CMPE260
PRINCIPLES OF PROGRAMMING LANGUAGES

HASAN ÖZTÜRK
2017400258

CMPE 260 – PROLOG PROJECT

PROGRAMMING PROJECT

SUBMISSION DATE: 23.04.2018 – 12.00 AM

INTRODUCTION

The project is about a football league. There are teams and matches in this league. This information is given in the knowledge base. The information about the teams are their names and their home location. The information about the matches are the week that the match is played, the hometeam, hometeam score, awayteam and the awayteam score. This knowledge base is given in a file called **cl_base.pl.** There are four tasks in this project which is about generating data about the teams and the matches.

We are expected to use Prolog programming language to implement this task. Unlike imperative programming languages prolog uses an inference mechanism based on logic rules. I wrote predicates according to these rules in order to achieve the goals. The predicates that I wrote is in the file called **predicate.pl.**

PROGRAM INTERFACE

User must open the program by writing "prolog" to the terminal. Then a interactive mode is appeared. First the **cl_base.pl** knowledge base must be consulted by writing the full path of the file within the [' '] string. Then **predicate.pl** must be consulted with the same way. After that the wanted queries can be written. Program will respond to these queries according the knowledge base and the predicates. User could terminate the program by pressing "**Ctrl + D**".

PROGRAM EXECUTION

In the previous part basic steps to run the program is explained. In this part the format of queries will be explained.

allTeams(L,N).
The program will output the list of teams and the total number of teams. By pressing **;** other combinations of the elements in the list can be obtained.

allteams([galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad, shaktard, bleverkusen, omarseille, arsenal, fcnapoli, bdortmund], 12).
The program will output **true** if the team list and the total number of teams are correct, **false** otherwise.

wins(T,W,L,N).
T → team
W → week
L → list
N → number

This predicate gives the list L containing the teams defeated by the team T when we are in week W, and the length of the list N.

losses(T,W,L,N).
T → team
W → week
L → list
N → number

This predicate gives the list L containing the teams thats defeats the team T when we are in week W, and the length of the list N.

draws(T,W,L,N)

T → team
W → week
L → list
N → number

This predicate gives the list L containing the teams that team T could not defeat also did not lose to.

scored(T,W,S)
T → team
W → week
S → total score

This predicate gives the total score scored by the team T when we are in week W.

conceded(T,W,S)
T → team
W → week
S → total score

This predicate gives the total score conceded by the team T when we are in week W.

average(T,W,A)
T → team
W → week
A → average

This predicate gives average A of team T in week W.

order(L, W).
L → List
W → week

This gives the order of the teams according to their averages in week W.

topThree(L,W)
This gives the top 3 team in the ordered list.

PROGRAM STRUCTURE

This section explains the mechanisms of the predicates.

TASK1:

```
allTeams(L,N) :-
 findall(X,team(X,Y),M),
 length(M,N),
  permutation(M,L).
```

This predicate finds all the teams by looking all the team predicates,
then puts them in a list called M by means of the findall predicate.
After that it finds the length of the that list using the length predicate,
then it finds all the permutations of that list using permutation predicate and
puts them into the variable L.

TASK 2:

```
winsHelper(T,W,O) :-
  match(X,T,TS,O,OS),
  TS > OS,
  X =< W.
```

This predicate finds all the matches which the team T wins when it is hometeam.
This matches are played before the week W including the week W.

```
winsHelper(T,W,O) :-
  match(X,O,OS,T,TS),
  TS > OS,
  X =< W.
```

This predicate finds all the matches which the team T wins when it is away team.
This matches are played before the week W including the week W.

```
wins(T,W,L,N) :-
  findall(O,winsHelper(T,W,O),L),
  length(L,N).
```

this predicate finds all the matches which the team T wins by winsHelper predicate,
and then it puts the opponent teams played in these matches to a list called L. After that
it finds the length of that list.

```
lossesHelper(T,W,O) :-
  match(X,T,TS,O,OS),
  TS < OS,
  X =< W.
```

This predicate finds all the matches which the team T loses when it is hometeam.This matches are played before the week W including the week W.

```
lossesHelper(T,W,O) :-
  match(X,O,OS,T,TS),
  TS < OS,
  X =< W.
```

This predicate finds all the matches which the team T loses when it is awayteam.

This matches are played before the week W including the week W.

```
losses(T,W,L,N) :-
  findall(O,lossesHelper(T,W,O),L),
  length(L,N).
```

this predicate finds all the matches which the team T loses by lossesHelper predicate,
and then it puts the opponent teams played in these matches to a list called L. After that
it finds the length of that list.

```
drawsHelper(T,W,O) :-
  match(X,T,TS,O,OS),
  TS == OS,
  X =< W.
```

This predicate finds all the matches which the team T doesnt win or lose when it is hometeam.
This matches are played before the week W including the week W.

```
drawsHelper(T,W,O) :-
  match(X,O,OS,T,TS),
  TS == OS,
  X =< W.
```

This predicate finds all the matches which the team T doesnt win or lose when it is awayteam.
This matches are played before the week W including the week W.

```
draws(T,W,L,N) :-
  findall(O,drawsHelper(T,W,O),L),
  length(L,N).
```

this predicate finds all the matches which the team T doesnt win or lose by drawsHelper predicate,
and then it puts the opponent teams played in these matches to a list called L. After that
it finds the length of that list

TASK3:

```
listSoum([H|T],Sum) :-
  listSumHelper(T,H,Sum).
```

This predicate starts the summing process by initializing the accumulator with the head of the list
and calls the helper predicate with tail of the list.

```
listSumHelper([],Acc,Acc).
```

Base step of the recursive listSumHelper predicate which tells that stop the recursion when the list
is empty and make Sum equal to accumulator.

```
listSumHelper([H|T],Acc,Sum) :-
  NewAcc is H + Acc,
  listSumHelper(T,NewAcc,Sum).
```

This is a recursive predicate in which an accumulator sums up the elements of the list. It always takes the head of the list and adds it to the accumulator and calls the predicate with the tail of the list until the tail becomes an empty list.

```
scoredHelper(T,W,S) :-
  match(X,T,S,O,C),
  X =< W.
```

This is a helper predicate which finds all the scores scored by the given team before the given week including the given week when the team is hometeam.

```
scoredHelper(T,W,S) :-
  match(X,O,C,T,S),
  X =< W.
```

This is a helper predicate which finds all the scores scored by the given team before the given week including the given week when the team is awayteam.

```
scoredListGenerator(T,W,L) :-
  findall(S,scoredHelper(T,W,S),L),
  length(L,N).
```

This predicate generates a list which contains the scores scored by the given team before the given week including the given week.

```
concededHelper(T,W,C) :-
  match(X,T,S,O,C),
  X =< W.
```

This is a helper predicate which finds all the scores conceded by the given team before the given week including the given week when the team is hometeam.

```
concededHelper(T,W,C) :-
  match(X,O,C,T,S),
  X =< W.
```

This is a helper predicate which finds all the scores scored by the given team before the given week including the given week when the team is awayteam.

```
concededListGenerator(T,W,L) :-
  findall(C,concededHelper(T,W,C),L),
  length(L,N).
```

This predicate generates a list which contains the scores conceded by the given team before the given week including the given week.

```
scored(T,W,S) :-
  scoredListGenerator(T,W,L),
  listSum(L,S).
```

This predicate finds the total summation of the scores scored by the given team before the given week including the given week.

```
conceded(T,W,C) :-
  concededListGenerator(T,W,L),
  listSum(L,C).
```

This predicate finds the total summation of the scores conceded by the given team before the given week including the given week.

```
average(T,W,A) :-
  scored(T,W,S),
  conceded(T,W,C),
  A is S - C.
```

This predicate finds the average score of a team by substracting the scored points from the conceded points.

TASK4:

```
allTeamsHelper(L,N) :-
  findall(X,team(X,Y),M),
  length(M,N).
```

This predicate finds list of the teams without calculating the permutation.

```
orderHelper(W,L2,[Head|Tail],Final) :-
  findall(A-Head,average(Head,W,A),X),
  append(X,L2,MyList),
  length(MyList,N),
  allTeamsHelper(L3,M),
  N =< M,
  orderHelper(W,MyList,Tail,Final).
```

This helper predicate takes 4 arguments.
1st argument = week
2nd argument = list of the average-Team pairs which is calculated within the predicate
3rd argument = list of the teams
4th argument = Final list of the average-Team pairs.

This is a recursive predicate. In each step it finds a average-Team pair and adds it to a list by append predicate.
It stops when the total number of teams added to the list becomes equal to the original number of teams in the knowledge base.

```
orderHelper(W,L2,L4,Final) :-
  Final = L2,
  length(Final,N),
  allTeamsHelper(L,M),
  N == M.
```

This predicate is the one where the final form of the list is constituted.

```
order(L,W) :-
  findall(X,team(X,Y),L2),
  orderHelper(W,[],L2,L3),
  keysort(L3,Pairs),
  pairs_values(Pairs, L4),
  reverse(L4,L).
```

In this predicate, first the team list is generated and assigned to L2 variable. Then orderHelper predicate is called and the list which contains the average-Team pairs are generated and assigned to L3 variable. After that this list is sorted using the keysort predicate according to the averages in a ascending order. Then the average part of the pairs are removed by the pairs_values predicate and new list is assigned to L4. Finally the list is reversed by the reverse predicate and the desending order team list is generated and assigned to L.

```
topThree([T1,T2,T3|T], W) :-
  order([X,Y,Z|Tail],W),
  T1 = X,
  T2 = Y,
  T3 = Z,
  T = [].
```

In this predicate top three teams of the ordered list is fetched and assigned to the first argument of the top three predicate.

EXAMPLES

```
?- allTeams(L,N).

L = [galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad, shaktard, bleverkusen, omarseille, arsenal, fcnapoli, bdortmund]

N = 12;

L = [galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad, shaktard, bleverkusen, omarseille, arsenal, bdortmund, fcnapoli]
```

```
N = 12;

L = [galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad, shaktard, bleverkusen,
omarseille, bdortmund, arsenal, fcnapoli]

N = 12
True

?- allteams([galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad, shaktard,
bleverkusen, omarseille, arsenal, fcnapoli, bdortmund], 12).

True

?- allteams([], 12)
.
False

?- allteams(L, 12).

L = [galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad, shaktard,
bleverkusen, omarseille, bdortmund, arsenal, fcnapoli]
True
?- allteams([galatasaray, realmadrid, juventus, kobenhavn, manutd, realsociedad, shaktard,
bleverkusen, omarseille, bdortmund, arsenal, fcnapoli], N).
N = 12
False

?- wins(galatasaray,4,L,N).
L = [kobenhavn]
N = 1 ;
False

?- losses(galatasaray,4,L,N).
2L = [realmadrid, kobenhavn]
N = 2 ;
False

?- draws(galatasaray,4,L,N).
L = [juventus]
N = 1 ;
False

?- draws(galatasaray,4,[juventus],N).
N = 1 ;
False

?- draws(galatasaray,4,L,1).
L = [juventus] ;
```

```
False

?- draws(galatasaray,4,[juventus],1).
True

?- scored(juventus,5,S).
S = 9

?- conceded(juventus,5,C).
C = 8

?- average(kobenhavn, 3, A).
A = -6

?- average(kobenhavn, 6, A).
A = -9


?- order(L, 6).
L = [realmadrid, manutd, bdortmund, arsenal, fcnapoli, shaktard, juventus, bleverkusen,
galatasaray, realsociedad, kobenhavn, omarseille]

?-topThree(L, 6).
L = [realmadrid, manutd, bdortmund]
3?- order([realmadrid, manutd, bdortmund, arsenal, fcnapoli, shaktard, juventus, bleverkusen,
galatas
```

DIFFICULTIES ENCOUNTERED

This was my first logic programming project. I had difficulties in understanding concept. After I understood the mechanism it was easy.
I also had difficulty in recursive predicates.

APPENDİCES

```
%%%%%%TASK1

% This predicate finds all the teams by looking all the team predicates,
% then puts them in a list called M by means of the findall predicate.
% After that it finds the length of the that list using the length predicate,
% then it finds all the permutations of that list using permutation predicate and
% puts them into the variable L.
allTeams(L,N) :-
  findall(X,team(X,Y),M),
  length(M,N),
  permutation(M,L).

%TASK2
```

```
% This predicate finds all the matches which the team T wins when it is hometeam.
% This matches are played before the week W including the week W.
winsHelper(T,W,O) :-
  match(X,T,TS,O,OS),
  TS > OS,
  X =< W.

% This predicate finds all the matches which the team T wins when it is away team.
% This matches are played before the week W including the week W.
winsHelper(T,W,O) :-
  match(X,O,OS,T,TS),
  TS > OS,
  X =< W.

% this predicate finds all the matches which the team T wins by winsHelper predicate,
% and then it puts the opponent teams played in these matches to a list called L. After that
% it finds the length of that list.
wins(T,W,L,N) :-
  findall(O,winsHelper(T,W,O),L),
  length(L,N).

% This predicate finds all the matches which the team T loses when it is hometeam.
% This matches are played before the week W including the week W.
lossesHelper(T,W,O) :-
  match(X,T,TS,O,OS),
  TS < OS,
  X =< W.

% This predicate finds all the matches which the team T loses when it is awayteam.
% This matches are played before the week W including the week W.
lossesHelper(T,W,O) :-
  match(X,O,OS,T,TS),
  TS < OS,
  X =< W.

% this predicate finds all the matches which the team T loses by lossesHelper predicate,
% and then it puts the opponent teams played in these matches to a list called L. After that
% it finds the length of that list.
losses(T,W,L,N) :-
  findall(O,lossesHelper(T,W,O),L),
  length(L,N).

% This predicate finds all the matches which the team T doesnt win or lose when it is hometeam.
% This matches are played before the week W including the week W.
drawsHelper(T,W,O) :-
  match(X,T,TS,O,OS),
  TS == OS,
  X =< W.
```

```
% This predicate finds all the matches which the team T doesnt win or lose when it is awayteam.
% This matches are played before the week W including the week W.
drawsHelper(T,W,O) :-
  match(X,O,OS,T,TS),
  TS == OS,
  X =< W.

% this predicate finds all the matches which the team T doesnt win or lose by drawsHelper
predicate,
% and then it puts the opponent teams played in these matches to a list called L. After that
% it finds the length of that list
draws(T,W,L,N) :-
  findall(O,drawsHelper(T,W,O),L),
  length(L,N).

%TASK3

%The function of the following three predicates are to sum up the elements of a given list
consisting of numbers.

%This predicate starts the summing process by initializing the accumulator with the head of the list
and calls
%the helper predicate with tail of the list.
listSoum([H|T],Sum) :-
  listSumHelper(T,H,Sum).

%Base step of the recursive listSumHelper predicate which tells that stop the recursion when the
list is empty and
%make Sum equal to accumulator.
listSumHelper([],Acc,Acc).

%This is a recursive predicate in which an accumulator sums up the elements of the list. It always
takes the head of the list
%and adds it to the accumulator and calls the predicate with the tail of the list until the tail
becomes an empty list.
listSumHelper([H|T],Acc,Sum) :-
  NewAcc is H + Acc,
  listSumHelper(T,NewAcc,Sum).

% This is a helper predicate which finds all the scores scored by the given team before the given
week including the given week
%when the team is hometeam.
scoredHelper(T,W,S) :-
  match(X,T,S,O,C),
  X =< W.

% This is a helper predicate which finds all the scores scored by the given team before the given
week including the given week
```

```
%when the team is awayteam.
scoredHelper(T,W,S) :-
  match(X,O,C,T,S),
  X =< W.

%This predicate generates a list which contains the scores scored by the given team before the
given week including the given week.
scoredListGenerator(T,W,L) :-
  findall(S,scoredHelper(T,W,S),L),
  length(L,N).

% This is a helper predicate which finds all the scores conceded by the given team before the given
week including the given week
%when the team is hometeam.
concededHelper(T,W,C) :-
  match(X,T,S,O,C),
  X =< W.

% This is a helper predicate which finds all the scores scored by the given team before the given
week including the given week
%when the team is awayteam.
concededHelper(T,W,C) :-
  match(X,O,C,T,S),
  X =< W.

%This predicate generates a list which contains the scores conceded by the given team before the
given week including the given week.
concededListGenerator(T,W,L) :-
  findall(C,concededHelper(T,W,C),L),
  length(L,N).

%This predicate finds the total summation of the scores scored by the given team before the given
week including the given week.
scored(T,W,S) :-
  scoredListGenerator(T,W,L),
  listSum(L,S).

%This predicate finds the total summation of the scores conceded by the given team before the
given week including the given week.
conceded(T,W,C) :-
  concededListGenerator(T,W,L),
  listSum(L,C).

%This predicate finds the average score of a team by substracting the scored points from the
conceded points.
average(T,W,A) :-
  scored(T,W,S),
  conceded(T,W,C),
  A is S - C.
```

```
%TASK4

%This predicate finds list of the teams without calculating the permutation.
allTeamsHelper(L,N) :-
  findall(X,team(X,Y),M),
  length(M,N).

% This helper predicate takes 4 arguments.
% 1st argument = week
% 2nd argument = list of the average-Team pairs which is calculated within the predicate
% 3rd argument = list of the teams
% 4th argument = Final list of the average-Team pairs.
%%% This is a recursive predicate. In each step it finds a average-Team pair and adds it to a list by
append predicate.
%%% It stops when the total number of teams added to the list becomes equal to the original
number of teams in the knowledge base.
orderHelper(W,L2,[Head|Tail],Final) :-
  findall(A-Head,average(Head,W,A),X),
  append(X,L2,MyList),
  length(MyList,N),
  allTeamsHelper(L3,M),
  N =< M,
  orderHelper(W,MyList,Tail,Final).

%%% This predicate is the one where the final form of the list is constituted.
orderHelper(W,L2,L4,Final) :-
  Final = L2,
  length(Final,N),
  allTeamsHelper(L,M),
  N == M.

% In this predicate, first the team list is generated and assigned to L2 variable. Then orderHelper
predicate is called and
% the list which contains the average-Team pairs are generated and assigned to L3 variable. After
that this list is sorted
% using the keysort predicate according to the averages in a ascending order. Then the average
part of the pairs are removed
% by the pairs_values predicate and new list is assigned to L4. Finally the list is reversed by the
reverse predicate and the
% desending order team list is generated and assigned to L.
order(L,W) :-
  findall(X,team(X,Y),L2),
  orderHelper(W,[],L2,L3),
  keysort(L3,Pairs),
  pairs_values(Pairs, L4),
  reverse(L4,L).
```

```
%In this predicate top three teams of the ordered list is fetched and assigned to the first argument
of the top three predicate.
topThree([T1,T2,T3|T], W) :-
  order([X,Y,Z|Tail],W),
  T1 = X,
  T2 = Y,
  T3 = Z,
  T = [].
```