Universidade do Minho

Sistemas de Representação de Conhecimento e Raciocínio

MIEI - 3º ANO - 2º SEMESTRE
UNIVERSIDADE DO MINHO

TRABALHO PRÁTICO

Grupo 19

André Sousa
A74813

Bernardo Viseu
A74618

Marco Gonçalves
A75480

Renato Cruzinha
A75310

Hugo Moreira
A43148

Braga, 3 de Maio de 2020

Resumo

O presente documento diz respeito ao trabalho prático proposto na unidade curricular de Sistemas de Representação de Conhecimento e Raciocınio da Universidade do Minho.

Este trabalho tem o objetivo de motivar para a utilização da extensão à programação em lógica, usando a linguagem de programação em lógica PROLOG.

Ao longo do presente relatório serão apresentadas as formas de conhecimento adotadas e as caracterısticas e funcionalidades do sistema, assim como também serão descritas as estratégias utilizadas para a concretização das mesmas.

De uma perspetiva geral, consideramos que a realização deste exercício foi relativamente bem sucedida, visto que pensamos ter cumprido com todos os requisitos mínimos propostos.

Conteúdo

1. Introdução

No âmbito da disciplina de Sistemas de Representação de Conhecimento e Raciocínio foi-nos proposta, a demonstração de funcionalidades subjacentes à programação em lógica estendida e à representação de conhecimento, num sistema com capacidade para caracterizar um universo de discurso, área da contratação pública para a realização contratos para a prestação de serviços.

O sistema de representação de conhecimento e raciocínio desenvolvido deverá permitir:

- Representar conhecimento positivo e negativo;
- Representar casos de conhecimento imperfeito, pela utilização de valores nulos de todos os tipos estudados;
- Manipular invariantes que designem restrições à inserção e à remoção de conhecimento do sistema;
- Lidar com a problemática da evolução do conhecimento, criando os procedimentos adequados;
- Desenvolver um sistema de inferência capaz de implementar os mecanismos de raciocínio inerentes a estes sistemas.

Este trabalho tem como objetivo motivar para a utilização da extensão à programção em lógica, usando a linguagem de programção em lógica PROLOG.

2. Preliminares

Este sistema tem o intuito de representar um universo de discurso na área da contratação pública para a realização de contratos para a prestação de serviços, contendo três fontes de conhecimento. Como interveniente temos o adjudicante, que está associado um Id, Nome, Número de identificação fiscal e Morada, o qual pode celebrar um contrato com uma adjudicatária, que a par do adjudicante também contém um Id, Nome, Número de identificação fiscal e uma morada. O contrato celebrado entre estas duas entidades é caraterizado por ter um Id, o Id do adjudicante, o Id da adjudicatária, o tipo de contrato, o tipo de processo, a descrição do contrato, o seu valor de assinatura, o seu prazo de término, o seu local de celebração e a data da mesma. É de notar que, por razões óbvias, todos os Ids são de carater único. Posto isto encontramos-nos aptos a prosseguir com a elaboração do trabalho em si.

3. Descrição do Trabalho e Análise de Resultados

3.1 Descrição do Sistema

Para o desenvolvimento do 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 serviços, foi considerado a seguinte representação:

- $Adjudicante : \#Id, Nome, Nif, Morada \rightarrow \{V, F, D\}$
- $Adjudicatria : \#Id, Nome, Nif, Morada \rightarrow \{V, F, D\}$
- $Contrato : \#Id, \#IdAdjudicante, \#IdAdjudicataria, TipoContrato, TipoProcesso, Descricao, Custo, Prazo, Local, \#IdData \rightarrow \{V, F, D\}$
- $Data : \#Id, Ano, Mes, Dia \rightarrow \{V, F, D\}$

```
:- dynamic adjudicante/4.
:- dynamic adjudicataria/4.
:- dynamic contrato/10.
:- dynamic data/4.
```

3.2 Predicados

De seguida iremos expôr os predicados usados neste trabalho.

3.2.1 Predicado Adjudicante/4

O predicado adjudicante tem como finalidade caraterizar uma entidade pública que celebra um contrato público para a prestação de serviços. Esta entidade é caraterizada na base de conhecimento através de um id, um nome, número de identificação fiscal e uma morada. Para tal utilizamos os seguintes factos:

```
adjudicante(1, cmb, 123456789, braga).
adjudicante(2, cmvm, 325824562, vieira_minho).
adjudicante(3, cmvl, 858617833, vila_real).
adjudicante(4, cmpv, 389646360, povoa_varzim).
adjudicante(5, cmpl, 809032071, povoa_lanhoso).
adjudicante(6, cmf, 928803358, fafe).
adjudicante(7, cmg, 261109115, guimaraes).
adjudicante(8, cmp, 974841598, porto).
adjudicante(9, cmtb, 919338755, terras_bouro).
adjudicante(10, cmvc, 369641730, vila_conde).
```

Após consultar o ficheiro relativo a este exercício no SICStus Prolog, podemos consultar o conhecimento adquirido acerca de adjudicante/4, recorrendo ao predicado:

`listing(adjudicante).`

```
| ?- listing(adjudicante).
adjudicante(1, cmb, 123456789, braga).
adjudicante(2, cmvm, 325824562, vieira_minho).
adjudicante(3, cmvl, 858617833, vila_real).
adjudicante(4, cmpv, 389646360, povoa_varzim).
adjudicante(5, cmpl, 809032071, povoa_lanhoso).
adjudicante(6, cmf, 928803358, fafe).
adjudicante(7, cmg, 261109115, guimaraes).
adjudicante(8, cmp, 974841598, porto).
adjudicante(9, cmtb, 919338755, terras_bouro).
adjudicante(10, cmvc, 369641730, vila_conde).
adjudicante(16, cml, sem_nif1, lisboa).
adjudicante(17, cmpl, 854894729, sem_morada1).

yes
```

3.2.2 Predicado Adjudicataria/4

O predicado adjudicatária tem como finalidade caraterizar uma entidade (pública ou privada) que corresponde à entidade com quem a entidade adjudicatária irá celebrar um contrato público. Esta entidade é caraterizada na base de conhecimento através de um id, um nome, número de identificação fiscal e uma morada. Para tal utilizamos os seguintes factos:

```
adjudicataria(1, aaum, 420123954, braga).
adjudicataria(2, bvvm, 420123954, vieira_minho).
adjudicataria(3, hpb, 420123954, braga).
adjudicataria(4, jeor, 420123954, povoa_varzim).
adjudicataria(5, vsc, 420123954, guimaraes).
adjudicataria(6, scb, 420123954, braga).
adjudicataria(7, aevl, 420123954, vila_real).
adjudicataria(8, pcne, 420123954, braga).
adjudicataria(9, drpus, 420123954, braga).
adjudicataria(10, mfc, 420123954, melgaco).
```

Após consultar o ficheiro relativo a este exercício no SICStus Prolog, podemos consultar o conhecimento adquirido acerca de adjudicataria/4, recorrendo ao predicado:

`listing(adjudicataria).`

```
| ?- listing(adjudicataria).
adjudicataria(1, aaum, 420123954, braga).
adjudicataria(2, bvvm, 420123954, vieira_minho).
adjudicataria(3, hpb, 420123954, braga).
adjudicataria(4, jeor, 420123954, povoa_varzim).
adjudicataria(5, vsc, 420123954, guimaraes).
adjudicataria(6, scb, 420123954, braga).
adjudicataria(7, aevl, 420123954, vila_real).
adjudicataria(8, pcne, 420123954, braga).
adjudicataria(9, drpus, 420123954, braga).
adjudicataria(10, mfc, 420123954, melgaco).
adjudicataria(16, ipdj, sem_nif2, lisboa).
adjudicataria(17, gdpeoes, 274917053, sem_morada2).

yes
```

3.2.3 Predicado Contrato/10

O predicado contrato tem como finalidade caraterizar um instrumento dado à administração pública utilizado para adquirir bens ou serviços a particulares. Esta entidade é caraterizada na base de conhecimento através de um id, o id do adjudicante, o id do adjudicatário, o tipo de contrato, o tipo de procedimento, a descrição do contrato, o seu custo, o prazo de término, o seu local de celebração e um id de uma data. Para tal utilizamos os seguintes factos:

```
contrato(1, 1, 1, aquisicao_servico, consulta_previa, assessoria, 13599, 547, braga, 1).
contrato(2, 5, 2, aquisicao_bens, ajuste_direto, assessoria, 1982, 53, viera_minho, 2).
contrato(3, 8, 6, locacao_bens, consulta_previa, assessoria, 13599, 547, povoa_varzim, 3).
contrato(4, 3, 4, aquisicao_servico, concurso_publico, assessoria, 13599, 547, vila_real, 4).
contrato(5, 9, 8, aquisicao_bens, consulta_previa, assessoria, 13599, 133, braga, 5).
contrato(6, 10,1, locacao_bens, concurso_publico, assessoria, 13599, 547, braga, 6).
contrato(7, 2, 2, aquisicao_bens, consulta_previa, assessoria, 13599, 547, lisboa, 7).
contrato(8, 6, 9, aquisicao_servico, concurso_publico, assessoria, 13599, 547, braga, 8).
contrato(9, 8, 3, aquisicao_bens, ajuste_direto, assessoria, 1359, 105, coimbra, 9).
contrato(10, 1, 1, locacao_bens, concurso_publico, assessoria, 13599, 547, braga, 10).
```

Após consultar o ficheiro relativo a este exercício no SICStus Prolog, podemos consultar o conhecimento adquirido acerca de adjudicataria/4, recorrendo ao predicado:

`listing(contrato).`

```
| ?- listing(contrato).
contrato(1, 1, 1, aquisicao_servico, consulta_previa, assessoria, 13599, 547, braga, 1).
contrato(2, 5, 2, aquisicao_bens, ajuste_direto, assessoria, 1982, 53, viera_minho, 2).
contrato(3, 8, 6, locacao_bens, consulta_previa, assessoria, 13599, 547, povoa_varzim, 3).
contrato(4, 3, 4, aquisicao_servico, concurso_publico, assessoria, 13599, 547, vila_real, 4).
contrato(5, 9, 8, aquisicao_bens, consulta_previa, assessoria, 13599, 133, braga, 5).
contrato(6, 10, 1, locacao_bens, concurso_publico, assessoria, 13599, 547, braga, 6).
contrato(7, 2, 2, aquisicao_bens, consulta_previa, assessoria, 13599, 547, lisboa, 7).
contrato(8, 6, 9, aquisicao_servico, concurso_publico, assessoria, 13599, 547, braga, 8).
contrato(9, 8, 3, aquisicao_bens, ajuste_direto, assessoria, 1359, 105, coimbra, 9).
contrato(10, 1, 1, locacao_bens, concurso_publico, assessoria, 13599, 547, braga, 10).
contrato(11, 3, 5, aquisicao_servico, consulta_previa, desc1, 4932, 360, braga, 10).

yes
```

3.2.4 Predicado Data/4

O predicado data tem como finalidade caraterizar uma data. Esta entidade é caraterizada na base de conhecimento através de um id, o ano, o mês e o dia. Para tal utilizamos os seguintes factos:

```
data(1,2020,1,1).
data(2,2020,1,1).
data(3,2020,1,1).
data(4,2020,1,2).
data(5,2020,1,2).
data(6,2020,1,3).
data(7,2020,1,3).
data(8,2020,1,3).
data(9,2020,1,4).
data(10,2020,1,4).
data(11,2020,1,5).
data(12,2020,2,5).
data(13,2020,2,6).
data(14,2020,2,6).
data(15,2020,2,7).
data(16,2020,1,2).
data(17,2020,3,5).
```

Após consultar o ficheiro relativo a este exercício no SICStus Prolog, podemos consultar o conhecimento adquirido acerca de adjudicantaria/4, recorrendo ao predicado:

`listing(data).`

```
| ?- listing(data).
data(1, 2020, 1, 1).
data(2, 2020, 1, 1).
data(3, 2020, 1, 1).
data(4, 2020, 1, 2).
data(5, 2020, 1, 2).
data(6, 2020, 1, 3).
data(7, 2020, 1, 3).
data(8, 2020, 1, 3).
data(9, 2020, 1, 4).
data(10, 2020, 1, 4).
data(11, 2020, 1, 5).
data(12, 2020, 2, 5).
data(13, 2020, 2, 6).
data(14, 2020, 2, 6).
data(15, 2020, 2, 7).
data(16, 2020, 1, 2).
data(17, 2020, 3, 5).

yes
```

3.2.5 Predicado Evolucao/1

O predicado evolucao tem como finalidade adicionar conhecimento à base de conhecimento. A extensão deste predicado é a seguinte:

```
evolucao(T) :-
    solucoes(I, +T :: I, S),
    insere(T),
    testa(S).
```

Se efetuarmos uma tentativa de introduzir conhecimento que já se encontra atribuído na base de conhecimento, este predicado não deixa executar, tendo em conta os invariantes de inserção:

```
| ?- evolucao(adjudicante(1,aaum,98123742,braga)).
no
```

3.2.6 Predicado Involucao/1

O predicado involucao tem como finalidade remover conhecimento à base de conhecimento. A extensão deste predicado é a seguinte:

```
involucao(T) :-
    solucoes(I, -T :: I, S),
    remove(T),
    testa(S).
```

Este predicado têm em atenção os invariantes definidos, como por exemplo, se tentarmos remover um adjudicante que tem um contrato celebrado, tal não será possível:

```
| ?- involucao(adjudicante(2,Nome,Nif,Morada)).
no
```

3.3 Conhecimento Negativo

Após apresentarmos o conhecimento positivo, iremos de seguida, apresentar o conhecimento negativo.

A representação de conhecimento negativo é realizada com base na negação forte, indicando que um facto é considerado falso caso não exista na base de conhecimento e não possua uma exceção a si associada. Isto é, a informação é falsa, caso não seja verdadeira nem desconhecida.

O meta-predicado responsável por verificar a existência de uma determinada questão na base de conhecimento é o nao, focando-se na negação por falha na prova. Caso uma questão não se encontre na base de conhecimento o predicado indica verdadeiro (Pressuposto do Mundo Fechado).

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

Como o meta-predicado que representa a negação por falha na prova é o nao e o que indica a presença de conhecimento imperfeito é o excecao, estes foram utilizados como condições da clausula abaixo.

```
-adjudicante(IdA,N,Nif,M) :- nao(adjudicante(IdA,N,Nif,M)), nao(excecao(adjudicante(IdA,N,Nif,M))).
```

A clausula acima representa conhecimento negativo para o predicado adjudicante. Para os restantes predicados utilizou-se o mesmo raciocinio.

3.4 Representação de casos de conhecimento imperfeito

Neste trabalho representou-se conhecimento imperfeito através da utilização de três tipos de valores nulos:

- Incerto;
- Impreciso;
- Interdito.

O conhecimento imperfeito, tal como foi dito anteriormente, não se trata de conhecimento verdadeiro nem falso, mas sim de conhecimento desconhecido. Para representar esse tipo de conhecimento foi criado o meta-predicado excecao. Uma questão que esteja associada ao meta-predicado excecao representa conhecimento imperfeito existente na base de conhecimento.

Posto isto, um caso de conhecimento imperfeito possuirá a si associado um valor nulo ou uma excecao, mediante o caso que for.

Outro fator importante e que, para se saber o tipo de conhecimento representado por uma questão, deve-se recorrer a um meta-predicado que indique se se trata de conhecimento verdadeiro, falso ou desconhecido. Com esta finalidade foi desenvolvido o meta-predicado demo, utilizado para verificar o tipo de conhecimento de uma questão.

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

Nas proximas três secções serão apresentados e explicados os diferentes casos de conhecimento imperfeito, bem como os invariantes usados para manipular esse mesmo conhecimento.

3.4.1 Conhecimento Incerto

No caso do conhecimento incerto, não se possui qualquer informaçao acerca da informação desconhecida. Sendo assim, e tal como podemos ver abaixo, o parâmetro desconhecido será caracterizado por um valor nulo, que neste caso corresponde a sem_nif1. Para além disso, a este valor nulo é associada uma excecao para que o conhecimento consigo relacionado seja representado como desconhecido.

De seguida encontra-se um exemplo de conhecimento incerto representado para o predicado adjudicante.

```
adjudicante(16, cml , sem_nif1 , lisboa).

excecao(adjudicante(ID,N,Nif,M)) :-
    adjudicante(ID,N,sem_nif1,M).
```

O facto e a clausula em cima indicam que para o adjudicante cml, se desconhece qualquer informaçao acerca do seu número de identificação fiscal. Dado isto, tal como podemos ver na figura abaixo, se recorrermos ao meta-predicado demo, este irá indicar a presença de conhecimento desconhecido, qualquer que seja o nif questionado.

```
| ?- demo(adjudicante(16, cml , 123456789, lisboa),R).
R = desconhecido ?
yes
| ?- demo(adjudicante(16, cml , 987654321, lisboa),R).
R = desconhecido ?
yes
```

À semelhança deste caso, foi também representado conhecimento incerto para os predicados adjudicataria e contrato.

3.4.2 Conhecimento Impreciso

O segundo caso caracteriza-se por representar conhecimento impreciso, isto é, conhecimento que, tal como o próprio nome indica, é desconhecido dentro de um conjunto de valores. Posto isto, deve-se representar como exceções os casos que se enquadram nesse conjunto de valores. Qualquer valor fora do conjunto é considerado conhecimento falso, caso não se encontre na base de conhecimento.

A imprecisão do conjunto de valores pode tanto ser restrita a valores pontuais, como a um conjunto de valores limitado dentro de um determinado dom ınio. Caso sejam casos pontuais, estes podem ser representados recorrendo a um facto, caso contrário devem ser representados por uma cláusula.

Vejam-se os dois casos de conhecimento impreciso representados abaixo.

```
excecao(adjudicante(20, cmg , 749285627 , guimaraes)).
excecao(adjudicante(20, cmg , 749285627 , gualtar)).
```

```
excecao(adjudicataria(20, gdpfel , I , felgueiras)) :-
I >= 123456789,
I =< 987654321.
```

No primeiro exemplo sabemos que o adjudicante tem morada em guimaraes ou em gualtar, e no segundo exemplo sabemos que a adjudicataria tem um nif variado entre 123456789 e 987654321, então representamos esse conhecimento impreciso dessa maneira.

A figura abaixo mostra um exemplo de utilização desse conhecimento através do SICSTUS.

```
| ?- demo(adjudicataria(20, gdpfel , 000105000, felgueiras),R).
R = falso ?
yes
| ?- demo(adjudicataria(20, gdpfel , 234567890, felgueiras),R).
R = desconhecido ?
yes
| ?-
```

3.4.3 Conhecimento Interdito

Por fim, o ultimo tipo de conhecimento imperfeito caracteriza-se por interdito, isto porque este tipo de conhecimento não deve ser especificado nem conhecido. Para indicar esta ”proibição” é necessário atribuir um valor nulo ao conhecimento que pretendemos ”bloquear”.

Tal como foi dito, não deve ser permitido incluir informação acerca do conhecimento classificado como interdito. Posto isto, foi criado o meta-predicado nuloInterdito que caracteriza os valores de nulo interdito.

Este meta-predicado deve ser utilizado num invariante que impeça a inserção de conhecimento interdito. Para além disto, este conhecimento não deixa de ser desconhecido e, como tal, é necessário associar-lhe uma excecao.

Para representar este tipo de conhecimento, criou-se o seguinte exemplo:

```
contrato(17,2,7,aquisicao_servico,consulta_previa,assessoria,sem_custo1,320,ponte_barca,13).

excecao(contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData)) :-
contrato(Id,IdA,IdAda,Tipo,Proc,Desc,sem_custo1,Prazo,Local,IdData).

nuloInterdito(sem_custo1).
```

Como se pretende que o valor do custo do contrato 17 seja interdito, foi necessário atribuir-lhe o valor nulo sem_custo1 ao parâmetro correspondente ao custo, para que não se pudesse representar ou conhecer essa informação.

Em adição a isto, foi necessário criar um invariante de inserção, que não permitisse a inclusão de factos na base de conhecimento que indicassem qual o custo deste contrato.

Este invariante será analisado na secção seguinte.

```
| ?- listing(contrato).
contrato(1, 1, 1, aquisicao_servico, consulta_previa, assessoria, 13599, 547, braga, 1).
contrato(2, 5, 2, aquisicao_bens, ajuste_direto, assessoria, 1982, 53, viera_minho, 2).
contrato(3, 8, 6, locacao_bens, consulta_previa, assessoria, 13599, 547, povoa_varzim, 3).
contrato(4, 3, 4, aquisicao_servico, concurso_publico, assessoria, 13599, 547, vila_real, 4).
contrato(5, 9, 8, aquisicao_bens, consulta_previa, assessoria, 13599, 133, braga, 5).
contrato(6, 10, 1, locacao_bens, concurso_publico, assessoria, 13599, 547, braga, 6).
contrato(7, 2, 2, aquisicao_bens, consulta_previa, assessoria, 13599, 547, lisboa, 7).
contrato(8, 6, 9, aquisicao_servico, concurso_publico, assessoria, 13599, 547, braga, 8).
contrato(9, 8, 3, aquisicao_bens, ajuste_direto, assessoria, 1359, 105, coimbra, 9).
contrato(10, 1, 1, locacao_bens, concurso_publico, assessoria, 13599, 547, braga, 10).
contrato(11, 3, 5, aquisicao_servico, consulta_previa, desc1, 4932, 360, braga, 10).
contrato(17, 2, 7, aquisicao_servico, consulta_previa, assessoria, sem_custo1, 320, ponte_barca, 13).

yes
| ?- evolucao(contrato(17, 2, 7, aquisicao_servico, consulta_previa, assessoria, 88, 320, ponte_barca, 13)).
no
| ?- evolucao(contrato(18,5,8,aquisicao_servico,ajuste_direto,assessoria,88,123,ponte_barca,15)).
yes
| ?- listing(contrato).
contrato(1, 1, 1, aquisicao_servico, consulta_previa, assessoria, 13599, 547, braga, 1).
contrato(2, 5, 2, aquisicao_bens, ajuste_direto, assessoria, 1982, 53, viera_minho, 2).
contrato(3, 8, 6, locacao_bens, consulta_previa, assessoria, 13599, 547, povoa_varzim, 3).
contrato(4, 3, 4, aquisicao_servico, concurso_publico, assessoria, 13599, 547, vila_real, 4).
contrato(5, 9, 8, aquisicao_bens, consulta_previa, assessoria, 13599, 133, braga, 5).
contrato(6, 10, 1, locacao_bens, concurso_publico, assessoria, 13599, 547, braga, 6).
contrato(7, 2, 2, aquisicao_bens, consulta_previa, assessoria, 13599, 547, lisboa, 7).
contrato(8, 6, 9, aquisicao_servico, concurso_publico, assessoria, 13599, 547, braga, 8).
contrato(9, 8, 3, aquisicao_bens, ajuste_direto, assessoria, 1359, 105, coimbra, 9).
contrato(10, 1, 1, locacao_bens, concurso_publico, assessoria, 13599, 547, braga, 10).
contrato(11, 3, 5, aquisicao_servico, consulta_previa, desc1, 4932, 360, braga, 10).
contrato(17, 2, 7, aquisicao_servico, consulta_previa, assessoria, sem_custo1, 320, ponte_barca, 13).
contrato(18, 5, 8, aquisicao_servico, ajuste_direto, assessoria, 88, 123, ponte_barca, 15).

yes
```

3.5 Invariantes

Ao construir invariantes, existem três assuntos principais a ter em conta, a estrutura do conhecimento, a referência a informação contida na base de conhecimento e a manipulação de conhecimento.

Os invariantes são utilizados para manutenção da verdade, tanto na inserção como na remoção de conhecimento.

Nesta secção será apresentada a construção dos diferentes invariantes estruturais e referenciais utilizados neste projeto.

3.5.1 Invariantes Estruturais

Os quatro invariantes estruturais de inserção que se apresentam de seguida permitem que seja adicionado à base de conhecimento informação cujo número de identificação seja único, isto é, cujo número não exista na base de conhecimento antes de este ser inserido. Logo, a quantidade de adjudicantes/adjudicatarias/contratos/datas com esse número de identificação tem de ser menor ou igual a 1.

Não podem ser adicionados adjudicantes com o mesmo id

```
+adjudicante(IdA,Nome,Nif,Morada)::((
    solucoes(IdA, adjudicante(IdA,A,B,C), Z),
    comprimento(Z,N),
    N=<1)).
```

Como podemos ver a seguir, se tentarmos adicionar um adjudicante que já tem um id atribuido, esse conhecimento não será adicionado à base de conhecimento.

```
| ?- evolucao(adjudicante(1,aaum,98123742,braga)).
no
```

Não podem ser adicionados adjudicantarias com o mesmo id

```
+adjudicataria(IdAda,Nome,Nif,Morada)::((
    solucoes(IdAda, adjudicataria(IdAda,A,B,C), Z),
    comprimento(Z,N),
    N=<1)).
```

Como podemos ver a seguir, se tentarmos adicionar um adjudicantarias que já tem um id atribuido, esse conhecimento não será adicionado à base de conhecimento.

```
| ?- evolucao(adjudicataria(1,aaum,982183918,braga)).
no
```

Não podem ser adicionados contratos com o mesmo id

```
+contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,Data) :: (
    solucoes(Id,(contrato(Id,A,B,C,D,E,F,G,H,I)),Z),
    comprimento(Z,N),
    N =< 1).
```

Como podemos ver a seguir, se tentarmos adicionar um contrato que já tem um id atribuido, esse conhecimento não será adicionado à base de conhecimento.

```
| ?- evolucao(contrato(1,3,4,aquisicao_servico,consulta_previa,assessoria, 293,105,braga,1)).
no
```

Não podem ser adicionados datas com o mesmo id

```
+data(IdD,_,_,_) :: (
    solucoes(data(IdD,Ano,Mes,Dia), data(IdD,_,_,_), R),
    comprimento(R,N),
    N ==1).
```

Como podemos ver a seguir, se tentarmos adicionar uma data que já tem um id atribuido, esse conhecimento não será adicionado à base de conhecimento.

```
| ?- evolucao(data(1,2020,4,20)).
no
```

3.5.2 Invariantes Referenciais

Os invariantes referencias de remoção apenas permitem remover adjudicantes/adjudicatarias se estes não tiverem nenhum contrato celebrado.

Não podem ser removidos adjudicantes associados a contratos

```
-adjudicante(IdA,Nome,Nif,Morada) :: (
    solucoes((IdA),(contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,Data)),S),
    comprimento(S,N),
    N == 0).
```

Como podemos ver a seguir, se tentarmos remover um adjudicante que tem um contrato celebrado, esse conhecimento não será removido da base de conhecimento.

```
| ?- involucao(adjudicante(1, cmb, 123456789, braga)).
no
```

Não podem ser removidos adjudicatarias associados a contratos

```
-adjudicataria(IdAda,Nome,Nif,Morada) :: (
    solucoes((IdAda),(contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,Data)),S),
    comprimento(S,N),
    N == 0).
```

Como podemos ver a seguir, se tentarmos remover uma adjudicataria que tem um contrato celebrado, esse conhecimento não será removido da base de conhecimento.

```
| ?- involucao(adjudicataria(1, aaum, 420123954, braga)).
no
```

Não podem ser adicionados contratos em que o adjudicante não existe

```
+contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,Data) :: (
    solucoes(IdA,(adjudicante(IdA,Nome,Nif,Morada)),S),
    comprimento(S,N),
    N == 1).
```

Como podemos ver a seguir, se tentarmos adicionar um contrato com um id que não corresponda a nenhum adjudicante, esse conhecimento não será adicionado à base de conhecimento.

```
| ?- evolucao(contrato(15,25,3,aquisicao_servico,consulta_previa,assessoria, 293,105,braga,1)).
no
```

Não podem ser adicionados contratos em que a adjudicataria não existe

```
+contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,Data) :: (
    solucoes(IdAda,(adjudicataria(IdAda,Nome,Nif,Morada)),S),
    comprimento(S,N),
    N == 1).
```

Como podemos ver a seguir, se tentarmos adicionar um contrato com um id que não corresponda a nenhuma adjudicataria, esse conhecimento não será adicionado à base de conhecimento.

```
| ?- evolucao(contrato(15,3,45,aquisicao_servico,consulta_previa,assessoria, 293,105,braga,1)).
no
```

3.5.3 Manipulação de Invariantes

Com o objetivo de impedir a inconsistência da base de conhecimento, provocada pela inserção ou remoção de conhecimento, foram criados vários tipos de invariantes, os quais serão apresentados de seguida.

Invariantes relativos a conhecimento nulo interdito

Invariante para o exemplo de conhecimento interdito apresentado anteriormente.

```
+contrato(Id,IdA,IdAda,Tipo,Proc,Desc,C,Prazo,Local,IdData) :: (
    solucoes(Custo,(contrato(17,2,7,aquisicao_servico,consulta_previa,assessoria,Custo,320,ponte_barca,13),
    nao(nuloInterdito(Custo))),S),
    comprimento(S,N),
    N==0).
```

Este invariante impede a inserção de informação relativa ao custo do contrato 17.

Invariante que impede a inserção de conhecimento negativo repetido

Embora possa existir mais do que um facto de conhecimento negativo associado a um id, não pode existir conhecimento negativo repetido, isto é, o mesmo facto não pode existir mais do que uma vez. Para este efeito foi criado o invariante apresentado de seguida.

```
+(-Q) :: (solucoes(Q, clause(-Q, true), S),
          comprimento(S,N),
          N =< 1).
```

Invariantes que impedem contradições

Após serem efetuadas as atualizações necessárias na base de conhecimento, é necessário certificarmo-nos de que o conhecimento inserido não gerou uma contradições, isto é, que não passaram a existir dois tipos de conhecimento associados à mesma informação.

Para este efeito foram criados os invariantes apresentados de seguida.

Invariantes que não permitam a existência do mesmo conhecimento negativo e positivo em simultaneo:

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

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

Invariante que não permite a existência de exatamente o mesmo conhecimento negativo e desconhecido em simultâneo:

```
+(-Q) :: (solucoes(Q,clause(excecao(Q), true),S),
                    comprimento(S,N),
                    N == 0).
```

3.6 Atualização de Conhecimento

3.6.1 Atualização de conhecimento positivo

Embora não possa existir conhecimento positivo repetido para um mesmo identificador, esse conhecimento deve poder ser alterado de forma a coincidir com a realidade.

Posto isto, quando se pretender inserir informação positiva, é necessário eliminar da base de conhecimento a desatualizada e inserir o novo conhecimento. Seguindo este raciocínio foi criada a cláusula abaixo.

```
atualizar(adjudicante(IdA,N,Nif,M)):-
    nao(adjudicante(IdA,N,Nif,M)),
    nao(excecao(adjudicante(IdA,N,Nif,M))),
    solucoes((adjudicante(IdA,_,_,_)),
             (adjudicante(IdA,_,_,_)),
             R),
    elimina(R),
    insere(adjudicante(IdA,N,Nif,M)).
```

A par desta cláusula foram também criadas cláusulas idênticas para os predicados de adjudicataria, contrato e data.

3.6.2 Atualizar conhecimento incerto/impreciso para conhecimento positivo

Embora esteja registado na base de conhecimento o desconhecimento acerca de determinado predicado, isto é, apesar de existirem exceções relativas a essa informação, esta pode ser atualizada. Sendo assim, considerou-se que qualquer conhecimento positivo inserido acerca de conhecimento incerto ou impreciso seria correto, logo, deveria inserido na base de conhecimento e as exceções deveriam ser eliminadas.

Posto isto, ao inserir conhecimento positivo relativo a conhecimento imperfeito, devem-se determinar as clausulas de conhecimento imperfeito relativas àquela informação, recorrendo aos meta-predicados clausImperfeito e solucoes, e eliminá-las da base de conhecimento. Desta forma o conhecimento inserido deixará de ser imperfeito e passar a a ser positivo, tal como pretendemos. Este raciocınio foi aplicado na clausula abaixo.

```
atualizar(Q):-
    demo(Q, desconhecido),
    solucoes(C,
             (clausImperfeito(Q, C)),
             R),
    elimina(R),
    insere(Q).
```

3.6.3 Inserir conhecimento negativo relativo a conhecimento incerto/impreciso

No que toca a conhecimento incerto, este pode ser atualizado com conhecimento negativo sendo que não é necessario realizar qualquer tipo de alteração. O mesmo acontece para o conhecimento impreciso especificado por um conjunto especıfico de valores. No entanto, quando se trata de conhecimento impreciso definido por valores pontuais, pode acontecer de a informação negativa que estamos a inserir se encontre expressa como informação desconhecida. Isto é, pode acontecer de o facto que pretendemos inserir como negativo, já se encontrar especificado por uma excecao. Neste caso é necessario eliminar essa exceção, visto que caso não o fizéssemos a base de conhecimento deixaria de estar consistente. Posto isto, para considerar o caso mencionado foi criada a clausula apresentada de seguida.

```
atualizar(-Q):-
    clause(excecao(Q), true),
    remove(excecao(Q)),
    insere(-Q).
```

3.6.4 Atualizar conhecimento negativo para conhecimento positivo

Outro dos aspetos tidos em conta na problemática da evolução do conhecimento é que, embora possa existir informação negativa na base de conhecimento, essa informação pode ser considerada positiva a qualquer momento. Isto significa que é possível alterar-se conhecimento negativo para conhecimento positivo, desde que o negativo seja removido da base de conhecimento, de forma a não gerar contradições. Para representar este caso foi criada a clausula abaixo.

```
atualizar(Q):-
    clause(-Q,true),
    remove(-Q),
    insere(Q).
```

3.6.5 Atualizar conhecimento positivo para conhecimento negativo

Da mesma forma que conhecimento negativo pode ser transformado em conhecimento positivo, também conhecimento positivo pode ser negado a qualquer momento, desde que a informação desatualizada seja eliminada da base de conhecimento. É importante referir que caso o facto a negar coincida com um facto existente na base de conhecimento relativo a conhecimento incerto, este não deve poder ser negado. Tendo isto em conta, foi criada a clausula abaixo.

```
atualizar(-Q):-
    solucoes(Q, excecao(Q), S),
    comprimento(S,N),
    N == 0,
    clause(Q,true),
    remove(Q),
    insere(-Q).
```

3.6.6 Inserir novo conhecimento positivo ou negativo

Caso a inserção de conhecimento perfeito não seja relativa a informação já existente na base de conhecimento, então não é necessário atualizar a base de conhecimento. Desta forma a única tarefa necessária será a verificação dos invariantes, que é feita pelo meta-predicado evolucao. Posto isto, a clausula do meta-predicado atualizar limita-se a inserir a informação na base de conhecimento para que esta seja testada, tal como podemos ver de seguida.

```
atualizar(Q):-
    insere(Q).
```

3.7 Funcionalidades Extras

Para que a nossa base de conhecimentos seja de maior facilidade na interação com o utilizador, foram criadas algumas funções que dinamizam o seu uso, e também foram criadas funções que podem dar umas informações extras caso o utilizador pretenda um pouco mais de informação.

Regista Adjudicante

Predicado que facilita o registo de um adjudicante.

```
registaAdjudicante(IdA,N,Nif,M) :-
    evolucao(adjudicante(IdA,N,Nif,M)).
```

Regista Adjudicataria

Predicado que facilita o registo de uma adjudicataria.

```
registaAdjudicataria(IdAda,Nome,Nif,Morada) :-
    evolucao(adjudicataria(IdAda,Nome,Nif,Morada)).
```

Regista Contrato

Predicado que facilita o registo de um contrato.

```
registaContrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData) :-
    evolucao(contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData)).
```

Remove Adjudicante

Predicado que facilita a remoção de um adjudicante, apenas pela introdução do seu Id.

```
removeAdjudicante(ID) :-
    involucao(adjudicante(ID,_,_,_)).
```

Remove Adjudicataria

Predicado que facilita a remoção de uma adjudicataria, apenas pela introdução do seu Id.

```
removeAdjudicataria(ID) :-
    involucao(adjudicataria(ID,_,_,_)).
```

Remove Contrato

Predicado que facilita a remoção de um contrato, apenas pela introdução do seu Id.

```
removeContrato(ID) :-
    involucao(contrato(ID,_,_,_,_,_,_,_,_,_)).
```

Identificar Adjudicantes

Predicado que facilita a seleção de adjudicantes, através dos seus diversos atributos.

```
adjudicanteID(IdA,R) :- solucoes(adjudicante(IdA,N,Nif,M), adjudicante(IdA,N,Nif,M), [R|_]).
adjudicanteNome(N,R) :- solucoes((IdA,N,Nif,M), adjudicante(IdA,N,Nif,M), R).
adjudicanteIdade(Nif,R) :- solucoes((IdA,N,Nif,M),adjudicante(IdA,N,Nif,M),R).
adjudicanteMor(M,R) :- solucoes((IdA,N,Nif,M),adjudicante(IdA,N,Nif,M),R).
```

Identificar Adjudicatarias

Predicado que facilita a seleção de adjudicatarias, através dos seus diversos atributos.

```
adjudicatariaID(IdAda,R) :- solucoes(adjudicataria(IdAda,N,Nif,M), adjudicataria(IdAda,N,Nif,M), [R|_]).
adjudicatariaNome(N,R) :- solucoes((IdAda,N,Nif,M), adjudicataria(IdAda,N,Nif,M), R).
adjudicatariaIdade(Nif,R) :- solucoes((IdAda,N,Nif,M),adjudicataria(IdAda,N,Nif,M),R).
adjudicatariaMor(M,R) :- solucoes((IdAda,N,Nif,M),adjudicataria(IdAda,N,Nif,M),R).
```

Identificar Contratos

Predicado que facilita a seleção de contratos, através dos seus diversos atributos.

```
contrato_Id(Id,R) :-
    solucoes((Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
    contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),R).
contrato_IdA(IdA,R) :-
    solucoes((Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
    contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),R).
contrato_IdAda(IdAda,R) :-
    solucoes((Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
    contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),R).
contrato_Tipo(Tipo,R) :-
    solucoes((Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
    contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),R).
contrato_Proc(Proc,R) :-
    solucoes((Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
    contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),R).
contrato_Local(Local,R) :-
    solucoes((Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
    contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),R).
contrato_Data(IdData,R) :-
    solucoes((Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
    contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),R).
```

Custos Totais

Predicado que calcula os custos totais de contratos através dos seus diferentes atributos.

```
custo_adjudicante(IdA,R) :-
    solucoes(Custo,
             contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
             R1),
    custo_total(R1,R).
custo_adjudicataria(IdAda,R) :-
    solucoes(Custo,
             contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
             R1),
    custo_total(R1,R).
custo_tipo(Tipo,R) :-
    solucoes(Custo,
             contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
             R1),
    custo_total(R1,R).
custo_local(Local,R) :-
    solucoes(Custo,
             contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
             R1),
    custo_total(R1,R).
custo_data(IdData,R) :-
    solucoes(Custo,
             contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
             R1),
    custo_total(R1,R).
```

Predicado auxiliar:

```
custo_total([X],X).
custo_total([X,Y|Z], R) :- custo_total([X+Y|Z], R1), R is R1.
```

Número de Adjudicantes

Predicado que calcula o número total de adjudicantes existes na base de conhecimento.

```
total_adjudicante(R) :-
    solucoes(IdA, adjudicante(IdA,N,Nif,M), L),
    comprimento(L,R).
```

Número de Adjudicatarias

Predicado que calcula o número total de adjudicatarias existes na base de conhecimento.

```
total_adjudicataria(R) :-
    solucoes(IdAda, adjudicataria(IdAda,N,Nif,M), L),
    comprimento(L,R).
```

Número de Contratos

Predicado que calcula o número total de contratos existes na base de conhecimento.

```
total_contrato(R) :-
    solucoes(Id, contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData), L),
    comprimento(L,R).
```

Mais Caro

Predicado que calcula qual é o contrato mais caro existente na base de conhecimento.

```
contrato_mais_caro(R) :-
    solucoes(Custo, contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData), L),
    maxLista(L,R1), solucoes((Id,IdA,IdAda,Tipo,Proc,Desc,R1,Prazo,Local,IdData),
    contrato(Id,IdA,IdAda,Tipo,Proc,Desc,R1,Prazo,Local,IdData), R).
```

Predicado auxiliar:

```
maxLista([H],R):- R is H.
maxLista([X|L],R) :- maxLista(L,N), X>N, R is X.
maxLista([X|L],R) :- maxLista(L,N), X=<N, R is N.
```

Número de Contratos por Tipo

Predicado que o número ce contratos existentes de um certo tipo na base de conhecimento.

```
nr_contrato_tipo(Tipo,R) :-
    solucoes(contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData),
    contrato(Id,IdA,IdAda,Tipo,Proc,Desc,Custo,Prazo,Local,IdData), R1),
    comprimento(R1,R).
```

3.8 Sistema de Inferência

À semelhança do meta-predicado demo que infere a veracidade de apenas uma questão, foi criado um novo meta-predicado denominado demoComp. Este meta-predicado deduz a veracidade de uma composição de questões, sendo essa composição constituída por disjunções e conjunções. Ao recorrer a este meta-predicado passamos a poder analisar um maior numero de informação, permitindo enriquecer o sistema de inferência.

```
demoComp(Q1 e Q2, R) :-
    demo(Q1,R1),
    demoComp(Q2,R2),
    conjuncao(R1,R2,R).
demoComp(Q1 ou Q2, R) :-
    demo(Q1,R1),
    demoComp(Q2,R2),
    disjuncao(R1,R2,R).
demoComp(Q, R) :-
    demo(Q,R).
```

Este meta-predicado permite testar se disjunções e conjunções de conhecimento são verdadeiras, falsas ou desconhecidas, obedecendo às regras lógicas da disjunções e conjunções. Contudo, como existe conhecimento imperfeito na base de conhecimento, este também tem de ser considerado nas tabelas lógicas.

e	**V**	D	**F**
V	V	D	F
D	D	D	F
F	F	F	F

Tabela de Conjunção

ou	**V**	D	**F**
V	V	V	V
D	V	D	D
F	V	D	F

Tabela de Disjunção

Estas tabelas são traduzidas em factos através dos meta-predicados conjuncao e disjuncao, da seguinte forma:

```
conjuncao(verdadeiro,verdadeiro,verdadeiro).
conjuncao(verdadeiro,falso,falso).
conjuncao(falso,verdadeiro,falso).
conjuncao(falso,falso,falso).
conjuncao(desconhecido,desconhecido,desconhecido).
conjuncao(desconhecido,verdadeiro,desconhecido).
conjuncao(verdadeiro,desconhecido,desconhecido).
conjuncao(desconhecido,falso,falso).
conjuncao(falso,desconhecido,falso).
```

```
disjuncao(verdadeiro,verdadeiro,verdadeiro).
disjuncao(verdadeiro,falso,verdadeiro).
disjuncao(falso,verdadeiro,verdadeiro).
disjuncao(falso,falso,falso).
disjuncao(desconhecido,desconhecido,desconhecido).
disjuncao(desconhecido,verdadeiro,verdadeiro).
disjuncao(verdadeiro,desconhecido,verdadeiro).
disjuncao(desconhecido,falso,desconhecido).
disjuncao(falso,desconhecido,desconhecido).
```

Para que o predicado demoComp pudesse analisar composições foi necessário definir dois novos operadores e e ou, correspondentes à conjuncao e disjuncao, respetivamente.

```
:- op( 900,xfy,'e'  ).
:- op( 900,xfy,'ou' ).
```

4. Conclusão

De uma perspetiva geral, consideramos que a realização deste exercıcio foi relativamente bem sucedida, visto que pensamos ter cumprido com todos os requisitos mínimos propostos e ainda conseguimos implementar novas funcionalidades e estender o conhecimento do sistema,

Após finalizarmos este trabalho, podemos afirmar que o nosso sistema é capaz de representar conhecimento perfeito e conhecimento imperfeito.

Este trabalho permitiu consolidar melhor os conceitos de adquiridos ao longo do semestre da cadeira de Sistemas de Representação de Conhecimento e Raciocínio, com o auxílio da linguagem de programação lógica PROLOG.