PROGRAMMING PARADIGMS, DT228A – DT228B/1

Dr. Pierpaolo Dondio

Assignment 1 – Prolog Project, Due Date: 1st December 2016

PART 1 (10% of final exam marks)

Your knowledge base **(ass1.pro)** contains information about flights. For each flight there is a prolog predicate `flight(city-a,city-b,airline,distance,time,cost)` meaning that there is a flight from A to B with a specific Airline, the distance between A and B, the flight time and the flight cost. For instance:

```
flight(london,dublin,aerlingus,500,45,150)
flight(rome,london,ba,1500,150,400)
flight(rome,paris,airfrance,1200,120,500)
flight(paris,dublin,airfrance,600,60,200)
```

The time is expressed in minutes, distance in km, price in euros. You have also information about the location of each city in the predicate `country(city,state)`. For instance:

```
country(london,uk)
country(dublin,ireland)
country(rome,italy)
country(paris,france)
country(cork,ireland)
country(shannon,ireland)
```
….
You are required to write the following PROLOG predicates

```
1.
list_airport(X,L)
```

where X (input) is a country and L is a list of all airports in that country. Example:

```
list_airport(ireland,L)
L=[dublin,cork,shannon]
```

```
2.
trip(X,Y,T)
```

i.e. a predicate to show the connections from city X to city Y (one by one).
X and Y are the two inputs (city) and T is the output. Each solution T is a list of all the cities connecting X to Y (X and Y included). Example:

```
trip(rome,dublin,T)
T=[rome,london,dublin] ; //first solution
T=[rome,paris,dublin] ; //second solution
```

```
3.
all_trip(X,Y,T)
```
i.e. a predicate that returns **all** the connections (trips) from X to Y and place them in T. Therefore T is a list of lists. Example:

```
all_trip(rome,Dublin,T).
T=[[rome,london,dublin], [rome,paris,dublin]]
```

```
4.
trip_dist(X,Y,[T,D])
```

the predicate returns the distance D of each trip T btw city X and Y (one by one). Note that the output is a list with first element the trip (which is a list) and second element the numeric distance.
Example:

```
trip_dist(rome,dublin,W)
W=[[rome,london,dublin],2000];
W=[[rome,paris,dublin],1800];
False
```

5.
`trip_cost(X,Y,[T,C])`

same as `trip_dist` but C is the total cost of the trip

6.
`trip_change(X,Y,[T,I])`

same as `trip_dist` but `I` is the total number of airplanes changed (=0 for direct connections).
Suggestion: use a predicate to compute the length of alist.

7.
`all_trip_noairline(X,Y,T,A).`

same as `all_trip`, but DISCARD all the trip containing a flight with airline `A` (for instance the customer would like to flight from `rome` to `dublin` avoiding `ryanair`)

8.
```
cheapest(X,Y,T,C)
shortest(X,Y,T,C)
fastest(X,Y,T,C)
```

finding the cheapest, shortest, fastest trip `T` from city `X` to city `Y`. `C` is the cost,distance or time.
Suggestion: using `findall`, put all the results of `trip_dist` (or `trip_cost` or `trip_time`) in a list, then find the minimum of the list, using the price/distance/time respectively as the key to compare two trips.
Therefore, a predicate to compute the max or min of a list is needed.

9.
`trip_to_nation(X,Y,T)`

showing all the connections from airport `X` to country `Y` (one by one). Example:

```
trip_to_nation(rome,ireland,T)
T=[rome,london,dublin] ; //first solution
T=[rome,paris,dublin] ; //second solution
T=[rome,paris,cork] ; //third solution
false.
```

Suggestion: first use list_airport to get all the airports in the country, and then look for the trips

10.
`all_trip_to_nation(X,Y,T)`

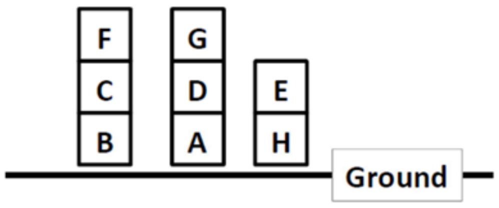
Important: to avoid trips with cycles, such as `[rome, paris, rome, london, dublin]`, keep a list of cities already visited for the trip. A city can be added only if it was not visited so far.

PART 2 (15% of final exam marks)

There are some messy letter blocks on the floor (8 in total). The blocks can only be stacked in three different places on the floor. You have to program a set of PROLOG predicates for a robot working with the blocks. Each pile is a list of element where the element on the left is the one on the ground. The status of your blocks is identified by a list containing three lists, one for each stack of blocks. For instance:

```
B = [[b,c,f],[a,d,g],[h,e]]
```

Describe the following situation:



1. Print the status of the blocks

You are required to define a PROLOG predicate `print_status` to print the status of the blocks.

```
print_status(B)
```

where `B` is a list of three lists describing the three piles of blocks. For instance:

```
?- print_status([[b,c,f],[a,d,g],[h,e]]).

|b|c|f|
|a|d|g|
|h|e|
```

2. Predicate `high(B,X,H)`

You are required to produce a predicate `high(X,H)` where `B` describes the three piles of blocks, `X` is a block and `H` is the height of the block from the ground. A block directly on the ground has height equal to zero. Example (referring to the picture above):

```
?- high([[b,c,f],[a,d,g],[h,e]],c,H)
H=1
?- high([[b,c,f],[a,d,g],[h,e]],g,H)
H=2
```

The predicate can also be used to find all the blocks at a certain high. For instance:

```
?- high([[b,c,f],[a,d,g],[h,e]],X,2)
X=f;
X=g.
```

3. Block at a specific height.

Write a PROLOG predicate that returns a list of all the blocks at a specific heights (from all the 3 stacks). The predicate is called `all_same_height(B,H,L)` where `B` again is the list of piles, `H` is the height (integer positive number) and `L` is the list of blocks that are at an height of `H`.

Example (referring to the picture above):

```
?- all_same_height([[b,c,f],[a,d,g],[h,e]],0,L)

L = [b,a,h]

?- all_same_height([[b,c,f],[a,d,g],[h,e]],2,L)

L = [f,g]
```

4. Predicate same height

Write a PROLOG predicate `same_height(B,X,Y)` that is true if block `X` and block `Y` have the same height (B is the usual description of the three piles).

5. Move blocks

Write a PROLOG predicate `moveblock(B,X,S1,S2)` to move a block from the stack `S1` to the stack `S2.` `S1` and `S2` are integer numbers identifying one of the three stacks (1,2 or 3). A block can be moved only ifitisatthetopofastack (otherwise it cannot be moved) and it can be only placed on top of another stack (or on the ground if the stack is empty). The predicate has to print the status of the blocks before and after the move (only if the block can be moved!).

For instance, let's refer to the above picture and let's move block G from stack 2 to stack 1. Therefore:

```
?- moveblock([[b,c,f],[a,d,g],[h,e]],g,2,1)

Before:

|b|c|f|
|a|d|g|
|h|e|


After:

|b|c|f|g|
|a|d|
|h|e|
```

6. Put the blocks in order

Write a predicate `order_blocks(S,S_order)` that takes as an input the three stacks of blocks (variable S) and put them in order (A on the ground, B over A and so on...) in any of the three stacks (output variable S_order).
There are only three stacks available (blocks can only be moved in one of the three stacks).
Each time the block is moved the program print the status of the blocks.