Programação em lógica estendida e Conhecimento imperfeito

Sistemas de Representação de Conhecimento e Raciocínio

Mestrado Integrado em Engenharia Informática

(2019/2020)

A85227 João Azevedo

A85729 Paulo Araújo

A83719 Pedro Machado

A89983 Paulo Lima

Braga

Abril 2020

Resumo

O trabalho desenvolvido pretendia abordar a programação em lógica através do **PROLOG** e do Conhecimento Imperfeito. Assim, era pretendido desenvolver um sistema capaz de armazenar conhecimento relativo aos Contratos Públicos e procurar através de predicados e invariantes resolver as problemáticas de evolução e involução de conhecimento.

Ao longo deste relatório vamos proceder à descrição do trabalho realizado bem como a metodologia que seguimos. Assim, vamos descrever todas as normas e procedimentos aplicados aos conhecimentos do sistema e, de seguida, demonstrar quais os resultados obtidos nos predicados e funcionalidades desenvolvidas.

Em suma, achamos que o trabalho desenvolvido cumpre os objetivos propostos pela equipa docente e que este possibilitou a uma melhor assimilação de conhecimentos teóricos e práticos dados pela cadeira de Sistemas de Representação de Conhecimento e Raciocínio, bem como da programação lógica em **PROLOG**.

Índice

Capítulo 1

Introdução

Este relatório é relativo ao trabalho prático da UC de Sistemas de Representação de Conhecimento e Raciocínio que tem como objetivos a motivação para o uso de linguagens lógicas, 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.

Para este trabalho era pretendido o desenvolvimento de um sistema de representação de conhecimento com capacidade para caracterizar um universo de discurso na área da **contratação pública** para a realização de contratos com intuito de possibilitar prestações de serviços.

Capítulo 2

Preliminares

Primeiramente, o grupo procedeu a uma investigação da área da Contratação pública com vista a compreender o universo que era pretendido representar. Para isso, o enunciado proposto e o Portal Base serviram como grande influência para este desenvolvimento. De realçar que o Portal Base é o portal de contratos públicos que se destina a divulgar a informação pública sobre estes contratos sujeitos ao regime do Código dos Contratos Públicos.

2.1 Definições relativas ao sistema

Torna-se, assim, necessário destacar algumas definições de entidades e do contrato para representar este conhecimento relativo aos contratos públicos, bem como o tipo de conhecimento que podem transmitir.

- **adjudicante**

  A entidade adjudicante (entidade pública) num contrato público conduz e decide o procedimento de formação de contrato e são responsáveis por introduzir informação sobre os contratos públicos celebrados.

  Uma entidade adjudicante é caracterizada pelo seu identificador único (*IdAd*), nome, número de identificação fiscal (*NIF*, que tem de ser um valor compreendido entre 0 a 999999999) e morada. De realçar que o seu *IdAd* apenas a identifica em relação a outras entidades adjudicantes, sendo que o *NIF* a identifica de forma geral. Assim, pode haver *IdAd* e *IdAda*(identificador da entidade adjudicatária) com o mesmo valor porque identificam entidades diferentes.

  ```
  adjudicante: #IdAd, Nome, NIF, Morada -> { V, F, D }
  ```

- **adjudicataria**

  A entidade adjudicatária num contrato público é titular da proposta que foi objeto da decisão de adjudicação e corresponde à entidade com quem a entidade adjudicante irá celebrar um contrato público. De realçar que uma entidade adjudicatária pode ser pública ou privada.

Uma entidade adjudicatária pode ser representada da mesma forma que a entidade adjudicante.

```
adjudicatária: #IdAda, Nome, NIF, Morada -> { V, F, D }
```

- **contrato**

  Um contrato público é o instrumento dado à administração pública utilizado para adquirir bens ou serviços a particulares. Este contrato realiza-se entre entidades adjudicantes e adjudicatárias e pode ser caracterizado pelo seu próprio identificador único (*IdC*), pelos identificadores das entidades que o contrato representa (*IdAd* e *IdAda*), o tipo de contrato, o tipo de procedimento, uma descrição, o custo associado a este, o prazo do qual vinca o contrato, o local de assinatura e a data.

```
contrato: #IdC, #IdAd, #IdAda, TipoDeContrato,
    TipoDeProcedimento, Descrição, Custo,
    Prazo, Local, Data -> { V, F, D }
```

2.2 Normas e Procedimentos previstas para Contratos Públicos

Para além destas entidades principais, foi necessário ter em atenção regras a nível dos contratos públicos segundo o enunciado recebido.

2.2.1 Tipos de Procedimento

Para este trabalho, consideramos apenas os seguintes tipos de procedimento: **Ajuste Direto, Consulta Prévia e Concurso Público**.

O **ajuste direto** é o procedimento em que a entidade adjudicante convida diretamente uma entidade, à sua escolha, a apresentar proposta.

A **consulta prévia** é o procedimento em que a entidade adjudicante convida diretamente pelo menos três entidades à sua escolha a apresentar proposta, podendo com elas negociar os aspetos da execução do contrato a celebrar, desde que tal possibilidade conste expressamente do convite.

Já o **concurso público** é um procedimento concorrencial, dado a conhecer através de anúncio publicado no Diário da República quando o valor do contrato a celebrar for superior aos limiares comunitários.

2.2.2 Contratos por Ajuste Direto

Para todos os contratos cujo tipo de procedimento fosse o ajuste direto, o grupo assumiu os seguintes requisitos. De realçar que o ajuste direto considerado foi o simplificado, que se pode consultar no Portal Base.

- Caso o tipo de contrato fosse **aquisição ou locação de bens móveis ou aquisição de serviços**, então o valor do contrato teria de ser menor ou igual a 5.000 euros.

- Caso tivesse **empreitadas de obras públicas** como tipo de contrato, o valor teria de ser menor ou igual a 10.000 euros.

- De notar que o contrato precisa sempre de ter um prazo até 1 ano, inclusive, a contar da decisão de adjudicação.

2.2.3 Regra dos 3 anos válida para todos os contratos

A regra dos 3 anos consiste no facto de uma entidade adjudicante não poder 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.

Para além destes procedimentos e normas dos contratos públicos, o grupo consultou o **Portal Base** com o intuito de acrescentar uma maior veracidade ao nosso sistema de representação de conhecimento e raciocínio.

Assim, o grupo considerou as seguintes normas:

2.2.4 Contratos por Consulta Prévia

Todos contratos que tivessem consulta prévia como tipo de procedimento, teriam de seguir o seguinte conjunto de regras.

- Caso o tipo de contrato fosse **aquisição ou locação de bens móveis ou aquisição de serviços**, o valor permitido nunca poderá ultrapassar ou chegar aos 75.000 euros.

- Caso o contrato fosse de **empreitadas de obras públicas** seria apenas possível um valor inferior a 150.000 euros.

- Caso fosse **outro** qualquer tipo de contrato, o valor só poderia ser inferior a 100.000 euros.

2.2.5 Contratos por Concurso Público

A nível dos contratos por concurso público, estes seguem o seguinte conjunto de regras:

- Caso o tipo de contrato seja **empreitadas de obras públicas**, então o valor do contrato seria inferior a 5.350.000 euros.
- Caso o tipo de contrato seja **aquisição ou locação de bens móveis ou aquisição de serviços**, teremos duas hipóteses:
  - Se a entidade adjudicante é o Estado, o valor tem de ser inferior a 139.000 euros;
  - Se a entidade adjudicante não é o Estado, o valor tem de ser inferior a 214.000 euros;

Assim, este conjunto de normas e procedimentos caracteriza os diferentes invariantes que estão explícitos no capítulo seguinte (Capítulo 3).

Capítulo 3

Descrição do Trabalho e Análise de Resultados

3.1 Povoamento de conhecimento no sistema

Depois de definidas os procedimentos e normas previstos aos contratos públicos, é necessário desenvolver predicados em **PROLOG** capazes de transmitir este conhecimento e raciocínio.

Assim, começamos por desenvolver um pequeno povoamento para as entidades adjudicantes e adjudicatárias, bem como para o contrato, recorrendo ao Portal Base para informações mais realistas.

De realçar que no povoamento não usamos qualquer tipo de acentos ou caracteres especiais.

```
% Extensão do Predicado adjudicante: #IdAd, Nome,
    NIF, Morada -> {V,F,D}
adjudicante(1, 'Estado Portugues', 1, 'Portugal').
                                ...

% Extensão do Predicado adjudicataria: #IdAda, Nome,
    NIF, Morada -> {V,F,D}
adjudicataria(1, 'SA LIMPA - SOCIEDADE DE LIMPEZAS, LDA.',
    504458086, 'Portugal').
                                ...

% Extensão do Predicado contrato:
    #IdC, #IdAd, #IdAda, Tipo de Contrato,
    % Tipo de Procedimento, Descriçao,
    % Custo, Preço, Local, Data
    % -> {V,F,D}
contrato(1, 2, 3, 'Aquisicao de servicos', 'Ajuste Direto',
    'Aquisicao de servicos para a elaboracao do estudo
    urbano sobre a zona norte da Cidade',
    5000, 112, 'Guimaraes', data(19, 11, 2019)).
                                ...
```

3.2 Evolução de conhecimento

De seguida, podemos começar a tratar da evolução do conhecimento do sistema através da implementação de invariantes. Estes invariantes foram definidos através das regras anteriormente indicadas (secção 2.2).

De notar que foram usados predicados auxiliares na construção destes invariantes, sendo que estes estão detalhados na secção 3.4.

3.2.1 Invariante para a entidade Adjudicante

Este invariante serve para controlar a evolução do predicado "adjudicante" e, assim, prevenir que adjudicantes inválidos possam ser inseridos no sistema.

Em suma, o invariante verifica se a identificação inserida é um número inteiro e se o NIF da entidade é válido e único. A identificação da entidade (Id) tem de ser única na lista de adjudicantes e, por último, valida se existe uma entidade adjudicatária com o mesmo NIF. No caso da existência do mesmo NIF numa entidade adjudicatária, é necessário que essa entidade possua o mesmo Nome e Morada, ou seja, seja a mesma entidade.

```
+adjudicante(Id, Nome, NIF, Morada) ::
(
    integer(Id),
    solucoes((I, Nif), adjudicante(I, N, Nif, M), Lista),
    isNif(NIF),
    parseIdNif(Lista, [], [], IList, NList),
    nao(pertence(Id, IList)),
    nao(pertence(NIF, NList)),
    solucoes((NOME, Mord), adjudicataria(IdAda, NOME, NIF, Mord), L),
    mesmaEnt((Nome, Morada), L)
).
```

3.2.2 Invariante para a entidade Adjudicatária

Este invariante é em muito semelhante ao anterior, pelo que a única diferença é da utilização deste noutra entidade, a entidade adjudicatária.

```
+adjudicataria(Id, Nome, NIF, Morada) ::
(
    integer(Id),
    solucoes((I, Nif), adjudicataria(I, N, Nif, M), Lista),
    isNif(NIF),
    parseIdNif(Lista, [], [], IList, NList),
    nao(pertence(Id, IList)),
    nao(pertence(NIF, NList)),
    solucoes((NOME, Mord), adjudicante(IdAd, NOME, NIF, Mord), L),
    mesmaEnt((Nome, Morada), L)
).
```

3.2.3 Invariantes para o Contrato

De forma a controlar a evolução dos Contratos, foram feitos vários invariantes para os diversos tipos de contratos e as muitas regras que estes necessitavam de seguir. Iremos então abordar cada um destes, explicitando o porquê do seu uso.

- **Invariante que define os requisitos mínimos**

  Este invariante serve para verificar se todos os campos do contrato inserido são válidos. Assim, qualquer contrato inserido passa por este invariante, pelo que, tem de ter as seguintes características:

  - Os identificadores têm de ser números inteiros;
  - O indentificador do contrato (IdC) tem de ser único na lista de contratos do sistema;
  - O tipo de procedimento tem de ser válido, ou seja, tem de ser "Ajuste Direto", "Consulta Previa"ou "Concurso Publico";
  - A data inserida tem de ser válida, ou seja, o ano estar entre 1900 e 2021 (estes valores foram assumidos pelo grupo) e ser um dia/mês válido.
  - O valor do contrato não ser negativo;
  - O prazo ser positivo;
  - As entidades adjudicantes e adjudicatárias do contrato precisam de existir no sistema;

  Estas regras são aplicadas na seguinte forma sob a linguagem **PROLOG**:

```
+contrato(IdC, IdAd, IdAda, TC, TP, DESC, Va, PR, L, Data) ::
(
    integer(IdC), integer(IdAd), integer(IdAda),
    solucoes(IdC, contrato(IdC, Id, IdA, Tc, Tp, Desc,
        V, Prazo, Local, D), Lista),
    tiraUltimo(Lista, [], R),
    nao(pertence(IdC, R)),
    tipoProcedimento(TP),
    Data,
    Va >= 0,
    PR > 0,
    solucoes(IdAd, adjudicante(IdAd, N, Nif, M), ListIdAd),
    comprimento(ListIdAd, 1),
    solucoes(IdAda, adjudicataria(IdAda, N, Nif, M), ListIdAda),
    comprimento(ListIdAda, 1)
).
```

- **Invariante para Contratos por Ajuste Direto**

  Todos os contratos que possuam o "Ajuste Direto" como tipo de procedimento necessitam de ir de acordo com as regras definidas na secção 2.2.2. De notar que o grupo assumiu o valor de 365 dias para 1 ano.

  De realçar que usamos o *if-else* do **PROLOG** neste invariante, bem como noutros invariantes/predicados desenvolvidos.

```
+contrato(IdC, IdAd, IdAda, TC, 'Ajuste Direto',
    DESC, Va, PR, L, D) ::
(
    PR =< 365,
    (TC == 'Empreitadas de obras publicas') -> Va =< 10000 ;
    (pertence(TC, ['Aquisicao de servicos',
    'Aquisicao de Bens Moveis','Locacao de Bens Moveis']))
    -> Va =< 5000
).
```

- **Invariante para Contratos por Consulta Prévia**

  Caso o contrato que se queira adicionar tenha "Consulta Previa", então tem-se de cumprir as normas estipuladas na secção 2.2.4.

```
+contrato(IdC, IdAd, IdAda, TC, 'Consulta Previa',
    DESC, Va, PR, L, D) ::
(
    (TC == 'Empreitadas de obras publicas') -> Va < 150000 ;
    (pertence(TC, ['Aquisicao de servicos',
    'Aquisicao de Bens Moveis', 'Locacao de Bens Moveis']))
    -> Va < 75000 ; Va < 100000
).
```

- **Invariante para Contratos por Concurso Público**

  Todos os contratos que tenham o procedimento "Concurso Publico" têm de seguir as normas definidas na secção 2.2.5. Aqui assumimos pelo povoamento feito, que o Estado possui como identificação 1.

```
+contrato(IdC, IdAd, IdAda, TC, 'Concurso Publico',
    DESC, Va, PR, L, D) ::
(
    (TC == 'Empreitadas de obras publicas') -> Va < 5350000 ;
    (pertence(TC, ['Aquisicao de servicos',
    'Aquisicao de Bens Moveis', 'Locacao de Bens Moveis']),
    IdAd == 1) -> Va < 139000 ;
    (pertence(TC, ['Aquisicao de servicos',
    'Aquisicao de Bens Moveis', 'Locacao de Bens Moveis']),
    IdAd \== 1) -> Va < 214000
).
```

- **Regra dos 3 Anos válida para todos os contratos**

  Este invariante, tal como o primeiro definido para os contratos, precisa de ser cumprido por todos os contratos. Assim, estes contratos precisam de seguir as regras da secção 2.2.3.

```
+contrato(IdC, IdAd, IdAda, TC, TP, DESC,
    Va, PR, L, data(DIns, MIns, AINS)) ::
(
    solucoes((Ano, V), contrato(I, IdAd, IdAda, TC, Tp, Desc,
        V, Prazo, Local, data(Dia, Mes, Ano)), Lista),
    c3Anos(AINS, Lista, -Va, N),
    N < 75000
).
```

3.3 Involução de conhecimento

Para além da necessidade do desenvolvimento de invariantes de evolução, também foi fulcral a implementação de invariantes de involução. Estes invariantes servem para controlar a remoção de predicados no sistema de forma a manter o conhecimento deste correto.

3.3.1 Invariante para a entidade Adjudicante

Este invariante indica que apenas é possível remover um predicado adjudicante caso ele não tenha nenhum contrato relacionado.

```
-adjudicante(Id, Nome, NIF, Morada) ::
(
    solucoes(IdC, contrato(IdC, Id, IdAda, TC, TP, DESC,
    Va, PR, L, D), Lista),
    comprimento(Lista, N),
    N == 0
).
```

3.3.2 Invariante para a entidade Adjudicatária

Este invariante é semelhante ao anterior, sendo apenas diferente no predicado aplicado.

```
-adjudicataria(Id, Nome, NIF, Morada) ::
(
    solucoes(IdC, contrato(IdC, IdAd, Id, TC, TP, DESC,
    Va, PR, L, D), Lista),
    comprimento(Lista, N),
    N == 0
).
```

3.4 Predicados auxiliares

Durante o desenvolvimento do trabalho, foram usadas várias funções/predicados com o intuito de auxiliar outras, tais como invariantes e/ou funcionalidades.

Assim, nesta secção iremos falar de alguns predicados desenvolvidos juntamente com invariantes de evolução e involução.

3.4.1 Evolução e Involução

De forma a ser possível evoluir o conhecimento do nosso sistema dinamicamente, precisamos de definir dois predicados base: *evolucao* e *involucao*. O *evolucao* representa a adição de conhecimento ao nosso sistema e o *involucao* serve para a remoção de conhecimento.

Para o predicado *evolucao* precisamos de definir outros 3 predicados: *solucoes, insercao e teste*.

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

O predicado *solucoes* deriva do predicado *findall* do **PROLOG**.
Já o predicado *insercao* e o *teste* são representados da seguinte forma:

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

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

No *insercao* se for possível ocorrer o *assert* de um termo, então é possível inseri-lo. Contudo, devido ao mecanismo do **PROLOG** de *backtracking*, a segunda cláusula apenas será feita se o predicado seguinte do *evolucao* falhar, neste caso o *teste*.

Assim, a primeira cláusula é verificada sempre que o *evolucao* é chamado e a segunda apenas se não foi possível inserir o termo por ter falhado algum invariante e, por isso mesmo, é necessário remove-lo através do *retract*. Após o retract é feito um *cut*(!) de forma a que o predicado *evolucao* retorne "no".

De notar que o predicado *teste* serve para testar uma lista de termos, ou seja, se o termo a inserir falha algum invariante.

O predicado *involucao* segue um raciocínio semelhante. Contudo, gostávamos de sublinhar que tivemos especial atenção à possibilidade da possível involução de uma entidade não inserida no conhecimento do sistema. Por isso mesmo, o *involucao* tem o predicado *Termo* que verifica se a entidade a remover do sistema já existe nele.

Caso não fosse feito tal verificação, o *involucao* iria acrescentar tal entidade ao sistema, pois o *retract* iria falhar e, por consequente, iria ser feito o *assert* desta.

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

Outra diferença entre o *involucao* e *evolucao*, é a existência do predicado *remocao*, em vez do *insercao*. De notar que estes dois são inversos.

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

3.4.2 data

Para a representação do conhecimento de datas, foi desenvolvido o predicado *data* que valida e representa a data segundo o dia, mês e ano em que o contrato foi assinado.

Apenas consideramos como datas válidas aquelas que representassem dias, meses e anos válidos (1900 a 2021). O predicado data chama outros predicados (anoV, mesV, diaV) que validam cada um dos elementos que compõem a data.

```
% Extensão do predicado data valida: Data -> {V,F}
data(Dia, Mes, Ano) :- anoV(Ano), mesV(Mes), diaV(Dia, Mes).
```

3.4.3 tipoProcedimento

O predicado *tipoProcedimento* serve para validar se o procedimento recebido é válido, ou seja, se é "Ajuste Direto", "Consulta Previa"ou "Concurso Publico".

```
% Extensão do predicado tipoProcedimento: TipoProcedimento -> {V,F}
tipoProcedimento(X) :- pertence(X, ['Ajuste Direto', 'Consulta Previa',
    'Concurso Publico']).
```

3.4.4 c3Anos

O predicado *c3Anos* serve como predicado auxiliar ao invariante do contrato relativo aos 3 anos. Neste predicado, verificamos se a lista recebida (lista de um par Ano-Valor) tem contratos no mesmo ano ou nos dois anteriores ao ano recebido como primeiro argumento (AInser).

O valor acumulado começa com o valor negativo do contrato que se pretende celebrar, visto que apenas queremos o valor dos contratos já celebrados e a lista recebida contém o valor do contrato por celebrar.

Assim, vamos acumulando o valor dos contratos no período de 3 anos pretendido e retornamos em R.

```
c3Anos(AInser, [], Acc, R) :- R is Acc.
c3Anos(AInser, [ (Ano, V)|T ], Acc, R) :-
((AInser =:= Ano; AInser =:= Ano+1; AInser =:= Ano+2)
-> H is (V+Acc) ; H is Acc),
c3Anos(AInser, T, H, R).
```

3.5 Demonstração das funcionalidades do Sistema

De forma a demonstrar as funcionalidades do nosso sistema desenvolvido, procedemos à criação de possíveis formas de representar conhecimento criando para cada uma delas, uma breve descrição do que era pretendido. Aqui iremos também demonstrar os resultados destas funcionalidades.

Nestas funcionalidades iremos demonstrar exemplos de vários tipos de conhecimento, bem como lidar com a problemática de evolução e involução do conhecimento do sistema.

De notar que todas as funcionalidades aqui especificadas vão de encontro ao que era pretendido delas.

3.5.1 Resultados obtidos nas evoluções e involuções

Iremos então demonstrar alguns exemplos de evoluções e involuções relativas aos invariantes definidos e, para cada um deles, vamos analisar o resultado.

Em primeiro lugar, é necessário demonstrar os conhecimentos já presentes no sistema até ao momento. Isto é possível de obter através do predicado *listing*. Contudo, devido ao número extenso de conhecimento por cada predicado existente, vamos proceder a apenas demonstrar alguns resultados.

```
| ?- listing(adjudicante).
adjudicante(1, 'Estado Portugues', 1, 'Portugal').
adjudicante(2, 'Municipio de Guimaraes', 505948605,
    'Portugal, Braga, Guimaraes').
..

| ?- listing(adjudicataria).
adjudicataria(1, 'SA LIMPA - SOCIEDADE DE LIMPEZAS, LDA.',
    504458086, 'Portugal').
adjudicataria(2, 'Vodafone', 506862747, 'Portugal').
..

contrato(2, 1, 8, 'Aquisicao de servicos', 'Concurso Publico',
    'Aquisicao de servicos de telecomunicacoes', 20000, 100,
    'Lisboa', data(12,5,2019)).
contrato(3, 7, 1, 'Aquisicao de bens moveis', 'Consulta Previa',
    'Aquisicao de materiais de Limpeza', 74000, 400, 'Gualtar',
    data(1,8,2019)).
..
```

- **Exemplo relativo à evolução do adjudicante (3.2.1)**

  Um exemplo de evolução de uma entidade adjudicante que não cumpre o invariante definido é o seguinte:

```
| ?- evolucao(adjudicante(1, 'Municipio de Alto', 103,
    'Portugal, Braga, Alto de Basto')).
no
```

  Através do uso *trace* disponibilizado pelo **PROLOG**, conseguimos verificar o caso em que o invariante falha.

```
    4 Exit: pertence(1,[306,11,10,9,8,7,6,5,4|...]) ?
    3 Fail: nao(pertence(1,[306,11,10,9,8,7,6,5|...])) ?
```

  Esta entidade adjudicante não é inserida ao sistema devido à existência de uma entidade adjudicante com o mesmo identificador, pelo que falha no invariante. Isto acontece de acordo com o esperado.

- **Exemplo relativo à evolução do contrato (3.2.3)**

  Para evoluir o conhecimento dos contratos, temos como o seguinte exemplo que terá de dar verdadeiro tanto ao invariante dos requisitos mínimos dos contratos, bem como ao invariante do ajuste direto e ao dos 3 anos.

```
| ?- evolucao(contrato(200, 5, 2, 'Empreitadas de obras publicas',
    'Ajuste Direto', 'Assessoria juridica', 7000, 300,
    'Alto de Basto', data(12, 05, 2009))).
yes

3 Exit: 7000>=0 ?
3 Call: 300>0 ?
3 Exit: 300>0 ?
..
4 Exit: 'Empreitadas de obras publicas'==
    'Empreitadas de obras publicas' ?
4 Call: 7000=<10000 ?
4 Exit: 7000=<10000
..
5 Exit: c3Anos(2009,[(2009,7000)],- (7000),0) ?
5 Call: 0<75000 ?
5 Exit: 0<75000 ?
```

Como observamos este contrato não falha em nenhum dos 3 invariantes, visto que segue as normas definidas em todos os invariantes. De realçar que o valor acumulado dos contratos até 3 anos já celebrados é 0, visto que não temos conhecimento de outro qualquer contrato relativo às entidades deste, como observamos no que é obtido pelo *trace*.

Assim, torna-se possível a inserção do contrato.

- **Exemplo relativo à involução da adjudicataria (3.3)**

  Neste exemplo iremos proceder à remoção de uma entidade adjudicatária do conhecimento do sistema, através do *involucao*.

```
| ?- involucao(adjudicataria(7, 'ARRIVA Portugal - Transportes Lda.',
    504426974, 'Portugal')).
yes
```

Se tentarmos fazer a involução de um conhecimento não presente no sistema, temos o seguinte resultado:

```
2 Call:adjudicataria(700000,'ARRIVA Portugal - Transportes Lda.',
    504426974,'Portugal') ?
2 Fail:adjudicataria(700000,'ARRIVA Portugal - Transportes Lda.',
    504426974,'Portugal') ?
1 Fail:involucao(adjudicataria(700000,'ARRIVA Portugal - Trans
    portes Lda.', 504426974,'Portugal')) ?
no
```

É possível verificar que a involução ocorre da forma que tinhamos planeado, pois é possível involuir conhecimento do sistema, consoante as regras definidas por nós. Também podemos observar que não é possível involuir conhecimento não presente no sistema e, ao fazer isto, não adicionamos qualquer tipo de conhecimento.

3.5.2 Conhecimento Imperfeito

Aqui iremos demonstrar alguns exemplos de representação de conhecimento imperfeito, ou seja, conhecimento derivado de informação incerta, imprecisa ou interdita.

- O Município de Braga indicou que celebrou um contrato por concurso público com uma entidade. Contudo, desconhece-se qual a entidade adjudicatária da qual foi celebrado o contrato. O contrato foi de aquisição de serviços informáticos no valor de 1.000 euros com um prazo de 50 dias. A data do contrato foi a 3 de Março, embora se desconheça o ano.

```
contrato(301, 10, ent1, 'Aquisicao de servicos', 'Concurso Publico',
    'Servicos Informaticos', 1000, 50, 'Braga', data(03, 12, ano1)).

excecao(contrato(IdC, IdAd, IdAda, TC, TP, DESC, Va, PR, L,
    data(DIns, MIns, AINS))) :-
        contrato(IdC, IdAd, ent1, TC, TP,DESC, Va, PR, L,
        data(DIns, MIns, ano1)).
```

  O resultado desta funcionalidade é o seguinte:

```
| ?- demo(contrato(301, 10, 1, 'Aquisicao de servicos',
    'Concurso Publico','Servicos Informaticos', 1000,
    50, 'Braga', data(03, 12, 2010)), R).
R = desconhecido ?
```

  Como observamos, o resultado "desconhecido" provém da informação **incerta** que a entidade adjudicatária e o ano do contrato contêm.

- A entidade adjucatária diz que celebrou um contrato com o Município de Alto de Basto por concurso público. Contudo, a imprensa afirma que foi por ajuste direto. A entidade adjudicante não confirma nem desmente nenhum dos valores.

```
excecao(contrato(302, 3, 7, 'Aquisicao de servicos', 'Concurso Publico',
    'Servicos de transporte', 4000, 100, 'Alto de Basto',
    data(06, 08, 2012))).
excecao(contrato(302, 3, 7, 'Aquisicao de servicos', 'Ajuste Direto',
    'Servicos de transporte', 4000, 100, 'Alto de Basto',
    data(06, 08, 2012))).
```

Tem-se o seguinte resultado:

```
| ?- demo(contrato(302, 3, 7, 'Aquisicao de servicos',
    'Concurso Publico','Servicos de transporte',
    4000, 100, 'Alto de Basto',
    data(06, 08, 2012)), R).
R = desconhecido ?

| ?- demo(contrato(302, 3, 7, 'Aquisicao de servicos',
    'Consulta Previa','Servicos de transporte', 4000,
    100, 'Alto de Basto',
    data(06, 08, 2012)), R).
R = falso ?
```

Como era esperado, se o procedimento do contrato for concurso público ou ajuste direto, teremos "desconhecido" como resultado. Contudo, se o tipo de contrato for outro, teremos então um "falso". Isto deve-se ao facto de estarmos a representar informação **imprecisa**.

- O papel do contrato estabelecido entre a empresa "Diversey Portugal-Sistemas de Higiene e Limpeza Unipessoal, Lda" e a "Faculdade de Engenharia da Universidade do Porto" perdeu-se, pelo que é impossível saber qual o tipo de procedimento adotado.

```
contrato(id5, 8, 5, 'Aquisicao de servicos', tp5,
    'Servicos de Limpeza', 1000, 500, 'Porto',
    data(30, 01, 1960)).

excecao(contrato(IdC, IdAd, IdAda, TC, TP, DESC, Va, PR, L, D)) :-
    contrato(id5,IdAd, IdAda, TC, tp5, DESC, Va, PR, L, D).
nulo(tp5).
+contrato(IdC, IdAd, IdAda, TC, TP, DESC, Va, PR, Local,
    data(DIns, MIns, AINS)) ::
(
    solucoes(Tp, (contrato(305, I, IA, Tc, Tp, Desc, V,
        P, L, D), nao(nulo(Tp))), Lista),
    comprimento(Lista, 0)
).
```

Neste exemplo representamos conhecimento imperfeito através de informação **interdita**. Como podemos ver no comando trace seguinte, é impossível proceder à evolução do conhecimento deste contrato, visto que dá *fail* neste mesmo invariante.

```
| ?- demo(contrato(305, 8, 5, 'Aquisicao de servicos',
    'Consulta Previa','Servicos de Limpeza', 1000, 500,
    'Porto', data(30, 01, 1960)),R).
R = desconhecido ?

| ?- evolucao(contrato(305, 8, 5, 'Aquisicao de servicos',
    'Consulta Previa','Servicos de Limpeza', 1000, 500,
    'Porto', data(30, 01, 1960))).
    8 Exit: nao(nulo('Consulta Previa')) ?
    ..
    6 Call: comprimento(['Consulta Previa'],0) ?
    6 Fail: comprimento(['Consulta Previa'],0) ?
```

3.5.3 Conhecimento positivo e negativo

Para além da representação de conhecimento imperfeito, é também possível representar conhecimento positivo e negativo neste sistema. O povoamento de conhecimento do sistema releva-se como conhecimento positivo, por isso iremos apenas demonstrar um exemplo de um conhecimento negativo.

A identificação da entidade adjudicatária "MEO" não é 1, mas sim 8.

```
| ?- demo(adjudicataria(1, 'MEO - Servicos de Comunicacao e multimedia,
    S. A.', 203755030, 'Portugal'), R).
R = falso ?

| ?- demo(adjudicataria(8, 'MEO - Servicos de Comunicacao e multimedia,
    S. A.', 203755030, 'Portugal'), R).
R = verdadeiro ?
```

Por último, gostávamos de realçar que o grupo desenvolveu mais funcionalidades e demonstrações de todos estes tipos de representação de conhecimento, que se encontram disponíveis no código do sistema desenvolvido.

Capítulo 4

Conclusão e Sugestões

Em suma, com este trabalho foi possível aplicar na prática todos os conhecimentos adquiridos nas aulas relativamente à representação de conhecimento imperfeito numa linguagem lógica (**PROLOG**). Além disso, pudemos transpor predicados sugeridos nas aulas práticas e teóricas que possibilitaram a evolução e involução do conhecimento do sistema.

Assim, achamos que cumprimos todos os objetivos propostos pela equipa docente para a realização deste trabalho de programação em lógica estendida e Conhecimento imperfeito, bem como tentamos tornar o sistema representado mais realista com o acréscimo de normas e procedimentos para além dos pedidos.

Apesar disso, consideramos que o relatório do trabalho ficou um pouco extenso devido à necessidade de descrição do trabalho por nós feito, de forma a uma melhor percepção da nossa metodologia de trabalho e dos predicados por nós definidos.

Referências

[1] Analide, C. e Neves, J. (1996) Representação de Informação Incompleta. Unidade de Ensino. Departamento de Informática. Universidade do Minho. Braga, Portugal. Novembro, 1996.

[2] Bratko I. (1986). Prolog programming for artificial intelligence. Wokingham: Addison-Wesley.

[3] IMPIC - Instituto dos Mercados Públicos, do Imobiliário e da Construção. Portal Base. `http://www.base.gov.pt/Base/pt/OPortal/Base`