



**UNIVERSIDADE FEDERAL
DE SANTA CATARINA**
Centro Tecnológico - CTC

DAS - Departamento de Automação e Sistemas
Programa de Pós-Graduação em Engenharia de Automação e Sistemas
DAS 410057 - Inteligência Artificial

Trabalho prático: Representação de conhecimento Agência de viagens

Autores:

Afonso da Fonseca Braga

Ana Caroline Bonafin

Matrícula:

201901886

201901660

February 5, 2021

Contents

1	Introdução	2
2	Problema Proposto	3
3	Desenvolvimento	4
3.1	Fatos	4
3.2	Regras	4
3.3	Consultas	6
4	Conclusão	10
5	Apêndice	12
5.1	Apendice 1	12

1 Introdução

A inteligência artificial tenta compreender e construir entidades inteligentes medindo seu sucesso comparando-o a um conceito ideal de inteligência, chamado de racionalidade, ou em termos de fidelidade ao desempenho humano.

Um sistema é racional se *faz a coisa certa* dado o que sabe. Uma das abordagens racionalistas tenta codificar o *pensamento correto* a partir de silogismos. Estes fornecem padrões para estruturas de argumentos que resultam em conclusões corretas, ao receberem premissas corretas. De acordo com [1] tal raciocínio dedutivo deveria conduzir a forma como a mente opera, surgindo assim o estudo da lógica.

No dia a dia, a linguagem natural pode ser utilizada para representar a lógica, entretanto, esta forma é passível de inúmeras interpretações. Uma mesma frase ou sentença pode ter diversos significados dependendo de seu interpretador. Uma forma de amenizar esse problema na hora de representar o conhecimento em sistemas computacionais é utilizar a lógica de predicados, a qual se destaca por sua proximidade com a linguagem natural.

Além de conectivos lógicos ($\neg, \wedge, \vee, \rightarrow$), as fórmulas da lógica de predicados são compostas por objetos, predicados (que estabelecem uma relação entre objetos), variáveis (estabelecem fatos a respeito dos objetos) e quantificadores (\forall, \exists).

Prolog é uma linguagem declarativa baseada na lógica de predicados de primeira ordem. Utiliza uma coleção de fatos e de relações lógicas, chamadas de regras, que expressam o domínio relacional do problema que se deseja resolver. O programa executa de modo iterativo a partir de consultas formuladas pelo usuário, uma base de dados, regras relacionais e um mecanismo de unificação, que juntos, produzem uma solução a partir métodos eficientes de prova.

2 Problema Proposto

Nos dias de hoje, no ramo do turismo existem infinitas opções de destinos e hotelaria à disposição dos viajantes. Agências são responsáveis por calcular rotas de transporte e hospedagem, criando pacotes muita das vezes exclusivos para seus clientes. Tendo em vista este cenário, foi proposto criar um programa em *Prolog* capaz de oferecer pacotes de viagens, sendo compostos por passagens aéreas e diárias, a partir de informações fornecidas pelos clientes.

3 Desenvolvimento

O capítulo encontra-se dividido em três partes. A primeira parte contém informações sobre os fatos que foram utilizados pelo programa. Em seguida são apresentadas todas as regras de inferência desenvolvidas para a aplicação. E, por último, são descritas algumas consultas realizadas, a critério de demonstração.

3.1 Fatos

Fatos são responsáveis por identificar objetos e indivíduos na linguagem *Prolog*. São utilizados para desenvolver a base de dados do programa.

Os tópicos a baixo representam os moldes de como os fatos foram criados. Os fatos relacionados com cada molde encontram-se no Apêndice 5.1.

- `moeda(cod_moeda, nome, conversao)`.
O tipo moeda é composto por um código único, nome da moeda e seu valor de conversão para o real (quantos reais equivalem a um valor da moeda).
- `pais(cod_pais, nome, cod_moeda)`.
O tipo país possui um código único, seu nome e o código da moeda utilizada no local.
- `cidade(cod_cidade, nome, cod_pais)`.
O tipo cidade é composto pelo código único da cidade, seu nome e o código do país onde ela se situa.
- `voo(cod_voo, origem, destino, preço, tempo_min)`.
O tipo voo é composto por seu código único, a cidade de origem, a cidade de destino, o preço e a duração do voo em minutos.
- `hotel(cod_hotel, nome, cod_cidade, estrelas, preco_diaria)`.
O tipo hotel é composto por um código único do hotel, seu nome, o código da cidade que está localizado, quantas estrelas possui e o preço de sua diária em moeda local.

3.2 Regras

Regras são criadas para relacionar diversos tipos de objetos e indivíduos. Os tópicos abaixo apresentam as regras utilizadas no programa. As regras relacionadas à recursividade não estão listadas abaixo. O código completo encontra-se no Apêndice 5.1.

- `viagem(X,Y, Cod_cidades_inicial, Cod_cidades_final, Cod_voos, Cod_voos_final,Custo,Total_aereo)`
Esta regra lista todos os voos disponíveis partindo da cidade X até a cidade Y. Ela existe também para a recursividade, caso entre as cidades X e Y existam

outras cidades (conexões). É necessário informar um vetor com as cidades já visitadas (*Cod_cidades_inicial*) e os códigos de voos (*Cod_voos*). Na variável *Cod_voos_final* e *Cod_cidades_final* encontram-se os resultados para a viagem. A variável *Custo* armazena o custo parcial, utilizado para a recursividade e a variável *Total_aereo* fornece o custo total da viagem em reais.

- *imprimir_voos_linha*(*Cod_voos*,*Cod_voos_Original*,*Total_aereo*)
É responsável por imprimir a linha dos voos, contendo seus códigos, a cidade de origem, conexões e cidade destino e o custo total aéreo.
- *busca_voos*(*X*,*Y*,*Cod1*,*Cod_voos_final*,*Total_aereo*)
Utilizada para simplificar a regra viagem. *Cod1* é o código da cidade de origem.
- *imprime_voo*(*Cod_voos_final*)
Responsável por imprimir o cabeçalho dos voos e está relacionada com a regra *imprimir_voos_linha*.
- *busca_hotel*(*Cod_cidade*,*Diarias*,*Cod_hotel*,*Total_hotel*,*Cod_moeda*)
Busca todos os hotéis disponíveis na cidade.
- *imprime_hotel*(*Cod_hotel*,*Diarias*,*Total_hotel*,*Cod_moeda*)
Imprime todos os hotéis disponíveis na cidade.
- *cod_cidade_cod_pais*(*Cod_pais*,*Cod_cidade*)
Relaciona o código de um país com sua cidade.
- *cod_cidade_pais*(*Pais*,*Cod_cidade*)
Relaciona o código de uma cidade com seu país.
- *cidade_cod_pais*(*Cod_pais*,*Cidade*)
Relaciona o nome de uma cidade com o código do país.
- *cidade_pais*(*Pais*,*Cidade*)
Relaciona o nome de uma cidade com o nome de um país.
- *cod_pais_cod_moeda*(*Cod_moeda*,*Cod_pais*)
Relaciona o código de um país com o código de uma moeda.
- *cod_pais_moeda*(*Moeda*,*Cod_pais*)
Relaciona o código de um país com o nome de uma moeda.
- *pais_moeda*(*Moeda*,*Pais*)
Relaciona o nome de um país com o nome de sua moeda.
- *cod_cidade_cod_moeda*(*Cod_moeda*,*Cod_cidade*)
Relaciona o código de uma cidade com o código de uma moeda.
- *cod_cidade_moeda*(*Moeda*,*Cod_cidade*)
Relaciona o código de uma cidade com o nome de uma moeda.
- *pacote*(*X*,*Y*,*Diarias*, *Max_voos*)
Imprime todas as possibilidades de pacotes saindo da cidade *X* até a cidade *Y*,

com o número de diárias e quantidade máxima de voos permitidos para cada trajeto separado.

- pacote_moeda(X,Moeda,Diarias,Max_voos)
Regra que imprime pacotes partindo da cidade X, para destinos que utilizam determinada moeda.
- pacote_pais(X,Pais,Diarias,Max_voos)
Regra utilizada para imprimir pacotes partindo da cidade X até o país escolhido.
- pacote_preco(X,Preco,Diarias,Max_voos)
Imprime todos os pacotes possíveis, a partir da cidade X até um valor total.

3.3 Consultas

A forma mais fácil de realizar consultas seria utilizando as regras que se iniciam com a palavra pacote. Por exemplo:

- pacote_pais(hannover,brasil,10,2).
- pacote_preco(florianopolis,2400,1,2).
- pacote_moeda(buenos_aires,dolar_americano,10,2).
- pacote(madrid,Cidade,10,2).

As figuras 1, 2, 3 e 4 apresentam as soluções encontradas pelo algoritmo quando as consultas dos itens (i), (ii), (iii) e (iv) são executadas, respectivamente.

```

Desktop : swipl — Konsole
File Edit View Bookmarks Settings Help

?- pacote_pais(hannover,brasil,10,2).

Voos
Preço  Códigos          Cidades
2100   [v029,v028]        [hannover,frankfurt,sao_paulo]

Voos
Preço  Códigos          Cidades
1500   [v009]           [sao_paulo,hannover]

Hoteis
Código Nome      Estrelas  Diarias  Preço (brl)  Total (brl)  Total (brl)
h002   plaza_Sampa  ***      10       120         1200         1200

TOTAL: 4800
true ;
false.

?-

```

Figure 1: Solução apresentada pelo algoritmo para a busca do item (i)

```

Desktop : swipl — Konsole
File Edit View Bookmarks Settings Help

?- pacote_preco(florianopolis,2400,1,2).

Voos
Preço      Códigos      Cidades
1200      [v004,v007]      [florianopolis,sao_paulo,rio_de_janeiro]

Voos
Preço      Códigos      Cidades
350       [v001,v002]      [rio_de_janeiro,sao_paulo,florianopolis]

Hoteis
Código Nome      Estrelas Diarias Preço (brl) Total (brl) Total (brl)
h001  plaza_Rio      ****      1      100      100      100

TOTAL: 1650
true ;

Voos
Preço      Códigos      Cidades
200       [v004]      [florianopolis,sao_paulo]

Voos
Preço      Códigos      Cidades
200       [v002]      [sao_paulo,florianopolis]

Hoteis
Código Nome      Estrelas Diarias Preço (brl) Total (brl) Total (brl)
h002  plaza_Sampa    ***      1      120      120      120

TOTAL: 520
true ;

```

Figure 2: Solução apresentada pelo algoritmo para a busca do item (ii)

```

Desktop : swipl — Konsole
File Edit View Bookmarks Settings Help

?- pacote_moeda(buenos_aires,dolar_americano,10,2).

Voos
Preço      Códigos      Cidades
200       [v031]      [buenos_aires,new_york]

Voos
Preço      Códigos      Cidades
4000      [v023]      [new_york,buenos_aires]

Hoteis
Código Nome      Estrelas Diarias Preço (usd) Total (usd) Total (brl)
h008  mercury      ***      10      180      1800      7200

TOTAL: 11400
true ;
false.

?-

```

Figure 3: Solução apresentada pelo algoritmo para a busca do item (iii)


```
Desktop : swipl — Konsole
File Edit View Bookmarks Settings Help

?- pacote(madrid,Cidade,10,2).

Voos
Preço  Códigos          Cidades
900    [v017,v018]      [madrid,paris,frankfurt]

Voos
Preço  Códigos          Cidades
200    [v012]          [frankfurt,madrid]

Hoteis
Código Nome          Estrelas  Diarias  Preço (eur)  Total (eur)  Total (brl)
h009   ibis_rich      ***      10      80          800          3600.0

TOTAL: 4700.0
Cidade = frankfurt ;

Voos
Preço  Códigos          Cidades
900    [v017,v018]      [madrid,paris,frankfurt]

Voos
Preço  Códigos          Cidades
250    [v013,v014]      [frankfurt,paris,madrid]

Hoteis
Código Nome          Estrelas  Diarias  Preço (eur)  Total (eur)  Total (brl)
h009   ibis_rich      ***      10      80          800          3600.0

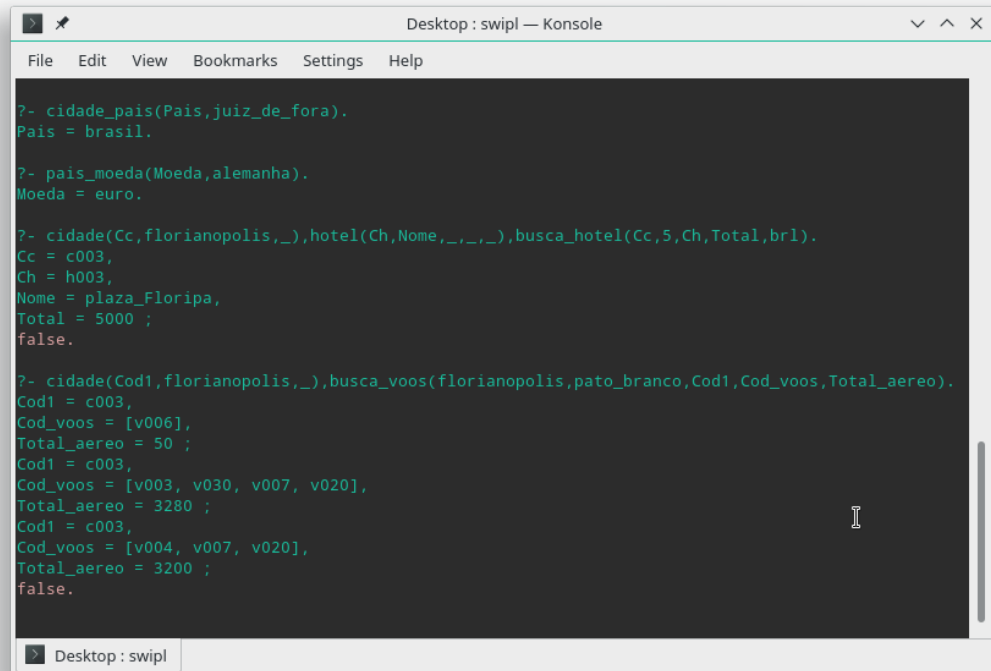
TOTAL: 4750.0
Cidade = frankfurt
```

Figure 4: Solução apresentada pelo algoritmo para a busca do item (iv)

Outras consultas, não envolvendo pacotes de viagens também são possíveis, como por exemplo:

- (v) cidade_pais(juiz_de_fora,Pais).
- (vi) pais_moeda(alemanha,Moeda).
- (vii) cidade(Cod1,florianopolis,-), busca_voos(florianopolis,pato_branco,Cod1, Cod_voos,Total_aereo).
- (viii) cidade(Cc,florianopolis,-), hotel(Ch,Nome,-,-), busca_hotel(Cc,5,Ch, Ctotal, brl).

A figura 5 apresenta o resultado das buscas, (v),(vi),(vii) e (viii), executadas pelo programa implementado.



```
Desktop : swipl — Konsole
File Edit View Bookmarks Settings Help

?- cidade_pais(Pais,juiz_de_fora).
Pais = brasil.

?- pais_moeda(Moeda,alemanha).
Moeda = euro.

?- cidade(Cc,florianopolis,_),hotel(Ch,Nome,_,_,_),busca_hotel(Cc,5,Ch,Total,brl).
Cc = c003,
Ch = h003,
Nome = plaza_Floripa,
Total = 5000 ;
false.

?- cidade(Cod1,florianopolis,_),busca_voos(florianopolis,pato_branco,Cod1,Cod_voos,Total_aereo).
Cod1 = c003,
Cod_voos = [v006],
Total_aereo = 50 ;
Cod1 = c003,
Cod_voos = [v003, v030, v007, v020],
Total_aereo = 3280 ;
Cod1 = c003,
Cod_voos = [v004, v007, v020],
Total_aereo = 3200 ;
false.
```

Figure 5: Solução apresentada pelo algoritmo para a busca das consultas (v), (vi), (vi), (vii) e (viii)

4 Conclusão

A linguagem *Prolog* torna possível a implementação de regras de inferência de modo simples e eficaz, visto que vários elementos básicos da linguagem foram herdados da lógica de predicados. Dispondo de todos os fatos e regras as consultas são executadas de forma rápida, podendo ser facilmente *debugadas*. Entretanto, por ser uma linguagem declarativa, pode haver dificuldades na programação, tendo em mente que muitos usuários estão acostumados com linguagens imperativas. O conhecimento prévio de lógica de predicados e regras de inferência se faz necessário para que o código fique limpo e apresentável.

O interpretador utilizado foi o *Swi-Prolog*, que é multiplataforma, não havendo problemas para executar o mesmo programa em diversos sistemas operacionais como *Mac OS*, *Linux* e *Windows*.

Neste trabalho, a maior dificuldade encontrada pela equipe ao desenvolver o programa foi no momento de imprimir os resultados na tela de forma organizada, sendo assim, foi necessário o uso de vetores. Adquirindo experiência foi possível solucionar outros obstáculos ao longo do trabalho.

Embora existam inúmeras vantagens, acredita-se que seja necessário a integração do *Prolog* com outras linguagens de programação para que seja desenvolvida uma aplicação completa. Um exemplo seria a programação de um robô, onde a parte de controle do hardware seria melhor desenvolvida em linguagens imperativas (*C++*, *Python*, etc) e o Agente desenvolvido em *Prolog*.

References

- [1] Peter Norvig and Stuart Russell. *Inteligência Artificial: Tradução da 3a Edição*, volume 1. Elsevier Brasil, 2014.

5 Apêndice

5.1 Apêndice 1

```
1  % -----
2  % |          FATOS          |
3  % -----
4
5  % voo(cod.voo, origem, destino, preco, tempo_min)
6
7  voo(v001,rio-de-janeiro,sao-paulo,150,52) .
8  voo(v002,sao-paulo,florianopolis,200,30) .
9  voo(v003,florianopolis,juiz-de-fora,200,30) .
10 voo(v004,florianopolis,sao-paulo,200,30) .
11 voo(v005,pato-branco,juiz-de-fora,50,30) .
12 voo(v006,florianopolis,pato-branco,50,30) .
13 voo(v007,sao-paulo,rio-de-janeiro,1000,45) .
14 voo(v008,sao-paulo,frankfurt,2000,600) .
15 voo(v009,sao-paulo,hannover,1500,660) .
16 voo(v010,rio-de-janeiro,berlin,2000,45) .
17 voo(v011,frankfurt,marburg,30,20) .
18 voo(v012,frankfurt,madrid,200,120) .
19 voo(v013,frankfurt,paris,120,30) .
20 voo(v014,paris,madrid,130,120) .
21 voo(v015,new-york,florianopolis,3500,700) .
22 voo(v016,berlin,madrid,350,180) .
23 voo(v017,madrid,paris,300,120) .
24 voo(v018,paris,frankfurt,600,60) .
25 voo(v019,marburg,hannover,200,30) .
26 voo(v020,rio-de-janeiro,pato-branco,2000,120) .
27 voo(v021,berlin,madrid,250,65) .
28 voo(v022,berlin,new-york,3000,540) .
29 voo(v023,new-york,buenos-aires,4000,500) .
30 voo(v024,buenos-aires,sao-paulo,1200,100) .
31 voo(v025,buenos-aires,berlim,3600,750) .
32 voo(v026,buenos-aires,madrid,2300,2600) .
33 voo(v027,madrid,pato-branco,3000,1200) .
34 voo(v028,frankfurt,sao-paulo,2000,600) .
35 voo(v029,hannover,frankfurt,100,40) .
36 voo(v030,juiz-de-fora,sao-paulo,80,30) .
37 voo(v031,buenos-aires,new-york,200,500) .
38 voo(v032,sao-paulo,buenos-aires,400,120) .
39
40 % cidade(cod.cidade, nome, pais)
41
42 cidade(c001,rio-de-janeiro,p001) .
43 cidade(c002,sao-paulo,p001) .
44 cidade(c003,florianopolis,p001) .
45 cidade(c004,juiz-de-fora,p001) .
46 cidade(c005,pato-branco,p001) .
47 cidade(c006,buenos-aires,p002) .
48 cidade(c007,new-york,p003) .
49 cidade(c008,frankfurt,p004) .
```

```

50 cidade(c009,marburg,p004).
51 cidade(c010,hannover,p004).
52 cidade(c011,berlin,p004).
53 cidade(c012,paris,p005).
54 cidade(c013,madrid,p006).
55
56
57 % Pais (cod.pais, nome, moeda)
58
59 pais(p001, brasil, brl).
60 pais(p002, argentina, ars).
61 pais(p003, estados_unidos, usd).
62 pais(p004, alemanha, eur).
63 pais(p005, franca, eur).
64 pais(p006, espanha, eur).
65
66
67 % Moeda (cod.moeda, nome, conversao)
68
69 moeda(brl, real, 1).
70 moeda(ars, peso_argentino, 0.55).
71 moeda(eur, euro, 4.5).
72 moeda(usd, dolar_americano, 4).
73
74 % Hotel (cod.hotel, nome, cidade, estrelas, preco.diaria)
75
76 hotel(h001,plaza_Rio,c001,'****',100).
77 hotel(h002,plaza_Sampa,c002,'***',120).
78 hotel(h003,plaza_Floripa,c003,'*****',1000).
79 hotel(h004,plaza_JF,c004,'***',220).
80 hotel(h005,plaza_JF3,c004,'***',40).
81 hotel(h006,plaza_PATO,c005,'***',80).
82 hotel(h007,hilton,c006,'***',1180).
83 hotel(h008,mercury,c007,'***',180).
84 hotel(h009,ibis_rich,c008,'***',80).
85 hotel(h010,ibis_budget,c009,'***',15).
86 hotel(h011,carlton,c0010,'***',110).
87 hotel(h012,linden,c011,'***',180).
88 hotel(h013,uova,c012,'***',130).
89 hotel(h014,salsalito,c013,'***',180).
90
91 % -----
92 % |          REGRAS          |
93 % -----
94
95 % Regra de voo
96
97 viagem(X,Y, Cod_cidades_inicial, Cod_cidades_final, Cod_voos, ...
    Cod_voos_final,Custo,Total_aereo) :- voo(Cod_voo,X,Y,Preco, ...
    -), % Busca se existe algum voo direto
98 cidade(Cod2, Y, -), % Buscar o codigo da cidade destino
99 not(member(Cod2,Cod_cidades_inicial)), % A cidade ...
    destino ja se encontra na lista de conexoes?
100 append(Cod_cidades_inicial,[Cod2],Cod_cidades_final), % ...
    Acrescenta no vetor cidade destino

```

```

101     append(Cod.voos,[Cod.voo],Cod.voos.final),
102     Total.aereo is Custo+Preco.    % Acrescenta o voo na lista de ...
        voos escolhidos
103
104
105     viagem(X,Y, Cod.cidades.inicial, Cod.cidades.final, Cod.voos, ...
        Cod.voos.final, Custo,Total.aereo) :- voo(Cod.voo,X,Z,Preco, ...
        _),
        % Busca uma conexao
106     cidade(Cod2, Z,_),    % Buscar o codigo da cidade destino
107     not(member(Cod2,Cod.cidades.inicial)), % A cidade destino ja ...
        se encontra na lista de conexoes?
108     append(Cod.cidades.inicial,[Cod2],Cod.cidades.final1), ...
        % Acrescenta ...
        no vetor cidade destino
109     append(Cod.voos,[Cod.voo],Cod.voos.1),
110     Parcial is Custo+Preco,    % Acrescenta o voo na lista de ...
        voos escolhidos
111     viagem(Z,Y, Cod.cidades.final1, Cod.cidades.final, ...
        Cod.voos.1, Cod.voos.final,Parcial,Total.aereo). % ...
        Recursividade para buscar nova conexao
112
113
114     imprimir_voos_linha(Cod.voos,Cod.voos.Original,Total.aereo) :- ...
        Cod.voos = [Cod|Listal],
115     voo(Cod,X,Y,Custo,_),
116     append([X],[Y],Cidades),
117     imprimir_voos_linha(Listal,Cod.voos.Original, ...
        Cidades,Custo,Total.aereo).
118
119     imprimir_voos_linha(Cod.voos,Cod.voos.Original,Custo) :- ...
        Cod.voos = [Cod|[]],
120     voo(Cod,X,Y,Custo,_),
121     append([X],[Y],Cidades),
122     write(Custo),write('\t'),write(Cod.voos.Original), ...
        write('\t\t'),write(Cidades),write('\n').
123
124     imprimir_voos_linha(Cod.voos,Cod.voos.Original, ...
        Cidades,Custo,Total.aereo) :- Cod.voos = [Cod|Listal],
125     voo(Cod,_,Y,Custo2,_),
126     append(Cidades,[Y],Cidades2),
127     Total is Custo+Custo2,
128     imprimir_voos_linha(Listal,Cod.voos.Original, ...
        Cidades2,Total,Total.aereo).
129
130     imprimir_voos_linha(Cod.voos,Cod.voos.Original, ...
        Cidades,Custo,Total) :- Cod.voos = [Cod|[]],
131     voo(Cod,_,Y,Custo2,_),
132     append(Cidades,[Y],Cidades2),
133     Total is Custo+Custo2,
134     write(Total),write('\t'),write(Cod.voos.Original) ...
        ,write('\t\t'),write(Cidades2),write('\n').
135
136
137     busca_voos(X,Y,Cod1,Cod.voos.final,Total.aereo) :- ...
        viagem(X,Y,[Cod1],Cod.cidades.final,[], ...

```

```

        Cod.voos_final,0,Total_aereo).
138
139 imprime_voo(Cod.voos_final) :- ...
        write('\nVoos\nPreco\tCodigos\t\t\t\t\tCidades\n'),
140        imprimir_voos_linha(Cod.voos_final,Cod.voos_final,0).
141
142
143 %Regras relacionando cidades, países e moedas
144 cod_cidade_cod_pais(Cod.pais,Cod.cidade) :- ...
        cidade(Cod.cidade,_,Cod.pais).
145
146 cod_cidade_pais(Pais,Cod.cidade) :- ...
        cod_cidade_cod_pais(Cod.pais,Cod.cidade),
147        pais(Cod.pais,Pais,_).
148 cidade_cod_pais(Cod.pais,Cidades) :- cidade(_,Cidades,Cod.pais).
149
150 cidade_pais(Pais,Cidade) :- ...
        cidade(_,Cidade,Cod.pais),pais(Cod.pais,Pais,_).
151
152 cod_pais_cod_moeda(Cod.moeda,Cod.pais) :- ...
        pais(Cod.pais,_,Cod.moeda).
153
154 cod_pais_moeda(Moeda,Cod.pais) :- ...
        cod_pais_cod_moeda(Cod.moeda,Cod.pais),
155        moeda(Cod.moeda,Moeda,_).
156
157 pais_moeda(Moeda,Pais) :- pais(_,Pais,Cod.moeda),
158        moeda(Cod.moeda,Moeda,_).
159
160 cod_cidade_cod_moeda(Cod.moeda,Cod.cidade) :- ...
        cod_cidade_cod_pais(Cod.pais,Cod.cidade),
161        cod_pais_cod_moeda(Cod.moeda,Cod.pais).
162
163 cod_cidade_moeda(Moeda,Cod.cidade) :- cidade(Cod.cidade,_,Cod.pais),
164        moeda(Cod.moeda,Moeda,_),pais(Cod.pais,_,Cod.moeda).
165
166 busca_hotel(Cod.cidade,Diarias,Cod.hotel,Total_hotel,Cod.moeda) ...
        :- hotel(Cod.hotel,_,Cod.cidade,_,Custo),
167        cod_cidade_cod_moeda(Cod.moeda,Cod.cidade),
168        moeda(Cod.moeda,_,Cotacao),
169        Total_hotel is Diarias*Custo*Cotacao.
170
171 imprime_hotel(Cod.hotel,Diarias,Total_hotel,Cod.moeda) :- ...
        hotel(Cod.hotel,Nome,_,Estrelas,Custo),
172        moeda(Cod.moeda,_,Cotacao),
173        write('\nHoteis\nCodigo\tNome\t\t\t\t\tEstrelas   Diarias   ...
        Preco   (',write(Cod.moeda),write('   Total ...
        (',write(Cod.moeda),write('   Total (brl)\n'),
174        Total_local is Custo*Diarias,
175        write(Cod.hotel),write('\t'),write(Nome), ...
        write('\t\t'),write(Estrelas),write('\t   ...
        '),write(Diarias),write('\t   ...
        '),write(Custo),write('\t\t'), ...
        write(Total_local),write('\t   ...
        '),write(Total_hotel),write('\n').

```



```

176
177
178 % Regra que busca um destino
179 pacote(X,Y,Diarias,Max.voos) :- cidade(Cod1, X, _),
180     cidade(Cod2, Y, _),
181     Cod1 \== Cod2,
182     busca.voos(X,Y,Cod1,Cod.voos.final_ida,Total.aereo.ida),
183     busca.voos(Y,X,Cod2,Cod.voos.final_volta,Total.aereo.volta),
184     busca.hotel(Cod2,Diarias,Cod.hotel,Total.hotel,Cod.moeda),
185     length(Cod.voos.final_ida,Limite1),
186     length(Cod.voos.final_volta,Limite2),
187     Limite1 =< Max.voos,
188     Limite2 =< Max.voos,
189     Total is Total.aereo.ida+Total.aereo.volta+Total.hotel,
190     imprime.voo(Cod.voos.final_ida),
191     imprime.voo(Cod.voos.final_volta),
192     imprime.hotel(Cod.hotel,Diarias,Total.hotel,Cod.moeda),
193     write('\nTOTAL: '),write(Total).
194
195 % Regra que busca um destino de acordo com sua moeda.
196 pacote.moeda(X,Moeda,Diarias,Max.voos) :- ...
197     cod.cidade.moeda(Moeda,Cod),
198     cidade(Cod,Y,_),
199     pacote(X,Y,Diarias,Max.voos).
200
201 % Regra que busca um destino no pais desejado.
202 pacote.pais(X,Pais,Diarias,Max.voos) :- cod.cidade.pais(Pais,Cod),
203     cidade(Cod,Y,_),
204     pacote(X,Y,Diarias,Max.voos).
205
206 % Regra que busca a partir do preco maximo um destino com ...
207     hospedagem.
208 pacote.preco(X,Preco,Diarias,Max.voos) :- cidade(Cod1, X, _),
209     cidade(Cod2, Y, _),
210     Cod1 \== Cod2,
211     busca.voos(X,Y,Cod1,Cod.voos.final_ida,Total.aereo.ida),
212     busca.voos(Y,X,Cod2,Cod.voos.final_volta,Total.aereo.volta),
213     length(Cod.voos.final_ida,Limite1),
214     length(Cod.voos.final_volta,Limite2),
215     Limite1 =< Max.voos,
216     Limite2 =< Max.voos,
217     busca.hotel(Cod2,Diarias,Cod.hotel,Total.hotel,Cod.moeda),
218     Total is Total.aereo.ida+Total.aereo.volta+Total.hotel,
219     Total < Preco,
220     imprime.voo(Cod.voos.final_ida),
221     imprime.voo(Cod.voos.final_volta),
222     imprime.hotel(Cod.hotel,Diarias,Total.hotel,Cod.moeda),
223     write('\nTOTAL: '),write(Total).
224
225 %EXEMPLO DE CONSULTAS
226 % pacote(sao.paulo,X,10,3).
227 % pacote.preco(florianopolis,3000,1,3).
228 % pacote.pais(sao.paulo,alemanha,10,3).
229 % pacote.moeda(sao.paulo,euro,10,3).

```