Universidade do Minho

Mestrado Integrado em Engenharia Informática
Licenciatura em Engenharia Informática

Inteligência Artificial

Trabalho Prático - Fase 1

Grupo 12

Ana Rebelo
(A90234)

Ana Murta
(A93284)

Ana Henriques
(A93268)

Joana Oliveira
(A87956)

2 de dezembro de 2021

Resumo

O presente relatório tem como objetivo descrever todas as decisões tomadas para solucionar a primeira fase da problemática proprosta na unidade curricular de Inteligência Artificial. De modo a utilizar a linguagem de programação em lógica, *PROLOG*, desenvolveu-se um sistema que fosse capaz de representar um universo de discurso na área da logística de distribuição de encomendas.

Para este panorama de conhecimento, caracterizado pelas encomendas, clientes e estafetas, procurou-se resolver o mais eficazmente possível as funcionalidades base exigidas, adicionando algumas extras que o grupo julgou pertinentes.

Conteúdo

1 Introdução

Nesta primeira fase do trabalho prático da disciplina de Inteligência Artificial, foi visada a resolução de um exercicío de forma a motivar os alunos a colocar em prática a linguagem de programação em lógica *PROLOG*, lecionada nas aulas.

Para a problemática proposta, foi solicitada a representação de um universo na área da logística da distribuição de encomendas pela empresa *Green Distribuition*. Esta empresa tem ao seu dispor vários meios de transporte, como bicicletas, motos e carros, procurando sempre privilegiar o mais ecológico.

Para esse efeito, implementámos as *queries* exigidas, tais como: a consulta das zonas com maior volume de entregas, a identificação do estafeta que utilizou um meio de transporte mais ecológico com mais frequência, o cálculo da classificação média atribuída a um estafeta pelos clientes e a determinação do valor faturado num determinado dia.

De modo a proteger a consistência da base de conhecimento, delineou-se uma quantidade de invariantes que teriam de ser respeitados, tanto na inserção, como na remoção de conhecimento. Tudo isto, à junção de algumas funcionalidades extras idealizadas, será detalhadamente explicitado ao longo deste relatório.

2 Preliminares

Para desenvolver esta primeira fase com êxito, foi indispensável o recurso aos conteúdos lecionados em Inteligência Artificial. Ao utilizar a linguagem de programação em lógica extendida, utilizou-se os dois valores lógicos seguintes:

- **Verdadeiro** - quando se verifica a existência de um termo na Base de Conhecimento.
- **Falso** - quando se verifica a negação de um termo ou, então, quando um termo não se encontra na Base de Conhecimento.

Adicionalmente, foi necessário ter em consideração alguns princípios:

- **Pressuposto dos Nomes Únicos** - entidades diferentes deverão possuir IDs diferentes a fim de poderem ser representados como objetos diferentes.
- **Pressuposto do Mundo Fechado** - não admite a existência de factos verdadeiros para além daqueles que se encontram na Base de Conhecimento.

3 Descrição do Trabalho e Análise de Resultados

O trabalho prático está estruturado num total de 6 partes, em que cada uma manipulará conhecimento acerca do universo abordado. A partir da Figura 1, consegue-se compreender com mais clareza a relação entre as várias partes previamente mencionadas.

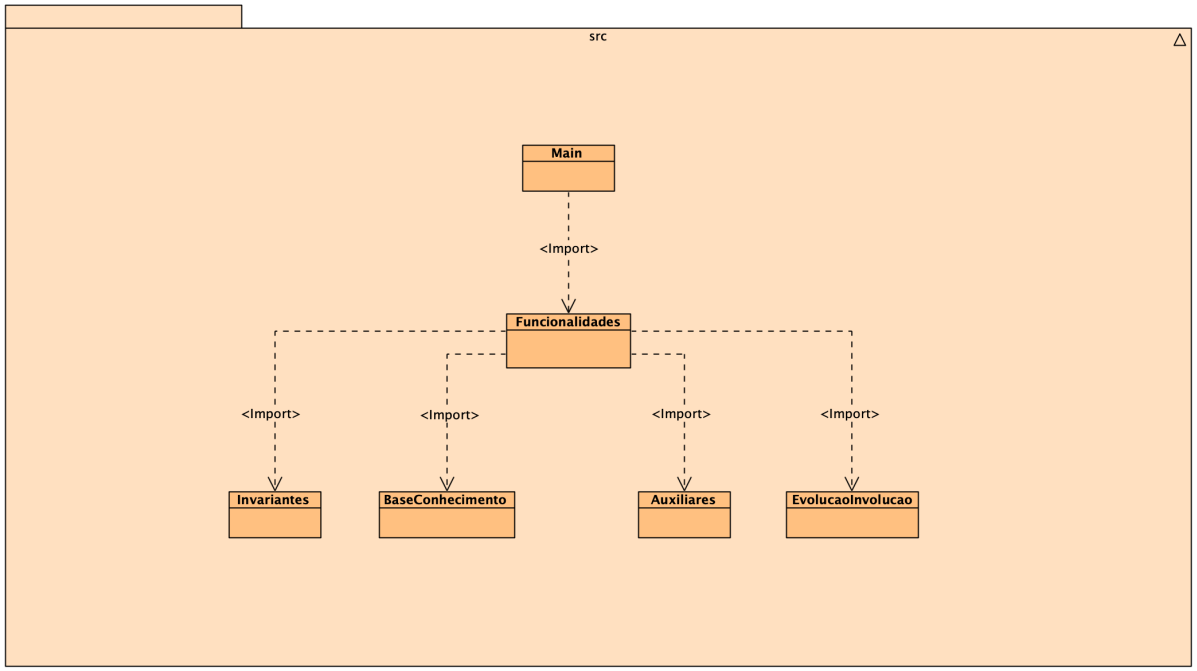

Figura 1: Diagrama da Estrutura do Trabalho

O ficheiro ***main.pl*** contém um menu onde são apresentadas todas as funcionalidades do sistema ao utilizador. Para as poder executar, este ficheiro tem de importar o ***funcionalidades.pl***, onde estão incorporados os predicados referentes a cada uma das funcionalidades exigidas e, ainda, a outras adicionais. No ***auxiliares.pl***, encontram-se, tal como o nome sugere, todos os predicados construídos como auxílio à elaboração das várias *queries* de forma a aproximarem-se o máximo possível da implementação mais eficiente.

Além destes ficheiros, temos o ***baseConhecimento.pl*** que contém todo o conhecimento do nosso sistema. Para poder ser implementada a mecância de inserção e remoção de conhecimento em ***evolucaoInvolucao.pl***, é imperativa a existência de invariantes que garantem a consistência da nossa Base de Conhecimento e, consequentemente, foi o criado o ficheiro ***invariantes.pl***.

3.1 Base de Conhecimento

A base de conhecimento tem como fontes de conhecimento as seguintes entidades:

- **encomenda**: IdEncomenda, IdCliente, Peso, Volume, Prazo, DataInicio, DataFim $\rightsquigarrow$ {$\mathbb{V}$,$\mathbb{F}$}
- **cliente**: IdCliente $\rightsquigarrow$ {$\mathbb{V}$,$\mathbb{F}$}
- **estafeta**: IdEstafeta, ListaEncomendas $\rightsquigarrow$ {$\mathbb{V}$,$\mathbb{F}$}

3.1.1 Encomenda

Cada encomenda é caracterizada pelo seu identificador (um inteiro de 12 dígitos), pelo identificador do cliente que a encomendou, pelo peso, pelo volume, pelo prazo de entrega ('Imediato', '2h', '3h', '6h', '12h, '1 dia', '3 dias'), pela data do início da entrega (sempre válida) e pela data do fim da entrega.

Ao longo deste trabalho prático, foi exigido trabalhar com intervalos de tempo e, por esse motivo, foi adotada a configuração data(Ano, Mes, Dia, Hora, Minuto) – ou, então, a configuração data(Ano, Mes, Dia) somente para duas *queries* – de forma a facilitar a manipulação e formatação de datas. Verifica-se que uma encomenda não chegou a ser, efetivamente, entregue se a dataFim for representada como data(0,0,0,0,0).

```
encomenda(100000000001,1,10,20,'6 horas',data(2021,7,25,10,0),data(2021,7,25,15,30)).
encomenda(123456789000,4,16,15,'Imediato',data(2021,8,1,12,30),data(2021,8,1,12,35)).
encomenda(300145999366,3,5,30,'2 horas',data(2021,4,10,17,20),data(0,0,0,0,0)). %não entregue
encomenda(411188745632,3,30,25,'3 horas',data(2021,5,29,13,10),data(2021,5,29,16,0)).
encomenda(512200534686,4,1,25,'6 horas',data(2021,6,30,9,0),data(2021,6,30,16,45)). %com atraso
encomenda(611111154895,2,70,40,'1 dia',data(2021,4,16,15,40),data(2021,4,18,15,0)). %com atraso
encomenda(792555468332,2,4,10,'3 horas',data(2021,2,28,16,12),data(0,0,0,0,0)). %não entregue
encomenda(800000000234,5,97,52,'12 horas',data(2021,6,24,8,30),data(2021,6,24,20,0)).
encomenda(910928382779,4,8,10,'4 horas',data(2021,8,1,14,0),data(2021,8,1,17,10)).
encomenda(100910101098,6,4,17,'1 hora',data(2021,1,31,12,27),data(2021,1,31,12,45)).
encomenda(113364968333,2,18,67,'1 dia',data(2021,12,19,21,29),data(2021,12,20,20,23)).
encomenda(121203928222,7,10,28,'Imediato',data(2021,10,13,11,43),data(2021,10,14,8,33)). %com atraso
encomenda(136666733413,5,20,41,'6 horas',data(2021,3,28,12,12),data(2021,3,28,15,53)).
encomenda(142019203922,2,7,21,'3 horas',data(2021,6,12,18,49),data(0,0,0,0,0)). %não entregue
encomenda(150393815151,8,30,77,'12 horas',data(2021,1,5,13,59),data(2021,1,5,22,12)).
encomenda(169998372344,4,87,87,'1 dia',data(2021,2,13,00,00),data(2021,2,14,13,00)).
encomenda(172123212430,9,9,18,'2 horas',data(2021,11,7,12,12),data(2021,11,7,13,45)).
```

3.1.2 Cliente

Um cliente é caracterizado pelo seu identificador (tal como a encomenda).

```
cliente(1).
cliente(2).
cliente(3).
cliente(4).
cliente(5).
cliente(6).
cliente(7).
cliente(8).
cliente(9).
```

3.1.3 Estafeta

Cada estafeta é representado pelo seu identificador e pela lista de entregas que fez. Inicialmente, o estafeta não estava caracterizado desta forma, porém, à medida que se iam elaborando as funcionalidades do nosso sistema, detetámos algumas dificuldades em associar um estafeta às encomendas que entregou. Assim sendo, incorporámos esta estrutura em busca de uma melhor organização do conhecimento.

Cada elemento desta lista é um conjunto de dados acerca de uma determinada entrega. O conjunto é, então, definido pelo identificador da encomenda em questão, pela classificação de satisfação atribuída pelo cliente, pela velocidade de deslocação, pelo meio de tansporte usado, pela rua e pela freguesia onde a entrega é feita.

```
estafeta(1,[(792555468332,0,10,'Bicicleta','Rua da Arcela','Gualtar')]).
estafeta(2,[(300145999366,0,10,'Bicicleta','Rua de São Victor-O-Velho','São Victor'),
            (100910101098,4.2,10,'Bicicleta','Rua das Mimosas','Gualtar'),
            (100000000001,4.7,10,'Bicicleta','Parque da Rodovia','São Victor')]).
estafeta(3,[(123456789000,4.7,35,'Mota','Rua José Antunes Guimarães','Gualtar'),
            (169998372344,4.7,25,'Carro','Rua da Graciosa','Esporões')]).
estafeta(4,[(512200534686,3.9,10,'Bicicleta','Av. Conde Dom Henriques','Maximinos'),
            (910928382779,4.3,35,'Mota','Av. São Pedro de Maximinos','Maximinos'),
            (113364968333,5.0,35,'Mota','Largo Senhora-A-Branca','São Victor'),
            (136666733413,5.0,35,'Mota','Largo do Bairro','Tadim'),
            (411188745632,4.3,25,'Carro','Ponte dos Falcões','Maximinos')]).
estafeta(5,[(611111154895,2.8,10,'Bicicleta','Rua da Veiga','Dume'),
            (150393815151,4.8,25,'Carro','Rua de Santo André','São Vicente')]).
estafeta(6,[(800000000234,4.9,25,'Carro','Rua 15 de maio','Gualtar'),
            (172123212430,4.9,25,'Carro','Rua da Boavista','Maximinos'),
            (142019203922,0,10,'Bicicleta','Rua da Arcela','Gualtar')]).
estafeta(7,[(121203928222,2.9,10,'Bicicleta','Largo Carlos Amarante','Maximinos')]).
```

3.2 Adição e Remoção de Conhecimento

A existência de invariantes, que especificam diversas restrições imprescindíveis à preservação da veracidade da base de informação, é o que possibilita a inserção e a remoção de conhecimento. Enquanto que a inserção de predicados é analisada como uma evolução de sistema, a remoção de um predicação já é vista como uma involução do sistema. Por conseguinte, sempre que algo é inserido ou removido da base de informação, a consistência do sistema é testada à custa dos invariantes. Caso alguma operação provoque anomalias, o mesmo voltará ao seu estado anterior.

```
% Após um termo ser inserido, é testada a consistência do sistema através de todos os
% invariantes relativos à inserção desse termo.
evolucao(Termo) :-
    solucoes(Invariantes,+Termo::Invariantes,Lista),
    insercao(Termo),
    teste(Lista).

% Após se verificar a existência do termo, este é removido e, posteriormente, é testada a
% consistência do sistema a partir de todos os invariantes relativos à remoção desse termo.
% Na situação do termo em questão não existir na base de conhecimento, então o processo
% de involução falha.
involucao(Termo) :- Termo,
            solucoes(Invariantes, -Termo::Invariantes, Lista),
            remove(Termo),
            teste(Lista).

% Insere um termo na base de conhecimento.
insercao(Termo) :- assert(Termo).
insercao(Termo) :- retract(Termo),!,fail.

% Remove um termo da base de conhecimento.
remocao(Termo) :- retract(Termo).
remocao(Termo) :- assert(Termo), !, fail.

% Testa a consistência do sistema.
teste([]).
teste([R|LR]) :- R, teste(LR).
```

3.2.1 Invariantes Gerais

Com o intuito de manutenir a coerência da base de informação, não é permitida a inserção de conhecimento positivo quando esse termo em questão já existe sobre a forma de conhecimento negativo. O mesmo se aplica à inserção de conhecimento negativo. Deste modo, os dois invariantes seguintes procuram evitar a inserção de contradições.

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

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

3.2.2 Invariantes Sobre Encomenda

Conferindo foco à inserção de entidades encomenda, o **primeiro** invariante prova que cada um dos campos do termo a inserir respeita o tipo de dados estipulado: o ID da encomenda, o ID do cliente, o peso e o volume devem ser inteiros, o prazo um átomo e a data do início da entrega válida. Em relação à data do fim da entrega, esta pode ser válida se a encomenda tiver sido entregue ou, então, inválida, caso contrário; daí não termos definido nenhuma restrição em relação à mesma.

```
% Invariante estrutural: os campos da Encomenda devem respeitar este tipo de dados
+encomenda(IdEnc,IdCliente,Peso,Volume,Prazo,DataInicio,DataFim) :: (
    integer(IdEnc),
    integer(IdCliente),
    integer(Peso),
    integer(Volume),
    atom(Prazo),
    dataTimeValida(DataInicio)).
```

O **segundo** invariante proíbe a entrada de um termo cujo identificador já exista no sistema a fim de evitar encomendas repetidas. Os restantes campos podem não ser únicos.

```
% Invariante estrutural: não permitir a entrada de uma encomenda cujo ID já exista
+encomenda(IdEnc,_,_,_,_,_,_) :: (
    solucoes(IdEnc,encomenda(IdEnc,_,_,_,_,_,_),L),
    comprimento(L,1)).
```

O **terceiro** invariante a respeito da inserção de conhecimento não permite a entrada de uma encomenda que esteja associada a um cliente que não exista no sistema. Isto impossibilitaria a finalização da entrega visto ser desconhecido o destinatário.

```
% Invariante estrutural: não permitir a entrada de uma encomenda que esteja associada
% a um cliente que não exista
+encomenda(IdEnc,IdCliente,_,_,_,_,_) :: (
    solucoes(IdCliente,encomenda(IdEnc,IdCliente,_,_,_,_,_),L),
    comprimento(L,1)).
```

Por fim, o **último** invariante impede a remoção desta entidade caso se encontre na lista de encomendas entregues por um estafeta. Para o efeito, é necessário aceder à lista de encomendas de todos os estafetas do sistema para comprovar que, de facto, a encomenda não pertence a nenhuma.

```
% Invariante referencial: não permitir a remoção de uma encomenda que esteja na
% lista de encomendas entregues por um estafeta
-encomenda(IdEnc,_,_,_,_,_,_) :: (
    solucoes(IdEstaf,estafeta(IdEstaf,_),ListaEstaf),
    encomendaNaoTemEstafeta(ListaEstaf,IdEnc)).
```

3.2.3 Invariantes Sobre Cliente

No que diz respeito à inserção e remoção de entidades cliente, temos 2 invariantes estruturais e 1 invariante referencial. O **primeiro** invariante assemelha-se a um da entidade encomenda, aferindo que o ID do cliente tem de ser um inteiro.

```
% Invariante estrutural: o ID de um cliente deve ser um inteiro
+cliente(IdCliente) :: (
    integer(IdCliente)).
```

O **segundo** invariante proíbe a inserção de um cliente cujo ID já exista no sistema, não autorizando que haja repetição de termos.

```
% Invariante estrutural: não permitir a entrada de um cliente cujo ID já exista
+cliente(IdCliente) :: (
    solucoes(IdCliente,cliente(IdCliente),L),
    comprimento(L,1)).
```

O **único** invariante referencial sobre esta entidade não permite a remoção de um cliente que tenha feito uma encomenda; de outra forma, o processo de entrega não poderia ser finalizado se o destinatário fosse retirado da base de informação.

```
% Invariante referencial: não permitir a remoção de um cliente que esteja associado
% a uma encomenda
-cliente(IdCliente) :: (
    solucoes(IdCliente,encomenda(_,IdCliente,_,_,_,_,_,_),L),
    comprimento(L,0)).
```

3.2.4 Invariantes Sobre Estafeta

Finalmente, falta enunciar os invariantes sobre a entidade estafeta. Para começar, o **primeiro** invariante estrutural assegura que, tal como para as outras entidades, cada um dos campos do termo a inserir respeita o tipo de dados estipulado: o ID do estafeta deve ser um inteiro e, dentro de cada um dos elementos da lista, o ID da encomenda deve ser um inteiro, a classificação um float, a velocidade um inteiro e o meio de transporte, a rua e a freguesia têm de ser átomos.

```
% Invariante estrutural: os campos do Estafeta devem respeitar este tipo de dados
+estafeta(IdEstaf,Lista) :: (
    integer(IdEstaf),
    verificaDadosLista(Lista)).
```

O **segundo** invariante não permite a inserção de um estafeta cujo ID já exista na base de informação de modo a evitar a repetição de conhecimento.

```
% Invariante estrutural: não permitir a entrada de um estafeta cujo ID já exista
+estafeta(IdEstaf,_) :: (
    solucoes(IdEstaf,estafeta(IdEstaf,_),L),
    comprimento(L,1)).
```

Uma das condições definidas, nesta primeira fase, foi as limitações em relação aos meios de transporte, no que concerne à velocidade e ao peso da encomenda a carregar. As bicicletas só podem transportar, a uma velocidade média de 10km/h, encomendas cujo peso não ultrapassa os 5kg; as motos apenas se podem deslocar a 35km/h, carregando, no máximo, um peso de 20kg; e os carros têm uma velocidade média de 25km/h e conseguem aguentar encomendas com um limite máximo de 100kg. Assim, o **terceiro** invariante proíbe a entrada de um estafeta cujo meio de transporte não suporta o peso da encomenda ou a velocidade de deslocação.

```
% Invariante estrutural: não permitir a entrada de um estafeta cujo meio de transporte
% não suporte o peso da encomenda ou a velocidade de deslocação
+estafeta(IdEstaf,Lista) :: (
    transportePesoVelocidade(Lista)).
```

O **quarto** invariante não permite a inserção de estafetas cujas avaliações relativas a cada encomenda não estejam de acordo com os limites estabelecidos. Primeiramente, através do predicado ***verificaClafMaiorQueX***, é feita a verificação de que todas as avaliações não ultrapassam o limite calculado com base no atraso da entrega da encomenda associada. De seguida, através do predicado ***verificaClafZero*** é verificado que todas encomendas não entregues têm zero como classificação associada.

```
% Invariante estrutural: não permitir a entrada de um estafeta cujas classficações
% não estejam de acordo com os atrasos nas entregas
+estafeta(IdEstaf,Lista) :: (
    verificaClafMaiorQueX(Lista),verificaClafZero(Lista)).
```

Tendo em consideração que, ao registar a conclusão de uma entrega, é importante identificar o responsável pela operação, o **último** invariante impossibilita a remoção de uma entidade estafeta que tenha feito entregas.

```
% Invariante referencial: não permitir a remoção de um estafeta que tenha feito entregas
-estafeta(IdEstaf,Lista) :: (
    estafeta(IdEstaf,[])).
```

3.3 Funcionalidades

3.3.1 Funcionalidade 1

"Identificar o estafeta que utilizou mais vezes um meio de transporte mais ecológico."

O predicado ***estafetaMaisEcologico*** identifica o estafeta que utilizou mais vezes o meio de transporte mais ecológico – a bicicleta –, recorrendo ao uso de alguns predicados auxiliares.

Inicialmente, através do predicado ***solucoes***, obtemos os IDs de todos os estafetas do sistema e, em seguida, com o ***encomendasPorBicicleta***, devolvemos o número de encomendas que foram transportadas de bicicleta por um estafeta. Depois de aplicarmos o mesmo a todos os estafetas da lista, devolvemos aquele que utilizou mais vezes a bicicleta como meio de transporte.

```
estafetaMaisEcologico(Ids) :-
        solucoes(IdEstaf,estafeta(IdEstaf,_),ListaEstaf),
        estafetaMaisEcologico(ListaEstaf,Max,Ids).

estafetaMaisEcologico([],0,[]).
estafetaMaisEcologico([IdEstaf],Max,[IdEstaf]) :-
        encomendasPorBicicleta(IdEstaf,Contador), Max = Contador.
estafetaMaisEcologico([IdEstaf|T],Max,IdsMax) :-
        encomendasPorBicicleta(IdEstaf,Contador),
        estafetaMaisEcologico(T,ContadorMax,Ids),
        ((Contador > ContadorMax -> Max = Contador, IdsMax = [IdEstaf]);
        (Contador == ContadorMax -> Max = ContadorMax, IdsMax = [IdEstaf|Ids]);
         Max = ContadorMax, IdsMax = Ids).
```

3.3.2 Funcionalidade 2

"Identificar que estafetas entregaram determinada(s) encomenda(s) a um determinado cliente."

O predicado ***estafetasEncomendasCliente*** identifica que estafetas entregaram determinadas encomendas, pelo que recebe como argumento a lista que contém essas mesmas encomendas.

Através do predicado ***estafetaEncCliente*** conseguimos obter o estafeta que entregou a encomenda que se encontra na cabeça da lista. Posto isto, ***estafetasEncomendasCliente*** é chamado recursivamente e, no fim de tudo, são adicionados os pares com o ID da encomenda e o estafeta que fez a sua entrega.

```
estafetasEncomendasCliente([],[]).
estafetasEncomendasCliente([IdEnc],L) :- estafetaEncCliente(IdEnc,R), adiciona((IdEnc,R),L1,L).
estafetasEncomendasCliente([IdEnc|Es],L) :-
        estafetaEncCliente(IdEnc,R), estafetasEncomendasCliente(Es,L1), adiciona((IdEnc,R),L1,L).
```

3.3.3 Funcionalidade 3

"Identificar os clientes servidos por um determinado estafeta."

O predicado ***clientesPorEstafeta*** identifica os clientes servidos por um determinado estafeta e, por isso, recebe o ID desse estafeta e recorre ao uso de alguns predicados auxiliares.

Para começar, através do ***encomendasDoEstafeta***, obtemos os IDs das encomendas de um estafeta e, posteriormente, o predicado ***listaClientesDasEnc*** retorna os IDs dos clientes que estão associados aos IDs dessas encomendas. Para terminar, o predicado ***sort*** ordena crescentemente a lista dos IDs dos clientes.

```
clientesPorEstafeta(IdEstaf,ListaR) :-
    encomendasDoEstafeta(IdEstaf,Lista0),
    listaClientesDasEnc(Lista0,Lista1),
    sort(0,@<,Lista1,ListaR).
```

3.3.4 Funcionalidade 4

"Calcular o valor faturado pela *Green Distribution* num determinado dia."

O predicado ***valorFaturadoDia*** calcula o valor faturado num determinado dia.

Primeiramente, é obtida a lista com todas as encomendas feita num dia, através do predicado ***encomendasDia***. Posto isto, esta lista é, então, passada ao predicado ***precosListaEncomendas*** que, para cada encomenda, calcula o seu preço, adicionando-o a uma nova lista. Depois, é feita a soma de todos os seus elementos, obtendo-se o total.

```
valorFaturadoDia(A,M,D,V) :-
        encomendasDia(A,M,D,L),
        precosListaEncomendas(L,R),
        totalEncomendas(R,V).
```

3.3.5 Funcionalidade 5

"Identificar quais as zonas (e.g., rua ou freguesia) com maior volume de entregas por parte da *Green Distribution*."

O predicado ***freguesiasMaisFrequentes*** identifica as freguesias com maior frequência de entregas por parte da *Green Distribuition*, recorrendo ao uso de alguns predicados auxiliares.

Antes de tudo, através do ***solucoes***, obtemos os IDs de todos os estafetas do sistema e, com esses IDs, o predicado ***todasAsFreguesias*** guarda, numa lista, todas as freguesias em que foram feitas entregas. O ***contaEntregasFreguesias*** devolve o número de encomendas entregues numa

freguesia e, por isso, depois de aplicarmos o mesmo a todas as freguesias, devolvemos aquela com o maior número de entregas.

```
freguesiasMaisFrequentes(ListaMaisFrequentes) :-
        solucoes(IdEstaf,estafeta(IdEstaf,_),ListaEstaf),
        todasAsFreguesias(ListaEstaf,ListaTodasFreg),
        freguesiasMaisFrequentes(ListaTodasFreg,ListaEstaf,Max,ListaMaisFrequentes).

freguesiasMaisFrequentes([],_,0,[]).
freguesiasMaisFrequentes([Freguesia],ListaEstaf,Max,[Freguesia]) :-
    contaEntregasFreguesia(Freguesia,ListaEstaf,Contador),
    Max = Contador.
freguesiasMaisFrequentes([Freguesia|T],ListaEstaf,Max,ListaMaisFrequentes) :-
    contaEntregasFreguesia(Freguesia,ListaEstaf,Contador),
    freguesiasMaisFrequentes(T,ListaEstaf,ContadorMax,ListaFregAux),
    ((Contador > ContadorMax -> Max = Contador, ListaMaisFrequentes = [Freguesia]);
     (Contador == ContadorMax -> Max = ContadorMax, ListaMaisFrequentes = [Freguesia|ListaFregAux]);
     (Contador < ContadorMax, Max = ContadorMax, ListaMaisFrequentes = ListaFregAux)).
```

3.3.6 Funcionalidade 6

"Calcular a classificação média de satisfação de cliente para um determinado estafeta."

O predicado ***mediaSatisfacaoEstafeta*** calcula a classificação média de um estafeta recebendo, por isso, o ID do estafeta como argumento.

Em primeiro lugar, é chamado o predicado ***classificacoesDoEstafeta***, que nos permite obter a lista com todas as classificações recebidas pelo estafeta. Em seguida, é feita a soma de todos os elementos da lista e calculado o seu comprimento. Finalmente, a média é dada dividindo o total da soma pelo comprimento da lista.

```
mediaSatisfacaoEstafeta(IdEstaf,Media) :-
        classificacoesDoEstafeta(IdEstaf,L),
        soma(L,S),
        comprimento(L,C),
        Media is S / C.
```

3.3.7 Funcionalidade 7

"Identificar o número total de entregas pelos diferentes meios de transporte, num determinado intervalo de tempo."

O predicado ***numeroTotalEntregasTransporte*** calcula o número total de entregas pelos diferentes meios de transporte, num determinado período de tempo. Assim sendo, devolve 3 valores:

o número de entregas por carro, por mota e por bicicleta. Para tal, este recebe a data inicial e a data final do dito intervalo de tempo e recorre, ainda, ao uso de alguns predicados auxiliares.

Inicialmente, utilizamos o predicado ***solucoes*** para devolver os IDs de todas as encomendas do sistema e, seguidamente, através do ***contaEntregasIntervalo***, extraímos a lista dos IDs das encomendas entregues naquele intervalo de tempo. Finalmente, o ***contaPorTransporteIntervalo*** verifica quais foram os meios de transporte usados na distribuição dessas encomendas e calcula o número total respetivo a cada um desses meios.

```
numeroTotalEntregasTransporte(data(AI,MI,DI,HI,MinI),data(AF,MF,DF,HF,MinF),
                              ContaCarro,ContaMota,ContaBicicleta) :-
    solucoes(IdEstaf,estafeta(IdEstaf,_),ListaEstaf),
    contaEntregasIntervalo(data(AI,MI,DI,HI,MinI),data(AF,MF,DF,HF,MinF),ListaEntregasPeriodo,_),
    contaPorTransporteIntervalo(ListaEstaf,ListaEntregasPeriodo,data(AI,MI,DI,HI,MinI),
                              data(AF,MF,DF,HF,MinF),ContaCarro,ContaMota,ContaBicicleta).
```

3.3.8 Funcionalidade 8

"Identificar o número total de entregas pelos estafetas, num determinado intervalo de tempo."

O predicado ***totalEntregasEstafetas*** calcula o número total de encomendas entregues, num dado intervalo de tempo.

Para tal, é primeiramente usado o predicado ***solucoes*** que nos permite obter todas as encomendas existentes. Esta lista é, depois, passada ao predicado ***contaEntregasIntervalo***, juntamente com as datas de início e de fim, que retorna, então, o total de entregas nesse intervalo.

```
totalEntregasEstafetas(data(AI,MI,DI,HI,MinI),data(AF,MF,DF,HF,MinF),Total) :-
        solucoes(IdEnc,encomenda(IdEnc,_,_,_,_,_,_),Encs),
        contaEntregasIntervalo(Encs,data(AI,MI,DI,HI,MinI),data(AF,MF,DF,HF,MinF),Lista),
        comprimento(Lista,Total).
```

3.3.9 Funcionalidade 9

"Calcular o número de encomendas entregues e não entregues pela *Green Distribution*, num determinado período de tempo."

O predicado ***numEntregasNaoEntregas*** calcula o número total de encomendas entregues, não entregues e nunca entregues pela *Green Distribution*, num determinado período de tempo. Para tal, este recebe a data inicial e a data final do dito intervalo de tempo e recorre, ainda, ao uso de alguns predicados auxiliares.

Primeiramente, a partir do ***solucoes***, extraímos a lista dos IDs de todas as encomendas do sistema e, posteriormente, com o objetivo de determinar o número de encomendas entregues naquele intervalo de tempo, aplicamos o predicado ***contaEntregasIntervalo***. Para terminar, o ***contaNaoEntreguesIntervalo*** calcula o número de encomendas não entregues durante aquele período e o número daquelas nunca entregues.

```
%Contador1: número de encomendas entregues naquele intervalo de tempo
%Contador2: número de encomendas não entregues naquele intervalo de tempo
%Contador3: número de encomendas nunca entregues

numEntregasNaoEntregas(data(AI,MI,DI,HI,MinI),data(AF,MF,DF,HF,MinF),
                           Contador1,Contador2,Contador3) :-
    solucoes(IdEnc,encomenda(IdEnc,_,_,_,_,_,_),ListaTodasEnc),
    contaEntregasIntervalo(data(AI,MI,DI,HI,MinI),data(AF,MF,DF,HF,MinF),
                           ListaEntregasPeriodo,Contador1),
    contaNaoEntregasIntervalo(ListaTodasEnc,data(AI,MI,DI,HI,MinI),data(AF,MF,DF,HF,MinF),
                           Contador2,Contador3).
```

3.3.10 Funcionalidade 10

"Calcular o peso total transportado por estafeta num determinado dia."

O predicado ***pesoTotalEstafetasDia*** calcula, para cada estafeta, o peso total transportado num determinado dia.

Para começar é invocado o predicado ***encomendasDia*** que nos dá a lista de todas as encomendas realizadas no dia em questão. Após isso, é chamado ***listaPesoTotalDia***, que cria uma lista de pares com cada estafeta e o respetivo peso total transportado.

```
pesoTotalEstafetasDia(A,M,D,L) :-
  encomendasDia(A,M,D,L1),
  listaPesoTotalDia(L1,[],R),
  sort(R,L).
```

3.3.11 Funcionalidade Extra1

"Identificar o cliente que fez mais encomendas."

O predicado ***clienteMaisEncomendas*** indica o ID do cliente que fez mais encomendas, entregues pela *Green Distribuition*.

Começámos, então, por usar o predicado ***solucoes*** para devolver os IDs de todos os clientes do sistema e, seguidamente, extraímos a lista de encomendas de cada um. Aquele cliente, cuja lista de encomendas tiver maior comprimento, será o valor resultante deste predicado.

```
clienteMaisEncomendas(Ids) :-
        solucoes(IdCliente,cliente(IdCliente),ListaClientes),
    clienteMaisEncomendas(ListaClientes,Max,Ids).

clienteMaisEncomendas([],0,[]).
clienteMaisEncomendas([IdCliente],Max,[IdCliente]) :-
    solucoes(IdEnc,encomenda(IdEnc,IdCliente,_,_,_,_,_),ListaEncCliente),
    comprimento(ListaEncCliente,Contador),
    Max = Contador.
clienteMaisEncomendas([IdCliente|T],Max,ListaIdMax) :-
    solucoes(IdEnc,encomenda(IdEnc,IdCliente,_,_,_,_,_),ListaEncCliente),
    comprimento(ListaEncCliente,Contador),
    clienteMaisEncomendas(T,ContadorMax,ListaAux),
    ((Contador > ContadorMax -> Max = Contador, ListaIdMax = [IdCliente]);
     (Contador == ContadorMax -> Max = ContadorMax, ListaIdMax = [IdCliente|ListaAux]);
     Max = ContadorMax, ListaIdMax = ListaAux).
```

3.3.12 Funcionalidade Extra2

"Identificar os estafetas que são menos pontuais, ou seja, quais têm um maior rácio entre encomendas não entregues/entregues com atraso e encomendas entregues."

O predicado ***estafetasMenosPontuais*** identifica, tal como o nome assim o sugere, os estafetas que são menos pontuais.

Em primeiro lugar, são obtidos os IDs de todos os estafetas, bem como a lista de encomendas não entregues e entregues com atraso. Através do predicado ***racioEstafetasAux***, é calculado o maior rácio existente entre todos os estafetas. De seguida, é invocado o predicado ***estafetasMaiorRacio***, que procura e guarda, numa lista, todos os estafetas cujo rácio seja igual ao calculado anteriormente.

```
estafetasMenosPontuais(L) :-
        getIdsEstafetas(Etfs),
    encomendasNaoEntreguesEAtrasadas(R),
    racioEstafetasAux(Etfs,R,Ratio),
    estafetasMaiorRacio(Ratio,R,L).
```

3.4 Main

O predicado ***main*** é responsável por interagir com o utilizador, criando um *loop*, em que são apresentadas as 12 funcionalidades do sistema e, ainda, a opção de sair do *loop*. O utilizador só tem de escolher a opção que pretende executar e, dependendo da funcionalidade, poderá ter de introduzir dados necessários para a mesma poder ser executada.

```
main :-

    repeat, nl,
    write('---------------------------------MENU---------------------------------'), nl,

    write('1. Consultar o estafeta que utilizou mais vezes um meio de transporte
              mais ecológico.'), nl,
    write('2. Consultar os estafetas que entregaram determinada(s) encomenda(s)
              a um determinado cliente.'), nl,
    write('3. Consultar os clientes servidos por um determinado estafeta.'), nl,
    write('4. Calcular o valor faturado pela Green Distribution num determinado dia.'), nl,
    write('5. Consultar quais as zonas (e.g., rua ou freguesia) com maior volume de
              entregas por parte da Green Distribution.'), nl,
    write('6. Calcular a classificação média de satisfação de cliente para um
              determinado estafeta.'), nl,
    write('7. Consultar o número total de entregas pelos diferentes meios de
              transporte, num determinado intervalo de tempo.'), nl,
    write('8. Consultar o número total de entregas pelos estafetas, num determinado
              intervalo de tempo.'), nl,
    write('9. Calcular o número de encomendas entregues e não entregues pela Green
              Distribution, num determinado período de tempo.'), nl,
    write('10. Calcular o peso total transportado por estafeta num determinado dia.'), nl,
    write('11. Consultar o cliente que fez mais encomendas.'), nl,
    write('12. Consultar os estafetas menos pontuais a fazer as suas entregas.'),nl,
    write('0. Sair'), nl,

    write('--------------------------------------------------------------------------'), nl,nl,

    write('Escolha um: '),
    read(X),
    (X = 0 -> !, fail; true), nl,
    funcionalidade(X), fail.
```

Com o predicado ***funcionalidade***, será executada a opção feita pelo utilizador que representa uma das *queries* do sistema. A seguir, é mostrado, como exemplo, a implementação deste predicado para a funcionalidade 10, consoante os seus requisitos:

```
funcionalidade(10) :-
    write('Indique uma data no formato - data(Ano,Mês,Dia): '),!,nl,
    read(Input),nl,
    ((dataValida(Input),
    elementosData(Input,A,M,D),
    pesoTotalEstafetasDia(A,M,D,L),nl,
    write('Peso total transportado por estafeta em '),
    write(D),write('/'),write(M),write('/'),write(A),write(': '),write(L),nl);
    write('A data que inseriu não é válida.'),nl),!.
```

Na Figura 2, é ilustrado um exemplo de como funciona a interface do nosso sistema.

```
?- main.

------------------------------------------------------MENU------------------------------------------------------
1. Consultar o estafeta que utilizou mais vezes um meio de transporte mais ecológico.
2. Consultar os estafetas que entregaram determinada(s) encomenda(s) a um determinado cliente.
3. Consultar os clientes servidos por um determinado estafeta.
4. Calcular o valor faturado pela Green Distribution num determinado dia.
5. Consultar quais as zonas (e.g., rua ou freguesia) com maior volume de entregas por parte da Green Distribution.
6. Calcular a classificação média de satisfação de cliente para um determinado estafeta.
7. Consultar o número total de entregas pelos diferentes meios de transporte, num determinado intervalo de tempo.
8. Consultar o número total de entregas pelos estafetas, num determinado intervalo de tempo.
9. Calcular o número de encomendas entregues e não entregues pela Green Distribution, num determinado período de tempo.
10. Calcular o peso total transportado por estafeta num determinado dia.
11. Consultar o cliente que fez mais encomendas.
12. Consultar os estefetas menos pontuais a fazer as suas entregas.
0. Sair
----------------------------------------------------------------------------------------------------------------

Escolha um:
|: 10.

Indique uma data no formato - data(Ano,Mês,Dia):
|: data(2021,5,29).

Peso total transportado por estafeta em 29/5/2021: [(4,30)]
```

Figura 2: Exemplo da Interface

3.5 Auxiliares

No ficheiro *auxiliares.pl*, são apresentados os predicados auxiliares que foram cruciais para a resolução desta primeira fase, permitindo uma implementação mais eficiente das *queries* do projeto e, ainda, facilitando a análise das mesmas.

Seguidamente, serão explicadas as estratégias adotadas pelo grupo para trabalhar a penalização de um estafeta quando este não cumpre o prazo de entrega e definir o preço de uma encomenda.

3.5.1 Preço da Encomenda

Para fazer o cálculo do preço de cada encomenda foi criado o predicado ***precoEncomenda*** apresentado de seguida.

```
precoEncomenda(IdEnc,P) :-
        encomenda(IdEnc,_,Peso,Vol,Prazo,_,_),
        transporteEncomenda(IdEnc,Transporte),
        calculaPrecoPorTransporte(Transporte,PT),
        getPrazoEncomendaHoras(Prazo,Horas),
        calculaPrecoPorPrazo(Horas,PP),
        P is 3*Peso + 2*Vol + PT + PP.
```

Primeiramente é calculado o preço com base no transporte utilizado.

```
calculaPrecoPorTransporte(Transporte,P) :-
        (Transporte == 'Bicicleta' -> P is 5;
          Transporte == 'Mota' -> P is 10;
          Transporte == 'Carro' -> P is 15).
```

De seguida, é calculado o preço com base no prazo, que foi previamente transformado em horas, caso já não estivesse. Para tal, este foi organizado da seguinte maneira: caso este seja inferior a 7 horas, terá o seu valor específico, caso contrário, o seu valor será calculado de acordo com o intervalo de tempo em que se encontrar.

```
calculaPrecoPorPrazo(PrazoH,Preco) :-
        (PrazoH < 7 -> calculaPrecoAte7Horas(PrazoH,Preco);
         PrazoH < 13 -> Preco is 8;
         PrazoH < 25 -> Preco is 5;
         PrazoH < 73 -> Preco is 4;
         PrazoH < 121 -> Preco is 3;
         Preco is 2).
```

O preço total da encomenda é calculado com base no peso, volume, transporte e prazo.

```
P is 3*Peso + 2*Vol + PT + PP.
```

3.5.2 Penalização do Estafeta

A implementação da penalização de um estafeta se este entregar uma encomenda com atraso ou, então, não a entregar, é feito através do predicado ***estafetaPenalizacao***. Este permite calcular a classificação máxima que um estafeta poderá obter, relativamente à entrega de uma encomenda, conforme o seu atraso.

```
estafetaPenalizacao(DataI,DataF,Prazo,Max):-
        calculaAtraso(DataI,DataF,Prazo,A),
        calculaPenalizacaoPorAtraso(A,P),
        Max is 5-5*P.
```

Primeiramente é invocado o predicado ***calculaAtraso***. Este recebe a data de pedido da encomenda, a data de entrega da encomenda e o prazo estipulado para a sua entrega e, com isto, calcula o atraso na entrega.

```
calculaAtraso(DataI,DataF,Prazo,Atraso) :-
      getPrazoEncomendaHoras(Prazo,PH),
      diferencaDatas(DataI,DataF,H),
      (H =< PH -> Atraso is 0;
       H > PH  -> Atraso is H-PH).
```

De seguida, é chamado o predicado ***calculaPenalizacaoPorAtraso***, que recebe o atraso na entrega de uma encomenda e decide qual a penalização a ser aplicada.

```
calculaPenalizacaoPorAtraso(Atraso,Penalizacao) :-
        (Atraso < 1 -> Penalizacao is 0;
         Atraso < 24 -> Penalizacao is 0.2;
     Atraso < 48 -> Penalizacao is 0.3;
     Penalizacao is 0.5).
```

Como auxiliares do invariante que não permite a entrada de estafetas que não tenham penalizações corretas, temos, ainda, os seguintes predicados.

O predicado ***verificaClafMaiorQueX*** percorre a lista de todas encomendas associadas ao estafeta e, confirma que, caso uma encomenda tenha sido entregue com atraso, a sua avaliação correspondente está dentro dos limites.

```
verificaClafMaiorQueX([]).
verificaClafMaiorQueX([(IdEnc,Nota,_,_,_,_)]) :-
    encEntregueAtraso(IdEnc), encomenda(IdEnc,_,_,_,Prazo,DataI,DataF),
    estafetaPenalizacao(DataI,DataF,Prazo,P), Nota =< P.
```

```
verificaClafMaiorQueX([(IdEnc,_,_,_,_,_)]) :- nao(encEntregueAtraso(IdEnc)).
verificaClafMaiorQueX([(IdEnc,Nota,_,_,_,_) | T]) :-
    (encEntregueAtraso(IdEnc) ->
        ((encomenda(IdEnc,_,_,_,Prazo,DataI,DataF),
        estafetaPenalizacao(DataI,DataF,Prazo,P), Nota =< P) ->
        verificaClafMaiorQueX(T); fail);
    verificaClafMaiorQueX(T)).
```

O predicado ***verificaClafZero***, tal como no predicado anterior, percorre a lista de todas as encomendas associadas ao estafeta, só que, nesta circunstância, confirma que, caso uma encomenda não tenha chegado ao seu destinatário, a classificação correspondente é igual a 0.

```
verificaClafZero([]).
verificaClafZero([(IdEnc,0,_,_,_,_)]) :- encNaoEntregue(IdEnc).
verificaClafZero([(IdEnc,_,_,_,_,_)]) :- nao(encNaoEntregue(IdEnc)).
verificaClafZero([(IdEnc,Nota,_,_,_,_) | T]) :-
    (encNaoEntregue(IdEnc) -> (Nota == 0 -> verificaClafZero(T); fail); verificaClafZero(T)).
```

4 Conclusão

A realização desta primeira frase do trabalho prático permitiu consolidar e pôr em prática os conceitos abordados nas aulas, relativos à programação em lógica.

Sentimos que a sua realização foi concluída com sucesso, uma vez que foram implementadas todas as *queries* pedidas, juntamente com duas extras, e, também, uma base de conhecimento. Adicionalmente, criámos os predicados que permitem calcular o preço de uma encomenda e penalizar um estafeta caso este se atrase na entrega de uma encomenda. Para concluir, adicionámos invariantes de modo a possibilitar a inserção e remoção de conhecimento.

As nossas maiores dificuldades concentraram-se na implementação dos predicados relativos ao preço de encomenda e penalização do estafeta já que estes exigiram uma maior complexidade relativamente às restantes funcionalidades. No entanto, achamos que fomos capazes de as ultrapassar e apresentar um bom resultado.

Como trabalho futuro, gostaríamos apenas de implementar mais algumas funcionalidades extra e, talvez, melhorar um pouco a eficiência das já existentes.

5 Referências Bibliográficas

- Stuart Russell e Peter Norvig
  Artificial Intelligence - A Modern Approach, 4ª edição.
- Cesar Analide, Paulo Novais e José Neves
  Sugestões para a Elaboração de Relatórios, novembro de 2001.