

CSE 341 PROGRAMMING LANGUAGES HW-4

REPORT

In this homework, I used online prolog terminal to test my program:

<https://swish.swi-prolog.org/>

PART 1

For this part, first I have written all possible flights. After writing all facts, I controlled directed or connected route between two cities.

Below, you can see some part of the written rules and facts:

```
%Facts.
flight(istanbul,rize).
flight(istanbul,izmir).
flight(istanbul,ankara).
flight(istanbul,van).
flight(istanbul,gaziantep).
flight(istanbul,antalya).
flight(izmir,ismarta).
flight(izmir,istanbul).

% rules that find if there is a route between given X,Y.
route(X, Y) :- routex(X, Y, []).
routex(X, Y, VisitList) :- flight(X, Z), not(member(Z, VisitList))
    , (Y = Z; routex(Z, Y, [X | VisitList])).
```

Test Results:



To test part 1, first I have written the following predicates:

`route(antalya,konya).` = True

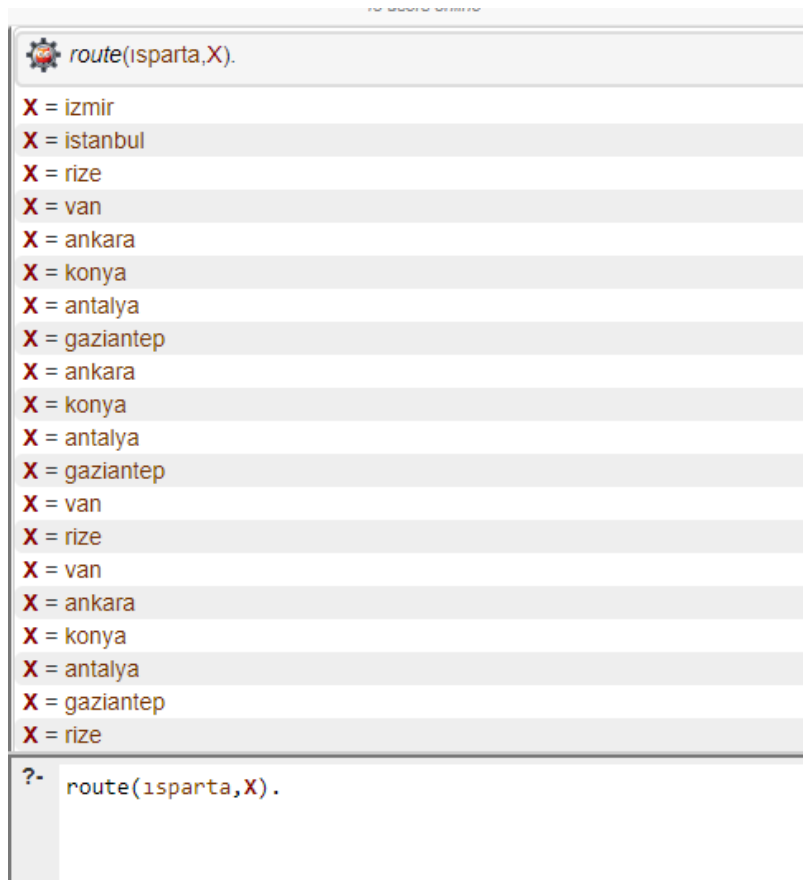
`route(istanbul,erzincan).` = False

`route(izmir,ismarta).` = True

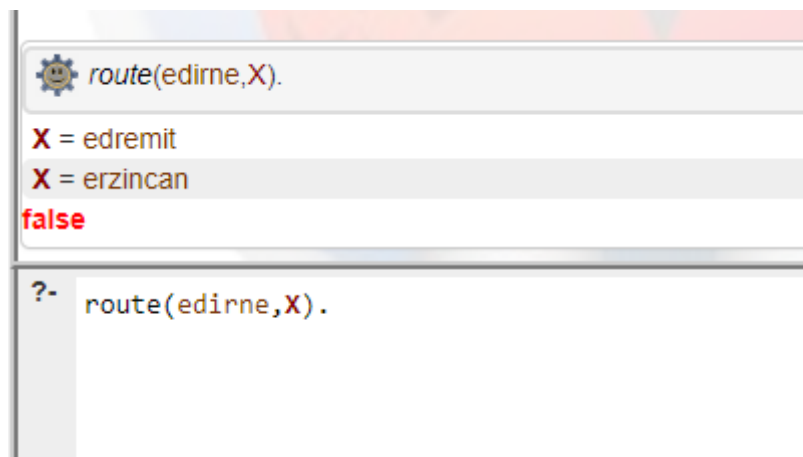
route(istanbul,izmir). =True

Then i have written following predicates to test see all flights from Isparta and Edirne:

route(isparta,X). result:



route(edirne,X). result:



PART 2

For this part, to be able to write fact part for distances; I calculated distances between cities using given source. Then I wrote distances for cities and rules:

```
distance(istanbul,rize,967.79).
distance(istanbul,izmir,328.80).
distance(istanbul,ankara,351.50).
distance(istanbul,van,1262.37).
distance(istanbul,gaziantep,847.42).
distance(istanbul,antalya,482.75).
distance(izmir,isparta,308.55).
distance(izmir,istanbul,328.80).
distance(isparta,izmir,308.55).
distance(isparta,burdur,24.60).
distance(burdur,isparta,24.60).
distance(edirne,edremit,914.67).
distance(edremit,edirne,914.67).
distance(edremit,erzincan,736.34).
distance(erzincan,edremit,736.34).
distance(antalya,istanbul,482.75).
distance(antalya,konya,192.28).
distance(antalya,gaziantep,592.33).
distance(konya,ankara,227.34).
distance(konya,antalya,192.28).
distance(gaziantep,antalya,592.33).
distance(gaziantep,istanbul,847.42).
distance(ankara,istanbul,351.50).
distance(ankara,konya,227.34).
distance(ankara,van,920.31).
distance(van,istanbul,1262.37).
distance(van,ankara,920.31).
distance(van,rize,373.01).
distance(rize,istanbul,967.79).
distance(rize,van,373.01).
```

```
% Rule that finds distance between two cities.
sroute(A,B,C):-distance(A,B,C).
sroute(A,B,C):- distance(A,Z,D1),
                  distance(Z,B,D2),
                  C is D1+D2.
```


Test Results:

If there exists direct flight, program shows the distance, if not it returns false.

For example, in the map there is no direct flight from Izmir to Konya so output is false.


The screenshot shows a Prolog interpreter window with the following content:

- Query: `sroute(rize,istanbul,X).`
Result: `X = 967.79`
Buttons: Next, 10, 100, 1,000, Stop
- Query: `sroute(izmir,konya,X).`
Result: `false`
- Query: `sroute(izmir,isparta,X).`
Result: `X = 308.55`
Buttons: Next, 10, 100, 1,000, Stop
- Query: `sroute(edremit,erzincan,X).`
Result: `X = 736.34`
Buttons: Next, 10, 100, 1,000, Stop


 `sroute(van,ankara,X).`

X = 920.31


Next

 `sroute(rize,konya,X).`

false


 `sroute(burdur,konya,X).`

false

 `sroute(burdur,isparta,X).`

X = 24.6

Next

 `sroute(istanbul,izmir,X).`

X = 328.8

Next

PART 3

At first step, I created base with given database about classes, student enrollment and classrooms.

And I created graph and querying database about student, class, time and room information.

Classes		
Class	Time	Room
102	10	z23
108	12	z11
341	14	z06
455	16	207
452	17	207

Enrollment	
Student	Class
a	102
a	108
b	102
c	108
d	341
e	455

Fact part:

```
%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).

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

3.1

Predicate Schedule, uses enroll, where, when.

```
schedule(S, P, T) :- enroll(S, C), where(C, P), when(C, T).
```

S: student name, P: classroom , T: time

Schedule predicate takes student name, it returns place and time of the courses that student takes.

enroll(S,C) : C classes that associated entered student.

where(C,P): P place of course.

when(C,T): T time of the course.

Test Result:

The screenshot shows a Prolog test interface with five test cases for the `schedule` predicate. Each case is displayed in a separate panel with a gear icon and the query `schedule(a,P,T).` (where 'a' is replaced by 'b', 'c', 'd', and 'e' for the subsequent cases). Below the query, the results for `P` and `T` are shown. At the bottom of the first panel, there are buttons for 'Next', '10', '100', '1,000', and 'Stop'.

Test Case	Query	Result P	Result T
1	<code>schedule(a,P,T).</code>	<code>P = z23,</code>	<code>T = 10</code>
2	<code>schedule(b,P,T).</code>	<code>P = z23,</code>	<code>T = 10</code>
3	<code>schedule(c,P,T).</code>	<code>P = z11,</code>	<code>T = 12</code>
4	<code>schedule(d,P,T).</code>	<code>P = z06,</code>	<code>T = 14</code>
5	<code>schedule(e,P,T).</code>	<code>P = 207,</code>	<code>T = 16</code>

3.2






Predicate usage(P,T) written. P is classroom and T is time. This predicate returns usage time of classroom.

where(C,P): C classes and P is the classroom

when(C,T): T is time

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

Test Result:



 <code>usage(z11,T).</code>
<code>T = 12</code>
 <code>usage(207,T).</code>
<code>T = 16</code>
<code>T = 17</code>
 <code>usage(z23,T).</code>
<code>T = 10</code>
 <code>usage(z06,T).</code>
<code>T = 14</code>
 <code>usage(abc,T).</code>
<code>false</code>

3.3

Predicate `conflict(X,Y)` written. True for if X and Y conflicts for classroom or time.

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

Test Result:





 <code>conflict(455,452).</code>
<code>true</code>
 <code>conflict(102,341).</code>
<code>false</code>

3.4

Predicate `meet` written by using `enroll`. If X and Y students are taking same course, they meet

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

Test Result:

 <code>meet(a,b).</code>
<code>true</code>
 <code>meet(a,c).</code>
<code>true</code>
 <code>meet(a,d).</code>
<code>false</code>
 <code>meet(b,c).</code>
<code>false</code>

PART 4

4.1

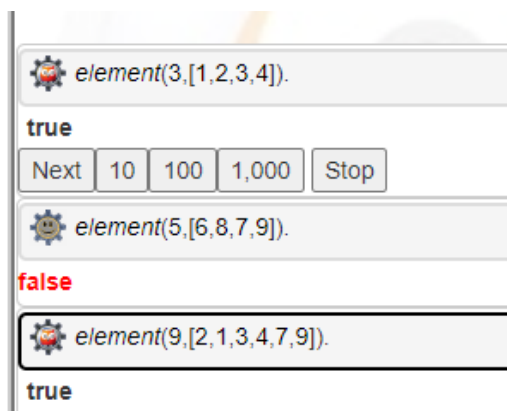
In the first predicate E represents element and S represents list.

`element(E, [E|_]).` means that if element exist, return true

`element(E, [_|S]):- element(E, S).` means a recursive call with the tail of the list

```
element(E, [E|_]).
element(E, [_|S]):- element(E, S).
```

Test Result:



4.2

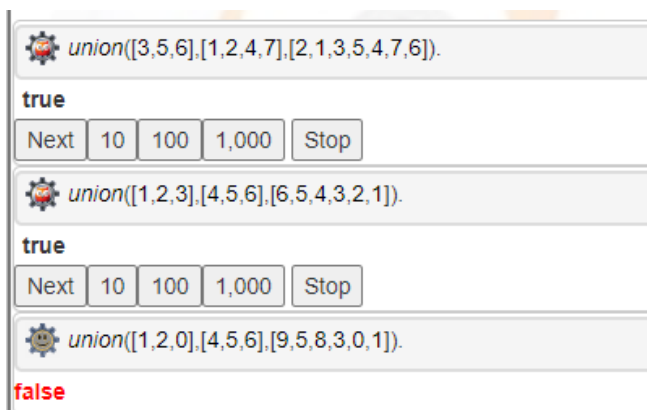
Predicate union written which returns true if S3 is the union of S2 and S1. All S values are represents list.

```
union(S1,S2,S3) :- union2(S1,S2,X), equivalent(X,S3).
```

Line 4- means predict union set can be

random order

Test Result:

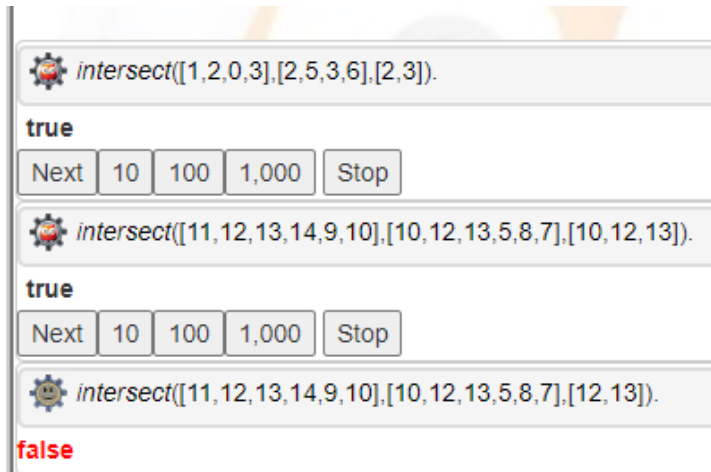


4.3

Predicate intersect written that returns true if S3 is the intersection of S1 and S2.

$\text{intersect}(S1, S2, S3) \text{ :- } \text{intersect2}(S1, S2, X), \text{equivalent}(X, S3).$

Test Result:

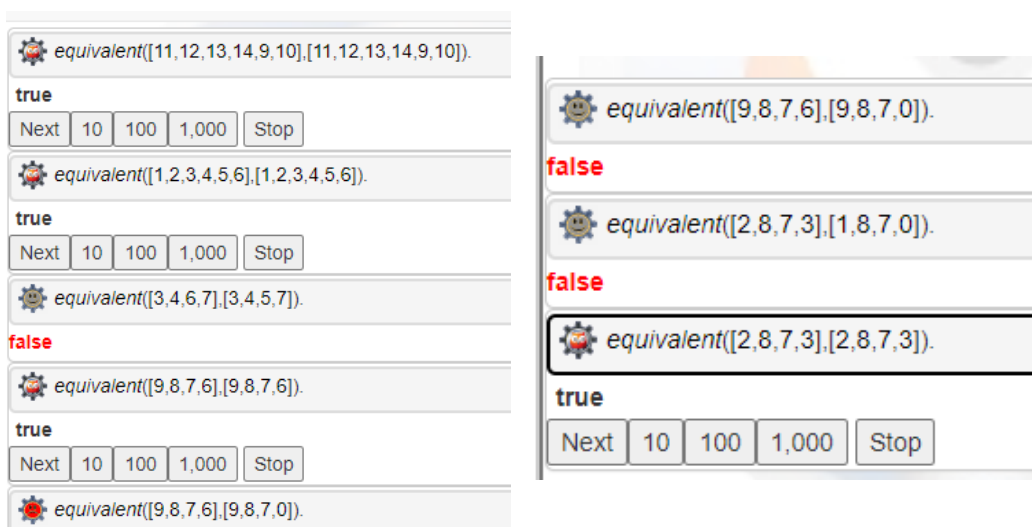


4.4

Predicate equivalent written to check if two sets are equal or not. Returns true if sets are equal, false otherwise

$\text{equivalent}(S1, S2) \text{ :- } \text{equivalent2}(S1, S2), \text{equivalent2}(S2, S1).$

Test Result:



PART 5

I tested this part on my ubuntu machine.

Predicate `equation(L,LT,RT)` :- written, L is the list of numbers which are the leaves in the arithmetic terms LT and RT.

term(L,T) :- L represents the list of numbers which are the leaves in the arithmetic term T.

binterm(LT,RT,T) :- T is a combined binary term constructed from left-hand term LT and right-hand term RT

do(L) :- finds all solutions to the problem as given by the list of numbers L, and print to the terminal.

Test Result:

```
File Edit View Search Terminal Help
esra@ubuntu:~/Desktop$
esra@ubuntu:~/Desktop$ swipl part4.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- do([2,3,6,7,11]).
2*(3+6) = 7+11
2*(3+6)-7 = 11
2/(3/6)+7 = 11
2/3*6+7 = 11
true.

?- do([5,3,5,7,13]).
5+3*5 = 7+13
5*3+5 = 7+13
5+(3*5-7) = 13
5*3+(5-7) = 13
5+3*5-7 = 13
5*3+5-7 = 13
true.
```