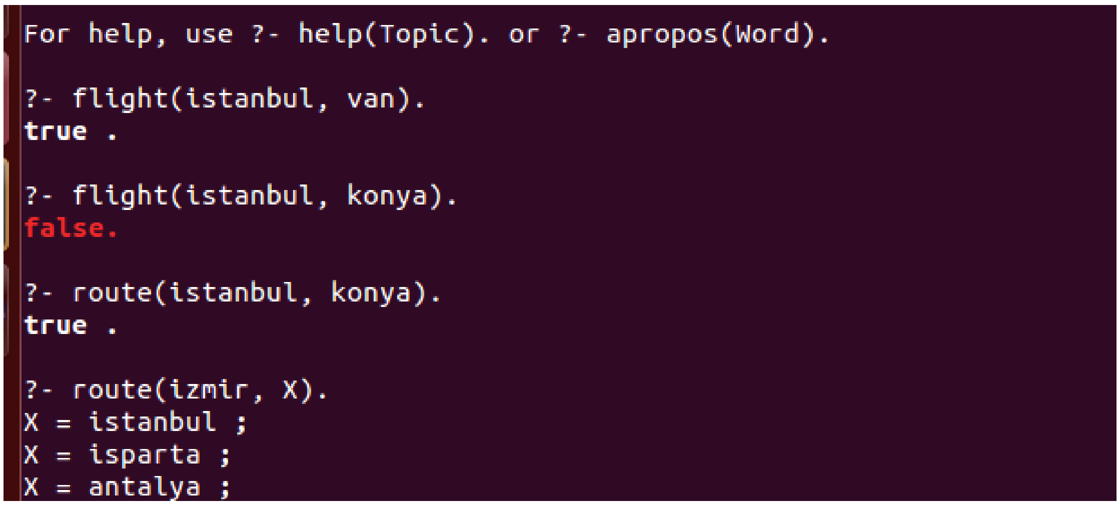
CSE 341 – HW #4

ELİF KELEŞ – 161044033

Q1)

First, I wrote all the facts. Then for route rule, I wrote a new rule called notdeadend(X). This returns true if a city X has two different flights to two different cities, (so it is not a dead end city for this program).

Route(X, Y) first checks if there is a direct flight between these cities, if not, then it checks for a route between a new city Z and a flight between Y and Z. The important point here is, Z should not be a dead end city so that program halts.

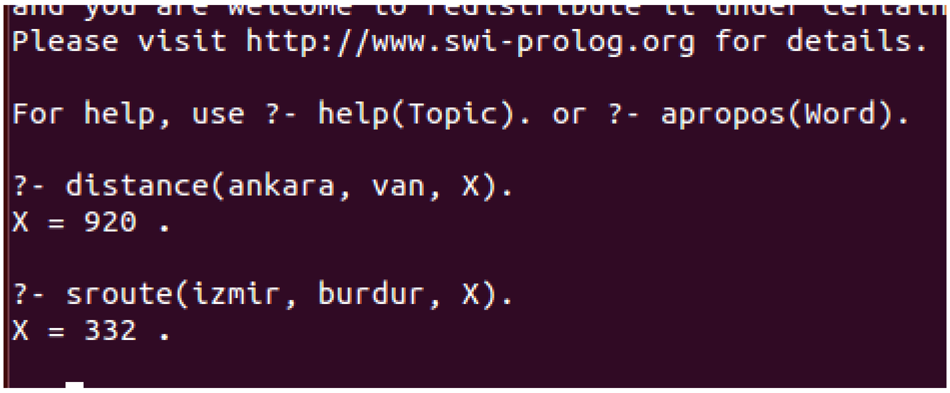


```
For help, use ?- help(Topic). or ?- apropos(Word).

?- flight(istanbul, van).
true .

?- flight(istanbul, konya).
false.

?- route(istanbul, konya).
true .

?- route(izmir, X).
X = istanbul ;
X = isparta ;
X = antalya ;
```

Q2)

First I added the distance relations to my first code. I calculated distance values in https://www.distancecalculator.net.

I used the same rule called notdeadend(X) for this part too. (So I used the flight rules here too) My sroute rule is basically doing the same thing as route before, and it checks for all the possibilities of a city and its distance relations. If it finds a proper city, it calculates the distance between.



```
and you are welcome to redistribute it under certain
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- distance(ankara, van, X).
X = 920 .

?- sroute(izmir, burdur, X).
X = 332 .
```

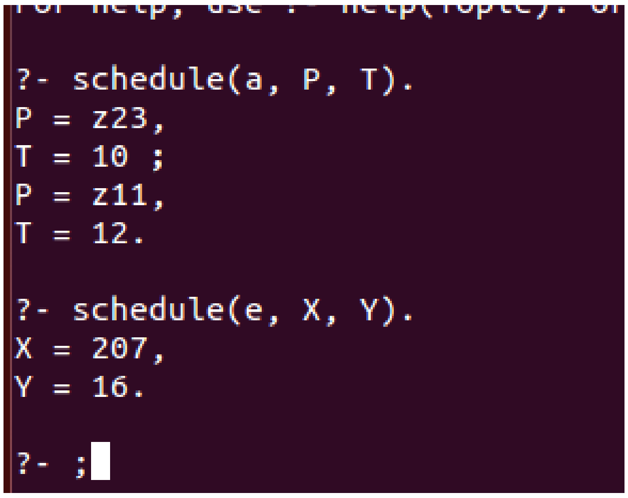
Q3)

I created the knowledge base according to the given tables in the homework PDF.

For 3.1)

I created the predicate schedule(S, P, T) taht associates a student to a place and class time. (S is student name, P is place and T is time)

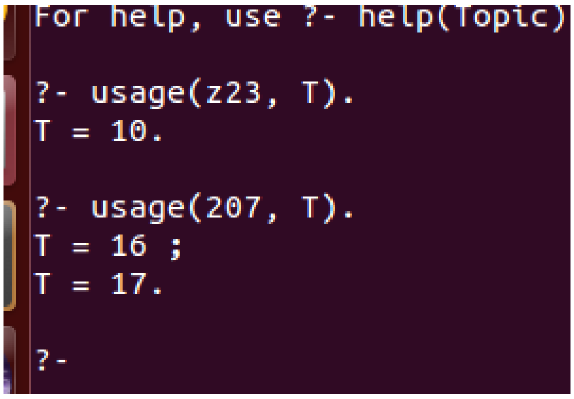
Schedule looks for the enroll, where and when rules, to create a schedule for a student.



```
For help, use ?- help(Topic). or

?- schedule(a, P, T).
P = z23,
T = 10 ;
P = z11,
T = 12.

?- schedule(e, X, Y).
X = 207,
Y = 16.

?- ;
```

For 3.2)

I created usage(P, T). Which takes a place and time and shows the usage times af a classroom. usage checks where and when rules.

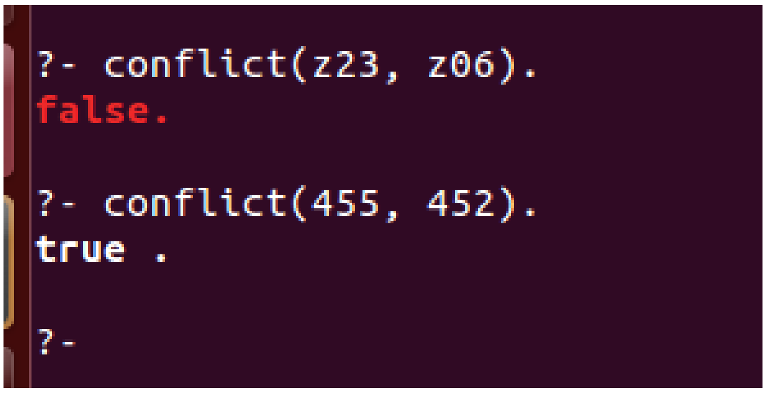


```
For help, use ?- help(Topic)

?- usage(z23, T).
T = 10.

?- usage(207, T).
T = 16 ;
T = 17.

?-
```

For 3.3)

Predicate conflict (X, Y) takes two arguments. Looks for a conflict between X and Y values in time or classroom.

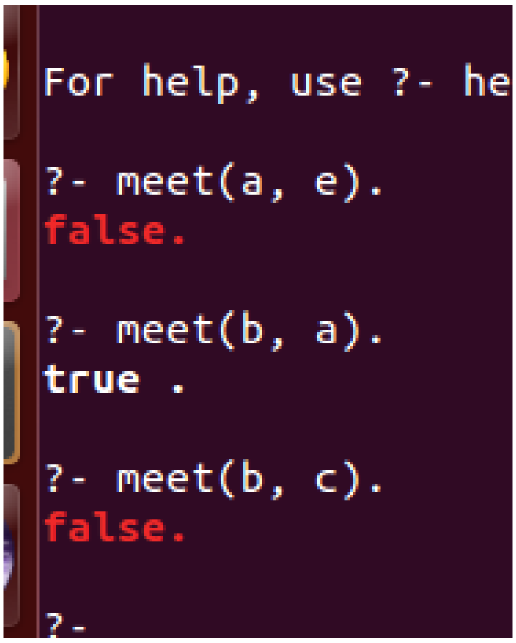
So it uses the rules of where and when.



```
?- conflict(z23, z06).
false.

?- conflict(455, 452).
true .

?-
```

For 3.4)

The predicate meet (X, Y), again takes two arguments. Here X and Y are students. meet checks if, X and Y are in a same classroom at the same time, or not. So it uses the enroll rules.

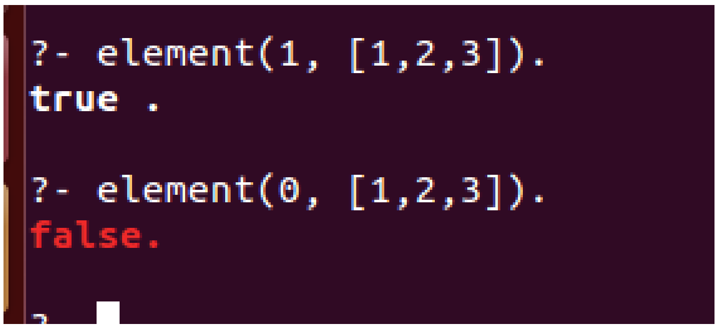


```
For help, use ?- he

?- meet(a, e).
false.

?- meet(b, a).
true .

?- meet(b, c).
false.

?-
```

Q4)

For 4.1)

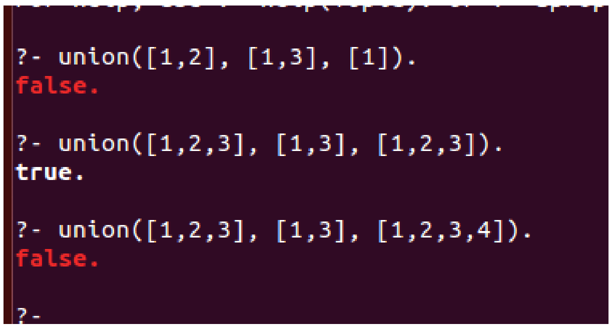
element (E, S) returns true if E is in set S. So I checked for E in S with “member(X, []).” of Prolog.



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

?- element(0, [1,2,3]).
false.
```

For 4.2)

union (S1, S2, S3) returns true if S3 is the union of S1 and S2. I used forecah to look up for every element in sets. First looked for the entity of the elements of S3 in S1 and S2 sets. Then checked for the extra elements that S3 may have (which S1 and S2 don't), and made sure there is none.

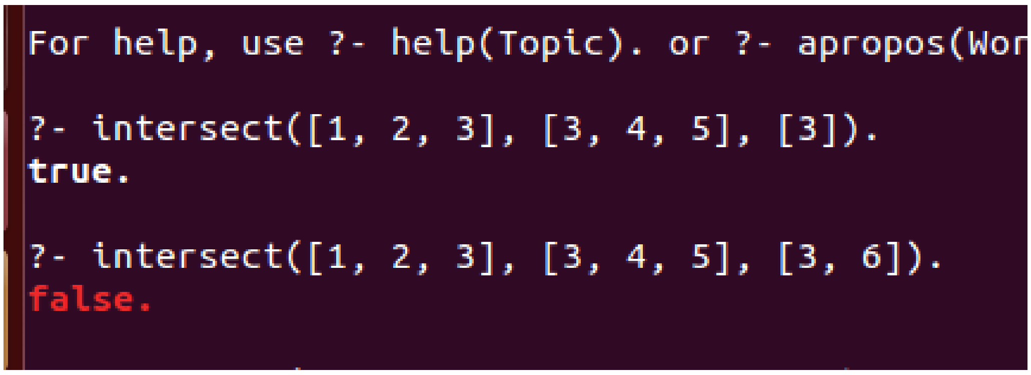


```
?- union([1,2], [1,3], [1]).
false.

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

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

?-
```

For 4.3)

İntersect (S1, S2, S3) returns true if S3 is the intersection of S1 and S2. For this again, I used foreach of Prolog, to check every element in sets.

Every element in S3 should be either in S1 or in S2.

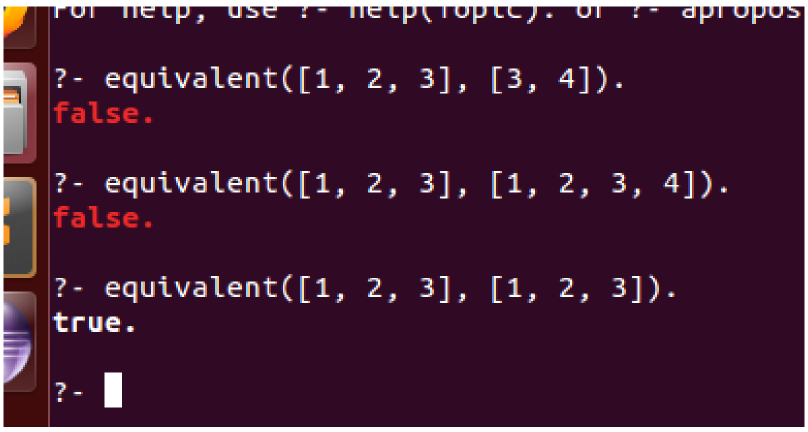


```
For help, use ?- help(Topic). or ?- apropos(Wor

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

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

For 4.4)

I wrote equivalent (S1, S2) that returns true if the sets are equal.

To do this one, I simply checked that if the given two sets are equal or not, using == operator.



```
For help, use ?- help(Topic). or ?- apropos

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

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

?- equivalent([1, 2, 3], [1, 2, 3]).
true.

?-
```

** I could not implement the parts 5 and 6.