CSE341 – Programming Languages

Homework #4

Report

OĞUZHAN AGKUŞ
161044003

Note: I used SWI-Prolog.

Part 1

Firstly, I defined facts which are indicates flights between two cities. Since, all flights are bidirectional I defined a return flight for each pair of cities. Then, I wrote a predict that check a route between two cities. The route can be direct or indirect.

```
% Facts
flight(istanbul, izmir).
flight(izmir, istanbul).
flight(istanbul, antalya).
flight(antalya, istanbul).
flight(istanbul, gaziantep).
flight(gaziantep, istanbul).
flight(istanbul, ankara).
flight(ankara, istanbul).
flight(istanbul, van).
flight(van, istanbul).
flight(istanbul, rize).
flight(rize, istanbul).
flight(edirne, edremit).
flight(edremit, edirne).
flight(erzincan, edremit).
flight(edremit, erzincan).
flight(izmir, ısparta).
flight(ısparta, izmir).
flight(burdur, ısparta).
flight(ısparta, burdur).
flight(konya, antalya).
flight(antalya, konya).
flight(konya, ankara).
flight(ankara, konya).
flight(antalya, gaziantep).
flight(gaziantep, antalya).
flight(van, ankara).
flight(ankara, van).
flight(van, rize).
flight(rize, van).
```

```
% Rules
route(X, Y) :-
  route_(X, Y, []).
route_(X, Y, R) :-
  flight(X, Z),
  not(member(Z, R)),
  (Y = Z; route_(Z, Y, [X | R])).
```

Test Results

I tested with a few inputs. You can check results according to given graph in the assignment file.

```
?- flight(rize, van).
true.

?- flight(edirne, edremit).
true.

?- flight(rize, ankara).
false.

?- flight(gaziantep, ısparta).
false.

?- flight(burdur, istanbul).
false.
```

```
?- route(ankara, rize).
true .

?- route(istanbul, burdur).
true .

?- route(edremit, ankara).
false.

?- route(ankara, erzincan).
false.

?- route(istanbul, konya).
true .
```

```
?- route(edremit, X).
X = edirne ;
X = erzincan ;
```

Part 2

In this part, I added distance facts (image on left) which are indicates the distance between two cities. Then, I wrote some predicates (image on right) to find shortest route between two cities. It finds all possible routes and choose the shortest one. If there are no route, it returns false.

```
distance(istanbul, izmir, 329).
distance(izmir, istanbul, 329).
distance(istanbul, antalya, 483).
distance(antalya, istanbul, 483).
distance(istanbul, gaziantep, 847).
distance(gaziantep, istanbul, 847).
distance(istanbul, ankara, 352).
distance(ankara, istanbul, 352).
distance(istanbul, van, 1262).
distance(van, istanbul, 1262).
distance(istanbul, rize, 968).
distance(rize, istanbul, 968).
distance(edirne, edremit, 244).
distance(edremit, edirne, 244).
distance(erzincan, edremit, 1027).
distance(edremit, erzincan, 1027).
distance(izmir, isparta, 309).
distance(isparta, izmir, 309).
distance(burdur, isparta, 25).
distance(isparta, burdur, 25).
distance(konya, antalya, 192).
distance(antalya, konya, 192).
distance(konya, ankara, 227).
distance(ankara, konya, 227).
distance(antalya, gaziantep, 592).
distance(gaziantep, antalya, 592).
distance(van, ankara, 920).
distance(ankara, van, 920).
distance(van, rize, 373).
distance(rize, van, 373).
```

```
% Rules
sroute(X, Y, D) :- shortest_route(X, Y, D).

shortest_route(X, Y, D) :-
  route_distance(X, Y, Route, Minimum_Distance),
  not(shorter_route(X, Y, _, _, Minimum_Distance)),
  D is Minimum_Distance,
  !.

shorter_route(X, Y, Route, Cost, Min_Cost) :-
  route_distance(X, Y, Route, Cost),
  Cost < Min_Cost.

route_distance(X, Y, Route, Distance) :-
  route_one_distance(X, [Y], 0, Route, Distance).

route_one_distance(X, [X | Route1], Distance1, [X | Route1], Distance1).
route_one_distance(X, [Y | Route1], Distance1, Route, Distance) :-
  distance(N, Y, D),
  not(member(N, Route1)),
  Distance2 is Distance1 + D,
  route_one_distance(X, [N, Y | Route1], Distance2, Route, Distance).

member(X,[X | _]).
member(X,[_ | T]) :-
  member(X, T).
```

“sroute” predicate is a wrapper for “shortest_route” predicate. I thought “sroute” can be meaningless sometimes, so I prefer to use “shortest_route” name.

Test Results

I tested with a few inputs. You can check results according to given graph in the assignment file and given distance calculator website. There are different possible inputs. Konya-Edremit has not route, Edirne-Edremit has direct route, İstanbul-Burdur has indirect route, İstanbul-Konya has indirect and multiple routes. In İstanbul-Konya test, it finds shortest route.

```
?- sroute(konya, edremit, D).
false.

?- sroute(edirne, edremit, D).
D = 244.

?- sroute(istanbul, burdur, D).
D = 663.

?- sroute(istanbul, konya, D).
D = 579.
```

Part 3

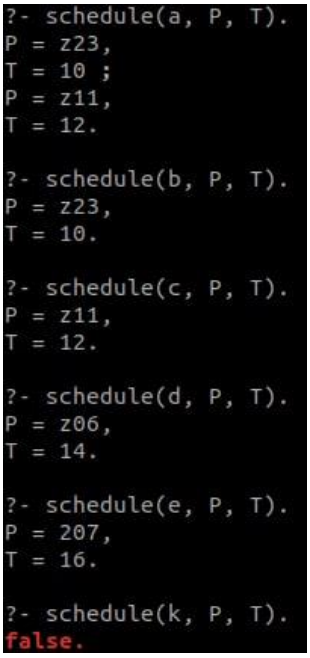
Firstly, I defined facts (image on left) according to assignment file. Then, I wrote predicates (image on right) asked in the assignment file.

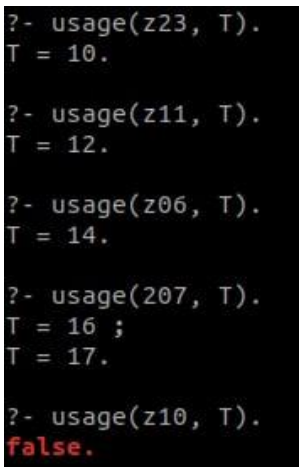
```
% Facts
when(102, 10).
when(108, 12).
when(341, 14).
when(455, 16).
when(452, 17).

where(102, z23).
where(108, z11).
where(341, z06).
where(455, 207).
where(452, 207).

enrollment(a, 102).
enrollment(a, 108).
enrollment(b, 102).
enrollment(c, 108).
enrollment(d, 341).
enrollment(e, 455).
```

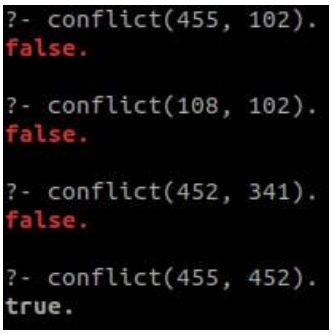
```
% Rules
schedule(S, P, T) :-
  enrollment(S, C),
  where(C, P),
  when(C,T).

usage(P, T) :-
  where(C, P),
  when(C, T).

conflict(X, Y) :-
  (when(X, T1), when(Y, T2), T1==T2);
  (where(X, P1), where(Y, P2), P1==P2).

meet(X, Y) :-
  enrollment(X, C1),
  enrollment(Y, C2),
  C1==C2.
```

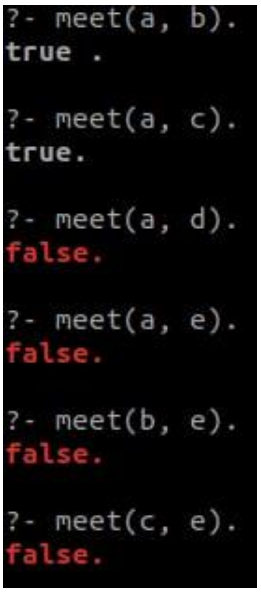
Test Results

I tested with a few inputs. You can check results according to given tables in the assignment file. There are multiple different tests for each predicate.









Part 4

In this part, I wrote predicates for some set operations. I need some extra helper predicates.

```
% Element
element(E, S) :-
    member(E, S).

% Union
union(S1, S2, S3) :-
    cover_1(S1, S3),
    cover_1(S2, S3),
    cover_both_1(S1, S2, S3).

% Intersect
intersect(S1,S2,S3) :-
    cover_2(S1, S2, S3),
    cover_both_2(S1, S2, S3).

% Equivalent
equivalent(S1, S2) :-
    cover_1(S1, S2),
    cover_1(S2, S1).
```

```
% Helpers
cover_1(X, Y) :-
    foreach(element(I, X), element(I, Y)).

cover_both_1(X, Y, Z) :-
    foreach(element(I, Z), element(I, X); element(I, Y)).

cover_2(X, Y, Z) :-
    foreach((element(I, X), element(I, Y)), element(I, Z)).

cover_both_2(X, Y, Z) :-
    foreach(element(I, Z), (element(I, X), element(I, Y))).
```

Test Results

I tested with a few inputs.

```
?- element(3, [1, 2, 3, 4, 5]).
true .

?- element(6, [1, 2, 3, 4, 5]).
false.

?- element(X, [1, 2, 3, 4, 5]).
X = 1 ;
X = 2 ;
X = 3 ;
X = 4 ;
X = 5.
```

```
?- union([1, 2], [2, 3, 4], [1, 2, 3, 4]).
true.

?- union([1, 2], [2, 3, 4], [1, 2, 3]).
false.
```

```
?- intersect([1, 2], [2, 3, 4], [2]).
true.

?- intersect([1, 2], [2, 3, 4], [1, 2]).
false.
```

```
?- equivalent([1, 5, 7, 10], [1, 10, 7, 5]).
true.

?- equivalent([1, 5, 7, 10], [1, 10, 7, 5, 4]).
false.
```

Part 5

In this part, I implemented a equation creator. It reads a list of integers from file named “input.txt”. Then it tries to insert arithmetic operators to obtain current equations. Finally, it writes the possible equations to the output file named “output.txt”.

```
main :-
  open('input.txt', read, InputStream),
  read(InputStream, Line),
  close(InputStream),
  string_to_list(Line, List),
  open('output.txt', write, OutputStream),
  calculate(List, OutputStream),
  close(OutputStream).

calculate(List, OutputStream) :-
  equation(List, LeftTerm, RightTerm),
  write(OutputStream, LeftTerm),
  write(OutputStream, '='),
  write(OutputStream, RightTerm),
  write(OutputStream, '\n'),
  fail.
calculate(_, OutputStream).
```

```
equation(List, LeftTerm, RightTerm) :-
  split(List, LeftList, RightList),
  term(LeftList, LeftTerm),
  term(RightList, RightTerm),
  LeftTerm =:= RightTerm.

split(List, List1, List2) :-
  append(List1, List2, List), List1 = [_|_], List2 = [_|_].

term([X], X).
term(List, Term) :-
  split(List, LeftList, RightList),
  term(LeftList, LeftTerm),
  term(RightList, RightTerm),
  combine(LeftTerm, RightTerm, Term).

combine(LeftTerm, RightTerm, LeftTerm + RightTerm).
combine(LeftTerm, RightTerm, LeftTerm - RightTerm).
combine(LeftTerm, RightTerm, LeftTerm * RightTerm).
combine(LeftTerm, RightTerm, LeftTerm / RightTerm) :- RightTerm =\= 0.
```

To run the program, supply a input file named “input.txt” in the same directory of the program. It’s format should be like that:

input.txt

```
1  [5,3,5,7,49].
```

Then call main predicate by “main().”. It creates an output file to same directory. It fills with all possible equations if they exists.

```
?- [part5].
true.

?- main().
true.
```

output.txt

```
1  (5-(3-5))*7=49
2  (5-3+5)*7=49
```

Part 6

In this part, I tried to solve the puzzle. There are defined 3 test cases which are given in assignment file. It writes both terminal and file named “output.txt”.

To run the program, call main predicate by “main(test_no)”. It solves puzzle and prints the output.

I can try only first test case because the other ones last very long, they need powerful processor. My hardware is not good for it. But I tried, I waited very long time but it did not finished. If you wish, you can try. May be I have problem with my algorithm.

The results for first test case.

```
?- [part6].
true.

?- main(1).

   X X X           3
 X X   X           2 1
   X X X       X X 3 2
     X X       X X 2 2
     X X X X X X 6
 X   X X X X X   1 5
 X X X X X X     6
         X       1
       X X       2
 1 3 1 7 5 3 4 3
 2 1 5 1
true .
```

output.txt

```
 1      X X X           3
 2    X X   X           2 1
 3      X X X       X X 3 2
 4        X X       X X 2 2
 5        X X X X X X 6
 6    X     X X X X X   1 5
 7    X X X X X X       6
 8              X       1
 9            X X       2
10    1 3 1 7 5 3 4 3
11    2 1 5 1
```