Trabalho Prático - Fase 1

Diogo Pires, a93239 Gonçalo Soares, a93286
Marco Costa, a93283 Rita Teixeira, a89494

4 de agosto de 2023

Inteligência Artificial

Licenciatura em Engenharia Informática

Índice

Capítulo 1

Introdução

Nesta fase do trabalho procuramos representar um sistema de entregas de encomendas, utilizando como base dessa representação conceitos lecionados na disciplina de Inteligência Artificial. Para isso, e tendo em conta as funcionalidades pedidas, procurámos criar uma base de conhecimento que tornasses as funcionalidades mais eficientes. No contexto deste trabalho, pretende-se entender o conceito de lógica na Inteligência Artificial e conseguir aplicá-lo na utilização da linguagem de programação em lógica PROLOG. Este tema, permitir-nos-à uma maior e melhor representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas.

Quanto à estrutura do relatório acerca deste projeto, decidimos começar por descrever como interpretámos o enunciado e estruturamos a informação (*Base de conhecimento* (2.1)). Nessa mesma secção, também se abordaram as decisões tomadas de forma a facilitar a realização das *queries*. Seguidamente, o subcapítulo *Funcionalidades* (2.2) trata da maneira como as *queries* do enunciado foram resolvidas. Porém, antes do tratamento das *queries*, achou-se por bem descrever o funcionamento de alguns predicados predefinidos usados. Esta explicação deve-se ao facto de os predicados não terem sido gerados por nós. Posteriormente, a secção *Entregas de encomendas e Atribuições* (2.3) descreve o processo de criação de entregas e atribuições de estafetas e encomendas. Para além disso, abordamos algumas decisões tomadas, tendo em conta a nossa *Base de conhecimento* (2.1).

Por fim, na secção *Inserir/Remover conhecimento - Invariantes* (2.4) iremos descrever as regras de inserção/remoção de conhecimento que garantem a consistência da base de conhecimento.

Capítulo 2

Desenvolvimento

2.1 Base de conhecimento

Para representarmos o sistema de entregas, decidimos criar várias entidades individuais: rua; cliente; encomenda; entrega; atribuído; estafeta e transporte. De forma a simplificarmos o problema do projeto, guardamos nestas apenas a informação que consideramos essencial. Vamos agora explicar as relações entre elas:

Rua

Uma rua tem um identificador, pertence a uma cidade e tem um nome. De forma a simplificar o problema deste projeto decidimos não implementar as casas individuais mas apenas entregar as encomendas às ruas.

Cliente

Ao cliente é-lhe apenas atribuído um ID e um nome. Não o restringimos a uma rua ou localidade de forma a permitir que este envie encomendas para outras ruas, seja porque mudou de casa ou apenas para enviar uma prenda a um(a) amigo(a).

Encomenda

Uma encomenda está associada a um identificador e é caracterizada pelos seguintes elementos:

- ID do cliente
- Peso
- Volume
- Prazo de entrega (Dia)
- Prazo de entrega (Hora do dia)
- Data de encomenda (Dia)
- Data de encomenda (Hora do dia)
- ID da rua

Esta estrutura apenas representa um pedido de encomenda. Não representa a entrega desta nem guarda o estafeta que lhe está atribuído, se estiver algum. Isto porque facilita a diferenciação de encomendas entregues das não entregues, e entre estas, as que têm um estafeta associado das que não têm.

Entrega

Aqui estão guardados todos os valores relevantes à entrega, tais como:

- ID do estafeta que entregou a encomenda
- Veículo usado
- ID da encomenda
- Avaliação de 0-5 feita pelo cliente
- Data de entrega
- Hora de entrega

Atribuído

Esta estrutura apenas associa um estafeta com uma encomenda através dos seus IDs. A atribuição não representa uma entrega, apenas que a encomenda foi dada ao estafeta. Porém este pode, por qualquer razão, não a entregar.

Estafeta

A cada estafeta está atribuído um número identificador único, um nome e uma localidade (não confundir com rua).

Transporte

Cada meio de transporte guarda um nome único, uma carga máxima, uma velocidade média e um preço por quilómetro. Esta implementação facilita a introdução de novos meios de transporte não restringindo aos três referidos no enunciado.

Ratings

Outro elemento da *Base de conhecimento* (2.1) é o rating dos estafetas. Esses ratings são calculados pela quantidade de entregas atribuídas que não conseguiu entregar, e podem penalizar o estafeta. A fórmula de cálculo do rating do estafeta é: $Rating = 5 - (n.atribuições - n.entregas)$. Assumimos que as entregas que ele realizou correspondem às atribuídas. Este rating é limitado entre 0 e 5, sendo 5 o melhor, e 0 o pior.

2.2 Funcionalidades

2.2.1 Considerações gerais

Como auxilio para a implementação das diversas funcionalidades, foram utilizados vários predicados de uma biblioteca chamada *apply*. Os predicados usados foram, nomeadamente, o *foldl* e *maplist*. Para além destes foi também usado o predicado *findall*.

foldl(:Goal, +List, +V0, -V)

O predicado *foldl* aplicado tem aridade 4. É usado para reduzir uma lista a apenas um valor. Como argumentos, temos o objetivo - *Goal* - que vai ser um aplicado durante a iteração da lista - List. Como valor inicial para o acumulador temos o *V0* e como valor de retorno o *V*.

maplist(:Goal, ?List1, ?List2)

Este predicado tem aridade 3. É usado para mapear um lista numa outra. Como argumentos, temos o objetivo - *Goal* - que vai ser aplicado a cada elemento da lista - *List1* - formando um elemento da nova lista *List2*.

findall(+Template, :Goal, -Bag)

Este predicado tem aridade 3. É usado para encontrar todos os valores que concretizam um dado objetivo.

2.2.2 Queries individuais

Query 1

Para a resolução desta *query* começamos por procurar por todas as entregas guardadas na *Base de conhecimento* (2.1). A partir desta, criamos uma lista auxiliar que contém apenas os elementos únicos. De seguida, para cada elemento da lista filtrada, contamos o número de vezes que esta aparece na lista completa e, por fim, filtramos a lista originada para uma nova com apenas os estafetas que mais vezes usaram cada veículo.

Com esta implementação podemos procurar, não só pelo estafeta mais ecológico, mas também pelo estafeta que usou mais vezes cada veículo. Para além disso é genérica, não requerendo qualquer alteração após a adição de novos meios de transporte.

```
estafetaMaisEco(R) :-
    findall(Estafeta/Veiculo, entrega(Estafeta,Veiculo,_,_,_,_), Entregas),
    encontraUnicos(Entregas,Filtrados),
    aplicaLista(contaElem,Filtrados,Entregas,Contados),
    contaVeiculos(Contados,R)
.
```

Listing 2.1: Query 1

Query 2

Esta *query* tem como objetivo identificar que estafetas entregaram determinada(s) encomenda(s) a um determinado cliente. Deste modo, começamos por encontrar todas as encomendas referentes a um ID de cliente, e depois, através do predicado *maplist* conseguimos descobrir todos os estafetas que entregaram as encomendas.

```
estafetas_clientes(EstafetasID, EncomendasID, ClienteID) :-
  encomendas_cliente(ClienteID, EncomendasID),
  EncID = EncomendasID,
  maplist(estafeta_entregou_encomenda, EncID, EstafetasID).
```

Listing 2.2: Query 2

Query 3

Este predicado é bastante simples. Com um *findall*, procuramos todos os clientes a quem o estafeta entregou uma encomenda e filtramos a lista resultante de forma a remover os clientes repetidos.

```
clientesServidos(Estafeta,Clientes) :-
    findall(Cliente,
    (entrega(Estafeta,_,EncID,_,_,_),encomenda(EncID,Cliente,_,_,_,_,_,_,_)),
    Clientes1),
    encontraUnicos(Clientes1, Clientes)
.
```

Listing 2.3: Query 3

Query 4

Esta *query* tem como objetivo calcular o valor faturado pela *Green Distribution* num determinado dia. Para isto, descobrimos todas as entregas que foram efetuadas num dado dia. Calculamos o preço de cada uma destas entregas e, por fim, somamos os preços para obter o preço total.

```
valorFaturado(data(Dia, Mes, Ano), Valor) :-
    findall(entrega(A, B, C, D, data(Dia, Mes, Ano), E), entrega(A, B, C, D,
    data(Dia, Mes, Ano), E), Encomendas),
    maplist(preco, Encomendas, Precos),
    foldl(plusFloat, Precos, 0, Valor).

plusFloat(N1, N2, N) :- N is N1 + N2.
```

Listing 2.4: Query 4

Query 5

O objetivo desta *query* é procurar quais as ruas com maior volume de entregas por parte da *Green Distribuition*.

Este predicado funciona de forma muito semelhante à *query* 1. Começamos por pesquisar por todas as ruas em que houve uma entrega. Em seguida, geramos uma lista auxiliar sem elementos repetidos e, por fim, contamos quantas vezes cada elemento da lista auxiliar aparece na lista de ruas.

```
entregasEmCadaRua(R) :-
    findall(Rua,
    (entrega(_,_,EncID,_,_,_),encomenda(EncID,_,_,_,_,_,_,_,_,Rua)), Ruas),
    encontraUnicos(Ruas,Filtrados),
    aplicaLista(contaElem,Filtrados,Ruas,R)
.
```

Listing 2.5: Query 5

Query 6

Com esta *query* queremos calcular a satisfação média para um dado estafeta. Para tal procuramos todas as entregas desse estafeta e guardamos todos os ratings numa lista. Depois somamos todos esses ratings e dividimos pelo comprimento da lista.

Se o estafeta não tiver quaisquer entregas feitas ou não existir, o predicado retorna 0.

```
satisfacClienteParaEstafeta(Estafeta,N) :-
    findall(Rating, entrega(Estafeta,_,EncID,Rating,_,_), Ratings),
    sum(Ratings,Sum),
    length(Ratings, NRatings),
    NRatings > 0, N is Sum/NRatings, !
.
satisfacClienteParaEstafeta(_,0).
```

Listing 2.6: Query 6

Query 7

Nesta *query* temos de identificar o número total de entregas pelos diferentes meios de transporte, num determinado intervalo de tempo. Primeiro, procuramos todas as entregas realizadas entre essas duas datas, e guardamos os meios de transporte utilizados. Depois percorremos a lista dos meios de transporte e contamos quantas vezes cada um surge.

```
entregasPorMeioTransporte(Data1, Data2, ListaRes) :-
    findall(X, filtraEntregas(Data1, Data2, X), ListaVeiculos),
    contaPares(ListaVeiculos, ListaRes).
```

Listing 2.7: Query 7

Query 8

Nesta *query* temos de identificar o número total de entregas pelos estafetas, num determinado intervalo de tempo. Esta *query* foi feita duma forma bastante parecida com a anterior. Primeiro, procuramos todas as entregas realizadas entre essas duas datas, e guardamos o ID do estafeta que entregou. Por fim, contamos quantas vezes cada um dos estafetas aparece nessa lista.

```
numeroEncomendas(Data1, Data2, RespostaPares) :-
    findall(IdEstafeta, entregaEntreDatas(Data1, Data2, IdEstafeta), Pares),
    contaPares(Pares, RespostaPares).
```

Listing 2.8: Query 8

Query 9

Nesta *query* temos de calcular o número de encomendas entregues e não entregues pela *Green Distribution*, num determinado período de tempo. Primeiro procuramos quais as encomendas entregues no período de tempo fornecido, guardando o ID da encomenda numa lista. Depois calculamos quantas dessas encomendas foram entregues, e quantas não foram. Para isso, procuramos nas entregas se alguma tem guardada o ID dessa encomenda.

```
encomendasNEntregues(D1, D2, NTotal, NTotalNEntregues) :-
    findall(X, filtraEncomendas(D1, D2, X), ListaEncomendas),
    quaisForamEntregues(ListaEncomendas, NTotal, NTotalNEntregues).
```

Listing 2.9: Qurery 9

Query 10

Nesta *query* temos de calcular o peso total transportado por um estafeta num determinado dia. Primeiro, procuramos as entregas que um estafeta entregou, e guardamos os IDs dessas entregas. Como cada entrega tem o ID da encomenda correspondente e essa encomenda tem a informação do peso, podemos descobrir o peso a partir do ID da entrega. No fim, temos de somar todos os pesos associados aos IDs das entregas.

```
totalPesoEstafeta(IdEstafeta, Data, PesoTotal) :-
  entregasDoEstafeta(IdEstafeta, Data, IdsEnTregasFeitas),
  calculaPesoPorEncomendas(IdsEnTregasFeitas, PesoTotal).
```

Listing 2.10: Query 10

2.3 Entregas de encomendas e Atribuições

Entregas de encomendas

De modo a entregar as encomendas, foi criado um predicado *entregaEncomendas* que descobre todas as encomendas não entregues e aplica-lhes o predicado *entregaEncomenda* de modo a que estas passem a ser entregues.

```
entregaEncomendas(Entregas) :-
    findall(encomenda(Id, ClienteID , Peso, Volume, PrazoEntrega,
    HorasPrazoEntrega, DataDeEncomenda, HorasDataEncomenda, Rua),
    (encomenda(Id, ClienteID , Peso, Volume, PrazoEntrega, HorasPrazoEntrega,
    DataDeEncomenda, HorasDataEncomenda, Rua), \+ entrega(_, _, Id, _, _, _)),
    Encs), % Todas as encomendas nao entregues.
    maplist(entregaEncomenda, Encs, Entregas).
```

Listing 2.11: Predicado entregaEncomendas

Relativamente à *entregaEncomenda*, determinamos, através do predicado *atribuido*, cujo funcionamento é explicado posteriormente, o *EstafetaId*. Já o veiculo das encomendas é constante, sendo sempre o veiculo *carro*. De modo a determinar o rating, usamos o predicado *random* para gerar um número aleatório de 0 a 5. A data de entrega e hora de entrega foi colocado como o prazo de entrega e horas do prazo de entrega, de modo a simplificar. De seguida, é adicionado à *Base de conhecimento* (2.1) através do predicado *evolucao*.

```
entregaEncomenda(encomenda(EncomendaID, ClienteID , Peso, Volume,
    PrazoEntrega, HorasPrazoEntrega, DataDeEncomenda, HorasDataEncomenda,
    Rua), Entrega) :-
    atribuido(EstafetaId, EncomendaID),
    Veiculo = carro,
    random(0, 5, Rating),
    DataEntrega = PrazoEntrega,
    HoraEntrega = HorasPrazoEntrega,
    Entrega = entrega(EstafetaId, Veiculo, EncomendaID, Rating, DataEntrega,
    HoraEntrega),
    evolucao(Entrega).
```

Listing 2.12: Predicado entregaEncomenda

Gerar atribuições

De modo a gerar atribuições, começamos por procurar quais as encomendas não atribuídas. Depois geramos uma atribuição para cada encomenda, como explicado a seguir.

```
listaEncomendasNAtribuida(Res) :-
    findall(EncId, encomenda(EncId, _, _, _, _, _, _, _, _), ListaIDs ),
    findall(EncE, atribuido(_, EncE), ListaAtribuidas ),
    removeOneList(ListaIDs, ListaAtribuidas, Res).
```

Listing 2.13: Procurar encomendas não atribuídas

Em primeiro lugar, procuramos que estafetas podem entregar essa encomenda, por causa da localização associada à encomenda, utilizando o predicado *filtraCidadeEncomenda*.

Depois, gostaríamos de ordenar a lista pelo rating dos estafetas, mas não conseguimos fazer isso. Desta forma, conseguiríamos penalizar os estafetas que não conseguem entregar as encomendas. No entanto, consideramos já existir uma penalização nos estafetas através dos ratings, no caso de ser mau.

A seguir, verificamos se é possível o primeiro estafeta da lista entregar a encomenda. Para saber isso, vemos se ele já atingiu o peso máximo do carro, o veículo com maior capacidade. Se puder entregar, é-lhe atribuído a encomenda. Caso não hajam mais estafetas, o último estafeta da lista de estafetas possíveis tem essa encomenda atribuída.

```
procuraEstafetaLivreEnc(_, _, []).
procuraEstafetaLivreEnc(EncId, _, [Est1|T]) :-  tamLista(T,
    0),evolucao(atribuido(Est1, EncId)).
procuraEstafetaLivreEnc(EncId, Data, [Est1|T]) :- \+ tamLista(T, 0),
    estafetaLivre(Est1, Data),evolucao(atribuido(Est1, EncId)).
procuraEstafetaLivreEnc(EncId, Data, [Est1|T]) :- \+ tamLista(T, 0), \+
    estafetaLivre(Est1, Data),procuraEstafetaLivreEnc(EncId, Data, T).
```

Listing 2.14: Verifica estafeta e adiciona conhecimento

2.4 Inserir/Remover conhecimento - Invariantes

De forma a evitarmos contradições com a adição/remoção de conhecimento, criamos um conjunto de invariantes que garantem a permanente consistência deste.

Uma regra que é comum a todas as entidades é que o seu ID é único. Para além desta, cada estrutura tem as suas regras de adição/remoção:

Encomenda

```
+encomenda(EncID,Cliente,Peso,Volume,Prazo,HoraEnt,DataEnc,HoraEnc,Rua) ::
(
    findall(EncID,encomenda(EncID,_,_,_,_,_,_,_,_),R),
    length(R,L),
    L == 1,
    cliente(Cliente,_),
    Peso > 0,
    Volume > 0,
    Prazo,
    HoraEnt,
    DataEnc,
    HoraEnc,
    rua(Rua,_,_)
).
-encomenda(EncID,_,_,_,_,_,_,_,_) ::
(
    \+ entrega(_,_,EncID,_,_,_),
    \+ atribuido(_,EncID)
).
```

Listing 2.15: Invariantes da encomenda

O invariante de inserção não permite a existência de encomendas com identificadores repetidos nem clientes e/ou ruas inválidas. Adicionalmente o peso e volume devem ser positivos e ambas datas e horas devem ser válidas.

Relativamente à remoção, uma encomenda não pode ser removida se esta tiver entregas ou atribuições que a referenciem.

Entrega

```
+entrega(IDEstafeta,Veiculo,EncID,Rating,Data,Hora) ::
(
    findall(EncID,(entrega(_,_,EncID,_,_,_)),R),
    length( R,N ),
   N == 1,
    estafeta(IDEstafeta,_,_),
    encomenda(EncID,_,Peso,_,_,_,_,_,_),
    transporte(Veiculo,Carga,_,_),
    atribuido(IDEstafeta,EncID),
    Peso =< Carga,
    Rating >= 0,
    Rating =< 5,
    Data,
    Hora
).
-entrega(_,_,_,_,_,_) :: (true).
```

Listing 2.16: Invariantes da entrega

Na adição de uma entrega, obviamente, não só o próprio deve ser único como também o estafeta, a encomenda e o transporte devem ser válidos. Para além disso, esta só pode ser adicionada se a encomenda tiver sido atribuída a esse estafeta em específico. O invariante também obriga a que o veículo usado consiga suportar o peso da encomenda, que o *rating* seja um valor entre 0 e 5 e que a data e hora sejam ambas válidas.

Em contrapartida, o invariante de remoção, visto que não existe qualquer outra estrutura que dependa da existência da entrega, pode ser sempre removida.

Estafeta

```
+estafeta(ID,_,_) ::
(
    findall(ID,estafeta(ID,_,_),R),
    length(R,L),
    L == 1
).
-estafeta(ID,_,_) ::
(
    \+ entrega(ID,_,_,_,_,_),
    \+ atribuido(ID,_)
).
```

Listing 2.17: Invariantes do estafeta

Na adição de conhecimento o estafeta apenas é restrito no seu ID. Porém, na sua remoção, este não deve ter entregas nem encomendas atribuídas que o refiram.

Transporte

```
+transporte(Veiculo,Carga,Velocidade,Preco) ::
(
    findall(Veiculo,transporte(Veiculo,_,_),R),
    length(R,L),
    L == 1,
    Carga > 0,
    Velocidade > 0,
    Preco > 0
).
-transporte(Veiculo,_,_,_) ::
(
    \+ entrega(_,Veiculo,_,_,_,_)
).
```

Listing 2.18: Invariantes do transporte

Em conjunto com a unicidade do seu nome, não faz sentido que o transporte tenha uma carga, velocidade, ou preço por km negativos.

Na remoção, visto que a entrega faz referência ao meio de transporte usado, não podemos remover um veículo que esteja registado numa entrega.

Cliente

```
+cliente(ID,_) ::
(
    findall(ID,cliente(ID,_),R),
    length(R,L),
    L == 1
).
-cliente(ID,_) ::
(
    \+ encomenda(_,ID,_,_,_,_,_,_,_)
).
```

Listing 2.19: Invariantes do cliente

Ao cliente, pela sua simplicidade, é-lhe apenas restringido o ID. Para além disso, a sua remoção só pode ser feita caso este não tenha feito nenhuma encomenda.

Rua

```
+rua(ID,_,_) ::
(
    findall(ID, rua(ID,_,_),R),
    length(R,L),
    L == 1
).
-rua(ID,_,_) ::
(
    \+ encomenda(_,_,_,_,_,_,_,_,ID)
).
```

Listing 2.20: Invariantes da rua

De igual modo ao cliente, a rua só é obrigada a ter ID único e a sua remoção só pode ser executada caso não apareça em nenhuma encomenda.

Capítulo 3

Conclusão

Concluídas as tarefas propostas, damos por finalizado este trabalho prático.

Este projeto permitiu consolidar a matéria lecionada nas aulas da unidade curricular, mas também obter um maior conhecimento acerca do funcionamento da lógica quando aplicada à inteligência artificial e também sobre a linguagem PROLOG.

De um modo geral, consideramos o projeto bem sucedido porque cumprimos os objetivos delineados no enunciado e entre nós como equipa.