuste Direto Regime Geral', 'PARQUE INFANTIL - RUA DA MOUTA, LOMA', '67051.12', '90', 'Portugal, Braga, Braga','23-03-2018').
contrato(2,'510837581','502621559','Empreitadas de obras publicas','Ajuste Direto Regime Geral','CONSTRUCAO DE MURO DE SUPORTE - LOTEAMENTO DE SOUTO NOVAL - LOMAR','5675.00','30','Portugal,Braga,Braga','17-03-2016').
contrato(3,'509133843','513210865','Aquisicao de servicos','Ajuste Direto Regime Geral', '122_2017 Consultoria Juridica', '74900.00', '180', 'Portugal, Beja, Beja','17-01-2018' ).
contrato(4,'506901173','738570569','Aquisicao de servicos','Ajuste Direto Regime Geral', 'ADCM/1/2020/DCP', '40000.00', '15', 'Portugal, Braga, Braga','25-03-2020' ).
contrato(5,'506901173','505927764','Aquisicao de bens moveis','Ajuste Direto Regime Geral', 'ADG/2/18/DACPGP', '6553.00', '306', 'Portugal, Braga, Braga','01-03-2018' ).
contrato(6,'600078965','508831989','Aquisicao de bens moveis','Concurso Previa', 'CP.2019.10 Economato', '4910.12', '365', 'Portugal, Porto, Porto','30-12-2019').
contrato(7,'600078965','501063013','Aquisicao de bens moveis','Consulta Previa', 'CP.2019.06. Bens Alimentares Restauracao', '2444.04', '106', 'Portugal, Porto, Porto','23-09-2019').
contrato(8,'510878440','510078214','Empreitadas de obras públicas','Ajuste Direto Regime Geral','Reconstrução e ampliação do cemiterio de Arcozelo','92835.34','180','Portugal, Braga, Vila Verde','15-05-2018').
contrato(9,'705330336','702675112','Aquisicao de serviços','Consulta Previa','Assessoria Juridica','13599','547','Alto de Basto','11-02-2020').
```

Figure 4: Conhecimento Perfeito positivo do contrato

5.1.2 Conhecimento Perfeito Negativo

No caso deste tipo de conhecimento, adicionamos então factos com **negação forte**. Foi também desenvolvido para além da negação forte, nefgação por falha no caso do adjudicante e na adjudicatária.

```
%Conhecimento Perfeito Negativo
-adjudicante( 8,'Municipio de Vila Verde','385049328','Portugal,Braga, Vila Verde').
-adjudicante( 9,'Municipio de Barcelos','283239329', 'Portugal, Braga, Barcelos').
```

Figure 5: Conhecimento Perfeito negativo do adjudicante

```
% Conhecimento Perfeito Negativo
-adjudicataria( 10,'OLMAR - Artigos de Papelaria, Lda.','508831989','Portugal').
-adjudicataria( 11,'Jose Barros Moreira & Gomes Lda.','510078214','Portugal').
```

Figure 6: Conhecimento Perfeito negativo do adjudicatário

```
% Conhecimento Perfeito Negativo
-contrato(10,'600078965','501169580','Aquisicao de bens moveis','Ajuste Direto Regime Geral', '133/2020 - LABESFAL', '64.00', '60', 'Portugal, Lisboa, Lisboa','06-04-2020').
-contrato(11,'600014576','509697887','Aquisicao de servicos','Concurso publico','Aquisição de Servicos de Apoio e a Direcao de Servicos de Administracao Financeira do Ministerio
                                                                                    dos Negocios Estrangeiros (CATGest)','92835.34','1095','Portugal, Lisboa, Braga','06-04-2020').
```

Figure 7: Conhecimento Perfeito negativo do contrato

Como referido anteriormente, para além do conhecimento perfeito negativo, recorremos também à **negação por falha**.

```
% Confirmar de o adjudicante não faz parte da base de Conhecimento
% tenham parâmetros diferentes do que os que estão guardados
-adjudicante(Id,N,I,M) :- nao(adjudicante(Id,N,I,M)),
                          nao(excecao(adjudicante(Id,N,I,M))).
```

Figure 8: Conhecimento negativo por falha do adjudicante

```
% Confirmar de o adjudicataria não faz parte da base de Conhecimento
% tenham parâmetros diferentes do que os que estão guardados
-adjudicataria( Id , Nome , Nif , Morada) :- nao(adjudicataria(Id,Nome,Nif,Morada)),
                                             nao(excecao(adjudicataria(Id,Nome,Nif,Morada))).
```

Figure 9: Conhecimento negativo por falha da adjudicatária

5.2 Conhecimento Imperfeito

Com a extensão à Programação em Lógica obteve-se um novo dominio de respostas, tendo agora a possibilidade de obter respostas **desconhecidas**, além das já conhecidas do panorama do conhecimento perfeito, o verdadeiro e falso. Estas novas respostas são caracterizadas como conhecimento imperfeito, as quais aboradam três tipos de valores nulos: o **Incerto**, o **Impreciso** e o **Interdito**.

5.2.1 Conhecimento Incerto

O conhecimento imperfeito incerto siginifica que se desconhece um valor de um determinado campo de um predicado, dentro de um conjunto ilimitado de hipoteses. No nosso caso de estudo, colocamos o caso em que o contrato com id 34, a abjudicataria alterou o valor do contrato, sendo que ainda ninguem sabe o mesmo. Com isto, representamos a informação de seguinte modo:

Figure 10: Conhecimento incerto sobre contrato

5.2.2 Conhecimento Impreciso

O conhecimento imperfeito impreciso siginifica que se desconhece um valor de um determinado campo de um predicado, dentro de um conjunto finito de hipoteses. No nosso caso de estudo, colocamos o caso em que a adjudicataria com id 26 no inicio não tinha, posteriormente foi adicionado que poderia Portugal ou Belgica. Com isto, representamos a informação de seguinte modo:

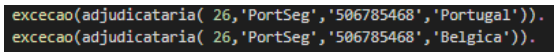


Figure 11: Conhecimento impreciso sobre abjudicataria

5.2.3 Conhecimento Interdito

O conhecimento imperfeito interdito siginifica que se desconhece um valor de um determinado campo de um predicado,e não pode vir a ser conhecido. No nosso caso de estudo, colocamos o caso em que a adjudicataria com id 25 por motivos de segurança nao permite que se saiba a sua morada. Com isto, representamos a informação de seguinte modo:

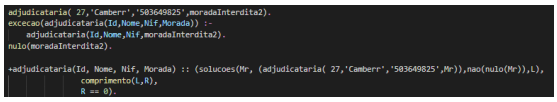


Figure 12: Conhecimento interdito sobre abjudicataria

6 Invariantes

De forma a garantir uma correta adição e remoção de conhecimento na base de conhecimento do nosso sistema construimos invariantes estruturais e referências tanto para os predicados em geral como para cada um do predicados construidos, de forma a respeitar as restrições lógicas, como a não existência de IDs repetidos, tal como, as restrições impostas pelo enunciado deste trabalho.

6.1 Invariantes universais

Como existem invariantes que se enquadram em todos os predicados decidimos construi-los de forma genérica para que fossem aplicáveis a todos.
Entre os quais temos os invariantes que garantem que na nossa base de conhecimento não existe conhecimento perfeito, tanto positivo como negativo, repetido:

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

Figure 13: Invariante conhecimento repetido

De modo a garantir a coerência entre os predicados, foi necessário criar também um invariante que não permite a adição de um conhecimento perfeito positivo que contradiz um conhecimento perfeito negativo e vice-versa.

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

Figure 14: Invariante conhecimento contraditório

Por último foi também criado um invariante que não permite excessões repetidas.

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

Figure 15: Invariante exceções repetidas

6.2 Invariantes de Adjudicante e Adjudicataria

Como ambos os predicados Adjudicante e Adjudicataria dizem respeito a entidades caraterizadas pelos mesmos atributos e que apenas diferem no papel que desenpenham no contrato, os invariantes aplicados serão semelhantes.
Posto isto, de forma a simplificar o relatório decidimos juntá-los numa só secção.

Primeiramente começamos por criar um invariante que garanta que não exista

um adjudicante/adjudicataria com o mesmo ID, tanto para um conhecimento perfeito positivos tanto para um conhecimeno perfeito negativo.Como o NIF tende a ser único para cada entidade achamos por bem criar um invariante que garanta que não há NIFs repetidos

```
+adjudicante(Id,Nome,Nif,Morada) :: (solucoes((Id),adjudicante(Id,_,_,_), L),comprimento(L, 1)).
+(-adjudicante(Id,Nome,Nif,Morada)) :: (solucoes(Id, -adjudicante(Id,_,_,_,_), L),comprimento(L, 1)).
```

Figure 16: Invariante ID não repetido- Adjudicante

```
+adjudicataria(Id,Nome,Nif,Morada) :: (solucoes((Id), adjudicataria(Id,_,_,_), L),comprimento(L, 1)).
+(-adjudicataria(Id,Nome,Nif,Morada)) :: (solucoes(Id, -adjudicataria(Id,_,_,_,_), L),comprimento(L, 1)).
```

Figure 17: Invariante ID não repetido- Adjudicataria

```
+adjudicante(Id,Nome,Nif,Morada) :: (solucoes((Nif), adjudicante(_,_,Nif,_), L),comprimento(L, 1)).
+(-adjudicante(Id,Nome,Nif,Morada)) :: (solucoes((Nif), -adjudicante(_,_,Nif,_), L),comprimento(L, 1)).
```

Figure 18: Invariante NIF não repetido- Adjudicante

```
+adjudicataria(Id,Nome,Nif,Morada) :: (solucoes((Nif), adjudicataria(_,_,Nif,_), L),comprimento(L, 1)).
+(-adjudicataria(Id,Nome,Nif,Morada)) :: (solucoes((Nif), -adjudicataria(_,_,Nif,_), L),comprimento(L, 1)).
```

Figure 19: Invariante NIF não repetido- Adjudicataria

De forma a garantir que para IDs diferentes a restante informação também é diferente foi criado mais um invariante:

```
+adjudicante(Id,Nome,Nif,Morada) :: (solucoes((Nome,Nif,Morada), adjudicante(_,Nome,Nif,Morada), R),comprimento(R, 1)).
+(-adjudicante(Id,Nome,Nif,Morada)) :: (solucoes((Nome,Nif,Morada), -adjudicante(_,Nome,Nif,Morada), R),comprimento(R, 1)).
```

Figure 20: Invariante informação não repetido- Adjudicante

```
+adjudicataria(Id,Nome,Nif,Morada) :: (solucoes((Nome,Nif,Morada), adjudicataria(_,Nome,Nif,Morada), R),comprimento(R, 1)).
+(-adjudicataria(Id,Nome,Nif,Morada)) :: (solucoes((Nome,Nif,Morada), -adjudicataria(_,Nome,Nif,Morada), R),comprimento(R, 1)).
```

Figure 21: Invariante informação não repetido- Adjudicataria

Por último, para estes dois predicados foi também essencial não permitir a remoção de alguma entidade caso esta esteja presente em algum contrato.

```
+adjudicataria(Id,Nome,Nif,Morada) :: (solucoes((Nome,Nif,Morada), adjudicataria(_,Nome,Nif,Morada), R),comprimento(R, 1)).
+(-adjudicataria(Id,Nome,Nif,Morada)) :: (solucoes((Nome,Nif,Morada), -adjudicataria(_,Nome,Nif,Morada), R),comprimento(R, 1)).
```

Figure 22: Invariante que impede a remoção de uma entidade caso ela esteja presente num contrato

6.3 Invariantes de Contrato

Primeiramente, começamos por definir invariantes básicos, como garantir que não existem contratos com o mesmo ID e que os NIFs associados a um contrato correspondem a um adjucante e a uma adjudicataria.

```
% Invariante que nao permite dois contratos com o mesmo id para conhecimento perfeito positivo
+contrato(IdC,IdAd,IdAda,TipoC,TipoP,Des,Custo,Prazo,Loca,Data) :: (solucoes((IdC), contrato(IdC,_,_,_,_,_,_,_,_,_), L),comprimento(L, 1)).

% Invariante que nao permite dois contratos com o mesmo id para conhecimento perfeito negativo
+(-contrato(IdC,IdAd,IdAda,TipoC,TipoP,Des,Custo,Prazo,Loca,Data)) :: (solucoes(IdC, -contrato(IdC,_,_,_,_,_,_,_,_,_), L),comprimento(L, 1)).
```

Figure 23: Invariante ID não repetido

```
%garante que os NIFs associados ao adjudicante e à adjudicataria existem
+contrato(IdC,IdAd,IdAda,TipoC,TipoP,Des,Custo,Prazo,Loca,Data) :: (adjudicante(_,_,IdAd,_),adjudicataria(_,_,IdAda,_)).
```

Figure 24: Invariante de NIFs existentes

Os contratos públicos seguem procedimentos e normas previstas no Código dos Contratos Públicos. Tendo isto em conta, no enunciado é pedido que apenas se considere algumas das indicações, apresentadas de seguida:

- Os Tipos de Procedimentos são: Ajuste Direto, Consulta Previa e Concurso Publico.

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

%garante que o tipo de Procidemento pertence a : {'Ajuste Direto','Consulta Previa','Concurso Publico'}
+contrato(IdC,IdAd,IdAda,TipoC,TipoP,Des,Custo,Prazo,Loca,Data):: tipoP(TipoP).
```

Figure 25: Invariante dos tipos de procedimento

- Condições obrigatórias do contrato por ajuste direto
  - O valor do custo igual ou inferior a 5000 euros. Apesar de nada ser referido faz sentido limitar os custos de qualquer tipo de contrato para valores positivos, acrescentando assim um invariante que garanta isso.

```
custoValido(C) :-   C >=0.
custoAjDiretoVALIDO(C) :- C=<5000 , C>=0.

%garante que os custos associados aos contratos são válidos
+contrato(IdC,IdAd,IdAda,TipoC,'Consulta Previa',Des,Custo,Prazo,Loca,Data) :: custoValido(Custo).
+contrato(IdC,IdAd,IdAda,TipoC,'Concurso Publico',Des,Custo,Prazo,Loca,Data) :: custoValido(Custo).
+contrato(IdC,IdAd,IdAda,TipoC,'Ajuste Direto',Des,Custo,Prazo,Loca,Data) :: custoADVALIDO(Custo).
```

Figure 26: Invariantes do custo válido

  - Contrato de aquisição ou locação de bens móveis ou aquisição de serviços

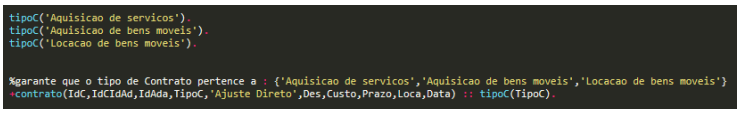


Figure 27: Invariante dos tipos de contrato

- Prazo de vigência até 1 ano, inclusive, a contar da decisão de adjudicação

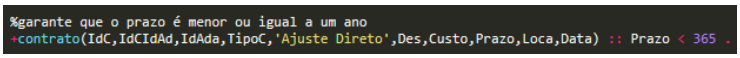


Figure 28: Invariante do prazo

- Regra dos 3 anos válida para todos os contratos
  - Uma entidade adjudicante não pode convidar a mesma empresa para celebrar um contrato com prestações de serviço do mesmo tipo ou idênticas às de contratos que já lhe foram atribuídos, no ano económico em curso e nos dois anos económicos anteriores, sempre que: O preço contratual acumulado dos contratos já celebrados (não incluindo o contrato que se pretende celebrar) seja igual ou superior a 75.000 euros.

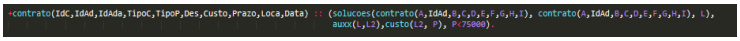


Figure 29: Invariante do prazo

Onde a função auxx filtra todos os contratos que foram celebrados há 2 ou menos anos e a função custo calcula o valor total dos restantes contratos.

7 Evolução do Conhecimento

De maneira a ser possível a evolução de todo o tipo de conhecimento respeitando todos os invariantes referidos na secção anterior, optámos neste projeto por criar procedimentos específicos para cada um destes tipos: conhecimento perfeito positivo e negativo, imperfeito incerto e imperfeito impreciso. Não foram criados procedimentos para o conhecimento imperfeito interdito uma vez que este jamais pode ser evoluído. Nesta secção iremos retratar e explicar todos os mecanismos e metodologias utilizados.

7.1 Evolução do conhecimento perfeito positivo e negativo

Numa primeira fase, e de maneira a todo o conhecimento positivo ser evoluído,ou seja, ser adicionado à base de conhecimento, foi criado o predicado evolução.

Este é idêntico ao utilizado nas aulas práticas.

Para ser possível adicionar conhecimento, o predicado evolução verifica primeiro se o elemento a ser adicionado respeita todos os invariantes, caso seja verdade então este é adicionado à base de conhecimento.

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

Figure 30: Evolução positiva de conhecimento

De maneira a ser exequível a adição de conhecimento perfeito negativo na base de conhecimento, o predicado criado é parecido ao interior, mas neste caso é necessário especificar que o conhecimento a adicionar é negativo. Neste caso os invariantes a respeitar são os que dizem respeito à inserção de conhecimento negativo.

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

Figure 31: Evolução de conhecimento negativo

Nas duas figuras seguintes são apresentados exemplos de outputs da evolução de conhecimento positivo e negativo.

```
yes
| ?- evolucao(adjudicante(12,'Municipio de Braga','95154485','Portugal,Braga, Esposende
')).
yes
| ?- listing(adjudicante).
adjudicante(1, 'Munincipio de Alto de Basto', '705330336', 'Portugal, Braga, Alto de Ba
sto').
adjudicante(2, 'Agrupamento de Escolas do Cerco, Porto', '600078965', 'Portugal, Porto'
).
adjudicante(3, 'Munincipio de Braga', '506901173', 'Portugal, Braga, Braga').
adjudicante(4, 'Munincipio de Amares', '506797627', 'Portugal, Braga, Amares').
adjudicante(5, 'Cooperativa Agricola de Barcelos , CRL', '500967580', 'Portugal, Braga,
 Barcelos').
adjudicante(6, 'AgdA - Aguas Publicas do Alentejo, S. A.', '509133843', 'Portugal, Beja
, Beja').
adjudicante(7, 'Uniao das Freguesias de Lomar e Arcos', '510837581', 'Portugal').
adjudicante(15, 'Munincipio de Vizela', '705330343', morada_desconhecida).
adjudicante(17, 'Munincipio de Vizela', '705330345', moradaInterdita).
adjudicante(12, 'Municipio de Braga', '95154485', 'Portugal,Braga, Esposende').
```

Figure 32: Output da evolução do conhecimento positivo de um adjudicante

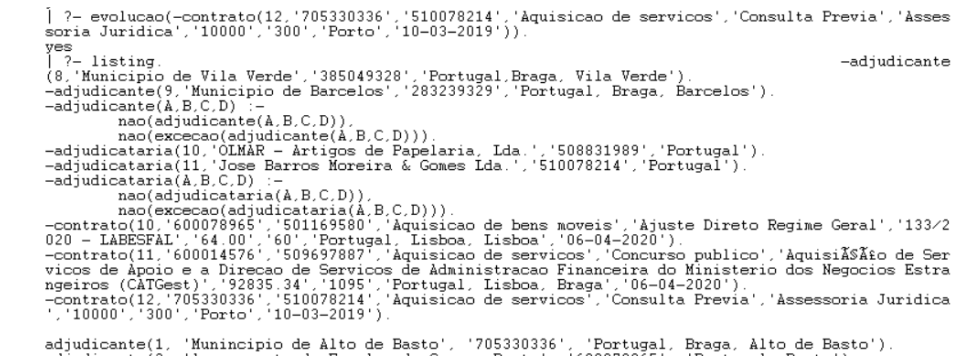


Figure 33: Output da evolução do conhecimento negativo de um contrato

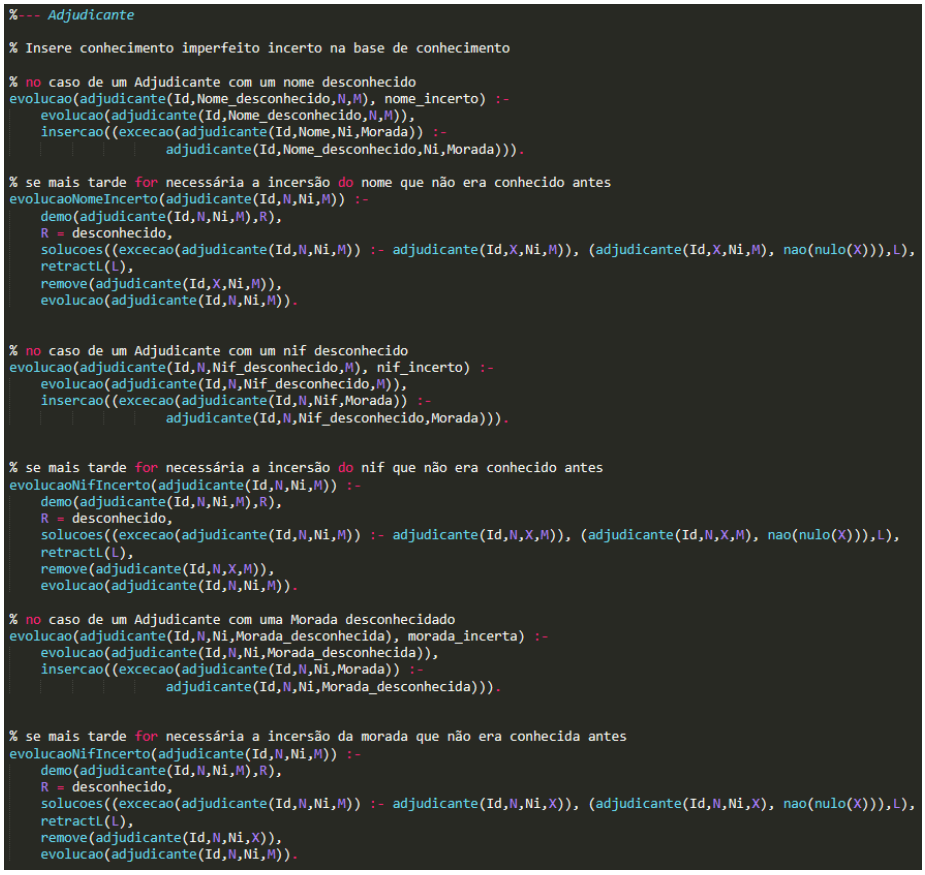
7.2 Evolução do conhecimento imperfeito incerto

O conhecimento imperfeito incerto consiste em existir algo desconhecido de um conjunto indeterminado de hipóteses. Apesar de existerem certas vezes dados desconhecidos, continua a ser necessária a sua inserção na base de conhecimento. Para tal foram criados procedimentos que permitem essa evolução.

Uma vez que temos três tipos de predicados diferentes (adjudicante, adjudicataria e contrato), cada um deles com diferentes possibilidades de campos incertos, foi necessária a elaboração de um predicado evolução para cada um deles. Este possui como argumentos o predicado a adicionar (com o respetivo campo a desconhecido) e o que se encontra incerto. O que acontece nestes casos é que é adicionado o pretendido na base de conhecimento e criada uma exceção associada ao campo desconhecido.

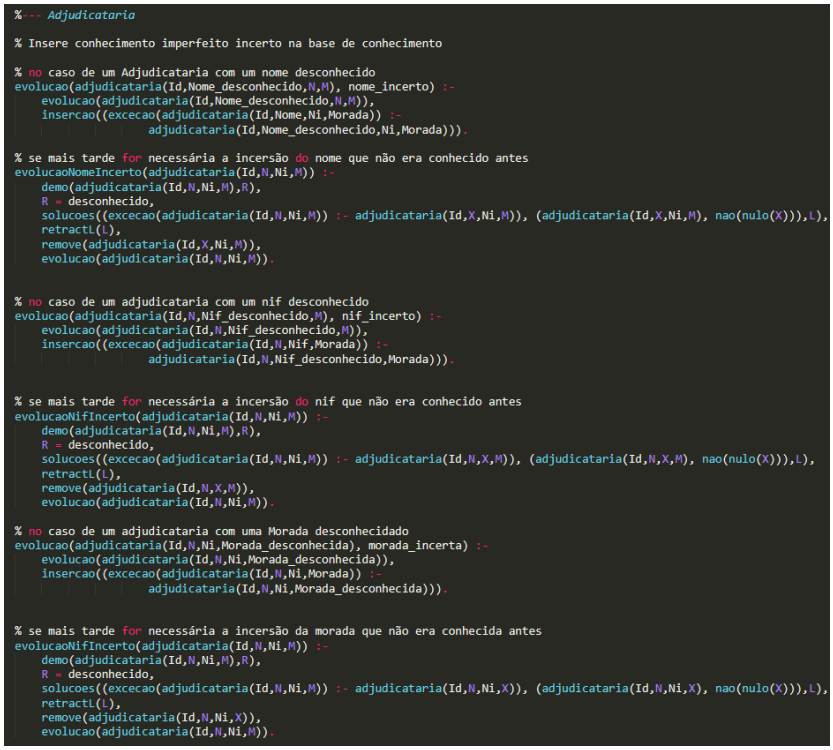
Foram também desenvolvidos procedimentos para que mais tarde, se já se souber o que deveria estar no campo desconhecido, ser possível a sua inserção. O que decorre é a remoção do predicado das execeções e a atualização do conhecimento do mesmo.

Nas figuras seguintes encontram-se todos os mecanismos criados na evolução de conhecimento incerto, tanto para o adjudicante como para o adjudicataria e ,por fim, para o contrato (como no contrato foram criados vários métodos apenas se encontram na figura os primeiros) .



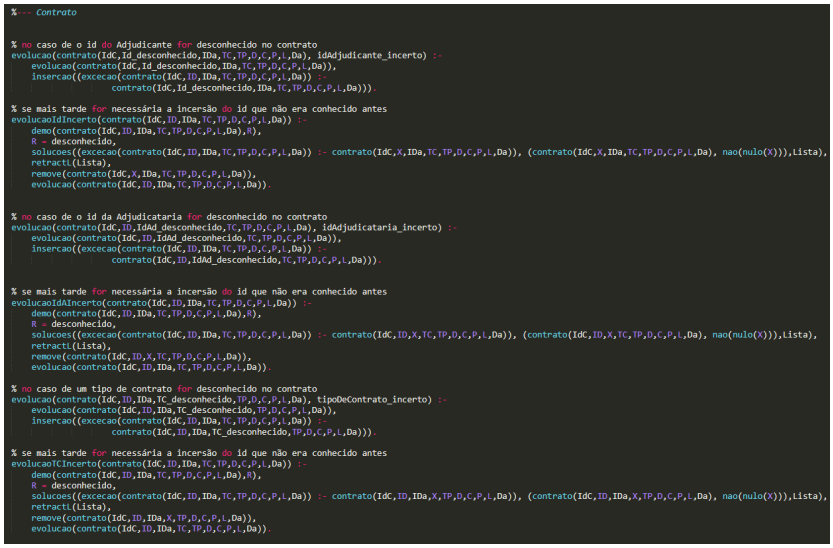
```
%--- Adjudicante

% Insere conhecimento imperfeito incerto na base de conhecimento

% no caso de um Adjudicante com um nome desconhecido
evolucao(adjudicante(Id,Nome_desconhecido,N,M), nome_incerto) :-
    evolucao(adjudicante(Id,Nome_desconhecido,N,M)),
    insercao((excecao(adjudicante(Id,Nome,Ni,Morada)) :-
                adjudicante(Id,Nome_desconhecido,Ni,Morada))).

% se mais tarde for necessária a incersão do nome que não era conhecido antes
evolucaoNomeIncerto(adjudicante(Id,N,Ni,M)) :-
    demo(adjudicante(Id,N,Ni,M),R),
    R = desconhecido,
    solucoes((excecao(adjudicante(Id,N,Ni,M)) :- adjudicante(Id,X,Ni,M)), (adjudicante(Id,X,Ni,M), nao(nulo(X))),L),
    retractL(L),
    remove(adjudicante(Id,X,Ni,M)),
    evolucao(adjudicante(Id,N,Ni,M)).


% no caso de um Adjudicante com um nif desconhecido
evolucao(adjudicante(Id,N,Nif_desconhecido,M), nif_incerto) :-
    evolucao(adjudicante(Id,N,Nif_desconhecido,M)),
    insercao((excecao(adjudicante(Id,N,Nif,Morada)) :-
                adjudicante(Id,N,Nif_desconhecido,Morada))).


% se mais tarde for necessária a incersão do nif que não era conhecido antes
evolucaoNifIncerto(adjudicante(Id,N,Ni,M)) :-
    demo(adjudicante(Id,N,Ni,M),R),
    R = desconhecido,
    solucoes((excecao(adjudicante(Id,N,Ni,M)) :- adjudicante(Id,N,X,M)), (adjudicante(Id,N,X,M), nao(nulo(X))),L),
    retractL(L),
    remove(adjudicante(Id,N,X,M)),
    evolucao(adjudicante(Id,N,Ni,M)).

% no caso de um Adjudicante com uma Morada desconhecidado
evolucao(adjudicante(Id,N,Ni,Morada_desconhecida), morada_incerta) :-
    evolucao(adjudicante(Id,N,Ni,Morada_desconhecida)),
    insercao((excecao(adjudicante(Id,N,Ni,Morada)) :-
                adjudicante(Id,N,Ni,Morada_desconhecida))).


% se mais tarde for necessária a incersão da morada que não era conhecida antes
evolucaoNifIncerto(adjudicante(Id,N,Ni,M)) :-
    demo(adjudicante(Id,N,Ni,M),R),
    R = desconhecido,
    solucoes((excecao(adjudicante(Id,N,Ni,M)) :- adjudicante(Id,N,Ni,X)), (adjudicante(Id,N,Ni,X), nao(nulo(X))),L),
    retractL(L),
    remove(adjudicante(Id,N,Ni,X)),
    evolucao(adjudicante(Id,N,Ni,M)).
```

Figure 34: Evolução de conhecimento imperfeito incerto no cado de Adjudicante



```
%--- Adjudicataria

% Insere conhecimento imperfeito incerto na base de conhecimento

% no caso de um Adjudicataria com um nome desconhecido
evolucao(adjudicataria(Id,Nome_desconhecido,N,M), nome_incerto) :-
    evolucao(adjudicataria(Id,Nome_desconhecido,N,M)),
    insercao((excecao(adjudicataria(Id,Nome,Ni,Morada)) :-
                adjudicataria(Id,Nome_desconhecido,Ni,Morada))).

% se mais tarde for necessária a incersão do nome que não era conhecido antes
evolucaoNomeIncerto(adjudicataria(Id,N,Ni,M)) :-
    demo(adjudicataria(Id,N,Ni,M),R),
    R = desconhecido,
    solucoes((excecao(adjudicataria(Id,N,Ni,M)) :- adjudicataria(Id,X,Ni,M)), (adjudicataria(Id,X,Ni,M), nao(nulo(X))),L),
    retractL(L),
    remove(adjudicataria(Id,X,Ni,M)),
    evolucao(adjudicataria(Id,N,Ni,M)).


% no caso de um adjudicataria com um nif desconhecido
evolucao(adjudicataria(Id,N,Nif_desconhecido,M), nif_incerto) :-
    evolucao(adjudicataria(Id,N,Nif_desconhecido,M)),
    insercao((excecao(adjudicataria(Id,N,Nif,Morada)) :-
                adjudicataria(Id,N,Nif_desconhecido,Morada))).


% se mais tarde for necessária a incersão do nif que não era conhecido antes
evolucaoNifIncerto(adjudicataria(Id,N,Ni,M)) :-
    demo(adjudicataria(Id,N,Ni,M),R),
    R = desconhecido,
    solucoes((excecao(adjudicataria(Id,N,Ni,M)) :- adjudicataria(Id,N,X,M)), (adjudicataria(Id,N,X,M), nao(nulo(X))),L),
    retractL(L),
    remove(adjudicataria(Id,N,X,M)),
    evolucao(adjudicataria(Id,N,Ni,M)).

% no caso de um adjudicataria com uma Morada desconhecidado
evolucao(adjudicataria(Id,N,Ni,Morada_desconhecida), morada_incerta) :-
    evolucao(adjudicataria(Id,N,Ni,Morada_desconhecida)),
    insercao((excecao(adjudicataria(Id,N,Ni,Morada)) :-
                adjudicataria(Id,N,Ni,Morada_desconhecida))).


% se mais tarde for necessária a incersão da morada que não era conhecida antes
evolucaoNifIncerto(adjudicataria(Id,N,Ni,M)) :-
    demo(adjudicataria(Id,N,Ni,M),R),
    R = desconhecido,
    solucoes((excecao(adjudicataria(Id,N,Ni,M)) :- adjudicataria(Id,N,Ni,X)), (adjudicataria(Id,N,Ni,X), nao(nulo(X))),L),
    retractL(L),
    remove(adjudicataria(Id,N,Ni,X)),
    evolucao(adjudicataria(Id,N,Ni,M)).
```

Figure 35: Evolução de conhecimento imperfeito incerto no cado de Adjudicataria



```
%-- Contrato


% no caso de o id do Adjudicante for desconhecido no contrato
evolucao(contrato(IdC,Id_desconhecido,IDa,TC,TP,D,C,P,L,Da), idAdjudicante_incerto) :-
    evolucao(contrato(IdC,Id_desconhecido,IDa,TC,TP,D,C,P,L,Da)),
    insercao((excecao(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)) :-
                contrato(IdC,Id_desconhecido,IDa,TC,TP,D,C,P,L,Da))).

% se mais tarde for necessária a incersão do id que não era conhecido antes
evolucaoIdIncerto(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)) :-
    demo(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da),R),
    R = desconhecido,
    solucoes((excecao(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)) :- contrato(IdC,X,IDa,TC,TP,D,C,P,L,Da)), (contrato(IdC,X,IDa,TC,TP,D,C,P,L,Da), nao(nulo(X))),Lista),
    retractL(Lista),
    remove(contrato(IdC,X,IDa,TC,TP,D,C,P,L,Da)),
    evolucao(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)).


% no caso de o id da Adjudicataria for desconhecido no contrato
evolucao(contrato(IdC,ID,IdAd_desconhecido,TC,TP,D,C,P,L,Da), idAdjudicataria_incerto) :-
    evolucao(contrato(IdC,ID,IdAd_desconhecido,TC,TP,D,C,P,L,Da)),
    insercao((excecao(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)) :-
                contrato(IdC,ID,IdAd_desconhecido,TC,TP,D,C,P,L,Da))).


% se mais tarde for necessária a incersão do id que não era conhecido antes
evolucaoIdAIncerto(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)) :-
    demo(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da),R),
    R = desconhecido,
    solucoes((excecao(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)) :- contrato(IdC,ID,X,TC,TP,D,C,P,L,Da)), (contrato(IdC,ID,X,TC,TP,D,C,P,L,Da), nao(nulo(X))),Lista),
    retractL(Lista),
    remove(contrato(IdC,ID,X,TC,TP,D,C,P,L,Da)),
    evolucao(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)).

% no caso de um tipo de contrato for desconhecido no contrato
evolucao(contrato(IdC,ID,IDa,TC_desconhecido,TP,D,C,P,L,Da), tipoDeContrato_incerto) :-
    evolucao(contrato(IdC,ID,IDa,TC_desconhecido,TP,D,C,P,L,Da)),
    insercao((excecao(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)) :-
                contrato(IdC,ID,IDa,TC_desconhecido,TP,D,C,P,L,Da))).

% se mais tarde for necessária a incersão do id que não era conhecido antes
evolucaoTCIncerto(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)) :-
    demo(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da),R),
    R = desconhecido,
    solucoes((excecao(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)) :- contrato(IdC,ID,IDa,X,TP,D,C,P,L,Da)), (contrato(IdC,ID,IDa,X,TP,D,C,P,L,Da), nao(nulo(X))),Lista),
    retractL(Lista),
    remove(contrato(IdC,ID,IDa,X,TP,D,C,P,L,Da)),
    evolucao(contrato(IdC,ID,IDa,TC,TP,D,C,P,L,Da)).
```

Figure 36: Evolução de conhecimento imperfeito incerto no caso de Contrato

Exemplo de output da adição de um adjudicante com um nome desconhecido:

```
 ?- evolucao(adjudicante(14,incerto,'95154485','Portugal,Braga, Esposende'),nome_incerto).
yes
| ?- listing(adjudicante).
adjudicante(1, 'Munincipio de Alto de Basto', '705330336', 'Portugal, Braga, Alto de Basto').
adjudicante(2, 'Agrupamento de Escolas do Cerco, Porto', '600078965', 'Portugal, Porto').
adjudicante(3, 'Munincipio de Braga', '506901173', 'Portugal, Braga, Braga').
adjudicante(4, 'Munincipio de Amares', '506797627', 'Portugal, Braga, Amares').
adjudicante(5, 'Cooperativa Agricola de Barcelos , CRL', '500967580', 'Portugal, Braga, Barcelos')
.
adjudicante(6, 'AgdA - Aguas Publicas do Alentejo, S. A.', '509133843', 'Portugal, Beja, Beja').
adjudicante(7, 'Uniao das Freguesias de Lomar e Arcos', '510837581', 'Portugal').
adjudicante(15, 'Munincipio de Vizela', '705330343', morada_desconhecida).
adjudicante(17, 'Munincipio de Vizela', '705330345', moradaInterdita).
adjudicante(14, incerto, '95154485', 'Portugal,Braga, Esposende').

yes
| ?- listing(excecao).
excecao(adjudicante(A,B,C,_)) :-
        adjudicante(A, B, C, morada_desconhecida).
excecao(adjudicataria(A,B,_,C)) :-
        adjudicataria(A, B, nifDesc, C).
excecao(contrato(A,B,C,D,E,F,G,_,H,F)) :-
        contrato(A, B, C, D, E, F, G, valorDesc, H, F).
excecao(adjudicante(16,'Munincipio de Vizela','705330344','Portugal,Braga, Esposende')).
excecao(adjudicante(16,'Munincipio de Vizela','705330344','Portugal,Braga, Vizela')).
excecao(adjudicataria(26,'PortSeg','506785468','Portugal')).
excecao(contrato(35,'705330336','702675112','Aquisicao de serviÃ§os','Consulta Previa','Assessoria
 Juridica','5999','500','Alto de Basto','12-02-2020')).
excecao(contrato(35,'705330336','702675112','Aquisicao de serviÃ§os','Consulta Previa','Assessoria
 Juridica','6999','500','Alto de Basto','12-02-2020')).
excecao(adjudicante(A,B,C,_)) :-
        adjudicante(A, B, C, moradaInterdita).
excecao(adjudicataria(A,B,C,_)) :-
        adjudicataria(A, B, C, moradaInterdita2).
excecao(contrato(A,B,C,D,E,F,G,_,H,F)) :-
        contrato(A, B, C, D, E, F, G, valorDesc, H, F).
excecao(adjudicante(14,_,A,B)) :-
        adjudicante(14, incerto, A, B).
```

Figure 37: Adição de um adjudicante com um nome desconhecido

Exemplo de quando mais tarde já se sabe o nome que era anteriormente incerto e se pretende alterar:

```
| ?- evolucaoNomeIncerto(adjudicante(14,'Municipio de Braga','95154485','Portugal,Braga, E
sposende')).
yes
| ?- listing(adjudicante).
adjudicante(1, 'Munincipio de Alto de Basto', '705330336', 'Portugal, Braga, Alto de Basto
').
adjudicante(2, 'Agrupamento de Escolas do Cerco, Porto', '600078965', 'Portugal, Porto').
adjudicante(3, 'Munincipio de Braga', '506901173', 'Portugal, Braga, Braga').
adjudicante(4, 'Munincipio de Amares', '506797627', 'Portugal, Braga, Amares').
adjudicante(5, 'Cooperativa Agricola de Barcelos , CRL', '500967580', 'Portugal, Braga, Ba
rcelos').
adjudicante(6, 'AgdA - Aguas Publicas do Alentejo, S. A.', '509133843', 'Portugal, Beja, B
eja').
adjudicante(7, 'Uniao das Freguesias de Lomar e Arcos', '510837581', 'Portugal').
adjudicante(15, 'Munincipio de Vizela', '705330343', morada_desconhecida).
adjudicante(17, 'Munincipio de Vizela', '705330345', moradaInterdita).
adjudicante(14, 'Municipio de Braga', '95154485', 'Portugal,Braga, Esposende').
```

Figure 38: Adição de um adjudicante com o nome correto (anteriormente era incerto)

7.3 Evolução do conhecimento imperfeito impreciso

O conhecimento impreciso consiste em existir algo desconhecido mas de um conjunto determinado de hipóteses. Certas vezes não existe com total certeza se a uma entidade é associado um determinado dado ou outro. Para tal existe a necessidade de inserir na base de conhecimento todos as hipóteses que consideramos possíveis.

De maneira a evoluir o conhecimento imperfeito, foi criado um procedimento similar à evolução, mas com a diferença de ser necessária a especificação de que é impreciso. A abordagem utilizada foi a inserção do conhecimento nas exceções respeitando na mesma todos os invariantes.

```
evolucao(T, impreciso) :-
    solucoes(I, +(excecao(T))::I, Lint),
    insercao(excecao(T)),
    teste(Lint).
```

Figure 39: Evolução de conhecimento impreciso

De forma similia à evolução do conhecimento imperfeito incerto, neste caso também foram criados métodos que permitem a inserção dos dados considerados corretos dentro do conjunto de hipóteses que era dado. A abordagem usada consiste em remover as exceções e em adicionar o caso do conhecimento correto.

```
%Adjudicante
evolucaoImprecisoAdjudicante(adjudicante(ID,Nome,Nif,Morada)):-
        demo(adjudicante(ID,Nome,Nif,Morada),R),
        R = desconhecido,
        solucoes(excecao(adjudicante(ID,N,NI,M)), excecao(adjudicante(ID,N,NI,M)),Lista),
        retractL(Lista),
        evolucao(adjudicante(ID,Nome,Nif,Morada)).

%Adjudicataria
evolucaoImprecisoAdjudicataria(adjudicataria(ID,Nome,Nif,Morada)):-
        demo(adjudicataria(ID,Nome,Nif,Morada),R),
        R = desconhecido,
        solucoes(excecao(adjudicataria(ID,N,NI,M)), excecao(adjudicataria(ID,N,NI,M)),Lista),
        retractL(Lista),
        evolucao(adjudicataria(ID,Nome,Nif,Morada)).

%Contrato
evolucaoImprecisoContrato(contrato(IDC,ID,IDA,TipoDeContrato,TipoDeProcedimento,Descricao,Custo,Prazo,Local,Data)):-
        demo(contrato(IDC,ID,IDA,TipoDeContrato,TipoDeProcedimento,Descricao,Custo,Prazo,Local,Data),R),
        R = desconhecido,
        solucoes(excecao(contrato(IDC,ID,IDA,TC,TP,D,C,P,L,DA)), excecao(contrato(IDC,ID,IDA,TC,TP,D,C,P,L,DA)),Lista),
        retractL(Lista),
        evolucao(contrato(IDC,ID,IDA,TipoDeContrato,TipoDeProcedimento,Descricao,Custo,Prazo,Local,Data)).
```

Figure 40: Adição de conhecimento correto que antes era considerado impreciso

Na figura seguinte encontra-se um exemplo da inserção de um adjudicante cuja morada é imprecisa: ou é "Portugal,Braga,Esposende" ou "Portuga,Braga,Rio Tinto". Podemos verificar que ambas as inserções vão para as exceções.

```
| ?- evolucao(adjudicante(16,'Municipio de Braga','95154485','Portugal,Braga, Esposende'),
impreciso).
yes
| ?- evolucao(adjudicante(16,'Municipio de Braga','95154485','Portugal,Braga, Rio Tinto'),
impreciso).
yes
| ?- listing(excecao).
excecao(adjudicante(A,B,C,_)) :-
        adjudicante(A, B, C, morada_desconhecida).
excecao(adjudicataria(A,B,_,C)) :-
        adjudicataria(A, B, nifDesc, C).
excecao(contrato(A,B,C,D,E,F,G,_,H,F)) :-
        contrato(A, B, C, D, E, F, G, valorDesc, H, F).
excecao(adjudicante(16,'Munincipio de Vizela','705330344','Portugal,Braga, Esposende')).
excecao(adjudicante(16,'Munincipio de Vizela','705330344','Portugal,Braga, Vizela')).
excecao(adjudicataria(26,'PortSeg','506785468','Portugal')).
excecao(contrato(35,'705330336','702675112','Aquisicao de serviÃ§os','Consulta Previa','As
sessoria Juridica','5999','500','Alto de Basto','12-02-2020')).
excecao(contrato(35,'705330336','702675112','Aquisicao de serviÃ§os','Consulta Previa','As
sessoria Juridica','6999','500','Alto de Basto','12-02-2020')).
excecao(adjudicante(A,B,C,_)) :-
        adjudicante(A, B, C, moradaInterdita).
excecao(adjudicataria(A,B,C,_)) :-
        adjudicataria(A, B, C, moradaInterdita2).
excecao(contrato(A,B,C,D,E,F,G,_,H,F)) :-
        contrato(A, B, C, D, E, F, G, valorDesc, H, F).
excecao(adjudicante(16,'Municipio de Braga','95154485','Portugal,Braga, Esposende')).
excecao(adjudicante(16,'Municipio de Braga','95154485','Portugal,Braga, Rio Tinto')).
```

Figure 41: Output de adição de conhecimento impreciso

A seguinte figura mostra a possibilidade de mais tarde ser inserida a morada correta. Podemos verificar que agora o adjudicante é listado corretamente e retirado das exceções.

```
| ?- evolucaoImprecisoAdjudicante(adjudicante(16,'Municipio de Braga','95154485','Portugal
,Braga, Rio Tinto')).
yes
| ?- listing(adjudicante).
adjudicante(1, 'Munincipio de Alto de Basto', '705330336', 'Portugal, Braga, Alto de Basto
').
adjudicante(2, 'Agrupamento de Escolas do Cerco, Porto', '600078965', 'Portugal, Porto').
adjudicante(3, 'Munincipio de Braga', '506901173', 'Portugal, Braga, Braga').
adjudicante(4, 'Munincipio de Amares', '506797627', 'Portugal, Braga, Amares').
adjudicante(5, 'Cooperativa Agricola de Barcelos , CRL', '500967580', 'Portugal, Braga, Ba
rcelos').
adjudicante(6, 'AgdA - Aguas Publicas do Alentejo, S. A.', '509133843', 'Portugal, Beja, B
eja').
adjudicante(7, 'Uniao das Freguesias de Lomar e Arcos', '510837581', 'Portugal').
adjudicante(15, 'Munincipio de Vizela', '705330343', morada_desconhecida).
adjudicante(17, 'Munincipio de Vizela', '705330345', moradaInterdita).
adjudicante(16, 'Municipio de Braga', '95154485', 'Portugal,Braga, Rio Tinto').
```

Figure 42: Colocação de informação correta no conhecimento impreciso

8 Involução de conhecimento

Para ser possível a remoção de conhecimento da base de conhecimento, o raciocínio adotado foi análogo ao da problemática de evolução. Foram criados portanto procedimentos para a involução de conhecimento perfeito positivo e negativo e para a involução de conhecimento imperfeito incerto e impreciso.

8.1 Involução do conhecimento perfeito positivo e negativo

Foi criado o predicado involução análogo ao das aulas práticas para permitir a remoção de conhecimento da base de conhecimento verificando todos os invariantes necessários. Só em caso de sucesso é que o conhecimento é removido.

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

Figure 43: Involução do conhecimento

De maneira a permitir também a involução do conhceimento negativo, foi criado outro predicado involução mas com a particularidade de ser necessária a coloção de um **-** para indicar que é conhecimento negativo.

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

Figure 44: Involução do conhecimento negativo

Nas figuras seguintes são apresentados exemplos de output da involução de conhecimento positivo e negativo.

```
| ?- listing(adjudicante).
adjudicante(1, 'Munincipio de Alto de Basto', '705330336', 'Portugal, Braga, Alto de Basto
').
adjudicante(2, 'Agrupamento de Escolas do Cerco, Porto', '600078965', 'Portugal, Porto').
adjudicante(3, 'Munincipio de Braga', '506901173', 'Portugal, Braga, Braga').
adjudicante(4, 'Munincipio de Amares', '506797627', 'Portugal, Braga, Amares').
adjudicante(5, 'Cooperativa Agricola de Barcelos , CRL', '500967580', 'Portugal, Braga, Ba
rcelos').
adjudicante(6, 'AgdA - Aguas Publicas do Alentejo, S. A.', '509133843', 'Portugal, Beja, B
eja').
adjudicante(7, 'Uniao das Freguesias de Lomar e Arcos', '510837581', 'Portugal').
adjudicante(15, 'Munincipio de Vizela', '705330343', morada_desconhecida).
adjudicante(17, 'Munincipio de Vizela', '705330345', moradaInterdita).
adjudicante(16, 'Municipio de Braga', '95154485', 'Portugal,Braga, Rio Tinto').

yes
| ?- involucao(adjudicante(16,'Municipio de Braga','95154485','Portugal,Braga, Rio Tinto')
).
yes
| ?- listing(adjudicante).
adjudicante(1, 'Munincipio de Alto de Basto', '705330336', 'Portugal, Braga, Alto de Basto
').
adjudicante(2, 'Agrupamento de Escolas do Cerco, Porto', '600078965', 'Portugal, Porto').
adjudicante(3, 'Munincipio de Braga', '506901173', 'Portugal, Braga, Braga').
adjudicante(4, 'Munincipio de Amares', '506797627', 'Portugal, Braga, Amares').
adjudicante(5, 'Cooperativa Agricola de Barcelos , CRL', '500967580', 'Portugal, Braga, Ba
rcelos').
adjudicante(6, 'AgdA - Aguas Publicas do Alentejo, S. A.', '509133843', 'Portugal, Beja, B
eja').
adjudicante(7, 'Uniao das Freguesias de Lomar e Arcos', '510837581', 'Portugal').
adjudicante(15, 'Munincipio de Vizela', '705330343', morada_desconhecida).
adjudicante(17, 'Munincipio de Vizela', '705330345', moradaInterdita).
```

Figure 45: Involução positiva de um adjudicante

```
| ?- involucao(-contrato(10,'60078965','501169580','Aquisicao de bens moveis','Ajuste Dire
to Regime Geral','133/2020 - LABESFAL','64.00','60','Portugal,Lisboa,Lisboa','06-04-2020')
).
yes
```

Figure 46: Involução negativa de um contrato

8.2 Involução do conhecimento imperfeito incerto

De forma a ser possível a remoção do conhecimento imperfeito incerto da base de conhecimento e, de forma semelhante à evolução do mesmo, foram então criados procedimentos para cada um dos tipos de predicado existentes.

```
%---Adjudicante

% no caso de um Adjudicante com nome desconhecido
involucao(adjudicante(Id,Nome_desconhecido,N,M), nome_incerto) :-
    involucao(adjudicante(Id,Nome_desconhecido,N,M)),
    remocao((excecao(adjudicante(Id,Nome,Nif,Morada)) :-
                adjudicante(Id,Nome_desconhecido,Nif,Morada))).


% no caso de um Adjudicante com um nif desconhecido
involucao(adjudicante(Id,N,Nif_desconhecido,M), adjudicante, nif_incerto) :-
    involucao(adjudicante(Id,N,Nif_desconhecido,M)),
    remocao((excecao(adjudicante(Id,Nome,Nif,Morada)) :-
                adjudicante(Id,Nome,Nif_desconhecido,Morada))).

% no caso de um Adjudicante com uma Morada desconhecida
involucao(adjudicante(Id,Nome,Nif,Morada_desconhecida),morada_incerta) :-
    involucao(adjudicante(Id,Nome,Nif_desconhecido,Morada_desconhecida)),
    remocao((excecao(adjudicante(Id,Nome,Nif,Morada)) :-
                adjudicante(Id,Nome,Nif,Morada_desconhecida))).
```

Figure 47: Involução do conhecimento imperfeito incerto no caso do adjudicante

```
%--- Adjudicataria

% no caso de um Adjudicataria com nome desconhecido

involucao(adjudicataria(Id,Nome_desconhecido,N,M), nome_incerto) :-
    involucao(adjudicataria(Id,Nome_desconhecido,N,M)),
    remocao((excecao(adjudicataria(Id,Nome,Nif,Morada)) :-
                adjudicataria(Id,Nome_desconhecido,Nif,Morada))).


% no caso de um Adjudicataria com um nif desconhecido
involucao(adjudicataria(Id,N,Nif_desconhecido,M), nif_incerto) :-
    involucao(adjudicataria(Id,N,Nif_desconhecido,M)),
    remocao((excecao(adjudicataria(Id,Nome,Nif,Morada)) :-
                adjudicataria(Id,Nome,Nif_desconhecido,Morada))).

% no caso de um Adjudicataria com uma Morada desconhecida
involucao(adjudicataria(Id,Nome,Nif,Morada_desconhecida), morada_incerta) :-
    involucao(adjudicataria(Id,Nome,Nif_desconhecido,Morada_desconhecida), positivo),
    remocao((excecao(adjudicataria(Id,Nome,Nif,Morada)) :-
                adjudicataria(Id,Nome,Nif,Morada_desconhecida))).
```

Figure 48: Involução do conhecimento imperfeito incerto no caso da adjudicataria

Figure 49: Involução do conhecimento imperfeito incerto no caso do contrato

Na figura seguinte encontra-se um exemplo de involução de conhecimento de um adjudicante com o nome incerto.

```
| ?- listing(adjudicante).
adjudicante(1, 'Munincipio de Alto de Basto', '705330336', 'Portugal, Braga, Alto de Basto
').
adjudicante(2, 'Agrupamento de Escolas do Cerco, Porto', '600078965', 'Portugal, Porto').
adjudicante(3, 'Munincipio de Braga', '506901173', 'Portugal, Braga, Braga').
adjudicante(4, 'Munincipio de Amares', '506797627', 'Portugal, Braga, Amares').
adjudicante(5, 'Cooperativa Agricola de Barcelos , CRL', '500967580', 'Portugal, Braga, Ba
rcelos').
adjudicante(6, 'AgdA - Aguas Publicas do Alentejo, S. A.', '509133843', 'Portugal, Beja, B
eja').
adjudicante(7, 'Uniao das Freguesias de Lomar e Arcos', '510837581', 'Portugal').
adjudicante(15, 'Munincipio de Vizela', '705330343', morada_desconhecida).
adjudicante(17, 'Munincipio de Vizela', '705330345', moradaInterdita).
adjudicante(16, incerto, '95154485', 'Portugal,Braga, Rio Tinto').

yes
| ?- involucao(adjudicante(16,incerto,'95154485','Portugal,Braga, Rio Tinto'),nome_incerto
).
yes
| ?- listing(adjudicante).
adjudicante(1, 'Munincipio de Alto de Basto', '705330336', 'Portugal, Braga, Alto de Basto
').
adjudicante(2, 'Agrupamento de Escolas do Cerco, Porto', '600078965', 'Portugal, Porto').
adjudicante(3, 'Munincipio de Braga', '506901173', 'Portugal, Braga, Braga').
adjudicante(4, 'Munincipio de Amares', '506797627', 'Portugal, Braga, Amares').
adjudicante(5, 'Cooperativa Agricola de Barcelos , CRL', '500967580', 'Portugal, Braga, Ba
rcelos').
adjudicante(6, 'AgdA - Aguas Publicas do Alentejo, S. A.', '509133843', 'Portugal, Beja, B
eja').
adjudicante(7, 'Uniao das Freguesias de Lomar e Arcos', '510837581', 'Portugal').
adjudicante(15, 'Munincipio de Vizela', '705330343', morada_desconhecida).
adjudicante(17, 'Munincipio de Vizela', '705330345', moradaInterdita).
```

Figure 50: Output de involução de conhecimento imperfeito incerto

8.3 Involução do conhecimento imperfeito impreciso

Para ser possível a remoção do conhecimento imperfeito impreciso na base de conhecimento foi criado o procedimento que se encontra na figura seguinte.

```
involucao(Termo, impreciso) :-
    solucoes(Invariante, -(excecao(Termo))::Invariante, Lista),
    remove(excecao(Termo)),
    teste(Lista).
```

Figure 51: Involução do conhecimento imperfeito impreciso

Um exemplo de output de conhecimento imperfeito impreciso pode ser visto na próxima figura.

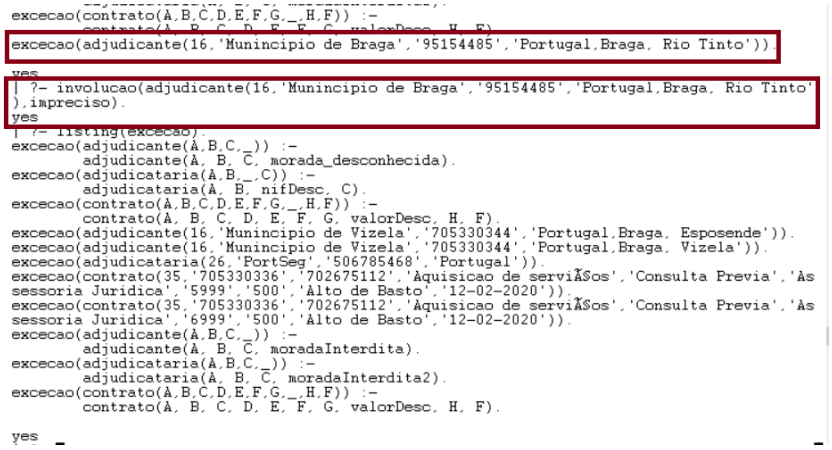


Figure 52: Output de involução de conhecimento imperfeito impreciso

9 Sistema de Inferência

De modo a conseguirmos verificar a veracidade de um determinado termo foi utilizado o predicado **demo**(dado nas aulas práticas da cadeira). Este retorna verdade caso exista uma prova explícita na base de conhecimento em que o parâmetro Questão seja verdadeiro, falso se existir na base de conhecimento uma negação forte da Questão, e desconhecido caso não existam informações sobre a Questão.

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

Figure 53: Predicado demo

Para aumentar a eficiência do nosso Sistema de Inferência foi também criado um predicado que permite responder a várias perguntas ao mesmo tempo, **demoLista**.

```
demoLista([],[]).
demoLista([Questao|Qs],[R|Rs]) :- demo(Questao,R),
                                  demoLista(Qs,Rs).
```

Figure 54: Predicado demoLista

Posto isto, de modo a aprofundar a relação entre questões foi criada um demo complementar que dadas duas Questoes e um tipo de operação, nomeadamente equivalência, conjunção e disjunçao, retorna o respetivo valor.

Para isso baseamo-nos na seguinte tabela de verdade:

p	q	p ∧ q	p v q	p ↔ q
1	1	1	1	1
1	0	0	1	0
0	1	0	1	0
0	0	0	0	1

Figure 55: Tabela de Verdade

```
%--------------------------------
% Extensao do predicado demo: Questao1, Tipo, Questao2, Flag -> {V, F, D}

demo(Q1, eq, Q2, F) :- demo(Q1, F1),
                       demo(Q2, F2),
                       equivalencia(F1, F2, F).

demo(Q1, ou, Q2, F) :- demo(Q1, F1),
                       demo(Q2, F2),
                       disjuncao(F1, F2, F).

demo(Q1, e, Q2, F) :- demo(Q1, F1),
                      demo(Q2, F2),
                      conjuncao(F1, F2, F).
```

Figure 56: Predicado demo com respetivas operações

10 Extras

De maneira a tornar o trabalho mais completo, foram desenvolvidos alguns procedimentos considerados necessários nesta problemática e que permitem auxiliar um possível utilizador do programa.

Deste modo, dois dos mecanismos considerados necessários foram a apresentação do total pago por um determinado adjudicante e do total recebido de uma determinada adjudicataria. Neste caso o argumento a passar é o Nif ou do adjudicantes ou da adjudicataria.
Na figura seguinte é possível vermos o código originado para o efeito.

```
%Valor total que um adjudicante já pagou
totalPagoAdjudicante(Adjudicante,T) :- solucoes(Custo, contrato(_,Adjudicante,_,_,_,_,Custo,_,_,_),Lista), somaLista(Lista,T).

%Valor total que uma adjudicataria já recebeu
totalGanhoAdjudicataria(Adjudicataria,T) :- solucoes(Custo, contrato(_,_,Adjudicataria,_,_,_,Custo,_,_,_),Lista), somaLista(Lista,T).
```

Figure 57: Funções que apresentam o total pago e recebido

Foram também criadas funções que para uma entidade, quer do tipo adjudicante e quer do tipo adjudicataria, dado o seu Nif devolve o a lista e o número de contratos realizados.

```
nContratosAdjudicante(IdAd,L,R) :- (solucoes(contrato(A,IdAd,B,C,D,E,F,G,H,I), contrato(A,IdAd,B,C,D,E,F,G,H,I),L),comprimento(L,R)).
nContratosAdjudicataria(IdAda,L,R) :- (solucoes(contrato(A,B,IdAda,C,D,E,F,G,H,I), contrato(A,B,IdAda,C,D,E,F,G,H,I),L),comprimento(L,R)).
```

Figure 58: Lista e número de contratos realizados por um Adjudicante/Adjudicatario

Num contexto semelhante, foram também criadas funções que para um adjudicante devolve uma lista de adjudicatárias com quem já teve um contrato, e vice versa:

```
%Dado um adjudicante, devolve a lista de adjudicatarias com quem já efetuou contratos
listaDeAdjudicatarios(NIF,R) :- solucoes(adjudicataria(_,_,NIF2,_),
      contrato(_,NIF,NIF2,_,_,_,_,_,_,_),R).

%Dado uma adjudicataria, devolve a lista de adjudicantes com quem já efetuou contratos
listaDeAdjudicantes(NIF,R) :- solucoes(adjudicante(_,_,NIF2,_),
      contrato(_,NIF2,NIF,_,_,_,_,_,_,_),R).
```

Figure 59: Função que devolve lista de adjudicatários e outra para devolver lista de adjudicantes, respetivamente

Decidimos ainda optar por desenvolver mais três funções simples mas importantes, estas devolvem o número de entidades de cada predicado:

```
%Predicado que devolve o total de cada entidade
%total de adjudicantes
numero_adjudicantes(R) :- solucoes(Id, adjudicante(Id,_,_,_),L), comprimento(L,R).

%total de adjudicatarias
numero_adjudicatarias(R) :- solucoes(Id, adjudicataria(Id,_,_,_),L),
                            comprimento(L,R).
%total de contratos
numero_contratos(R) :- solucoes(Id, contrato(Id,_,_,_,_,_,_,_,_,_),L),
                       comprimento(L,R).
```

Figure 60: Funções que devolvem o número de entidades de Adjudicantes, adjudicatários e contratos, respetivamente

Relativamente ao input de datas, fizemos três tipos de queries, as quais são sobre três tipos de datas, ou seja, vamos ter o tipo de data normal, no formato 'dd-mm-aaaa', o tipo de data mês, no formato 'mm-aaaa' e por fim o tipo de ano no formato 'aaaa', em que o conjunto de caracteres 'd' indica o dia, 'm' o mes e 'a' o ano.

```
%dado uma data no formato dd-mm-aaaa devolve o nr de contratos realizados nesse dia
nrContratosData(Data,T) :-
   solucoes(Data, contrato(_,_,_,_,_,_,_,_,_,Data),List),comprimento(List,T).

%dado uma data no formato mm-aaaa devolve o nr de contratos realizados nesse mes
nrContratosDataMes(Data,T) :-
  solucoes(Da, ( contrato(_,_,_,_,_,_,_,_,_,Da),compareDatasMA(Da,Data)),List),comprimento(List,T).

%dado uma data no formato aaaa devolve o nr de contratos realizados nesse ano
nrContratosDataAno(Data,T) :-
  solucoes(Da, ( contrato(_,_,_,_,_,_,_,_,_,Da),compareDatasA(Da,Data)),List),comprimento(List,T).
```

Figure 61: Funções que devolvem o número de contratos realizados em determinadas datas

```
%dado uma data no formato dd-mm-aaaa devolve os contratos realizados nesse dia
contratosData(Data,T) :-
    solucoes(contrato(Id,A,H,B,C,D,Custo,E,F,Data), contrato(Id,A,H,B,C,D,Custo,E,F,Data),T).

%dado uma data no formato mm-aaaa devolve os contratos realizados nesse mes
contratosDataMes(Data,T) :-
   solucoes(contrato(Id,A,H,B,C,D,Custo,E,F,Da), (contrato(Id,A,H,B,C,D,Custo,E,F,Da),compareDatasMA(Da,Data)),T).

%dado uma data no formato aaaa devolve os contratos realizados nesse ano
contratosDataAno(Data,T) :-
   solucoes(contrato(Id,A,H,B,C,D,Custo,E,F,Da), (contrato(Id,A,H,B,C,D,Custo,E,F,Da),compareDatasA(Da,Data)),T).
```

Figure 62: Funções que devolvem a lista de contratos realizados em determinadas datas

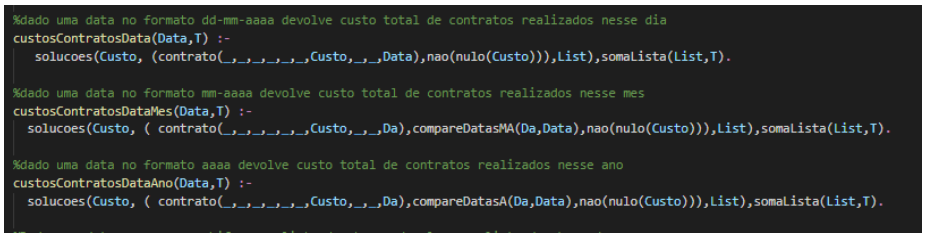


Figure 63: Funções que devolvem o custo total dos contratos realizados em determinadas datas

11 Conclusão

Concluímos então este exercicio da unidade curricular com o objetivo de desenvolver um sistema de caracterização de contratos, os seus adjudicantes e adjudicatários baseados nas fontes fornecidas pelos docentes. Este sistema inclui diversos tipos de conhecimento, nomenadamente, adjudicantes, adjudicatários e contratos.

Foram implementadas funcionalidades que representam todo o tipo de conhecimento estudado, nomeadamente, positivo e negativo como parte do conhecimento perfeito e incerto, impreciso e interdito do conhecimento imperfeito. Foi também analisada a problemática da evolução e da involução.

Assim, a realização deste exercício foi bastante importante para continuarmos a adquirir experiência e conhecimento, à medida que vamos entendendo como é que esta linguagem de programação lógica funciona e que problemas podem ser resolvidos e analisados com ela. Conseguimos, desta forma, consolidar não só todos os conhecimentos adquiridos nas aulas práticas, como também vários outros que foram sendo necessários durante a realização deste exercício.

12 Referências