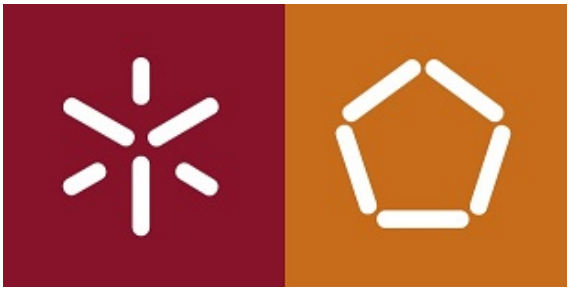
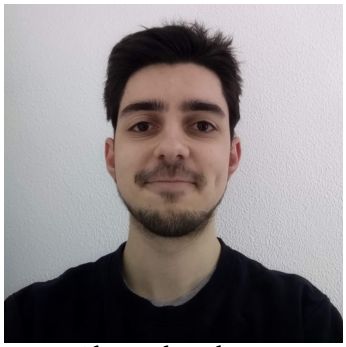


Universidade do Minho
Departamento de Informática

Inteligência Artificial
Grupo 5

2 de dezembro, 2021


Alexandre Flores
(a93220)


Mariana Rodrigues
(a93294)


Matilde Bravo
(a93246)


Pedro Alves (a93272)

Conteúdo

1 Introdução **3**
1.1 Descrição do Trabalho 3

2 Implementação **4**
2.1 Base de conhecimento 5
2.1.1 Freguesia 5
2.1.2 Rua 6
2.1.3 Morada 6
2.1.4 Transporte 6
2.1.5 Estafeta 7
2.1.6 Ranking 7
2.1.7 Cliente 7
2.1.8 Encomenda 8
2.1.9 Serviço 8
2.2 Queries 9
2.2.1 Query 1 9
2.2.2 Query 2 9
2.2.3 Query 3 10
2.2.4 Query 4 10
2.2.5 Query 5 10
2.2.6 Query 6 11
2.2.7 Query 7 11
2.2.8 Query 8 11
2.2.9 Query 9 12
2.2.10 Query 10 12
2.3 Funções Auxiliares 13
2.4 Funcionalidades Extra 15
2.4.1 Identificações 15
2.4.2 Invariantes 18
2.4.3 Adicionar Conhecimento 22
2.4.4 Remover Conhecimento 22
2.4.5 Guardar Estado 23
2.4.6 Outras funcionalidades 25

3 Conclusão **28**

Capítulo 1

Introdução

O presente relatório descreve o trabalho realizado no âmbito da Unidade Curricular de Inteligência Artificial.

Serão apresentados e devidamente justificados os predicados que foram criados para a realização de *queries* bem como os invariantes que permitem garantir a devida gestão da base de conhecimento. Serão também demonstradas as funcionalidades extra implementadas.

1.1 Descrição do Trabalho

Tal como descrito no enunciado, este projeto de *Prolog* reflete a gestão e consulta de conhecimento relativo à empresa de entregas *Green Distribution*.

Através dos predicados desenvolvidos na linguagem construímos uma série de ferramentas que nos permitem obter conhecimento válido inferido da base de conhecimento por nós definida, resolvendo assim as *queries* listadas no enunciado.

A base de conhecimento pode ser manipulada dinamicamente no **swiprolog**, através das funcionalidades de introdução e remoção de conhecimento desenvolvidas. Estas funcionalidades são suportadas por verificações de regras de validade para a base de conhecimento (os invariantes, também descritos neste relatório).

Capítulo 2

Implementação

O presente trabalho foi dividido em vários ficheiros por forma a obter uma melhor organização.

- **funcoes_auxiliares.pl**

  Contém predicados que, só por si, não são uma *query* ou funcionalidade principal do programa, sendo utilizados por outros predicados que o são.

- **queries.pl**

  Contém tanto os predicados das *queries* que foram propostas no enunciado como aqueles que das *queries* que foram por nós adicionadas.

- **base_dados.pl**

  Contém os predicados de factos acerca da nossa base de conhecimento (informações de clientes, meios de transporte, etc).

- **pesquisa.pl**

  Contém predicados que permitem pesquisas simples de conhecimento, como por exemplo pesquisas de freguesias por nome ou custo de visita.

- **writeStructs.pl**

  Contém predicados responsáveis por escrever, numa dada stream, o conhecimento atual do programa. Desta forma, é possível construir uma de base de conhecimento persistente guardando num ficheiro o seu estado atual.

- **trabalho.pl**

  O ficheiro principal, que inclui todos os outros ficheiros do projeto. É responsável também tanto pelos predicados relacionados com a adição e remoção de conhecimento, como por exemplo os invariantes.

```
:- include('base_dados.pl').
:- include('funcoes_auxiliares.pl').
:- include('pesquisa.pl').
:- include('writeStructs.pl').
:- include('queries.pl').
```

2.1 Base de conhecimento

Para uma melhor representação do sistema de conhecimento para este companhia de entregas, começamos primeiramente por estabelecer o tipo de conhecimento proposto criando predicados que o representam.

Finalmente, concluímos que iríamos precisar de definir:

- **freguesia**

  Para representar todas as freguesias conhecidas pelo sistema, bem como informações relevantes às mesmas.

- **rua**

  Para associar as ruas por nós conhecidas às suas freguesias.

- **transporte**

  Para representar todos os transportes que a *Green Distribution* possui, bem como as suas especificações.

- **estafeta**

  Para representar todos os trabalhadores da *Green Distribution* e os seus detalhes.

- **ranking**

  Para poder penalizar um estafeta caso este não cumprir o prazo determinado para a entrega de uma dada encomenda. Este *rank* representa a confiança atribuída a um estafeta, podendo este vir a diminuir com penalizações.

- **cliente**

  Para representar todos os clientes da *Green Distribution*, bem como as suas informações pessoais.

- **encomenda**

  Para representar todas as encomendas realizadas pelos clientes, quer estas já tenham ou não sido entregues.

- **serviço**

  Para representar um serviço, ou seja, toda a informação associada com o transporte e entrega da encomenda após a sua colocação pelo cliente.

2.1.1 Freguesia

```
% freguesia: Nome, Custo , Tempo: Horas/Minutos -> {V,F}
```

A **freguesia** é identificada através de um **nome**, estando a ela associados um **custo e tempo de visita**.

```
freguesia('Sao Vitor',13, 0/6 ).
freguesia('Nogueiró',20,0/9).
freguesia('Merelim',31, 0/14).
freguesia('Frossos',19, 0/8).
freguesia('Caldelas',60, 0/28).
freguesia('Briteiros',53, 0/25).
freguesia('Priscos',31,0/14).
freguesia('Adaúfe',28,0/11).
freguesia('Maximinos',6,0/3).
freguesia('Esporões',28,0/11).
freguesia('Lamas',26,0/10).
```

2.1.2 Rua

```
% rua: Nome, Nome da freguesia -> {V,F}
```

A **rua** é caracterizada por um **nome** e por uma dada **freguesia**.

```
rua('Rua do Taxa','Sao Vitor').
rua('Avenida Dom João II','Nogueiró').
rua('Rua de São Bento','Merelim').
rua('Rua Doutor José Alves Correia Da Silva','Frossos').
rua('Rua do Moinho','Caldelas').
rua('Rua Dr. Lindoso','Briteiros').
rua('Rua do Coucão','Priscos').
rua('Rua da Mota','Adaúfe').
rua('Rua Joãozinho Azeredo','Maximinos').
rua('Rua de Santa Marta','Esporões').
rua('Rua do Sol','Lamas').
```

2.1.3 Morada

A **morada** foi construída de forma a posteriormente associarmos uma **morada** a um **cliente**.

```
% morada: Nome da rua, Nome da freguesia -> {V,F}

morada(R,F):- rua(R,F), freguesia(F,_,_).
```

2.1.4 Transporte

```
% transporte: ID, Nome, Velocidade Média, Carga Máxima,
%              Nível Ecológico -> {V,F}
```

Cada **transporte** é identificado por um **ID** único, tendo também campos de **nome**, **velocidade média**, **carga máxima** e **nível ecológico**.

```
transporte(1,'bicicleta',10,5,5).
transporte(2,'mota',35,20,-2).
transporte(3,'carro',25,100,-5).
transporte(4,'bicicleta elétrica',40,15,3).
transporte(5,'bicicleta elétrica',40,15,3).
transporte(6,'bicicleta elétrica',40,15,3).
transporte(7,'carro',25,100,-5).
transporte(8,'carro',25,100,-5).
transporte(9,'carro',25,100,-5).
transporte(10,'carro',25,100,-5).
transporte(11,'mota',35,20,-2).
transporte(12,'mota',35,20,-2).
transporte(13,'mota',35,20,-2).
transporte(14,'mota',35,20,-2).
transporte(15,'bicicleta',10,5,5).
transporte(16,'bicicleta',10,5,5).
transporte(17,'bicicleta',10,5,5).
```

2.1.5 Estafeta

```
% estafeta: ID, Nome -> {V,F}
```

O **estafeta** é identificado por um **ID** único e tem ainda de um campo de **nome**.

```
estafeta(1,'Bernardo').
estafeta(2,'Sofia').
estafeta(3,'Costa').
estafeta(4,'Filipe').
estafeta(4,'Joana').
estafeta(5,'Filipa').
estafeta(6,'Pedro').
estafeta(7,'João').
estafeta(8,'Ricardo').
estafeta(8,'Catarina').
estafeta(9,'Mafalda').
estafeta(10,'Martim').
```

2.1.6 Ranking

```
% ranking: ID Estafeta, Classificação -> {V,F}
```

A um **ranking** é associado um dado ID de **estafeta** e a sua respetiva **classificação**.

```
ranking(1,5).
ranking(2,4.3).
ranking(3,3.2).
ranking(4,1.3).
ranking(5,2.6).
ranking(6,4.2).
ranking(7,4.6).
ranking(8,3.5).
ranking(9,4.9).
ranking(10,4.4).
```

2.1.7 Cliente

```
% cliente: ID, Nome, Morada -> {V,F}
```

O **cliente** é identificado por um **ID** único, e tem também associado a si um **nome** e uma **morada**.

```
cliente(1,'Alexandra Epifânio', morada('Rua do Taxa','Sao Vitor')).
cliente(2,'Filipa Simão'   ,   morada('Rua do Coucão','Priscos')).
cliente(3,'Ana Reigada',  morada('Rua de Santa Marta','Esporões')).
cliente(4,'Maria Dinis',    morada('Santo Antonio','Taipas')).
cliente(5,'Mafalda Bravo', morada('Velha','Briteiros')).
cliente(6,'Alexandre Rosas', morada('Rua da Mota','Adaúfe')).
```

2.1.8 Encomenda

```
% encomeda: ID, Id_Cliente, Peso, Volume, DiaPedido: D/M/Y/H/M,
%           Limite: D/H, -> {V,F}
```

A **encomenda** é identificada por um **ID** único, um ID de **cliente**, um **peso**, um **volume**, o **dia** em que o pedido foi efetuado e um **limite** de entrega.

```
encomenda(1, 1 ,7,  2,  10/11/2021/8/35, 0/2 ).
encomenda(2, 1 ,5,  10, 22/11/2021/23/45, 4/0 ).
encomenda(3, 3 ,2,  2,  23/11/2021/16/02, 1/0 ).
encomenda(4, 5 ,5,  2,  24/11/2021/9/03,  0/2 ).
encomenda(5, 6 ,10, 10, 03/12/2021/11/30, 4/0 ).
encomenda(6, 7 ,25,  2,  03/12/2021/17/0, 1/0 ).
encomenda(7, 3 ,20,  2,  03/12/2021/18/23, 2/0 ).
encomenda(8, 4 ,17,  2,  04/12/2021/17/0, 0/10 ).
encomenda(9, 2 ,80,  2,  10/12/2021/17/0, 0/8 ).
encomenda(10, 7 ,3,  2,  11/12/2021/17/0, 0/9 ).
```

2.1.9 Serviço

```
% servico: ID, Id_estafeta, Id_encomenda, Id_transporte,
%          DiaEntrega: D/M/Y/H/M, Classificacao -> {V,F}
```

O **serviço** é identificado por um **ID** único, ID de um **estafeta**, ID de **encomenda**, ID de **transporte**, **dia** em que a encomenda foi entregue e uma **classificação**.

```
servico(1, 3, 1, 3, 10/11/2021/9/11, 5).
servico(2, 4, 3, 1, 24/11/2021/9/0, 4).
servico(3, 1, 2, 15, 24/11/2021/14/0, 3).
servico(4, 2, 4, 1, 24/11/2021/20/43, 3).
servico(5, 6, 5, 2, 03/12/2021/13/00, 5).
servico(6, 2, 6, 7, 04/12/2021/07/03 ,4).
servico(8, 5, 8, 11, 05/12/2021/08/21 ,3).
servico(7, 3, 7, 14, 05/12/2021/18/23 ,1).
```

2.2 Queries

Como pedido no enunciado, implementamos várias *queries* que operam sobre a nossa base de conhecimento. Estas *queries* estão definidas em predicados no ficheiro **queries.pl**, usando predicados auxiliares desse e de outros ficheiros.

Estas *queries*, numeradas de 1 a 10, estão organizadas pela ordem em que se encontram no enunciado.

2.2.1 Query 1

```
estafetaServicoEcologico(servico(_,IdEstafeta,_,IdT,_,_),
↪  IdEstafeta):-
    nivelEcologicoByIdTransporte(IdT,E),
    -1 < E.

estafetasServicoEcologico([], []).
estafetasServicoEcologico([X|T], [ID|R]) :-
    estafetaServicoEcologico(X,ID),
    estafetasServicoEcologico(T,R).
estafetasServicoEcologico([X|T], R) :-
    not(estafetaServicoEcologico(X,_)),
    estafetasServicoEcologico(T,R).

estafetaMaisEcologico(R):-
    findall(servico(ID,E,Enc,T,D,C),servico(ID,E,Enc,T,D,C),L),
    estafetasServicoEcologico(L,LS),
    maxFreq(LS,IdEstafeta),
    estafetaById(IdEstafeta,R).
```

Para a resolução desta *query*, encontramos primeiro todos os serviços realizados pela companhia. filtramos aqueles que foram realizados usando meios de transporte ecológicos (Nível Ecológico positivo ou nulo), obtendo a lista de todos os estafetas que realizaram "serviços ecológicos". Posteriormente, utilizando o predicado **maxFreq**, encontramos o estafeta mais frequente dessa lista, assim concluindo a *query*.

2.2.2 Query 2

```
estafetasQueEntregaram(IdsEncomendas, R) :-
↪  maplist(estafetaQueEntregou, IdsEncomendas , R).

estafetaQueEntregou(IdEncomenda, R) :-
    servico(_,IdEstafeta,IdEncomenda,_,_,_),
    findall(estafeta(IdEstafeta,Nome),
    estafeta(IdEstafeta,Nome),[R|_]).
```

Para esta *query* usamos dois predicados. O primeiro unifica a **R** a lista de Estafetas que entregaram as encomendas identificadas pelos **IdsEncomendas**, através do uso de um **maplist** e do segundo predicado.

O segundo predicado unifica a **R** o estafeta que entregou a encomenda identificada por **IdEncomenda**. Se esta encomenda não foi entregue, o predicado falha e o primeiro predicado também falhará por consequência.

2.2.3 Query 3

```
clientesServidosIdEstafeta(ID,C):-
    estafetaById(ID,_), % verificar se existe estafeta
    findall(IdEncomenda,servico(_,ID,IdEncomenda,_,_,_),L),
    maplist(clienteByIdEncomenda,L,C_),
    eliminaRepetidos(C_,C).
```

Nesta *query* primeiro verificamos a existência do estafeta associado ao **ID**. Posteriormente, encontramos todos os **IdEncomenda** para quais as respetivas encomendas foram entregues (se foram entregues, ou seja, se existe um serviço para essa encomenda) por este estafeta. Finalmente, encontramos a lista de clientes que efetuaram essas encomendas sem duplicados.

2.2.4 Query 4

```
valorFaturado(Dia/Mes/Ano,Valor) :-
    findall(servico(A,B,C,D,Dia/Mes/Ano/Hora/Minuto,F),
    servico(A,B,C,D,Dia/Mes/Ano/Hora/Minuto,F),L),
    servicos_para_custo(L,Valor).

servicos_para_custo([],0).
servicos_para_custo([servico(_,_,IDEnc,_,_,_)|S],Total) :-
    encomenda(IDEnc,IDCliente,_,_,_,_),
    cliente(IDCliente,_,morada(_,Freguesia)),
    freguesia(Freguesia,Custo,_),
    servicos_para_custo(S,Resto),
    Total is Custo + Resto.
```

A *query* 4 procura obter o somatório dos custos de visita às freguesias para todos os serviços efetuados no dia fornecido. Para isso encontramos primeiro todos os serviço executados no dia, e convertemos essa lista de serviço a um custo. Para tal efeito utilizamos um predicado auxiliar que percorre a lista de serviços e soma todos os seus custos para um total, o **Valor**.

2.2.5 Query 5

```
freguesiasMaisFrequentes(R) :-
    freguesiasEncomendas(Freguesias),
    freq(Freguesias, [], Freqs),
    sort(1, @>=, Freqs, R).

moradasMaisFrequentes(R) :-
    moradasEncomendas(Moradas),
    freq(Moradas, [], Freqs),
    sort(1, @>=, Freqs, R).
```

A *query* 5 procura encontrar as freguesias/ruas com mais entregas. Para tal, foram criados dois predicados auxiliares, **freguesiasEncomendas** e **moradasEncomendas**, que encontram todas as freguesias ou moradas de encomendas, respetivamente. É importante notar que estes predicados não removem elementos duplicados, o que significa que é depois possível utilizar a função auxiliar **freq** para calcular quantas vezes cada morada aparece. Depois, basta ordenar por frequência e obtemos uma lista das moradas mais frequentes.

2.2.6 Query 6

```
classificacaoEstafeta(IdEstafeta, Media) :-
    findall(C, servico(_,IdEstafeta,_,_,_,C), [X|L]),
    avg([X|L], Media).
```

A *query* 5 permite encontrar a média das classificações atribuídas aos serviços executados pelo estafeta identificado por **IdEstafeta**. Para tal, usamos um **findall** para encontrar a lista de todas as classificações do estafeta, e posteriormente encontramos a média dessas classificações.

2.2.7 Query 7

```
total_entregas_por_transporte(Init,Fin,Freq) :-
    findall(servico(A,B,C,D,E,F),
            (servico(A,B,C,D,E,F),isBetween(E,Init,Fin)),
            Servicos),
    servicos_para_transportes(Servicos, Transportes),
    freq(Transportes,[],Freq).

servicos_para_transportes([],[]).
servicos_para_transportes(
        [servico(_,_,_,IdTrs,_,_)|Ss],
        [transporte(IdTrs,A,B,C,D)|Ts]
        ) :-
    transporte(IdTrs,A,B,C,D),
    servicos_para_transportes(Ss,Ts).
```

A *query* 7 identifica a tabela de frequências (lista de pares frequência-transporte) dos transportes usados nas entregas de encomendas de um dado intervalo de tempo. Para tal, encontramos primeiro todos os serviços efetuados no intervalo de tempo. Posteriormente, obtemos da lista de serviços a lista dos transportes que neles foram utilizados. Finalmente utilizamos a função auxiliar **freq** para obter a tabela de frequências de elementos dessa lista de transportes.

2.2.8 Query 8

```
getEstafeta(servico(_,ID,_,_,_,_), R) :-
    estafetaById(ID, R).

% Trocamos a ordem dos argumentos do isBetween para funcionar como
↪  filtro
isBetweenFilter(I, F, servico(_,_,_,_,Data,_)) :- isBetween(Data,
↪  I, F).

servicosEntre(I, F, Servicos) :-
    findall(
        servico(ID,IdE,IdEnc,IdTrans,Data,Class),
        servico(ID,IdE,IdEnc,IdTrans,Data,Class),
        Lista),
    include(isBetweenFilter(I, F), Lista, Servicos).

servicosPorEstafetaEntre(I, F, R) :-
    servicosEntre(I, F, Servicos),
    maplist(getEstafeta, Servicos, Estafetas),
```

```
        freq(Estafetas, [], Freqs),
        sort(1, @>=, Freqs, R).
```

A *query 8* determina os estafetas com mais entregas num determinado intervalo de tempo. Para isto, foi criado um predicado que encontra todos os serviços entre duas datas, que funciona procurando todos os serviços e removendo todos os que não estejam no intervalo das datas. Depois, é utilizado um método semelhante à *query* 5, criando uma lista com os estafetas responsáveis por este serviço, gerando a tabela de frequências e finalmente ordenando esta lista.

2.2.9 Query 9

```
todasEncomendas(I,F,Entregues,NaoEntregues) :-
    encomendasEntregues(I,F,Entregues),
    encomendasNaoEntregues(Entregues,NaoEntregues).

encomendasEntregues(I,F,R) :-
    findall(E,
    foiEntregueEntre(E,I,F),
    R).

encomendasNaoEntregues(Entregues,R) :-
    findall(encomenda(ID,A,B,C,D,E),
    encomenda(ID,A,B,C,D,E), Todas),
    subtract(Todas,Entregues,R).

foiEntregueEntre(encomenda(ID,A,B,C,D,E), I,F) :-
    encomenda(ID,A,B,C,D,E), servico(_,_,ID,_,Data,_),
    isBetween(Data,I,F).

isBetween(D/Mon/Y/H/Min, D1/Mon1/Y1/H1/Min1, D2/Mon2/Y2/H2/Min2) :-
    Y/Mon/D/H/Min @< Y2/Mon2/D2/H2/Min2,
    Y/Mon/D/H/Min @> Y1/Mon1/D1/H1/Min1.
```

A *query* 9 procura encontrar as listas de encomendas entregues e não entregues num dado intervalo de tempo. Para as encomendas entregues procuramos todos os serviços executados no intervalo de tempo para as encomendas entregues. Para as encomendas não entregues, subtraímos a todas as encomendas as que foram realizadas nesse intervalo, de forma a ficarmos apenas com as que não foram entregues. O índica da lista onde o estafeta se encontra corresponde ao índice da encomenda que este entregou.

2.2.10 Query 10

```
cargaEstafetaDia(Id1,D/M/Y,R):-
   findall(ID,servico(_,Id1,ID,_,D/M/Y/_/_,_),R1),
   maplist(cargaEncomendaById,R1,R).

totalCargaEstafetaDia(ID,D,R):-
   cargaEstafetaDia(ID,D,L),
   sum(L,R).

pesoTotalByEstafetaNoDia(estafeta(ID,Nome),D, R) :-
    estafeta(ID,Nome), % verificar se o estafeta é válido
    totalCargaEstafetaDia(ID,D,R).
```

```
tuplePesoTotalByEstafetaByDia(E,D,(E,P)):-
↪  pesoTotalByEstafetaNoDia(E,D,P).

pesoAllAux([],_,[]).
pesoAllAux([X|T],D,[ A | R ]):-
    tuplePesoTotalByEstafetaByDia(X,D,A),
    pesoAllAux(T,D,R).

pesoTotalAllEstafetaNoDia(D,R):-
    findall(estafeta(ID,Nome),estafeta(ID,Nome),L),
    pesoAllAux(L,D,R).

pesoTotalAux([],0).
pesoTotalAux([(_,N)|T], R):-
    pesoTotalAux(T,Rs),
    R is Rs + N.

pesoTotalByDia(D,R):-
    pesoTotalAllEstafetaNoDia(D,L),
    pesoTotalAux(L,R).
```

A *query* 10 começa por determinar, para todos os estafetas, o peso total de cada encomenda que cada um fez num dado dia. Com isto, para cada estafeta, basta fazer a soma de das cargas dessas encomendas obtidas. Apresentando o resultado numa lista de *tuples* (Estafeta, carga total no dado dia).

2.3 Funções Auxiliares

Ao longo do decorrer de todo o trabalho foi fundamental recorrermos a diversas funções auxiliares que nós foram extremamente úteis.

Como tal iremos fazer uma breve apresentação de cada uma delas.

O predicado **len**, que unifica *NS* ao comprimento de uma dada lista passada como argumento.

```
len([],0).
len([_|T],NS) :-
    len(T,N), NS is N+1.
```

O predicado **valid**, que testa se todos os predicados em *[A|T]* são verdadeiros.

```
valid([]).
valid([A|T]):- A, valid(T).
```

O predicado **inserir** serve para inserirmos *New* na base de conhecimento em caso de sucesso, e retira *New*, no caso de haver insucesso.

```
inserir(New):- assert(New).
inserir(New):- retract(New), !, fail.
```

O predicado **remover** faz o oposto do predicado **inserir**. Remove *X* da nossa base de conhecimento no caso de sucesso, e adiciona *X* no caso de haver retrocesso.

```
remover(X):- retract(X).
remover(X):- assert(X), !, fail.
```

O predicado **new_predicado** encontra-se responsável por adicionar novo conhecimento na base de conhecimento, certificando-se que todos os invariantes estruturais e referências de adição do conhecimento que se pretende adicionar continuam verdadeiros. Somente em caso de isso se verificar é que o novo conhecimento fica efetivamente registado, caso contrário, o conhecimento outrora inserido é removido.

```
new_predicado(P):-
    findall(X,+P::X,R),
    inserir(P),
    valid(R).
```

O predicado **remover_predicado** encontra-se responsável por remover conhecimento da base de conhecimento, garantindo que todos os invariantes estruturais e referenciais de remoção do conhecimento que se pretende remover *P* continuam verdadeiros. Somente em caso de sucesso é que o conhecimento é efetivamente removido, caso contrário, o conhecimento é novamente inserido.

```
remover_predicado(P):-
    findall(X,-P::X,R),
    remover(P),
    valid(R).
```

O predicado **iguais**, que coloca em *R* todos os elementos que pertencem a ambas as 2 listas dadas como argumento.

```
iguais([],_,[]).
iguais([X|T],L2,[X|R]):-
    member(X,L2), !,
    iguais(T,L2,R)
.
iguais([_|T],L2,R):-
    iguais(T,L2,R).
```

O predicado **eliminaRepetidos**, que coloca em *R* a lista fornecida sem elementos repetidos.

```
eliminaRepetidos(X, R) :- eliminaRepAux(X,[],R).

eliminaRepAux([],Acc,Acc).
eliminaRepAux([X|XS],Acc,R) :- member(X,Acc),!, eliminaRepAux(XS,Acc,R).
eliminaRepAux([X|XS],Acc,R) :- eliminaRepAux(XS,[X|Acc],R).
```

O predicado **sum** unifica *N* ao somatório de todos os valores de uma dada lista dada como argumento.

```
sum([],0).
sum([X|T],N):- sum(T,R), N is R+X.
```

O predicado **avg** unifica com *R* a média calculada a partir de uma dada lista fornecida.

```
avg(L, R) :- sum(L,Soma), length(L,Len), R is Soma/Len.
```

O predicado **freq** unifica com *R* todas as frequências de uma dada lista fornecida

```
% Como usar: freq([1,2,3,1,2,1,3,4], [], R).
freq([], R, R).
freq([X | Xs], Temp, R) :-
    select((N, X), Temp, Outros), % Encontra o elemento X e o seu
    ↪  numero de ocorrencias
    M is N + 1,
    freq(Xs, [ (M, X) | Outros], R).
freq([X | Xs], Temp, R) :-
    not(select((_, X), Temp, _)),
    freq(Xs, [ (1, X) | Temp], R).
```

O predicado **maxFreq** unifica com *R* o elemento com maior frequência de uma lista dada como argumento.

```
maxFreq(L,R):-
    freq(L,[],RL),
    sort(1, @>=, RL, [(_,R)|_]).
```

2.4 Funcionalidades Extra

2.4.1 Identificações

Identificar Freguesia

A **freguesia** pode ser identificada através dos seus 3 campos, *nome*, *custo* e *tempo*.

```
freguesiaByName(N,R):-
↪  findall(freguesia(N,C,T),freguesia(N,C,T),[R|_]).

freguesiaByCusto(C,R):-
↪  findall(freguesia(N,C,T),freguesia(N,C,T),R).

freguesiaByTempo(T,R):-
↪  findall(freguesia(N,C,T),freguesia(N,C,T),R).
```

Identificar Rua

A **rua** pode ser identificada a partir de qualquer um dos seus campos, *nome* e *nome de freguesia*.

```
ruaByName(N,R):- findall(rua(N,F),rua(N,F),[R|_]).

ruaByNomeFreguesia(N,R):- findall(rua(N,F),rua(N,F),R).

ruaByFreguesia(freguesia(F,_,_),R):- findall(rua(N,F),rua(N,F),R).
```

Identificar Transporte

O **transporte** pode ser identificado através dos seus 5 campos: *id*, *nome*, *velocidade máxima*, *carga* ou *nível ecológico*.

```
transporteById(ID,T):-
    findall(transporte(ID,N,V,C,P), transporte(ID,N,V,C,P),[T|_]).

transporteByName(N,T):-
    findall(transporte(ID,N,V,C,P), transporte(ID,N,V,C,P),T).

transporteByVelocidade(V,T):-
    findall(transporte(ID,N,V,C,P), transporte(ID,N,V,C,P),T).

transporteByCarga(C,T):-
    findall(transporte(ID,N,V,C,P), transporte(ID,N,V,C,P),T).

transporteByPontosEcologicos(P,T):-
    findall(transporte(ID,N,V,C,P), transporte(ID,N,V,C,P),T).
```

Decidimos ser possível obter o nível ecológico de um qualquer transporte através do seu ID.

```
nivelEcologicoByIdTransporte(ID,E):-
    transporte(ID,_,_,_,E).
```

De modo a conseguirmos obter somente os transportes ecológicos criamos o predicado **trasnportesEcologicos**

```
transportesEcologicos(R):-
    findall(transporte(I,N,V,C,P), (transporte(I,N,V,C,P), P >
    ↪  0),R).
```

Identificar Estafeta

Uma vez que o **estafeta** possui 2 campos, construímos 2 predicados capazes de o conseguir identificar. Um através do seu ID e outro através do seu nome.

```
estafetaById(ID,E):-
    findall(estafeta(ID,N),estafeta(ID,N),[E|_]).

estafetaByNome(N,E):-
    findall(estafeta(ID,N),estafeta(ID,N),[E|_]).
```

Identificar Ranking

O **ranking** pode ser obtido através de cada um dos seus dois campos, *id do estafeta* ou *classificação*.

```
rankingByIdEstafeta(ID,R):-
↪  findall(ranking(ID,C),ranking(ID,C),[R|_]).

rankingByClassificacao(C,R):-
↪  findall(ranking(ID,C),ranking(ID,C),R).
```

Identificar Cliente

O **cliente** pode ser identificado através do seu *id*, *nome* ou *morada*.

```
clienteById(ID, X):-
    findall(cliente(ID,N,M),cliente(ID,N,M),[X|_]).

clienteByNome(N, R):-
    findall(cliente(ID,N,M),cliente(ID,N,M),R).

clienteByMorada(M, R):-
    findall(cliente(ID,N,M),cliente(ID,N,M),R).
```

Fora isso, foram ainda criadas mais alguns predicados.

```
clienteByIdEncomenda(ID, R):-
    idClienteByIdEncomenda(ID, X),
    clienteById(X,R).

idClienteByIdEncomenda(ID, X):-
↪   findall(Id1,encomenda(ID,Id1,_,_,_,_),[X|_]).

clienteByIdServico(ID,C):-
    servico(ID,Id1,_,_,_,_),
    encomenda(Id1,CId,_,_,_,_),
    clienteById(CId,C).
```

Identificar encomenda

Uma vez que a **encomenda** possui 6 campos, foram construídos 6 predicados capazes de o conseguir identificar. Através do seu *id*, *id de cliente*, *peso*, *volume*, *dia de pedido*, *limmite*.

```
encomendaById(ID, X):-

    ↪  findall(encomenda(ID,Id1,P,V,D,L),encomenda(ID,Id1,P,V,D,L),[X|_]).

encomendaByIdCliente(ID,R):-
    findall(encomenda(Id1,ID,P,V,D,L),encomenda(Id1,ID,P,V,D,L),R).

encomendaByPeso(P,R):-
    findall(encomenda(Id1,ID,P,V,D,L),encomenda(Id1,ID,P,V,D,L),R).

encomendaByVolume(V,R):-
    findall(encomenda(Id1,ID,P,V,D,L),encomenda(Id1,ID,P,V,D,L),R).

encomendaByDiaDePedido(D/M/Y,R):-
    findall(encomenda(Id1,ID,P,V,D/M/Y/H/Min,L),
            encomenda(Id1,ID,P,V,D/M/Y/H/Min,L),R).

encomendaByDiaDePedidoCompelto(D,R):-
    findall(encomenda(Id1,ID,P,V,D,L),encomenda(Id1,ID,P,V,D,L),R).

encomendaByLimite(L,R):-
    findall(encomenda(Id1,ID,P,V,D,L),encomenda(Id1,ID,P,V,D,L),R).
```

Identificar Serviço

O **serviço** pode ser identificado através de cada um dos seus 6 campos, do seu *id*, *id de estafeta*, *id de encomenda*, *id de transporte*, *dia da entrega*, *classificação*.

```
servicoById(ID,R):-
    findall(servico(ID,Id1,E,T,D,C),servico(ID,Id1,E,T,D,C),[R|_]).

servicoByIdEstafeta(ID,R):-
    findall(servico(Id1,ID,E,T,D,C),servico(Id1,ID,E,T,D,C),R).

servicoByIdEncomenda(E,R):-
    findall(servico(ID,Id1,E,T,D,C),servico(ID,Id1,E,T,D,C),R).

servicoByIdTransporte(T,R):-
    findall(servico(ID,Id1,E,T,D,C),servico(ID,Id1,E,T,D,C),R).

servicoByDiaEntregaCompleto(D,R):-
    findall(servico(ID,Id1,E,T,D,C),servico(ID,Id1,E,T,D,C),R).

servicoByDiaEntrega(D/M/Y,R):-
    findall(servico(ID,Id1,E,T,D/M/Y/H/Min,C),
            servico(ID,Id1,E,T,D/M/Y/H/Min,C),R).

servicoByClassificacao(C,R):-
    findall(servico(ID,Id1,E,T,D,C),servico(ID,Id1,E,T,D,C),R).
```

2.4.2 Invariantes

Para conseguirmos garantir certas regras, tanto na adição como na remoção de conhecimento na base de dados, começamos por definir invariantes estruturais e referências para os predicados mencionados anteriormente.

Invariantes da Freguesia

```
%%% Freguesia %%%
% Garantir que o nome de cada freguesia é único
+freguesia(Nome,_,_) :: (findall(Nome,freguesia(Nome,_,_),R),
                         len(R,1)).

% Garantir que o Custo, as Horas e os Minutos inseridos são números
↪  válidos
+freguesia(_,Custo,H/M) :: (number(Custo),number(H),number(M)
                            , -1 < H, H < 24 , -1 < M, M < 60,
                            C > 0 ).

% Garantir que não é possível remover nenhuma freguesia que
↪  pertença a uma
% morada de algum cliente
-freguesia(Nome,_,_) ::
↪  (findall(Nome,cliente(_,_,morada(_,Nome)),R),
                       len(R,0)).
```

Invariantes da Rua

```
%%% Rua %%%
% Garantir que o nome de cada rua é único
+rua(Nome,_) :: (findall(Nome,rua(Nome,_),R),
                        len(R,1)).

% Garantir que o nome de freguesia dado é verdadeiro e existe na
↪  nossa base de conhecimento
+rua(_,NomeFreguesia) :: (freguesia(NomeFreguesia,_,_)).

% Garantir que não é possível remover nenhuma rua no caso de esta
↪  encontrar-se
% associada a alguma morada de um cliente
-rua(Nome,_) :: (findall(Nome,cliente(_,_,morada(Nome,_)),R),
                        len(R,0)).
```

Invariantes de Transporte

```
%%%  Transportes  %%%%
% Garantir que o ID dos transportes é único
+transporte(ID,_,_,_,_) :: (findall(ID,transporte(ID,_,_,_,_), R),
                            len(R,1)).

% Garantir que os dados inseridos encontram-se no formato certo
+transporte(ID,_,V,C,E) :: (number(ID),
↪  number(V),number(C),number(E)
                             ,-1 < V, -1 < C, -6 < E, E < 6
                            ).

% Garantir que um transporte só pode ser removido no caso de não
↪  estar presente
% em nenhum serviço
-transporte(ID,_,_,_,_) :: (findall(ID,servico(_,_,_,ID,_,_),R),
                            len(R,0)).
```

Invariantes de Estafeta

```
% Garantir que o ID dos estafetas é único
+estafeta(ID,_) :: (findall(ID,estafeta(ID,_),R),
                    len(R,1)).

% Garantir que os dados inseridos encontram-se no formato certo
+estafeta(ID,_) :: (number(ID)).

% Garantir que um estafeta só pode ser removido no caso de não
↪  estar associado a nenhum serviço
-estafeta(ID,_) :: (findall(ID,servico(_,ID,_,_,_,_),R),
                     len(R,0)).
```

Invariantes de Ranking

```
%%% Ranking %%%
% Garantir que só se consegue associar um ranking a um estafeta
↪  já existente
+ranking(ID,_) :: ( findall(ID,estafeta(ID,_),R),
                    len(R,N), 0 < N).

% Garantir que só existe 1 ranking para cada estafeta
+ranking(ID,_) :: ( findall(ID,ranking(ID,_),R), len(R,1)).

% Garantir que o valor do ranking encontra-se entre 0 e 5.
+ranking(ID,C) :: (number(ID),number(C), -1 < C, C < 6).
```

Invariantes de Cliente

```
%%% Cliente %%%
% Garantir que o ID dos clientes são únicos e a morada dada é
↪  válida.
+cliente(ID,_,Morada) :: (
     number(ID),
     Morada,
     findall(ID, cliente(ID,_,_), R),
     len(R,1)).

% Garantir que não é possível remover nenhum cliente que se
↪  encontre associado a uma dada encomenda.
-cliente(ID,_,_) :: (
     findall(ID, encomenda(_,ID,_,_,_,_),R),
     len(R,0)).
```

Invariantes de Encomenda

```
%%% Encomenda %%%
% Garantir que os dados inseridos encontram-se no formato correto.
+encomenda(ID,C,P,V,D/M/Y/H/Min,D1/H1) :: (
     number(ID),number(C), number(P),
     number(V),number(D), number(M),
     number(Y),number(H), number(D1),
     -1 < D, D < 32 ,
     -1 < M, M < 60,
     -1 < H, H < 24 ,
     -1 < P,
     -1 < V,
     -1 < Min, Min < 60,
     number(H1), number(Min),
     -1 < H1, H1 < 24).

% Garantir que o ID das encomendas é único e o cliente associado a
↪  ela é válido.
+encomenda(ID,C,_,_,_,_) :: (
```

```
        findall(ID,encomenda(ID,_,_,_,_,_),R),
        len(R,1),
        findall(C,cliente(C,_,_),R1),
        len(R1,1)
        ).

% Garantir que não é possível remover uma encomenda no caso de
↪  esta, se
% encontrar associada a um serviço
-encomenda(ID,_,_,_,_,_) :: (
        findall(ID,servico(_,_,ID,_,_,_),R),
        len(R,0)).
```

Invariantes de Serviço

```
%%% Serviço %%%
% Garantir que o ID dos serviços são únicos
+servico(ID,_,_,_,_,_) :: (
    findall(ID,servico(ID,_,_,_,_,_),R),
    len(R,1)).

% Garantir que os estafetas associados são válidos
+servico(_,E,_,_,_,_) :: (
    findall(E,estafeta(E,_),R1),
    len(R1,1)).

% Garantir que as encomendas associadas são válidos
+servico(_,_,Enc,_,_,_) :: (
    findall(Enc,encomenda(Enc,_,_,_,_,_),R2),
    len(R2,1)).

% Garantir que os transportes associados são válidos
+servico(_,_,_,T,_,_) :: (
    findall(T,transporte(T,_,_,_,_),R3), len(R3,1)).

% Garantir que os dados inseridos encontram-se no formato correto.
+servico(ID,E,Enc,T,D/M/Y/H/Min,C) :: (
     number(ID),number(E), number(Enc),
     number(T),number(D), number(M),
     number(Y),number(H), number(C),
     -1 < D, D < 32 ,
     -1 < M, M < 60,
     -1 < H, H < 24 ,
     -1 < Min, Min < 60,
     -1 < E,
     -1 < ID,
     -1 < Enc,
     -1 < C,
     number(Min)).
```

2.4.3 Adicionar Conhecimento

Para ser possível termos a funcionalidade de registar novo conhecimento, foi fundamental temos:

```
% Verificar se é valido
valid([]).
valid([A|T]):- A, valid(T).

% Inserir um novo conhecimento
inserir(New):- assert(New).
inserir(New):- retract(New), !, fail.

% Inserir um novo conhecimento verificando se este se encontra valido
new_predicado(P):-
    %% not(P),
    findall(X,+P::X,R),
    inserir(P),
    valid(R).
```

Através disso, fomos capazes de conseguir inserir novo conhecimento garantindo que, após essa inserção os invariantes definidos anteriormente permanecem válidos.

```
newRua(Nome,F):- new_predicado(rua(Nome,F)).

newFreguesia(Nome,C,T):- new_predicado(freguesia(Nome,C,T)).

newTransporte(ID,N,V,C,E):- new_predicado(transporte(ID,N,V,C,E)).

newEstafeta(ID,Nome):- new_predicado(estafeta(ID,Nome)).

newRanking(ID,Ranking):- new_predicado(ranking(ID,Ranking)).

newCliente(ID,Nome,M):- new_predicado(cliente(ID,Nome,M)).

newEncomenda(ID,C,P,V,D,L):- new_predicado(encomenda(ID,C,P,V,D,L)).

newServico(ID,E,En,C,D,C):- new_predicado(servico(ID,E,En,C,D,C)).
```

2.4.4 Remover Conhecimento

Da mesma forma que pretendemos ser possível adicionar conhecimento, queremos conseguir remove-lo. Para tal, foi crucial termos definido:

```
% Remover conhecimento
remover(X):- retract(X).
remover(X):- assert(X), !, fail.

% Remover um dado conhecimento, garantindo que este pode ser removido
remover_predicado(P):-
    findall(X,-P::X,R),
    remover(P),
    valid(R).
```

Com isto conseguimos garantir que não é permitido remover um qualquer conhecimento.

```
removeRua(Nome):- rua(Nome,F), remover_predicado(rua(Nome,F)).

removeFreguesia(Nome):- freguesia(Nome,C,T),
↪  remover_predicado(freguesia(Nome,C,T)).

removeTransporte(ID):- transporte(ID,N,V,C,E) , remover_predicado(
↪  transporte(ID,N,V,C,E)).

removeRanking(ID):- ranking(ID,C),remover_predicado(ranking(ID,C)).

removeEstafeta(ID):- estafeta(ID,Nome) , remover_predicado(
↪  estafeta(ID,Nome)).

removeCliente(ID):- cliente(ID,Nome,M),
↪  remover_predicado(cliente(ID,Nome,M)).

removeEncomenda(ID):- encomenda(ID,C,P,V,D,L),
↪  remover_predicado(encomenda(ID,C,P,V,D,L)).

removeServico(ID):-  servico(ID,E,En,C,D,C),
↪  encomenda(E,C1,P1,V1,D1,L1) ,
                    remover_predicado(servico(ID,E,En,C,D,C)),
                    remover_predicado(encomenda(E,C1,P1,V1,D1,L1)).
```

2.4.5 Guardar Estado

No caso de termos adicionado ou removido conhecimento, pretendemos que esse mesmo fique armazenado no nosso sistema. Para tal, construímos o predicado **save** que guarda no ficheiro **base_dados.pl** todos os factos que se encontram presentes no sistema.

Em nosso auxílio, recorremos a funções já existentes no prolog.

- **open**

  Abre o ficheiro com o nome dado em modo escrita, fornecendo a Stream responsável pelo *input/output*.

- **write**

  Serve para armazenar o conteúdo que pretendemos na Stream fornecida.

- **close**

  Por fim, o **close** serve para fechar a *Stream*, escrevendo o conteúdo na mesma .

```
saveIn(X) :-
    open(X,write,Stream),
    write(Stream, '\n% freguesia: Nome, Custo, Tempo: H/M -> {V,F}\n'),
    writeFreguesia(Stream),
    write(Stream, '\n% rua: Nome, Nome da Freguesia -> {V,F}\n'),
    writeRua(Stream),
    write(Stream,
'\n% transporte: ID, Nome, Velocidade Média, Carga Máxima, Nível Ecológico -> {V,F}\n'),
    writeTransporte(Stream),
    write(Stream, '\n% estafeta: ID, Nome -> {V,F}\n'),
    writeEstafeta(Stream),
    write(Stream, '\n% ranking: ID Estafeta, Classificação -> {V,F}\n'),
```

```
    writeRanking(Stream),
    write(Stream,
'\n% cliente: ID, Nome, morada(Nome da Rua,Nome da Freguesia) -> {V,F}\n'),
    writeCliente(Stream),
    write(Stream,
'\n% encomeda: ID, Id_Cliente, Peso, Volume, DiaPedido: D/M/Y/H/M, Limite: D/H, -> {V,F}\n'),
    writeEncomenda(Stream),
    write(Stream,
'\n% servico: ID, Id_estafeta, Id_encomenda, Id_transporte, DiaEntrega: D/M/Y/H/M, Classificacao
    writeServico(Stream),
    close(Stream)
.

save:- saveIn('base_dados.pl').
```

Visto cada um dos tipos de conhecimento armazenarem os seus próprios factos de forma diferente e única, cada um deles possui o seu próprio procedimento de guardar o conhecimento.

```
writeFreguesia(Stream):- freguesia(Nome,Custo,Tempo),
        write(Stream,'freguesia('),
        write(Stream,'\''), write(Stream,Nome), write(Stream,'\','),
        write(Stream,Custo), write(Stream, ','),
        write(Stream,Tempo), write(Stream, ').\n'),
        fail; true
.

writeRua(Stream):- rua(Nome,Freguesia),
        write(Stream,'rua('),
        write(Stream,'\''), write(Stream,Nome),
        write(Stream,'\','), write(Stream,'\''),
        write(Stream,Freguesia), write(Stream,'\').\n'),
        fail; true
.

writeTransporte(Stream):- transporte(ID,N,V,C,E),
        write(Stream,'transporte('),
        write(Stream,ID),
        write(Stream,',\''), write(Stream,N), write(Stream,'\','),
        write(Stream,V), write(Stream, ','),
        write(Stream,C), write(Stream, ','),
        write(Stream,E), write(Stream, ').\n'),
        fail; true
.

writeEstafeta(Stream):- estafeta(ID,Nome),
        write(Stream,'estafeta('),
        write(Stream,ID), write(Stream, ','),
        write(Stream,'\''), write(Stream,Nome), write(Stream,'\').\n'),
        fail; true
.

writeRanking(Stream):- ranking(ID,Nota),
        write(Stream,'ranking('),
        write(Stream,ID), write(Stream, ','),
        write(Stream,Nota), write(Stream, ').\n'),
```

```
        fail; true
.

writeCliente(Stream):- cliente(ID,Nome,morada(R,F)),
        write(Stream,'cliente('),
        write(Stream,ID), write(Stream, ','),
        write(Stream,'\''), write(Stream,Nome), write(Stream,'\','),
        write(Stream,'morada(\''),write(Stream,R), write(Stream,'\','),
        write(Stream,'\''),write(Stream,F), write(Stream,'\')).\n'),
        fail; true
.

writeEncomenda(Stream):- encomenda(ID, C, P,V,D,L),
        write(Stream,'encomenda('),
        write(Stream,ID), write(Stream, ','),
        write(Stream,C), write(Stream, ','),
        write(Stream,P), write(Stream, ','),
        write(Stream,V), write(Stream, ','),
        write(Stream,D), write(Stream, ','),
        write(Stream,L), write(Stream, ').\n'),
        fail; true
.

writeServico(Stream):- servico(ID,Es,E,T,D,C),
        write(Stream,'servico('),
        write(Stream,ID), write(Stream, ','),
        write(Stream,Es), write(Stream, ','),
        write(Stream,E), write(Stream, ','),
        write(Stream,T), write(Stream, ','),
        write(Stream,D), write(Stream, ','),
        write(Stream,C), write(Stream, ').\n'),
        fail; true
.
```

2.4.6 Outras funcionalidades

Cargas

Uma das funcionalidades extras é conseguirmos obter as cargas a partir de várias informações. Com isto, conseguimos obter a carga da encomenda através do seu ID.

```
cargaEncomendaById(ID,R):-
    findall(C,encomenda(ID,_,C,_,_,_),[R|_]).
```

Carga de todas as encomendas que um dado cliente fez, através do seuid.

```
cargaEncomendaByIdCliente(ID,R):-
    findall(P,encomenda(_,ID,P,_,_,_),R).
```

Carga de todos serviços que um dado estafeta realizou, através do seu ID.

```
cargaEstafeta(Id1,R):-
    findall(ID,servico(ID,Id1,_,_,_,_),R1), % buscar os ids das
    ↪  encomendas q ele realizou
    maplist(cargaEncomendaById,R1,R).
```

Carga de todos serviços que um dado estafeta realizou num certo dia, através do seu ID e de um dado dia.

```
cargaEstafetaDia(Id1,D/M/Y,R):-
    findall(ID,servico(_,Id1,ID,_,D/M/Y/_/_,_),R1), % buscar os ids
    ↪  das encomendas q ele realizou nesse dia
    maplist(cargaEncomendaById,R1,R).
```

Carga de todas as encomendas realizadas de um certo cliente, através do seu ID.

```
cargaCliente(Id1,R):-
    findall(ID,encomenda(ID,Id1,_,_,_,_),R2), % ids de todas as
    ↪  encomendas desse Cliente
    findall(ID,servico(_,_,ID,_,_,_),R1),     % ID de todas as
    ↪  encomendas feitas
    iguais(R1,R2,R3),
    maplist(cargaEncomendaById,R3,R).
```

Carga Total

Através de todos os predicados definidos outroora, conseguimos obter a carga total.

```
totalCargaEstafeta(ID,R):-
    cargaEstafeta(ID,L),
    sum(L,R).

totalCargaEstafetaDia(ID,D,R):-
    cargaEstafetaDia(ID,D,L),
    sum(L,R).

totalCargaEncomendaCliente(ID,R):-
    cargaEncomendaByIdCliente(ID,L),
    sum(L,R).

totalCargaServicoCliente(ID,R):-
    cargaCliente(ID,L),
    sum(L,R).
```

Identificar Estafetas que utilizaram um dado transporte

Através de um dado *ID* de um transporte somos capazes de identificar todos os *ids* dos estafetas que o usaram.

```
estafetasIdByIdTransporte(ID,E):-
    findall(Id1,servico(_,Id1,_,ID,_,_),E).
```

Identificar o Estafeta que mais utilizou um dado transporte

```
estafetaMaisUtilizouIdTransporte(ID,E):-
    findall(Id1,servico(_,Id1,_,ID,_,_),R),
    maxFreq(R,IdE),
    estafetaById(IdE,E),!.
```

```
estafetaMaisUtilizouIdTransporte(_,[]).

estafetaMaisUtilizouTransporte(transporte(ID,_,_,_,_),E):-
    estafetaMaisUtilizouIdTransporte(ID,E).
```

Identificar todos os Estafetas que mais utilizaram cada um dos transportes

Somos também capazes de conseguir determinar para cada um dos transportes existentes na nossa base de conhecimento o estafeta que o utilizou. Apresentando o resultado numa lista (*(Transporte, Estafeta que o mais utilizou*).

```
tupleEstafetaMaisUtilizouTransporte(E, (E,R) ):-
    estafetaMaisUtilizouTransporte(E,R).

listaEstafetasUtilizouMaisTransporte(R):-
    findall(transporte(ID,N,V,C,E),transporte(ID,N,V,C,E),LT),
    maplist(tupleEstafetaMaisUtilizouTransporte,LT,R).
```

Capítulo 3

Conclusão

Através da realização deste projeto pudemos observar em primeira mão a utilidade e as capacidades da linguagem *Prolog*. Através da definição de uma boa base de conhecimento conseguimos inferir conhecimento de uma forma analítica e controlada. Encontramos facilmente todas as possibilidades a um dado problema (neste caso, as *queries*) através de predicados lógicos cuidadosamente construídos.