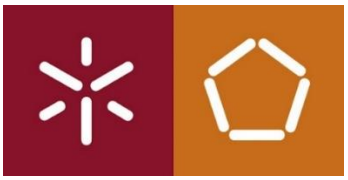

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

INTELIGÊNCIA ARTIFICIAL

2021/2022

REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO

Serviço de Encomendas - Green Distribution

GRUPO 22:

Joana Alves A93290
Maria Cunha A93264
Samuel Lira A94166
Vicente Moreira A93296

Data de Entrega: 02/12/2021

INTRODUÇÃO

Este trabalho consiste no desenvolvimento e construção de um sistema de representação de conhecimento e mecanismos de raciocínio com capacidade para caracterizar um universo de discurso na área da logística de distribuição de encomendas.

Assim, para o desenvolvimento desta fase, construímos a nossa própria base de conhecimento (*Data Set*) com os predicados que achamos mais convenientes. De seguida, passamos ao desenvolvimento de predicados de raciocínio básico sobre essa mesma base de conhecimento.

BASE DE CONHECIMENTO

Para a base de conhecimento, começamos por discutir a melhor forma de representar as várias entidades do sistema, ou seja, procedeu-se à definição da sintaxe e semântica dos vários predicados a serem usados, de forma a obter uma base de dados de fácil interpretação com acessos eficientes ao seu conteúdo. No entanto, devido à complexidade desta, assim como a extensão dos vários requisitos solicitados no enunciado, estipulamos uma estratégia inicial: começar por desenvolver predicados simples cuja função é identificar a informação mais atómica da base de dados e, a partir daí, desenvolver os restantes predicados conforme os requisitos.

No fim, obtivemos uma base de dados com a informação distribuída por predicados mais pequenos e por alguns predicados mais extensos. Apresentamos aqui os seguintes resultados da representação de entidades na base de dados da empresa *Green Distribution*, seguido de alguns exemplos:

- Veículo (ID do Veículo, Tipo de Veículo, Estado do Veículo).
```
veiculo(1, bicicleta, 0).
```
- Eco (Tipo de Veículo, Percentagem Ecologia).
```
eco(bicicleta,100).
```
- Peso (Tipo de Veículo, Peso Máximo Permitido).
```
peso(bicicleta, 5).
```
- Velocidade (Tipo de Veículo, Velocidade Máxima Permitida).
```
velocidade(moto, 35).
```
- Freguesia (ID da Freguesia, Nome da Freguesia).
```
freguesia(1, 'Gualtar').
```
- Cliente (ID do Cliente, Nome do Cliente, Idade).
```
cliente(1, 'Carlos',20).
```
- Estafeta (ID do Estafeta, Nome do Estafeta, Idade).
```
estafeta(1, 'Cláudio', 30).
```

- Entrega (ID da Entrega, ID do Estafeta, ID do Cliente).

```
entrega(10,4,10).
```

- Info (ID da Entrega, ID do Veículo, Tempo de Entrega, Preço, Data de Entrega, Satisfação, Estado, ID da Freguesia, Rua, Peso, Volume).

```
info(1 ,2,113,25,[2021,10,2 ,15,13],0.8,0,6,'Rua Conselheiro Lobato'     ,41,8 ).
```

Esclarecemos de seguida a sintaxe dos valores representados:

- **ID's:** Inteiro
- **Tipo de Veículo:** String
- **Estado do Veículo:** Inteiro → (0 - Operacional / 1 - Avariado)
- **Percentagem Ecologia:** Inteiro (0%-100%)
- **Peso Máximo Permitido:** Inteiro (Kg)
- **Velocidade Máxima Permitida:** Inteiro (Km/H)
- **Nomes:** String
- **Idade:** Inteiro (Anos)
- **Tempo de Entrega:** Inteiro (minutos)
- **Preço:** Inteiro (€)
- **Data de Entrega:** Lista de Inteiros → [Ano,Mês,Dia,Hora,Minuto]
- **Satisfação:** Float com 1 casa decimal (0-5)
- **Estado:** Inteiro → (0 – Entregue / 1 – Cancelado / 2 – Não Entregue)
- **Rua:** String
- **Peso:** Inteiro (Kg)
- **Volume:** Float c\ 1 casa decimal (dm^3)

Depois de gerarmos algumas freguesias, clientes e estafetas, procedemos à escrita de entregas para a base de dados. No entanto, rapidamente nos apercebemos que havia uma forte necessidade de automatizar este processo de escrita. Assim, decidimos escrever um pequeno programa em *Java* que nos permite gerar *N* entregas com valores aleatórios dentro de um limite definido para cada parâmetro.

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática
Inteligência Artificial

DESENVOLVIMENTO DE PREDICADOS

No decorrer do projeto, tivemos algumas dificuldades em conseguir perceber a dinâmica de alguns predicados pré-definidos na linguagem *Prolog*. No entanto, ao discutir o assunto em grupo acabamos por entender a utilidade destes, como o *findall*, que permitiu a simplificação do código a nível de pesquisa de informação, *maplist*, que manipula elementos de uma lista de forma simples. Este facilitaram no desenvolvimento do projeto, podendo obter facilmente um *set* de dados contendo a informação necessária, utilizando apenas os predicados *findall* e *maplist*.

Desenvolvemos também outros predicados auxiliares de forma a facilitar a compreensão dos predicados mais complexos, sendo estes úteis ao longo do projeto. Predicados como: *take* de listas; *sumlist* para somatórios, *remove_duplicates* para eliminar repetições numa lista e *entregasNumIntervalo* que, dado uma lista de entregas e duas datas (uma inicial e outra final), filtra a lista de entregas para devolver apenas as entregas realizadas dentro desse período de tempo.

No que toca aos requerimentos mínimos do trabalho, todos os predicados foram desenvolvidos. Assim, para melhor explicar o raciocínio por detrás de cada *query*, assim como o funcionamento dos predicados auxiliares, apresentamos uma breve explicação para cada uma das *queries* principais, assim como algumas das auxiliares mais utilizadas.

PREDICADOS AUXILIARES

- Predicados *Get*, utilizados para a devolução eficiente de conteúdos acerca de entregas.

```
getVeiculo(IDEntrega,IDVeiculo)     :- info(IDEntrega,IDVeiculo,_,_,_,_,_,_,_,_,_).
getTempoDeEntrega(IDEntrega,Tempo) :- info(IDEntrega,_,Tempo,_,_,_,_,_,_,_,_).
getPreco(IDEntrega,Preco)           :- info(IDEntrega,_,_,Preco,_,_,_,_,_,_,_).
getData(IDEntrega,Data)             :- info(IDEntrega,_,_,_,Data,_,_,_,_,_,_).
getSatisfacao(IDEntrega,Sat)        :- info(IDEntrega,_,_,_,_,Sat,_,_,_,_,_).
getEstado(IDEntrega,Estado)         :- info(IDEntrega,_,_,_,_,_,Estado,_,_,_,_).
getFreguesia(IDEntrega,Freg)        :- info(IDEntrega,_,_,_,_,_,_,Freg,_,_,_).
getRua(IDEntrega,Rua)               :- info(IDEntrega,_,_,_,_,_,_,_,Rua,_,_).
getPeso(IDEntrega,Peso)             :- info(IDEntrega,_,_,_,_,_,_,_,_,Peso,_).
getVolume(IDEntrega,Volume)         :- info(IDEntrega,_,_,_,_,_,_,_,_,_,Volume).
```

- *Remove_duplicates*, remove membros duplicados na lista, devolvendo uma lista com elementos únicos.

```
remove_duplicates([],[]).
remove_duplicates([H | T], L) :-
    member(H, T),
    remove_duplicates( T, L).
remove_duplicates([H | T], [H|T1]) :-
     \+member(H, T),
     remove_duplicates( T, T1).
```

- *take*, devolve uma lista com os *N* primeiros elementos de uma lista fornecida.

```
take(0,_,[]).
take(Num,[],[]) :- maior(Num,0,Num).
take(N,[Elem|T1],[Elem|T2]) :-
        NNovo is N-1,
        take(NNovo,T1,T2).
```

- *mediaLista*, calcula a média de elementos de uma lista. Retorna 0 caso a lista seja nula.

```
mediaLista(L,Media):-
        sumlist(L,Somatorio),
        length(L,Tamanho),
        (Tamanho > 0 -> Media is Somatorio/Tamanho; Media is 0).
```

- *maxPar*, dado uma lista de pares (Chave,Valor), é devolvido o par com maior valor no conjunto.

```
maxPar(L,Max) :- maxPar(L,(0,0),Max).
maxPar([],Max,Max).
maxPar([(ID,Valor)|T], (P1,P2),Max) :-
        VMax is max(Valor,P2),
        (VMax =:= Valor -> maxPar(T,(ID,Valor),Max);maxPar(T,(P1,P2),Max)).
```

- *entregasNumIntervalo*, fornecendo uma lista de entregas, assim como duas datas (uma inicial, outra final), filtra a lista de entregas, devolvendo apenas aquelas cujas datas de entregas correspondam ao período de tempo.

```
entregasNumIntervalo(_,_,[],[]).
entregasNumIntervalo(DataInicio,DataFim,[IdEntrega|T1],[IdEntrega|T2]) :-
            getData(IdEntrega,DataEntrega),
            testaDataNumIntervalo(DataInicio,DataFim,DataEntrega),
            entregasNumIntervalo(DataInicio,DataFim,T1,T2).
entregasNumIntervalo(DataInicio,DataFim,[_|T1],T2) :-
            entregasNumIntervalo(DataInicio,DataFim,T1,T2).
```

PREDICADOS PRINCIPAIS

Predicado 1 (identificar o estafeta que utilizou mais vezes um meio de transporte mais ecológico):

```
estafetaMaisEcologico(Nome) :-
      idMaximoEstafeta(Max),
      estafetaMaisEcologico(0,Max,Res),
      maxPar(Res,(ID,_)),
      getEstafeta(ID,Nome).
```

```
estafetaMaisEcologico(N,Max,_) :- N is Max+1.
estafetaMaisEcologico(Indice,Max,[(Indice,Media)|T]) :-
      entregasDeUmEstafeta(Indice,Entregas),
      maplist(getEcologiaEntrega(),Entregas,Ecologias),
      mediaLista(Ecologias,Media),
      N is Indice+1,
      estafetaMaisEcologico(N,Max,T).
```

Para este predicado, utilizamos vários predicados auxiliares para que a sua resolução se tornasse de fácil compreensão. Dessa forma, começamos por verificar o número máximo de estafetas presente na base de conhecimento naquele momento através do predicado auxiliar "idMaximoEstafeta". De seguida, através desse número, iteramos sobre os estafetas ("estafetaMaisEcologico") e armazenamos numa lista os pares (IDEstafeta, ValorEcologia) correspondentes ao identificador do estafeta na base de conhecimento e à sua média de ecologia. Terminando, calculamos o valor máximo de ecologia presente na lista de pares, verificando o identificador do estafeta correspondente e colocando no argumento "Nome" o nome do mesmo.

Predicado 2 (identificar que estafetas entregaram determinadas encomendas a um determinado cliente):

```
entregasDoCliente(_,[],[]).
entregasDoCliente(IdCliente,[IdEntrega|Entregas],[(Nome,IdEntrega)|T]) :-
      entrega(IdEntrega,IdEstafeta,IdCliente),
      estafeta(IdEstafeta,Nome,_),
      entregasDoCliente(IdCliente,Entregas,T).
entregasDoCliente(IdCliente,[_|Entregas],T) :-
      entregasDoCliente(IdCliente,Entregas,T).
```

Neste predicado, ao receber uma lista de identificadores de entregas, percorre-a e avalia se as mesmas pertencem ao cliente fornecido. Caso se verifique, armazena na lista resultado o par (Nome, IDEntrega), onde "Nome" é o nome associado ao estafeta e "IDEntrega" é o identificador da entrega feita pelo mesmo ao cliente.

Predicado 3 (identificar os clientes servidos por um determinado estafeta):

```
clientesDeUmEstafeta(IdEstafeta,Res) :-
      findall((Cliente), entrega(_,IdEstafeta,Cliente),Lista),
      remove_duplicates(Lista,L),
      getClientes(L,Res).
```

Neste predicado, através da utilização do predicado auxiliar *findall*, obtemos uma lista com os vários identificadores de clientes aos quais o estafeta fornecido entregou encomendas. De seguida, retiramos os identificadores repetidos para a informação não se tornar redundante. O último passo foi fazer a correspondência entre os identificadores e os nomes dos clientes, para, assim, apresentarmos uma solução mais "*user friendly"*.

Predicado 4 (calcular o valor faturado pela *Green Distribution* num determinado dia):

```
faturaUmDia(Data,Res) :-
      findall((IDEntrega), entrega(IDEntrega,_,_),AllEntregas),
      entregasDeUmDia(Data,AllEntregas,EntregasFiltradas),
      maplist(getPreco(),EntregasFiltradas,ListaPrecos),
      sumlist(ListaPrecos,Res).
```

Neste predicado, recorrendo ao predicado *findall*, obtemos todas as identificações de entregas, sendo depois submetida a um predicado auxiliar já referido, *entregasDeUmDia*, de modo a restringir a lista de encomendas a uma determinada data. De seguida, filtramos esta lista de modo que os preços de cada entrega sejam reunidos numa outra chamada de "ListaPrecos". Por fim, recorrendo ao predicado *sumlist*, somamos todos os preços reunidos anteriormente.

Predicado 5 (identificar quais as zonas com maior volume de entregas por parte da *Green Distribution*):

```
maiorVolumeFreguesia(Res) :-
      listaVolumesPorFreguesia(ListaPares),
      maxPar(ListaPares,(Id,_)),
      freguesia(Id,Res).
```

```
listaVolumesPorFreguesia(Lista) :-
      idMaximoFreguesia(Max),
      listaVolumesPorFreguesia(0,Max,Lista).
listaVolumesPorFreguesia(Max,Max,[]).
listaVolumesPorFreguesia(N,Max,[(Freg,Vol)|T]) :-
      Freg is N+1,
      volumePorFreguesia(Freg,Vol),
      listaVolumesPorFreguesia(Freg,Max,T).
```

Para este predicado, tivemos que recorrer a uma metodologia sequencial, ou seja, obtemos os valores dos volumes totais de cada freguesia invidualmente, e no fim recorremos ao tratamento dos dados obtidos.

Começamos por analisar o número máximo de freguesias presentes na *BD*, de forma a estabelecer um alvo, de seguida executamos de forma iterativa o predicado *volumePorFreguesia* para cada freguesia, obtendo uma lista de pares (chave,valor) onde por fim aplicamos o predicado *maxPar*, que devolve a freguesia com o maior volume de entregas.

Predicado 6 (calcular a classificação média de satisfação de cliente para um determinado estafeta):

```
mediaSatisfacaoEstafeta(IdEstafeta,Res) :-
      entregasDeUmEstafeta(IdEstafeta,ListaEntregas),
      maplist(getSatisfacao(),ListaEntregas,ListaSats),
      mediaLista(ListaSats,Res).
```

Este predicado começa por armazenar numa lista os identificadores das entregas feitas pelo estafeta em questão. De seguida, através do predicado auxiliar *maplist*, guarda numa outra lista as avaliações de cada entrega. Por fim, calcula uma média de avaliações através do predicado auxiliar "mediaLista", que apenas recebe uma lista e calcula uma média dos elementos presente nela.

Predicado 7 (identificar o número total de entregas pelos diferentes meios de transporte, num determinado intervalo de tempo):

```
totalEntregasPorTransporte(DataInicio,DataFinal,[("bicicleta",Bic),("carro",Carro),("mota",Mota)]) :-
    findall((IDEntrega),entrega(IDEntrega,_,_),AllEntregas),
    entregasNumIntervalo(DataInicio,DataFinal,AllEntregas,Filtrado),
    maplist(getTipoVeiculoEntrega(),Filtrado,ListaTipos),
    count(ListaTipos,bicicleta,Bic),
    count(ListaTipos,carro,Carro),
    count(ListaTipos,moto,Mota).
```

Para este predicado, começamos por recolher todas as entregas na base de dados, filtrando-as de acordo com o período de tempo definido. De seguida aplicamos o *maplist* de forma a obtermos uma lista com os tipos de veículos utilizados, onde executamos sequencialmente a contagem de tipo de veículo na lista, ou seja, contamos o número de vezes que cada tipo de veículo aparece, apresentando o resultado final.

Predicado 8 (identificar o número total de entregas pelos estafetas, num determinado intervalo de tempo):

```
numeroEntregasIntervalo(DataIncio,DataFinal,Res) :-
    findall((IDEntrega),entrega(IDEntrega,_,_),AllEntregas),
    entregasNumIntervalo(DataIncio,DataFinal,AllEntregas,Filtro),
    length(Filtro,Res).
```

A execução deste predicado é relativamente simples. Este começa por recolher todas as entregas na base de dados, as quais irá aplicar o predicado *entregasNumIntervalo* que irá filtrar esta lista de entregas. Por fim calcula o tamanho desta lista, pois irá corresponder ao número total de entregas realizados pelos estafetas nesse período de tempo.

Predicado 9 (calcular o número de encomendas entregues e não entregues pela *Green Distribution*, num determinado período de tempo):

```
estadoEntregasIntervalo(DataInicio,DataFinal,[("entregues",Entregue),("nao entregues",NaoEntregues)]) :-
      findall((IDEntrega),entrega(IDEntrega,_,_),AllEntregas),
      entregasNumIntervalo(DataInicio,DataFinal,AllEntregas,Filtro),
      maplist(getEstado(),Filtro,ListaEstados),
      count(ListaEstados,0,Entregue),
      count(ListaEstados,1,Cancelado),
      count(ListaEstados,2,NEntregues),
      NaoEntregues is Cancelado+NEntregues.
```

Para este predicado, começamos por recolher todas as entregas na base de dados, filtrando-as de acordo com o período de tempo definido. De seguido recolhemos os valores do estado de cada entrega, onde de seguida executamos sequencialmente a contagem de cada estado na lista, ou seja, contamos o número de vezes que cada estado aparece. Por fim somamos valores obtidos dos estados "Cancelados" e "Não Entregues" e apresentamos o resultado final.

Predicado 10 (calcular o peso total transportado por estafeta num determinado dia):

```
pesoBrutoDeUmEstafetaUmDia(IdEstafeta,Data,Res) :-
      entregasDeUmEstafeta(IdEstafeta,Entregas),
      entregasDeUmDia(Data,Entregas,EntregasFinais),
      maplist(getPeso(),EntregasFinais,ListaPesos),
      sumlist(ListaPesos,Res).
```

Este predicado recolhe numa lista "EntregasFinais", com o auxílio dos predicados *entregasDeUmEstafeta* e *entregasDeUmDia*, as entregas que este estafeta efetua num dia estipulado. Assim, recorrendo a *maplist* e a *sumlist*, filtramos a lista "EntregasFinais" obtendo o peso de cada entrega, realizando, por fim, um somatório desta.

CONCLUSÃO

Este projeto desenvolveu-se numa nova linguagem: *Prolog*. Inicialmente, devido à falta de conhecimento que os membros do nosso grupo apresentavam acerca desta, tornou-se um desafio implementar de forma eficiente e correta o que nos era pedido. Felizmente, à medida que o grupo se juntava para tentar esclarecer dúvidas e desenvolver código, estas dificuldades foram gradualmente ultrapassadas.

Assim, ao concluirmos esta fase do projeto, reparamos que o grupo adquiriu alguma familiaridade com a escrita de código de forma declarativa no âmbito de inteligência artificial, sendo esta relevante atualmente. Como tal, encontrámo-nos satisfeitos com o trabalho desenvolvido, denotando o desafio que este apresentou.