Universidade do Minho

Escola de Engenharia

Licenciatura em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Ano Lectivo de 2014/2015

Exercício 2

Grupo 28 & 29:

Adriano Teixeira, a67663

Carlos Morais, a64306

Filipe Ribeiro, a64315

João Farinha, a69302

Maio, 2015

Resumo

Neste relatório é apresentado um sistema de representação de conhecimento e raciocínio em que se pretende representar o universo do comercio automóvel.

Para a implementação do sistema é utilizada a linguagem de programação em lógica PROLOG e Java com a biblioteca JASPER.

No desenvolvimento do relatório serão apresentados todos os passos para a implementação do sistema. Será feita uma demonstração dos resultados obtidos sendo feita no final uma conclusão de todo o trabalho realizado.

Índice

Índice de Figuras

1. Introdução

Neste exercício é pretendido que se implemente um sistema para representação do universo do comércio automóvel.

O sistema a criar deverá representar o estado, cor, fabricante, marca, modelo, ano de fabrico, proprietário e registos de donos anteriores.

É pretendido que o sistema suporte a inserção de novo conhecimento inclusive conhecimento imperfeito ou incompleto.
Pretende-se também que a inserção e remoção de conhecimento respeite eventuais invariantes identificados no sistema.

O presente relatório é constituído por dois capítulos fundamentais sobre a implementação do sistema. No primeiro serão explicados todos os passos para o desenvolvimento do sistema e os motivos que nos fizeram tomar determinadas decisões. No último será feita uma apresentação de resultados obtidos para uma base de conhecimento previamente defina.

De forma a que a base de conhecimento tenha uma representação estruturada e organizada, a representação do conhecimento será feita à custa da matricula.

No final do relatório serão apresentadas algumas conclusões finais sobre o trabalho.

2. Desenvolvimento

2.1. Características do automóvel

O sistema a criar tem como base a representação das características de automóveis. Para a representação das características do automóvel foram implementados vários predicados.

Como base para a construção do universo de comercio automóvel é implementado o predicado matricula : CodigoMatricula -> {V,F}.

ex: matricula('AA-BB-00').

Neste exemplo o predicado matricula indica a matricula de um automóvel e vai servir para o identificar noutros predicados

A partir da matricula são construídos novos predicados para demonstrar as variadas características de um automóvel.

```
automovel(Matricula, Fabricante, Marca, Modelo).
cor(Matricula, Cor).
estado(Matricula, Estado).
anoMatricula(Matricula, Ano, Mês).
registo(Matricula, Individuo).
proprietario(Matricula, Proprietário).
```

O predicado registo representa antigos proprietários do veículo.

Foram também criados predicados que nos dão uma lista de todas as soluções que satisfazem uma determinada relação.

```
matriculas(R):- solucoes(M,matricula(M),R).
matriculas atribui a R a lista de todas as matriculas na base de conhecimento
automoveisMarca(Marca,R):-
solucoes(Matricula,automovel(Matricula,_,Marca,_),R).
automoveisMarca atribui a R a lista de todas as matriculas de
automóveis cuja marca coincide com Marca.
modeloPro(P,Mode) :- findall(M,(proprietario(Mat,P),
automovel(Mat,_,_,M)), Mode).
```

modeloPro atribui a Mode a lista dos modelos de todos os automoveis na posse do proprietario P.

listareg(Y,R) :- solucoes(N,registo(Y,N),R).

listareg atribui a R a lista de todos os proprietarios passados do veiculo com matricula Y.

2.2. Definição de Invariantes

De forma a que a representação da informação se mantenha consistente foram criados alguns invariantes.

Para se remover uma matricula não pode haver nenhuma relação com essa matricula.

-matricula(M) :: ((nao(automovel(M,_,_,_))), (nao(cor(M,_))), (nao(estado(M,_))), (nao(anoMatricula(M,_,_))), (nao(registo(M,_))), (nao(proprietario(M,_)))).

Não é permitida inserção de conhecimento repetido

+matricula(M) :: (solucoes(C,matricula(M),L) , comprimento(L,1)).

Só insere conhecimento se não houver conhecimento de que nao existe

+matricula(M) :: (nao(-matricula(M))).

Só insere conhecimento negativo se não houver conhecimento de que existe

+(-matricula(M)) :: (nao(matricula(M))).

Não permite Inserção de conhecimento negativo repetido

+(-matricula(M)) :: (solucoes(C,-matricula(M),L) , comprimento(L,N), N=<2).

Matricula tem de existir

+automovel(N,_,_,_) :: (matricula(N)).

Matricula do automóvel tem de ser única

+automovel(N,_,_,_) :: (solucoes(N,automovel(N,_,_,_),L) , comprimento(L,1)).

Só insere conhecimento se não houver conhecimento de que não existe

+automovel(M,X,Y,Z) :: (nao(-automovel(M,X,Y,Z))).

Só insere conhecimento negativo se não houver conhecimento de que existe

+(-automovel(M,X,Y,Z)) :: (nao(automovel(M,X,Y,Z))).

Não permite Inserção de conhecimento negativo repetido

+(-automovel(M,X,Y,Z)) :: (solucoes(C,-automovel(M,X,Y,Z),L) , comprimento(L,N), N=<2).

Matricula tem de existir

+cor(M,_) :: (matricula(M)).

Não admite cores duplicadas

+cor(M,_) :: (solucoes(C,cor(M,_),L) , comprimento(L,1)).

Não permite Inserção de conhecimento repetido

+cor(M,C) :: (solucoes(C,cor(M,C),L) , comprimento(L,1)).

Só insere conhecimento se não houver conhecimento de que nao existe

+cor(M,C) :: (nao(-cor(M,C))).

Só insere conhecimento negativo se não houver conhecimento de que existe

+(-cor(M,C)) :: (nao(cor(M,C))).

Não permite Inserção de conhecimento negativo repetido

+(-cor(M,C)) :: (solucoes(C,-cor(M,C) ,L) , comprimento(L,N), N=<2).

Matricula tem de existir

+estado(M,_) :: (matricula(M)).

Tipo de estado têm que ser válido

+estado(_,T) :: (tipo_estado(T)).

Não permite Inserção de conhecimento repetido
+estado(M,E) :: (solucoes(C,estado(M,E),L) , comprimento(L,1)).

Só insere conhecimento se não houver conhecimento de que nao existe
+estado(M,E) :: (nao(-estado(M,E))).

Só insere conhecimento negativo se não houver conhecimento de que existe
+(-estado(M,E)) :: (nao(estado(M,E))).

Não permite Inserção de conhecimento negativo repetido
+(-estado(M,E)) :: (solucoes(C,-estado(M,E),L) ,
comprimento(L,N), N=<2).

Matricula tem de existir
+anoMatricula(M,_,_) :: (matricula(M)).

Ano e mês tem de ser inteiros
+anoMatricula(_,A,M) :: (integer(A),integer(M), M=<12, M>=1).

Não permite Inserção de conhecimento repetido
+anoMatricula(M,A,Ms) :: (solucoes(C,anoMatricula(M,A,Ms),L) , comprimento(L,1)).

Só insere conhecimento se não houver conhecimento de que nao existe
+anoMatricula(M,A,Ms) :: (nao(-anoMatricula(M,A,Ms))).

Só insere conhecimento negativo se não houver conhecimento de que existe
+(-anoMatricula(M,A,Ms)) :: (nao(anoMatricula(M,A,Ms))).

Não permite Inserção de conhecimento negativo repetido
+(-anoMatricula(M,A,Ms)) :: (solucoes(C,-anoMatricula(M,A,Ms),L) , comprimento(L,N), N=<2).

Matricula tem de existir
+registo(M,_) :: (matricula(M)).

Não permite Inserção de conhecimento repetido
+registo(M,P) :: (solucoes(C,registo(M,P),L) , comprimento(L,1)).

Só insere conhecimento se não houver conhecimento de que nao existe
+registo(M,P) :: (nao(-registo(M,P))).

Só insere conhecimento negativo se não houver conhecimento de que existe
+(-registo(M,P)) :: (nao(registo(M,P))).

Não permite Inserção de conhecimento negativo repetido
+(-registo(M,P)) :: (solucoes(C,-registo(M,P),L) , comprimento(L,N), N=<2).

Não admite proprietario duplicados
+proprietario(M,_) :: (solucoes(P,proprietario(M,_),L) , comprimento(L,1)).

Matricula tem de existir
+proprietario(M,_) :: (matricula(M)).

Só insere conhecimento se não houver conhecimento de que nao existe
+proprietario(M,P) :: (nao(-proprietario(M,P))).

Só insere conhecimento negativo se não houver conhecimento de que existe
+(-proprietario(M,P)) :: (nao(proprietario(M,P))).

Não permite Inserção de conhecimento negativo repetido
+(-proprietario(M,P)) :: (solucoes(C,-proprietario(M,P),L) , comprimento(L,N), N=<2).

2.3. Inserção e remoção de conhecimento

De forma a permitir a inserção e remoção de conhecimento na base de conhecimento, foram implementados os predicados insercao e remocao.

O predicado insercao : Termo -> {V,F}, inicialmente faz a inserção do termo na base de conhecimento através do predicado evolucao e testa a inviolabilidade dos seus invariantes através do predicado teste. Caso um dos invariantes tenha sido violado, o termo é removido e a inserção falha (fail). É utilizado o operador ! (CUT), para que após a detecção de falha, o predicado evolucao não volte a repetir a inserção do termo.

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

insert( F ) :- assert( F ).
insert( F ) :- retract( F ), !, fail.

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

O predicado remocao : Termo -> {V,F}, faz a remoção na base de conhecimento do termo.

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

```
remove( F ) :- retract( F ).
remove( F ) :- assert( F ), !, fail.

        solucoes( T,Q,L ) :- findall( T,Q,L ).
```

Definimos também alguns predicados auxiliares que usámos nos invariantes.

```
        comprimento( [],0 ). comprimento( [_|T],R ) :- comprimento( T,R2 ), R is R2+1.
```

o predicado comprimento dá-nos o comprimento de uma lista

```
        concat([],L,L).
        concat([H|T],L,[H|Y]) :- concat(T,L,Y).
```

o predicado concat adiciona o elemento L à lista [H|T]

2.4. Conhecimento Negativo

Para representarmos a possibilidade de conhecimento imperfeito tivémos de introduzir o conceito de conhecimento negativo. Para este fim criámos os seguintes predicados:

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

O predicado demo indica se a questão e verdadeira falsa ou desconhecida.

```
        demoLista([],[]).
        demoLista([H|T],R) :- demo(H,X), demoLista(T,Y), concat([X],Y,R).
```

O predicado demoLista recebe uma lista de Questões e devolve uma lista de respostas

```
        demoLogico(L,falso) :- (demoLista(L,Raux), member(falso,Raux)).
        demoLogico(L,desconhecido) :- (demoLista(L,Raux), member(desconhecido,Raux)).
        demoLogico(L,verdadeiro).
```

O predicado demoLogico recebe uma lista de Questões e devolve verdadeiro se todos os membros da lista são iguais ao segundo argumento.

nao(Questao) :- Questao, !, fail.
nao(Questao).
O predicado nao verifica que Questao nao existe na base de conhecimento.

Para representar conhecimento incompleto usamos a relação excecao para denotar relações com informaçao desconhecida.

Para esta representação de conhecimento negativo funcionar tivemos de acrescentar alguns predicados de negação:

```
-matricula( M ) :- nao( matricula( M ) ), nao( excecao( matricula( M ) ) ).
-automovel(M,F,MA,MO):-(nao(automovel(M,F,MA,MO)), nao(excecao(automovel(M,F,MA,MO)))).
-cor(M,C):- (nao(cor(M,C)), nao(excecao(cor(M,C)))).
-estado(M,E):- (nao(estado(M,E)), nao(excecao(estado(M,E)))).
-anoMatricula(MA,AN,ME):-(nao(anoMatricula(MA,AN,ME)), nao(excecao(anoMatricula(MA,AN,ME)))).
-registo(M,R):- (nao(registo(M,R)), nao(excecao(registo(M,R)))).
-proprietario(M,P):-
    (nao(proprietario(M,P)),nao(excecao(proprietario(M,P)))).
```

2.5. Funcionalidades extra

Aqui descrevemos mais alguns predicádos que adicionámos:

```
evolucaoAutomovelNulo( automovel(M,F,Ma,impossivel) ) :- solucoes( Invariante,+automovel(M,F,Ma,impossivel)::Invariante,Lista ),
insert( automovel(M,F,Ma,impossivel) ),
teste( Lista ), assert( +automovel( M2,F2,Ma2,Md2 ) ::
( solucoes( Mds, (automovel(M,F,Ma,Mds), nao(nulo(Mds))), S), comprimento(S,N), N==0 ) ).
```

evolução usada para inserir automoveis cujo modelo é um caso de conhecimento interdito.

```
atualizarCor( M,C ) :- remocao( cor( M,_ ) ), evolucao( cor( M,C )).
```

Predicado que remove cor antiga do automovel de matricula M e adiciona a nova cor C.

```
atualizarProprietario( M,P ) :- (remocao( proprietario( M,_ ) ), evolucao( proprietario( M,P ))).
```

Predicado que remove o antigo proprietario do automovel de matricula M e adiciona o novo proprietario.

2.6. Java

A interação com o sistema foi implementada em JAVA com recurso à biblioteca JASPER.

A aplicação desenvolvida em Java, simula o interpretador do Prolog, alterando o seu output para os predicados implementados de forma a melhor a interação com o utilizador.

A aplicação apenas aceita os predicados implementados na Base de Conhecimento que permitem a evolução e remoção do conhecimento e a consulta do mesmo.

Para além dos predicados implementados no Prolog, a aplicação aceita o predicado infoMatricula(matricula). . Este embora não esteja implementado na Base de Conhecimento, é feito pela aplicação Java de forma á custa dos predicados implementados de forma a demonstrar outro tipo de funcionalidades com a biblioteca Jasper.

Os predicados aceites pela aplicação são os seguintes:

```
demo(T,R).
demoLista([Ts],R).
demoLogico([Ts],R).
evolucao(T).
evolucaoAutomovelNulo(T).
remocao(T).
modeloPro(T,R).
automoveisMarca(T,R).
automovel(T,R).
atualizarCor(T,R).
atualizarProprietario(T,R).
listareg(T,R).
infoMatricula(T,R).
```

Para terminar a aplicação deve ser introduzido o comando exit.

3. Resultados

Depois de feita a implementação do sistema, para testar a Base de Conhecimento construída foi utilizada a aplicação desenvolvida em Java.

São demonstrados todos os predicados implementados, contudo não é possível demonstra todos os invariantes impostos pois tornaria este capitulo demasiado extenso.

3.1. Evolução, demo e Invariante da matricula

```
evolucao(matricula('XX-90')).
R: YES!
demo(matricula('XX-90'),R).
R: a resposta ao demo é: verdadeiro
evolucao(matricula('XX-90')).
R: NO!
```

Figure 1 representação de matricula

Aqui podemos observar que após usarmos o predicado evolução para inserir na base de conhecimento "matricula('XX-90')", o demo devolve verdadeiro como seria de esperar. Quando tentamos reinserir a mesma matricula não nos é permitido tal como pretendido devido aos invariantes.

3.2. Evolução, demo e Invariante do automóvel

```
evolucao(automovel('XX-TT',mercedes,smart,xpt)).
R: NO!
evolucao(automovel('XX-90',mercedes,smart,xpt)).
R: YES!
demo(automovel('XX-90',mercedes,smart,xpt),R).
R: a resposta ao demo é: verdadeiro
```

Figure 2 representação de automovel

Neste exemplo tentamos inserir automovel('XX-TT',mercedes,smart,xpt) mas falha dado que esta matricula não existe na base de conhecimento. Quando tentamos inserir automovel('XX-90',mercedes,smart,xpt) e bem sucedido dado que esta matricula existe e como seria de esperar demo devolve verdadeiro já que este conhecimentos e encontra na base de conhecimento.

3.3. Evolução, demo e atualização da cor

```
evolucao(cor('XX-90',branco)).
R: YES!
evolucao(cor('XX-90',preto)).
R: NO!
atualizarCor('XX-90',preto).
R: YES!
demo(cor('XX-90',preto),R).
R: a resposta ao demo é: verdadeiro
```

Figure 3 representação de cor

Aqui inserimos a cor do carro de matricula XX-90(branco) no entanto quando tentamos inserir uma nova cor não funciona dado que se funciona-se o carro passaria a ter 2 cores. Para mudar a cor do carro usamos atualizarCor e

assim este veiculo passa a ter cor preto como podemos constatar pelo facto de o demo dar verdadeiro.

3.4. Evolução, demo e remoção do proprietário

```
evolucao(proprietario('XX-90',carlos)).
R: YES!
demo(proprietario('XX-90',carlos),R).
R: a resposta ao demo é: verdadeiro
remocao(proprietario('XX-90',carlos)).
R: YES!
demo(proprietario('XX-90',carlos),R).
R: a resposta ao demo é: falso
```

Figure 4 representação de proprietario

Neste exemplo inserimos o proprietário do carro(carlos) e testamos o respectivo demo que é bem sucedido. Em seguida usamos remocao para remover o proprietario do veiculo e como tal o demo seguinte da falso.

3.5. Evolução, demo e invariante do proprietário

```
evolucao(estado('XX-90',bombom)).
R: NO!
evolucao(estado('XX-90',novo)).
R: YES!
demo(estado('XX-90',novo),R).
R: a resposta ao demo é: verdadeiro
```

Figure 5 representação de estado

Quando tentamos inserir o estado do veiculo como sendo bombom a inserçao falha já que este não é um estado reconhecido. Quando tentamos inserir o

estado do veiculo como sendo novo este estado é reconhecido e a inserção e bem sucedida e como tal o demo da verdadeiro.

3.6. Evolução do ano da matricula

```
evolucao(anoMatricula('XX-90',2014,14)).
R: NO!
evolucao(anoMatricula('XX-90',2014,10)).
R: YES!
```

Figure 6 representação de anoMatricula

Neste exemplo podemos observar que a inserçao falha se o valor de mês do predicado anoMatricula não estiver dentro dos valores aceitáveis de mês(1-12).

3.7. Evolução e listagem de registos de uma matricula

```
evolucao(registo('XX-90',pedro)).
R: YES!
evolucao(registo('XX-90',andre)).
R: YES!
evolucao(registo('XX-90',ana)).
R: YES!
listareg('XX-90',R).
R: os registos são:
->pedro
->andre
->ana
```

Figure 7 representação de registo

Aqui podemos observar o predicado listareg a funcionar listando todos os indivíduos que foram inseridos previamente nos registos.

3.8. Informação da matricula

```
infoMatricula('XX-90').
Automovel: construtor=mercedes, marca=smart, modelo=smart;
Proprietario: carlos;
Estado: novo;
Cor: preto;
Registos: ->pedro
->andre
->ana
```

Figure 8 extra: infoMatricula

Neste exemplo observamos infoMatricula a funcionar devolvendo toda a informação que a base de conhecimento possui sobre o veiculo com uma dada matricula incluindo construtor, marca, modelo, proprietário, estado, cor, e registos.

3.9. Conhecimento Imperfeito

```
evolucao(matricula(xx)).
R: YES!
evolucao(-automovel(xx,mercedes,smart,z)).
R: YES!
demo(-automovel(xx,mercedes,smart,z),R).
R: a resposta ao demo é: verdadeiro
evolucao(automovel(xx,mercedes,smart,z)).
R: NO!
```

Figure 9 conhecimento imperfeito

Aqui inserimos uma nova matricula xx e em seguida inserimos conhecimento negativo a afirmar que o automovel de matricula xx não é contruido pela mercedes é um smart de modelo z. Quando usamos demo para testar se isto se verifica ele confirma que -automovel(xx,mercedes,smart,z) e se tentarmos

inserir automovel(xx,mercedes,smart,z) a inserção falha dado que entraria em conflito com o conhecimento previamente inserido.

3.10. Conhecimento Imperfeito: Incerto

```
evolucao(matricula(yy)).
R: YES!
evolucao(automovel(yy,mercedes,smart,desconhecido)).
R: YES!
demo(automovel(yy,mercedes,smart,y),R).
R: a resposta ao demo é: desconhecido
```

Figure 10 conhecimento imperfeito incerto

Neste exemplo constatamos que o demo devolve desconhecido para o veiculo yy dado que foi inserido previamente que o seu modelo é desconhecido.

3.11. Conhecimento Imperfeito: Impreciso

```
evolucao(excecao(automovel(zz,mercedes,smart,z))).
R: YES!
evolucao(excecao(automovel(zz,mercedes,smart,h))).
R: YES!
demo(automovel(zz,mercedes,smart,z),R).
R: a resposta ao demo é: desconhecido
demo(-automovel(zz,mercedes,smart,z),R).
R: a resposta ao demo é: desconhecido
```

Figure 11 conhecimento imperfeito impreciso

Neste exemplo inserimos uma exceção para o veículo zz de tal forma a que o seu modelo seja ou z ou h. Quando usamos o demo o programa confirma que o modelo do automovel ser ou não ser z é desconhecido.

3.12. Conhecimento Imperfeito: Nulo

```
evolucao(matricula(ww)).
R: YES!
evolucaoAutomovelNulo(automovel(ww,mercedes,smart,impossivel)).
R: YES!
demo(automovel(ww,mercedes,smart,impossivel),R).
R: a resposta ao demo é: verdadeiro
evolucao(automovel(ww,mercedes,smart,tt)).
R: NO!
```

Figure 12 conhecimento imperfeito nulo

O sistema permite também a inserçao de conhecimento nulo com o predicado evolucaoAutomovelNulo como podemos observar neste exemplo. Quando tentamos inserir o automovel com um modelo definido não nos e permitido.

4. Conclusão

Com a realização do segundo exercício foi possível interiorizar os conhecimento fundamentais da representação do conhecimento imperfeito com a linguagem de programação e lógica PROLOG e o uso da biblioteca jasper para criar uma interação entre esta linguagem e o java.

O sistema implementado apresenta todas as funcionalidades inicialmente pretendidas, sendo ainda adicionadas novas funcionalidades ao sistema de forma a obter um resultado final mais completo.

Anexos

I. Código Prolog

```
%---------------------------------- - - - - - - - - - - - - - -
%---------------------------------- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% SICStus PROLOG: Declarações iniciais
:- set_prolog_flag( discontiguous_warnings, off ).
:- set_prolog_flag( single_var_warnings, off ).
:- set_prolog_flag( unknown, fail ).

%---------------------------------- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% SICStus PROLOG: definicoes iniciais
:- op( 900,xfy,'::' ).
:- dynamic insert/1.
:- dynamic remove/1.
:- dynamic (-)/1.
:- dynamic matricula/1.
:- dynamic automovel/4.
:- dynamic cor/2.
:- dynamic registo/2.
:- dynamic proprietario/2.
:- dynamic anoMatricula/3.
:- dynamic estado/2.
:- dynamic excecao/1.
:- dynamic (::)/2.


%============================== Demo ==============================
%_____________________________________________________________________
% Extensao do predicado demo: Questao,Resposta -> {V,F,D}

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

```
   nao( Questao ),
   nao( -Questao ).

%_____________________________________________________________________________
% Extensao do predicado demoLista: [Questao],[Resposta] -> {V,F,D}

demoLista([],[]).
demoLista([H|T],R) :- demo(H,X), demoLista(T,Y), concat([X],Y,R).


%_____________________________________________________________________________
% Extensao do predicado demoLogico: [Questao],Resposta -> {V,F,D}

demoLogico(L,falso) :- (demoLista(L,Raux), member(falso,Raux)).
demoLogico(L,desconhecido) :- (demoLista(L,Raux), member(desconhecido,Raux)).
demoLogico(L,verdadeiro).


%============================== Não ==============================
% nao: Questao -> {V,F}

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

%============================== Matricula ==============================
%_____________________________________________________________________________
% matricula :: Matricula -> {V,F}

% Representacao de conhecimento positivo
matricula( 'AA-BB-00' ).
matricula( 'AA-BB-10' ).
matricula( 'AA-BB-11' ).
matricula( 'AA-BB-22' ).
matricula( 'AA-BB-33' ).
matricula( 'AA-BB-44' ).
matricula( 'AA-BB-66' ).
```

```
matricula( 'PP-BB-90' ).


% Representacao de conhecimento negativo
-matricula( 'ZZ-YY-99' ).
-matricula( 'ZZ-YY-88' ).
-matricula( M ) :-
    nao( matricula( M ) ),
    nao( excecao( matricula( M ) ) ).


excecao(matricula('XX-YY-10')).

%_____________________________________________________________________________
% AUX: -> Totos as Matriculas existentes
matriculas(R):- solucoes(M,matricula(M),R).


%_____________________________________________________________________________
% INVARIANTES DE MATRICULA

% Para remover, não pode ter nada que refere a matricula a remover
-matricula(M) :: ((nao(automovel(M,_,_,_))),
                  (nao(cor(M,_))),
                  (nao(estado(M,_))),
                  (nao(anoMatricula(M,_,_))),
                  (nao(registo(M,_))),
                  (nao(proprietario(M,_)))).

% não permite Inserção de conhecimento repetido
+matricula(M) :: (solucoes(C,matricula(M),L) , comprimento(L,1)).

% só insere conhecimento se não houver conhecimento de que nao existe
+matricula(M) :: (nao(-matricula(M))).

% só insere conhecimento negativo se não houver conhecimento de que existe
```

```
+(-matricula(M)) :: (nao(matricula(M))).

% não permite Inserção de conhecimento repetido
+(-matricula(M)) :: (solucoes(C,-matricula(M),L) , comprimento(L,N), N=<2).



%============================== AUTOMOVEL ==============================
%_____________________________________________________________________
% automovel :: Matricula, Fabricante, Marca, Modelo -> {V,F}

% Representacao de conhecimento positivo
automovel( 'AA-BB-10',mercedes,smart,fourtwo ).
automovel( 'AA-BB-11',renaut,renaut,clio ).
automovel( 'AA-BB-22',fiat,fiat,panda ).
automovel( 'AA-BB-33',ford,ford,fiesta ).
automovel( 'AA-BB-44',fiat,fiat,punto).



% Representacao de conhecimento negativo
-automovel( 'AA-BB-00',opel,opel,corsa ).

-automovel(M,F,MA,MO) :- (nao(automovel(M,F,MA,MO)),
                    nao(excecao(automovel(M,F,MA,MO)))).

%_____________________________________________________________________
% AUX:
% -> Modelos de todos os automoveis de um proprietario
modeloPro(P,Mode) :-  findall(M,(proprietario(Mat,P), automovel(Mat,_,_,M)),  Mode).

% -> Se existe um automovel com matricula....
automovel(X) :- (automovel(X,_,_,_)).

% -> Todos os automoveis(matricula) de uma marca
automoveisMarca(M,R):- solucoes(N,automovel(N,_,M,_),R).
```

```
%_____________________________________________________________________________
% INVARIANTES DE AUTOMÓVEL

% Matricula tem de existir
+automovel(N,_,_,_) :: (matricula(N)).

% Matricula do automovel tem de ser unica
+automovel(N,_,_,_) :: (solucoes(N,automovel(N,_,_,_),L) , comprimento(L,1)).

% só insere conhecimento se não houver conhecimento de que nao existe
+automovel(M,X,Y,Z) :: (nao(-automovel(M,X,Y,Z))).

%  só insere conhecimento negativo se não houver conhecimento de que existe
+(-automovel(M,X,Y,Z)) :: (nao(automovel(M,X,Y,Z))).

% só insere conhecimento se não houver conhecimento de que nao existe
+automovel(M,X,Y,Z) :: (nao(-automovel(M,X,Y,Z))).

% só insere conhecimento negativo se não houver conhecimento de que existe
+(-automovel(M,X,Y,Z)) :: (nao(automovel(M,X,Y,Z))).

% não permite Inserção de conhecimento repetido
+(-automovel(M,X,Y,Z)) :: (solucoes(C,-automovel(M,X,Y,Z),L) , comprimento(L,N), N=<2).


%_____________________________________________________________________________
% REPRESENTAÇÂO DE CONHECIMENTO INCERTO - (Tipo 1)
% - Sabe-se que o carro com matricula "aa-bb-55" é um ferrari, fabricado
%   pela mesma, só nao se sabe qual é o modelo

automovel( 'AA-BB-55' , ferrari , ferrari, desconhecido ).
excecao(automovel( A,F,M,MO )) :- automovel( A,F,M,desconhecido ).


%_____________________________________________________________________________
```

```
% REPRESENTAÇÂO DE CONHECIMENTO IMPRECISO - (Tipo 2)
% - Sabe-se que o carro com matricula "aa-bb-66" é um volkswagen, fabricado
%   pela mesma, o modelo é se tem a certeza, ou é um golf ou um polo

excecao( automovel( 'AA-BB-66',volkswagen,volkswagen,golf )).
excecao( automovel( 'AA-BB-66',volkswagen,volkswagen,polo )).



%_____________________________________________________________________________
% REPRESENTAÇÃO DE CONHECIMENTO INTERDITO - (Tipo 3)
% - Sabe-se que o novo carro da audi tem a matricula "aa-bb-77",
%   mas não se pode sabar qual o modelo

excecao(automovel(A,F,M,B)) :- automovel(A,F,M,impossivel).
nulo(impossivel).
automovel('PP-BB-90',aa,bb,impossivel).
+automovel(M2,F2,Ma2,Md2):: (solucoes(Mds,(automovel('PP-BB-
90',aa,bb,Mds),nao(nulo(Mds))),S),comprimento(S,N),N==0).

%_____________________________________________________________________________
% Predicado evolucaoAutomovelNulo
evolucaoAutomovelNulo( automovel(M,F,Ma,impossivel) ) :-
   solucoes( Invariante,+automovel(M,F,Ma,impossivel)::Invariante,Lista ),
                insert( automovel(M,F,Ma,impossivel) ),
                teste( Lista ),
                assert(
                   +automovel( M2,F2,Ma2,Md2 ) :: (
                      solucoes( Mds, (automovel(M,F,Ma,Mds), nao(nulo(Mds))), S),
                      comprimento(S,N),
                      N==0
                         )
                      ).


%============================== COR ==============================
```

```
%_____________________________________________________________________________
% cor :: Matricula, Cor -> {V,F}

% Representacao de conhecimento positivo
cor('AA-BB-00',amarelo).
cor('AA-BB_11',preto).
cor('AA-BB-22',azul).
cor('AA-BB-33',vermelho).

% Representacao de conhecimento negativo
-cor(M,C):- (nao(cor(M,C)), nao(excecao(cor(M,C)))).

%_____________________________________________________________________________
% INVARIANTES DE COR

% matricula tem de existir
+cor(M,_) :: (matricula(M)).

% não admite cores duplicados
+cor(M,_) :: (solucoes(C,cor(M,_),L) , comprimento(L,1)).

% não permite Inserção de conhecimento repetido
+cor(M,C) :: (solucoes(C,cor(M,C),L) , comprimento(L,1)).

% só insere conhecimento se não houver conhecimento de que nao existe
+cor(M,C)  :: (nao(-cor(M,C))).

% só insere conhecimento negativo se não houver conhecimento de que existe
+(-cor(M,C) ) :: (nao(cor(M,C))).

% não permite Inserção de conhecimento repetido
+(-cor(M,C) ) :: (solucoes(C,-cor(M,C) ,L) , comprimento(L,N), N=<2).

%_____________________________________________________________________________

% Predicado para ATUALIZAR a cor de um dado automovel
```

```
atualizarCor( M,C ) :- remocao( cor( M,_ ) ), evolucao( cor( M,C )).


%============================== ESTADO ==============================
%____________________________________________________________________
% tipo_estado :: Estado -> {V,F}
% todos os tipo possiveis para representar o estado de um automovel
tipo_estado( bom ).
tipo_estado( razoavel ).
tipo_estado( mau ).
tipo_estado( novo ).

%____________________________________________________________________
% estado :: Matricula, Estado -> {V,F}

% Representacao de conhecimento positivo
estado('AA-BB-00', bom).
estado('AA-BB-11', razoavel).
estado('AA-BB-33', razoavel).

-estado(M,E):- (nao(estado(M,E)), nao(excecao(estado(M,E)))).

%____________________________________________________________________
% INVARIANTES DO ESTADO

% Matricula tem de existir
+estado(M,_) :: (matricula(M)).

% Tipo de estado têm que ser válido
+estado( _,T ) :: (tipo_estado( T )).

% Não permite Inserção de conhecimento repetido
+estado(M,E) :: (solucoes(C,estado(M,E),L) , comprimento(L,1)).

% Só insere conhecimento se não houver conhecimento de que nao existe
+estado(M,E) :: (nao(-estado(M,E))).
```

```
% Só insere conhecimento negativo se não houver conhecimento de que existe
+(-estado(M,E)) :: (nao(estado(M,E))).

% Não permite Inserção de conhecimento repetido
+(-estado(M,E)) :: (solucoes(C,-estado(M,E),L) , comprimento(L,N), N=<2).


%_____________________________________________________________________________
% REPRESENTAÇÂO DE CONHECIMENTO IMPRECISO - (Tipo 2)
% - O o estado do automóvel com matricula "aa-bb-22", nao é novo,
%   nas nao se sabe se é bom ou razoavel

-estado('AA-BB-22',novo).
excecao(estado( 'AA-BB-22' , bom )).
excecao(estado( 'AA-BB-22' , razoavel )).



%============================  ANO MATRICULA ========================
%_____________________________________________________________________________
% anoMatricula :: Matricula, Ano, Mês -> {V,F}

% Representacao de conhecimento positivo
anoMatricula('AA-BB-00', 2012, 9).
anoMatricula('AA-BB-11', 2013, 10).
anoMatricula('AA-BB-22', 2014, 11).
anoMatricula('AA-BB-33', 2015, 12).

% Representacao de conhecimento negativo
-anoMatricula(MA,AN,ME):- (nao(anoMatricula(MA,AN,ME)),
                    nao(excecao(anoMatricula(MA,AN,ME)))).

%_____________________________________________________________________________
% INVARIANTES DE ANOMATRICULA
```

```
% matricula tem de existir
+anoMatricula(M,_,_) :: (matricula(M)).

% ano e mês tem de ser inteiros
+anoMatricula(_,A,M) :: (integer(A),integer(M), M=<12, M>=1).

% não permite Inserção de conhecimento repetido
+anoMatricula(M,A,Ms) :: (solucoes(C,anoMatricula(M,A,Ms),L) , comprimento(L,1)).

% só insere conhecimento se não houver conhecimento de que nao existe
+anoMatricula(M,A,Ms) :: (nao(-anoMatricula(M,A,Ms))).

% só insere conhecimento negativo se não houver conhecimento de que existe
+(-anoMatricula(M,A,Ms)) :: (nao(anoMatricula(M,A,Ms))).

% não permite Inserção de conhecimento repetido
+(-anoMatricula(M,A,Ms)) :: (solucoes(C,-anoMatricula(M,A,Ms),L) , comprimento(L,N), N=<2).


%============================== REGISTO ==============================
%_____________________________________________________________________
% registo :: Matricula, Proprietario -> {V,F}

% Representacao de conhecimento positivo
registo('AA-BB-00', joaquim_albero).
registo('AA-BB-00', pedro_sousa).


% Representacao de conhecimento negativo
-registo(M,R):- (nao(registo(M,R)),
                 nao(excecao(registo(M,R)))).


%_____________________________________________________________________
% AUX:
% -> Todos os Proprietarios de uma determinado automovel
```

```
listareg(Y,R) :- solucoes(N,registo(Y,N),R).

%_____________________________________________________________________
% INVARIANTES DE REGISTO

% matricula tem de existir
+registo(M,_) :: (matricula(M)).

% não permite Inserção de conhecimento repetido
+registo(M,P) :: (solucoes(C,registo(M,P),L) , comprimento(L,1)).

% só insere conhecimento se não houver conhecimento de que nao existe
+registo(M,P) :: (nao(-registo(M,P))).

% só insere conhecimento negativo se não houver conhecimento de que existe
+(-registo(M,P)) :: (nao(registo(M,P))).

% não permite Inserção de conhecimento repetido
+(-registo(M,P)) :: (solucoes(C,-registo(M,P),L) , comprimento(L,N), N=<2).


%============================== PROPRIETARIO
==============================
%_____________________________________________________________________
% proprietario :: Matricula, Proprietario -> {V,F}

% Representacao de conhecimento positivo
proprietario('AA-BB-00',pedro).
proprietario('AA-BB-11',daniel).

% Representacao de conhecimento negativo
-proprietario('AA-AA-00',alberto).

-proprietario(M,P):- (nao(proprietario(M,P)),
                      nao(excecao(proprietario(M,P)))).
```

```
%_____________________________________________________________________________

% Predicado para ATUALIZAR o proprietário de um dado automovel
atualizarProprietario( M,P ) :- (remocao( proprietario( M,_ ) ),
                                 evolucao( proprietario( M,P ))).

%_____________________________________________________________________________
% INVARIANTES DE PROPRIETARIOS

% nao admite proprietario duplicados
+proprietario(M,_) :: (solucoes(P,proprietario(M,_),L) , comprimento(L,1)).

% matricula tem de existir
+proprietario(M,_) :: (matricula(M)).

% só insere conhecimento se não houver conhecimento de que nao existe
+proprietario(M,P) :: (nao(-proprietario(M,P))).

% só insere conhecimento negativo se não houver conhecimento de que existe
+(-proprietario(M,P)) :: (nao(proprietario(M,P))).

% não permite Inserção de conhecimento repetido
+(-proprietario(M,P)) :: (solucoes(C,-proprietario(M,P),L) , comprimento(L,N), N=<2).

%_____________________________________________________________________________
% REPRESENTAÇÂO DE CONHECIMENTO INCERTO - (TIPO 1)

% - Sabe-se que existe um carro com matricula "aa-bb-33".
%   só nao se sabe onde quem é o Proprietario.

proprietario( 'AA-B3-33' , desconhecido ).
excecao(proprietario( A , P )) :- proprietario( A , desconhecido ).


%============================== AUXILIARES ============================
```

```
%_____________________________________________________________________
% Inserção de informação
% insert( Facto )

insert( F ) :- assert( F ).
insert( F ) :- retract( F ), !, fail.

%_____________________________________________________________________
% Remoção de informação
% remove( Facto )

remove( F ) :- retract( F ).
remove( F ) :- assert( F ), !, fail.

%_____________________________________________________________________
% Extensão do predicado que permite a adição de conhecimento

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

%_____________________________________________________________________
% Extensão do predicado que permite a remocao de conhecimento

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

%_____________________________________________________________________
% Extensao do predicado solucoes

solucoes( T,Q,L ) :- findall( T,Q,L ).

%_____________________________________________________________________
% Extensao do predicado comprimento
```

```
comprimento( [],0 ).
comprimento( [_|T],R ) :- comprimento( T,R2 ), R is R2+1.

%_____________________________________________________________________
% Extensao do predicado teste

teste( [] ).
teste( [H|T] ) :- H, teste( T ).

%_____________________________________________________________________
% Extensao do predicado concat

concat([],L,L).
concat([H|T],L,[H|Y]) :- concat(T,L,Y).

%_____________________________________________________________________
% Extensao do predicado excecao, para não permitir excecões duplicadas
+excecao(T) :: (solucoes(C,excecao(T),L) , comprimento(L,1)).
```

II. Código Java

PrologWorker.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package prolog;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import se.sics.jasper.ConversionFailedException;
import se.sics.jasper.IllegalTermException;
import se.sics.jasper.SICStus;
import se.sics.jasper.SPException;
import se.sics.jasper.SPTerm;
import se.sics.jasper.Query;



/**
 *
 * @author Carlos Morais
 */
public class PrologWorker {

    private SICStus sicstus;

    public PrologWorker(String path) throws SPException{
        sicstus = new SICStus();
        sicstus.load(path);
    }
```

```
    private List<String> parseTermListAsString(SPTerm spt) throws IllegalTermException,
ConversionFailedException{
        List<String> elementos = new ArrayList<>();
        SPTerm newSPT = new SPTerm(this.sicstus);

        while(!spt.isEmptyList()){
            spt.getList(newSPT, spt);
            elementos.add(newSPT.toString());
        }

        return elementos;
    }

    public String QueryQuestion(String q) throws SPException{
        HashMap map = new HashMap();
        boolean success = this.sicstus.query(q, map);

        if(success){
            Object R = null;
            for(Object aux : map.keySet())
                R = aux;

            SPTerm distTerm = (SPTerm) map.get(R);
            return distTerm.toString();
        }
        else
            return null;
    }

    public List<String> QueryQuestionList(String q) throws SPException{
        HashMap map = new HashMap();
        boolean success = this.sicstus.query(q, map);

        if(success){
            Object R = null;
            for(Object aux : map.keySet())
                R = aux;
```

```
        List<String> nodes = parseTermListAsString((SPTerm) map.get(R));
        return nodes;
    }
    else
        return null;
}

public boolean Query(String q) throws SPException{
    boolean success = this.sicstus.query(q, null);

    if(success)
        return true;
    else
        return false;
}

//nao é um predicado implementado na Base de Conhecimento!
public String infoMatricula(String matricula) throws SPException{
    List<String> respostas;
    String resposta;
    StringBuilder res = new StringBuilder();
    HashMap map = new HashMap();



    //Automovel
    if (this.sicstus.query("automovel("+matricula+",C,Ma,Mo).", map)){
        res.append("Automovel: construtor=");
        res.append(map.get("C").toString());

        res.append(", marca=");
        res.append(map.get("Ma").toString());

        res.append(", modelo=");
        res.append(map.get("Ma").toString());
        res.append(";\n");
    }
    map.clear();
```

```
    //proprietario
    if(this.sicstus.query("proprietario("+matricula+",R).", map)){
        res.append("Proprietario: ");
        res.append(map.get("R").toString());
        res.append(";\n");
    }
    map.clear();

    //estado
    if(this.sicstus.query("estado("+matricula+",R).", map)){
        res.append("Estado: ");
        res.append(map.get("R").toString());
        res.append(";\n");
    }
    map.clear();

    //cor
    if(this.sicstus.query("cor("+matricula+",R).", map)){
        res.append("Cor: ");
        res.append(map.get("R").toString());
        res.append(";\n");
    }
    map.clear();

    //registos
    if(this.sicstus.query("listareg("+matricula+",R).", map)){
        List<String> nodes = parseTermListAsString((SPTerm) map.get("R"));

        if(nodes.size()>0){
            res.append("Registos: ");
            for(String aux:nodes)
                res.append("->"+aux+"\n");
            res.append("\n");
        }
    }
    map.clear();

    return res.toString();
}
```

```
}
```


Prolog.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package prolog;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import se.sics.jasper.SPException;

/**
 *
 * @author Carlos Morais
 */
public class Prolog {

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     * @throws se.sics.jasper.SPException
     */
    public static void main(String[] args) throws IOException, SPException {
        boolean continua = true, found;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String leitura;
        PrologWorker prolog;

        prolog = new PrologWorker("TP2_SRCR.pl");
```

```
while(continua){
    found = false;
    leitura = br.readLine();

    //removes all whitespaces and non visible characters such as tab,\n
    leitura.replaceAll("\\s+","");

    if(leitura.startsWith("demo(")){
        found = true;
        String resposta = prolog.QueryQuestion(leitura);
        if(resposta!=null)
            System.out.println("R: a resposta ao demo é: "+resposta);
        else
            System.out.println("R: Erro no predicado!");

    }

    if(leitura.startsWith("demoLista(")){
        found = true;
        List<String> respostas = prolog.QueryQuestionList(leitura);

        if(respostas!=null){
            if(respostas.size() > 0){
                System.out.println("as respostas ao demoLista são: ");
                for(String res:respostas)
                    System.out.println("->"+res);
            }
            else
                System.out.println("R: não existem respostas!");
        }
        else
            System.out.println("R: Erro no predicado!");
    }

    if(leitura.startsWith("demoLogico(")){
        found = true;
        String resposta = prolog.QueryQuestion(leitura);

        if(resposta!=null)
            System.out.println("a resposta ao demoLogico é: "+resposta);
```

```
    else
        System.out.println("R: Erro no predicado!");
}

if(leitura.startsWith("evolucao(")){
    found = true;
    if(prolog.Query(leitura))
        System.out.println("R: YES!");
    else
        System.out.println("R: NO!");
}

if(leitura.startsWith("evolucaoAutomovelNulo(")){
    found = true;
    if(prolog.Query(leitura))
        System.out.println("R: YES!");
    else
        System.out.println("R: NO!");
}

if(leitura.startsWith("remocao(")){
    found = true;
    if(prolog.Query(leitura))
        System.out.println("R: YES!");
    else
        System.out.println("R: NO!");
}


if(leitura.startsWith("modeloPro(")){
    found = true;
    List<String> respostas = prolog.QueryQuestionList(leitura);
    if(respostas!=null){
        if(respostas.size() > 0){
            System.out.println("R: os modelos do proprietario sao:");
            for(String res:respostas)
                System.out.println("->"+res);
        }
        else
            System.out.println("R: não existem modelos!");
```

```
    }
    else
        System.out.println("R: Erro no predicado!");
}

if(leitura.startsWith("automoveisMarca(")){
    found = true;
    List<String> respostas = prolog.QueryQuestionList(leitura);
    if(respostas!=null){
        if(respostas.size() > 0){
            System.out.println("R: os automoveis da marca são:");
            for(String res:respostas)
                System.out.println("->"+res);
        }
        else
            System.out.println("R: não existem automoveis!");
    }
    else
        System.out.println("R: Erro no predicado!");
}

if(leitura.startsWith("automovel(")){
    found = true;
    if(prolog.Query(leitura))
        System.out.println("R: YES!");
    else
        System.out.println("R: NO!");
}

if(leitura.startsWith("atualizarCor(")){
    found = true;
    if(prolog.Query(leitura))
        System.out.println("R: YES!");
    else
        System.out.println("R: NO!");
}


if(leitura.startsWith("atualizarProprietario(")){
    found = true;
```

```
    if(prolog.Query(leitura))
        System.out.println("R: YES!");
    else
        System.out.println("R: NO!");
}

if(leitura.startsWith("listareg(")){
    found = true;
    List<String> respostas = prolog.QueryQuestionList(leitura);
    if(respostas!=null){
        if(respostas.size() > 0){
            System.out.println("R: os registos são:");
            for(String res:respostas)
                System.out.println("->"+res);
        }
        else
            System.out.println("R: não existem registos!");
    }
    else
        System.out.println("R: Erro no predicado!");
}

if(leitura.startsWith("exit")){
    found = true;
    continua = false;
}

//apresenta toda a informaçao na base de conhecimento para uma matricula
if(leitura.startsWith("infoMatricula(")){
    found = true;

    //matricula
    String mat = leitura.substring(14,(leitura.length()-2));

    // se a matricula existe
    if(prolog.Query("matricula("+mat+").")){
        String resposta = prolog.infoMatricula(mat);
        if(resposta.length()>0)
            System.out.println(resposta);
        else
```

```
                System.out.println("R: sem informação!");
            }
            else
                System.out.println("R: Não reconhece o a matricula!");
        }

        //nao reconhece o predicado
        if(!found){
            System.out.println("R: Não reconhece o predicado!");
        }


    }
  }

}
```