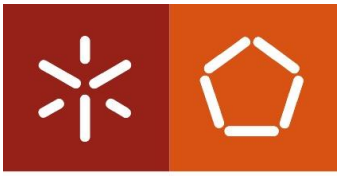

Universidade do Minho

Departamento de Informática

Inteligência Artificial (3º ano de LEI)

Trabalho Prático

1ª Fase

Relatório de Desenvolvimento

Grupo 13

Ariana Lousada (a87998) Miguel Gomes (a93294)

Rita Lino (a93196) Rui Armada (a90468)

4 de janeiro de 2022

Resumo

O trabalho prático foi desenvolvido no âmbito da unidade curricular Inteligência Artificial. O principal objetivo deste projeto consistiu em desenvolver um sistema de representação de conhecimento e raciocínio com capacidade para caracterizar um universo de discurso na área da logística de distribuição de encomendas, utilizando a linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas. Ao longo deste documento vai ser explicado todo o processo de desenvolvimento desde o início do projeto até à solução final concebida pela equipa de trabalho.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Nos dias de hoje os sistemas de entregas de encomendas estão a ser cada vez mais utilizados por diversos tipos de companhias.

O projeto desenvolvido nesta primeira fase do projeto consiste num sistema de representação de conhecimento e raciocínio aplicável à informação acerca das várias encomendas entregues a clientes por parte da organização *Green Distribution*.

Assim, com a definição de conhecimento baseado em *encomendas*, *clientes*, *estafetas* e *transportes*, o principal objetivo rege-se em construir um caso prático que seja capaz de demonstrar as funcionalidades subjacentes à utilização da linguagem de programação PROLOG. Estas funcionalidades vão ser exploradas extensivamente no capítulo 3 deste documento.

Capítulo 2

Preliminares

De modo a obter uma estrutura de conhecimento viável e completa, a equipa de trabalho decidiu desenvolver predicados representantes de *encomendas*, *clientes*, *estafetas* e *transportes*.

Grande parte das representações são feitas através de identificadores únicos, uma vez que se decidiu que as pesquisas de dados executadas ao longo do programa seriam mais diretas, diminuindo a probabilidade de obtenção de informação repetida. Para além disto, é necessário reter que apenas existem os três tipos de transporte já presentes no conhecimento base. Em fases futuras do projeto não deverá ser possível a remoção nem a adição de qualquer tipo de transporte.

Para além da estruturação do conhecimento, também foi necessário decidir os diferentes predicados de consulta deste. Todo o raciocínio envolvido no desenvolvimentos destes predicados é exposto nos capítulos e secções seguintes deste documento.

Capítulo 3

Descrição do Trabalho e Análise de Resultados

3.1 Formulação da Base de Conhecimento

Tal como já referido anteriormente, o conhecimento desenvolvido contém vários predicados necessários para a representação da informação:

- Encomenda;

```
encomenda(IdEncomenda, (IdEstafeta, idTransporte, idcliente), (Data,
    prazoEntrega), (Peso, Volume, Valor, Estado), (codigo, postal)) -> { V,F
    }
```

- Cliente;

```
cliente(IdCliente, Nome, [IdEncomenda]) -> { V,F }
```

- Estafeta;

```
estafeta(IdEstafeta, Nome, [(IdEncomenda, Classificacao)]) -> { V,F }
```

- Transporte.

```
transporte(idTransporte,Nome, PesoMaximo, VelocidadeMedia, nivelEcologico) ->
    { V,F }
```

Cada **encomenda** é definida pelo seu identificador único, um triplo que contém os identificadores do estafeta, tipo de transporte e cliente a quem foi entregue, um duplo constituído pela data e prazo de entrega, um quadruplo com o peso, volume, valor (preço) e estado (”entregue”,”pendente”ou ”cancelada”) e pelo código postal da rua na qual foi entregue.

Cada **cliente** é definido pelo seu identificador único, nome e lista de identificadores de encomendas que lhe foram entregues.

Cada **estafeta** é definido também pelo seu identificador único, nome e lista de duplos, cada um constituído pelo identificador de uma encomenda e pela classificação que a mesma teve, perante o serviço prestado.

Cada **transporte** é definido pelo respetivo identificador, nome, peso máximo que consegue transportar, velocidade média que este atinge e o nível ecológico. O tipo de transporte é tanto mais ecológico quanto maior for o valor guardado no nível. Com isto, definiram-se apenas três tipos de transporte de modo a simplificar a representação deste campo no conhecimento.

3.2 Formulação de predicados

3.2.1 Predicados principais

Para a consulta da informação retida na base de conhecimento desenvolveram-se diversos predicados de pesquisa. Para os testes realizados de cada predicado foi utilizada a seguinte base de conhecimento:

```
%estafeta(IdEstafeta, Nome, [(IdEncomenda, Classificacao)])
estafeta(1, 'Ana Paula', [(1,2),(3,4),(4,3)]).
estafeta(2, 'Ramiro Silva', [(2,1),(5,2)]).
estafeta(3, 'Mike Wazowski', [(0,4),(6,5),(7,4)]).


%cliente(IdCliente, Nome, [IdEncomenda])
cliente(1, 'Gina', [1,2,4]).
cliente(2, 'Marco', [0,3,6]).
cliente(3, 'Ermelinda Ribeiro',[5,7]).


%encomenda(IdEncomenda, (IdEstafeta, idTransporte, idcliente), (Data, prazoEntrega), (Peso, Volume, Valor, Estado), (codigo, postal))
% estado : entregue | pendente | cancelada

encomenda(0, (3,2,2), (19/01/2021,72), (16,2,25,pendente), (4700,691)).
encomenda(1, (1,3,1), (14/04/2019,41), (85.87,75,48,pendente),(1234,214)).
encomenda(2, (2,1,1), (10/08/2021,64), (4.13,85,509,entregue),(4672,451)).
encomenda(3, (1,1,2), (23/11/2020,21), (3.0,3,147,entregue),(4601,561)).
encomenda(4, (1,2,1), (27/07/2019,56), (19.25,32,355,entregue),(8214,823)).
encomenda(5, (2,1,3), (08/05/2021,44), (2,33,612,cancelada),(0502,145)).
encomenda(6, (3,3,2), (19/01/2021,21), (98.92,45,125,perdida),(5150,142)).
encomenda(7, (3,2,3), (26/04/2021,14), (14.14,48,342,cancelada),(4444,666)).


%transporte(idTransporte,Nome, PesoMaximo, VelocidadeMedia, nivelEcologico)
transporte(1, bicicleta, 5, 10, 3).
transporte(2, mota, 20, 35, 2).
transporte(3, carro, 100, 25, 1).
```

1. ecotrans

```
ecotrans(Res) :-
    transporte(IdBike, bicicleta, _, _, _),
    findall((Id, N)
        , (estafeta(Id, _, _)
            , findall(_, encomenda(_, (Id, IdBike, _), _, _, _), Es)
            , length(Es, N))
        , L),
    max_on_snd(L, (Res, _)).
```

O predicado **ecotrans** retorna o identificador do estafeta que utilizou transportes mais ecológicos.

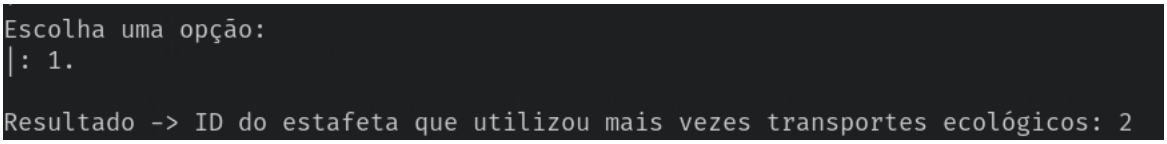

Figura 3.1: Teste do predicado `ecotrans`

2. estcliente

```
estcliente(IdCliente, Res) :-
    findall((IdEst,IdEnc), encomenda(IdEnc,(IdEst,_,IdCliente), _, _, _), Res).
```

O predicado **estcliente**, dado um identificador de um cliente existente na base de conhecimento, retorna os estafetas que lhe entregaram encomendas.

Figura 3.2: Teste do predicado `estcliente`

3. servEstaf

```
servEstaf(IdEst, Res) :-
    findall(IdCliente, encomenda(_,(IdEst, _, IdCliente), _, _,_), Res).
```

O predicado **servEstaf**, dado um identificador de um estafeta existente na base de conhecimento, retorna uma lista de identificadores de clientes a quem entregou encomendas.

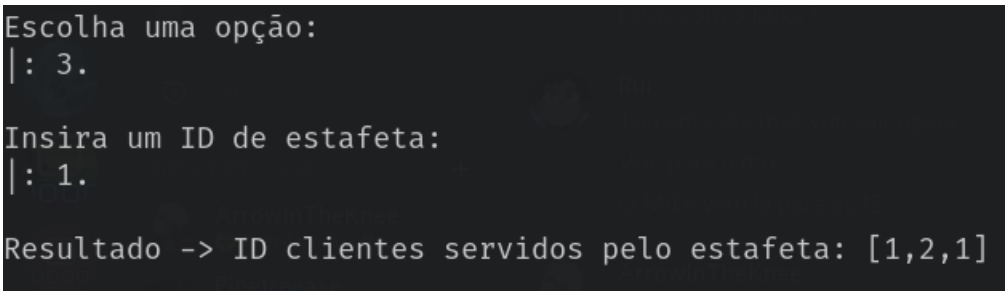

Figura 3.3: Teste do predicado `servEstaf`

4. valordia

```
valordia(Data, TotalFatorado) :-
    findall(Valor, encomenda(_, (_,_,_), (Data, _), (_,_,Valor,_), _), Values),
    sum_list(Values, TotalFatorado).
```

O predicado **valordia** calcula o total faturado num determinado dia pela *Green Distribution*.

```
Escolha uma opção:
|: 4.

Insira uma data de um dia:
|: 23/11/2020.

Resultado -> Valor faturado na dia inserido: 147
```

Figura 3.4: Teste do predicado `valordia`

5. freqZona

```
freqZona(Res):-
  findall((Concelho),encomenda(_,_,_,(_,_,_,entregue),Concelho),L),
  freqs(L,Res).
```

O predicado **freqZona** retorna as zonas (códigos postais) com maior volume de entregas assim como a quantidade das mesmas.

```
Escolha uma opção:
|: 5.

Resultado -> (Localidade, Número de encomendas): [(8214,1),(4601,1),(4672,1)]
```

Figura 3.5: Teste do predicado `freqZona`

6. classificacaoMedia

```
classificacaoMedia( IdEst, Media ):-
        estafeta(IdEst, _, L),
        sumTuples(2,L, Sum ),
        length( L, Length ),
        (Length > 0 -> Media is div(Sum,Length) ; Media is 0).
```

O predicado **classificacaoMedia**, dado um identificador de um estafeta existente na base de conhecimento, retorna a classificação média de satisfação do cliente.

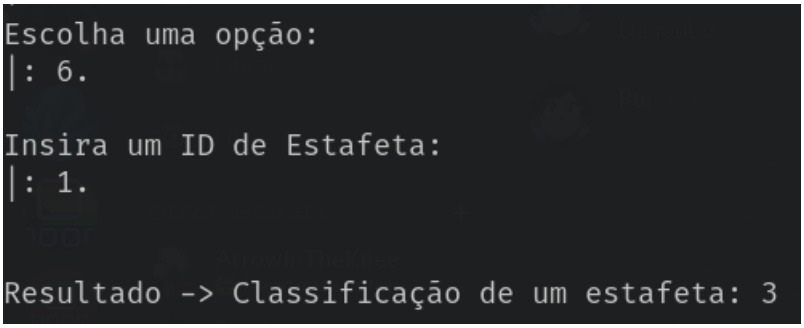

Figura 3.6: Teste do predicado `classificacaoMedia`

7. entregasTransporte

```
entregasTransporte(Di/Mi/Yi,Df/Mf/Yf,B,M,C):-
  findall((D/M/Y,IdTrans),encomenda(_,(_,IdTrans,_),(D/M/Y,_),(_,_,_,entregue),_),
    L),
  checkValidade(L,Di/Mi/Yi,Df/Mf/Yf,T),
  checkTransport(T,B,M,C).
```

O predicado **entregasTransporte**, dado um intervalo de tempo (uma data inicial e final), retorna uma lista de duplos com os identificadores dos transportes juntamente com o número de encomendas entregues.

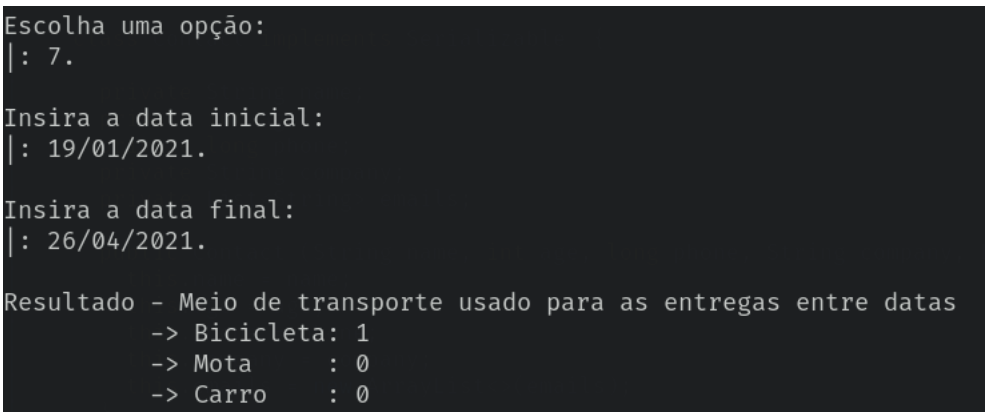

Figura 3.7: Teste do predicado `entregasTransportes`

8. entregasEstafeta

```
entregasEstafeta(Di/Mi/Yi,Df/Mf/Yf,Out):-
  findall((D/M/Y,IdEst),encomenda(_,(IdEst,_,_),(D/M/Y,_),(_,_,_,entregue),_),
    Estafetas),
  checkValidade(Estafetas,Di/Mi/Yi,Df/Mf/Yf,T),
  freqs(T,Out).
```

O predicado **entregasEstafetas**, dado um intervalo de tempo (uma data inicial e final), retorna uma lista de duplos com os identificadores dos estafetas juntamente com o número de encomendas entregues.

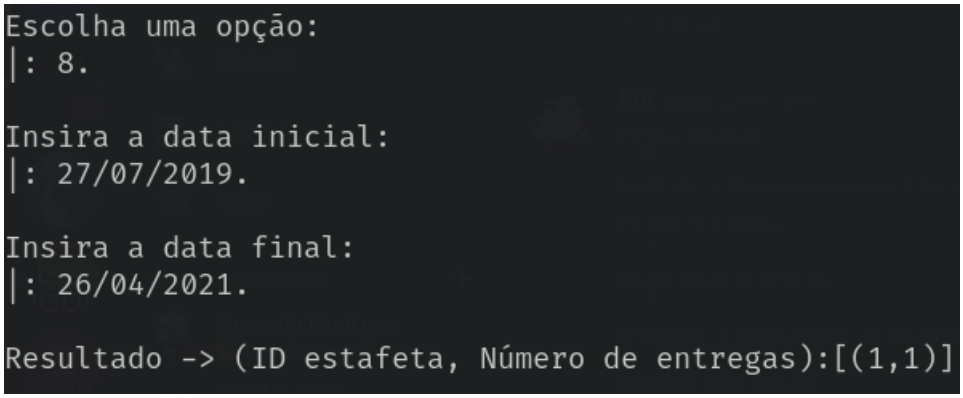

Figura 3.8: Teste do predicado `entregasEstafeta`

9. estadoIntrevalo

```
estadoIntrevalo(Di/Mi/Yi,Df/Mf/Yf,E,N):-
  findall((D/M/Y,Estado),encomenda(_,_,(D/M/Y,_),(_,_,_,Estado),_),L),
  checkValidade(L,Di/Mi/Yi,Df/Mf/Yf,Estados),
  checkState(Estados,E,N).
```

O predicado **estadoIntrevalo** calcula o número de encomendas entregues e não entregues num determinado intervalo de tempo (dada uma data inicial e uma data final).

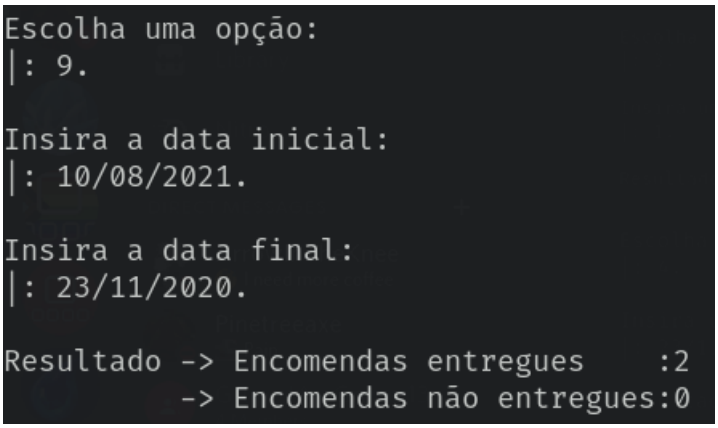

Figura 3.9: Teste do predicado `estadoIntrevalo`

10. pesoTotal

```
pesototal(IdEst, Data, Res) :-
    findall(Peso, encomenda(_,(IdEst,_,_),(Data,_),(Peso, _, _, _), _), Pesos),
    sum_list(Pesos, Res).
```

O predicado **pesoTotal** calcula o peso total transportado por estafeta num determinado dia.

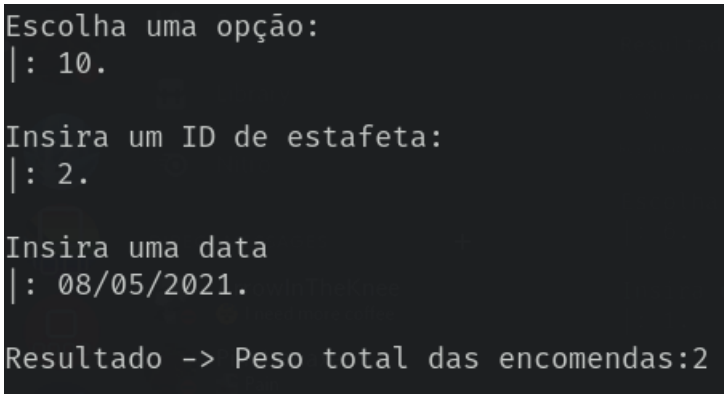

Figura 3.10: Teste do predicado `pesoTotal`

3.2.2 Predicados auxiliares

1. max_on_snd

```
% [(A, Number)] -> (A, Number)
max_on_snd([H], H).
max_on_snd([(A0,N0)|T], Res) :-
    max_on_snd(T, (A1,N1)),
    Max is max(N0, N1),
    (Max =:= N0 -> Res = (A0,N0); Res = (A1,N1)).
```

Dada uma lista de tuplos, o predicado retorna o maior valor presente no segundo campo de toda a lista.

2. freqs

```
%[X]->[(X,n)]
freqs([],[]).
freqs(In,Out):-
    freqs(In,[],Out).

freqs([],Out,Out).
freqs([X|Xs],Table,Out):-
    \+ member((X,_),Table),
    freqs(Xs,[(X,1)|Table],Out).

freqs([X|Xs],Table,Out) :-
    selectchk((X,N),Table, Others),
    M is N + 1,
    freqs(Xs, [(X,M)| Others], Out).
```

Dada uma lista de elementos, retorna uma lista de tuplos de cada elemento associado à sua frequência.

3. sumTuples

```
sumTuples(_,[],0).
sumTuples(Pos,[Tuple|T],Sum) :-
        sumTuples(Pos,T,Sum1),
        arg(Pos,Tuple,Elem),
        Sum is Sum1 + Elem.
```

Dada uma lista de tuplos, adiciona recursivamente os valores de um dado campo de cada elemento(escolhido pelo utilizador).

4. concat

```
concat([],L2,L2). concat([Head|Tail],L2,[Head|L3]) :- concat(Tail,L2,L3).
```

Dadas duas listas, o predicado concatena-as numa só lista como resultado.

5. checkValidade

```
% (D_inicial/Mi/Yi,D_final/Mf/Yf,[Sent],[NotSent],(D/M/Y,Estado)) -> [Lista]
checkValidade([],_,_,[]).
checkValidade([(D/M/Y,Estado)|T],Di/Mi/Yi,Df/Mf/Yf,Out):-
  checkValidade(T,Di/Mi/Yi,Df/Mf/Yf,Out1),
  (Y<2022, Y>1900, M<13, M>0,D<32, D>0->
    (((Yi<Y, Y<Yf, Mi=<M, M=<Mf);(Y==Yi, Mi=<M);(Y==Yf, M=<Mf);(Y==Yi, Mi==M, Di=<
    D);(Y==Yf, M==Mf, D=<Df))-> concat([Estado],Out1,Out);concat([],Out1,Out))).
```

Recebe duas datas e uma lista de objetos, retornando uma outra lista com os objetos (por exemplo, encomendas) que se enquadrem no período de tempo definido pelas datas recebidas.

6. checkState

```
% [Estados] -> XEntregues Y!Entregues
checkState([],0,0).
checkState([Estado|T],E,N):-
  checkState(T,E1,N1),
  (Estado = entregue-> (E is E1 + 1,N is N1); (E is E1, N is N1 + 1)).
```

Dada uma lista de estados(de encomendas), separa-os entre "Entregues"e "Não entregues".

7. checkTransport

```
% [Transporte] -> XEntregues YEntregues ZEntregues
checkTransport([],0,0,0).
checkTransport([IdTrans|T],B,M,C):-
  checkTransport(T,B1,M1,C1),
  (IdTrans = 1 -> B is B1 + 1,M is M1, C is C1;(IdTrans = 2 -> M is M1 + 1, B is
    B1, C is C1;C is C1 + 1, B is B1, M is M1)).
```

Dada uma lista de identificadores de transporte, calcula quantas encomendas foram transportadas por cada um.

3.3 Alteração da base de conhecimento

Para ser possível adicionar e remover conhecimento, foi necessário desenvolver predicados que tornariam isso possível de uma forma coerente, isto é, invariantes. Dito isto, foram utilizados os predicados de **evolução** e **involução** já disponibilizados pela equipa docente, que permitem a inserção e remoção de conhecimento. Para além destes, desenvolveram-se os predicados `atualizacao_c` e `atualizacao_e`, que permitem a atualização de conhecimento em determinadas situações. Estas atualizações são particularmente necessárias quando, por exemplo, são eliminadas ou adicionadas encomendas[1].

De modo a garantir que não seria possível possuir informação inválida, foi então necessária a implementação de vários invariantes, estabelecendo algumas regras para cada inserção e/ou remoção de cada predicado.

Estes invariantes podem ser consultados no ficheiro `invariantes.pl`

3.3.1 Estafeta

- Não será de possível inserção um estafeta com identificador já pertencente a outro contido na base de conhecimento.
- Um estafeta, ao ser eliminado, todas as encomendas a ele associadas irão ser atualizadas com o id '0'. Este '0' poderá ser interpretado como um estafeta que já não está presente no sistema.

```
?- add_estafeta('Eren Yeager').
true .

?- findall(Nome,estafeta(301,Nome,_),R).
R = ['Eren Yeager'].
```

Figura 3.11: Adição de um novo estafeta na base de conhecimento.

```
?- rem_estafeta(301).
true .

?- findall(Nome,estafeta(301,Nome,_),R).
R = [].
```

Figura 3.12: Remoção de um estafeta da base de conhecimento.

3.3.2 Cliente

- Não será de possível inserção um cliente com identificador já pertencente a outro contido na base de conhecimento.
- Um cliente, ao ser eliminado, todas as encomendas a ele associadas são eliminadas. Após a eliminação das encomendas os estafetas são atualizados.

[1] Uma vez que os estafetas e os clientes possuem informações acerca das encomendas, a inserção de uma nova exige a atualização da informação incluída em cada um.

```
?- add_cliente('Jeremias').
true .

?- findall(Nome,cliente(501,Nome,_),R).
R = ['Jeremias'].
```

Figura 3.13: Adição de um novo cliente na base de conhecimento.

```
?- rem_cliente(501).
true .

?- findall(Nome,cliente(501,Nome,_),R).
R = [].
```

Figura 3.14: Remoção de um cliente da base de conhecimento.

3.3.3 Encomenda

- Não será de possível inserção uma encomenda com identificador já pertencente a outro contido na base de conhecimento.
- Uma encomenda, ao ser eliminada, o respetivo cliente e estafeta são atualizados, dos quais se remove a referência à encomenda que se pretende eliminar.

```
?- add_encomenda(2,3,02/12/2021,21,12,32,41,1234,567).
true .

?- findall(Id,encomenda(5001,Id,_,_,_),R).
R = [(2, 1, 3)].
```

Figura 3.15: Adição de uma nova encomenda na base de conhecimento.

```
?- rem_encomenda(5001).
true .

?- findall(Id,encomenda(5001,Id,_,_,_),R).
R = [].
```

Figura 3.16: Remoção de uma encomenda da base de conhecimento.

3.3.4 Transporte

- Não será possível remover ou adicionar qualquer tipo de transporte à base de conhecimento.[2]

[2]Note-se que de modo a isto ser possível, não se definiu qualquer invariante relacionado com o tipo de transportes.

3.4 Extras

3.4.1 Menu

De modo a facilitar a utilização do sistema, a equipa de trabalho desenvolveu um menu que apresenta todas as escolhas possíveis no programa para utilizar os predicados principais desenvolvidos expostos na secção 3.2.1, juntamente com uma opção de consultar a base de conhecimento do sistema e de armazenamento da base de conhecimento num ficheiro.

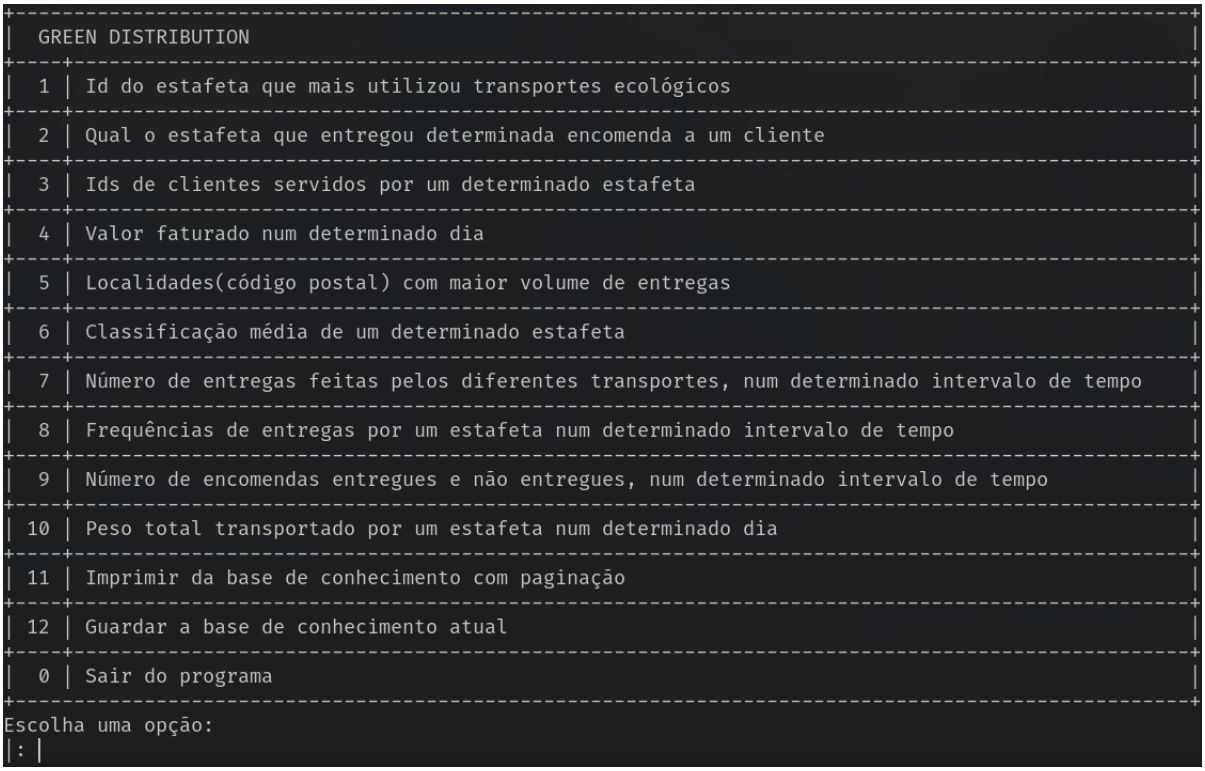

Figura 3.17: Menu do sistema

3.4.2 Paginação do conhecimento

De modo a possibilitar a consulta de toda a base de conhecimento, desenvolveu-se um predicado para aplicar a paginação. Para demonstrar esta parte do programa, foi utilizado um programa desenvolvido em Python[3] para gerar uma outra base de conhecimento já de tamanho considerável, que foi armazenada separadamente nos ficheiros `estafetas.pl`, `clientes.pl`, `transportes.pl` e `encomendas.pl`.

Esta base de conhecimento possui cerca de 5000 encomendas, 300 estafetas, 500 clientes e 3 tipos de transporte.

[3]Este programa pode ser consultado no ficheiro `python.py`

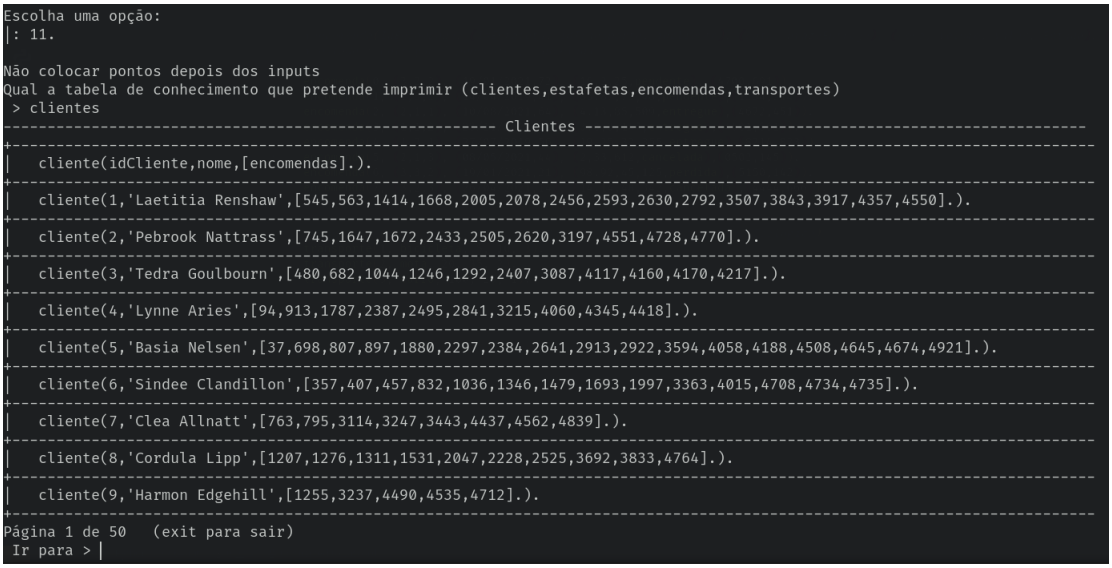

Figura 3.18: Paginação dos estafetas da base de conhecimento.

3.4.3 Salvaguarda da base de conhecimento em ficheiro

O sistema de reconhecimento possibilita também o armazenamento da base de conhecimento num ficheiro, com auxílio do predicado `saveConhecimento`.[4]

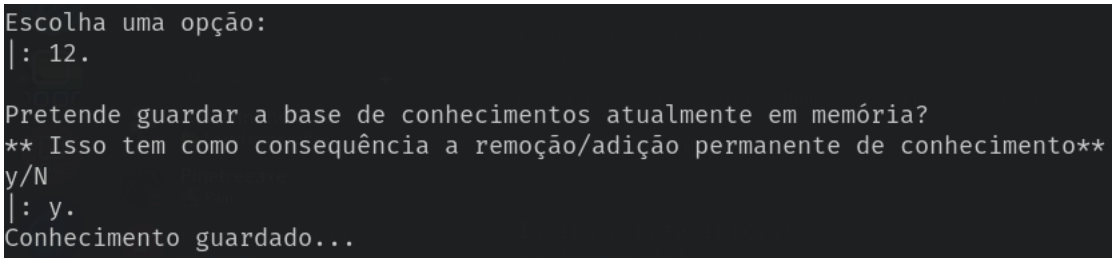

Figura 3.19: Armazenamento da base de dados em ficheiro

[4] É de notar que sempre que se guardar a base de conhecimento as cópias anteriores são eliminadas, uma vez que o predicado `saveConhecimento` escreve sempre nos mesmos ficheiros.

Capítulo 4

Manual de utilização do sistema de reconhecimento

Para ser possível executar o sistema desenvolvido, em primeiro lugar, é necessário instalar o Python. A instalação pode ser feita através da consulta do seguinte link : `https://www.python.org/downloads/`

Depois disto, na pasta do projeto, inserir o seguinte comando: `swipl main.pl`

Já dentro do sistema, apenas é necessário selecionar a opção pretendida no menu.

Caso se pretenda adicionar ou remover conhecimento (estafetas, clientes ou encomendas), é necessário selecionar a opção '0 - Sair do programa' e inserir os seguintes predicados diretamente na linha de comandos do Swi-Prolog:

- `add_estafeta(Nome)` - Insere um estafeta novo dado um nome.
- `rem_estafeta(IdE)` - Remove um estafeta dado o seu identificador.
- `add_cliente(Nome)` - Insere um cliente novo dado um nome.
- `rem_cliente(IdC)` - Remove um cliente dado o seu identificador.
- `add_encomenda(IdE,IdC,Data,Prazo,Peso,Volume,Valor,Codigo,Postal)` - Insere uma encomenda dado um identificador de estafeta e cliente (já existentes), uma data, prazo de entrega, peso, volume, preço e código postal.
- `rem_encomenda(IdEnc)` - Remove uma encomenda dado o seu identificador.

Capítulo 5

Problemas de Implementação

Durante o desenvolvimento do trabalho, a equipa de trabalho deparou-se com apenas um problema relacionado com a alteração da base de conhecimento.

Uma vez que o menu do sistema executa em *loop*, é possível que a adição e remoção de conhecimento não funcionem durante a sua execução. Contudo, é possível adicionar e remover conhecimento saindo do menu do sistema antes da inserção de qualquer predicado de alteração de conhecimento.

Capítulo 6

Conclusões e Sugestões

Com a elaboração deste trabalho prático foi possível aplicar os conceitos lecionados nas aulas da unidade curricular de uma forma mais prática, o que permitiu uma melhor consolidação da utilização da linguagem Prolog.

Parte do raciocínio foi facilitado com a utilização de certos métodos exemplificados nas aulas práticas, nomeadamente na pesquisa de determinada informação contida no conhecimento, na sua evolução e involução e ainda na formulação dos predicados necessários para atingir todas as metas inicialmente estabelecidas.

Em fases futuras do projeto poderá ser possível expandir as suas funcionalidades de modo a dar mais escolhas ao utilizador acerca do tipo de informação que pretende procurar.

Consideramos então que os principais objetivos do projeto foram atingidos, tendo culminado no desenvolvimento de um programa útil na atualidade, dada a crescente demanda em entregas de encomendas com gradual informatização dos serviços.

Bibliografia

[1] Swi-Prolog: https://www.swi-prolog.org/pldoc/