# CMPE 260 PRINCIPLES OF PROGRAMMING LANGUAGES

Prolog Football Knowledge Base Assignment (Assignment I)

Programming Project

Student Name: Serkan ÖZEL

Name of the submitted person: Tunga GÜNGÖR

Introduction:

In this project we will make kind of a data analysis/useful information extraction from raw data. Suppose we are given a prolog knowledge base file, let's say its name is cl\_base.pl, (as we will use this name for illustration example) basically, it can have any name. This knowledge base includes some facts and rules about matches between football teams organized week by week. Now we can extract other useful information by using match information given in our knowledge base. We do this by adding new rules and facts to the knowledge base. Then, in prolog we obtain results from knowledge bases by posing queries.

Terminology:

Knowledge base: This means a collection of facts and rules in prolog programming language. In prolog data and program are the same thing.

Facts: Something we say to be true to the program. It can have parameters in it to be evaluated according to them and be true or false. Written like this: fact(a).

Rules: Something we say to be true to the program if some facts are true. It can have parameters in it to be evaluated according to them and be true or false. Written like this rule(a):- fact(b),fact(c).

Query: A fact or rule written in console view to extract related information from the knowledge base. Based on the query result can be multiple or singular value of some variables or just a true/false.

Constant: Atoms or numbers Please refer here for more: <http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse2>

Variable: A Prolog variable can represent anything; a number, a name, a structure, an array, something as complicated as the known universe. A Prolog program works by constraining the variables until eventually they have particular values; then telling you what the values are. Source:<https://en.wikibooks.org/wiki/Prolog/Variables> Access Time: 22.04.2018 00:11

Program Interface:

We are using swi-prolog in this project you can reach its official site here: <http://www.swi-prolog.org/>.

1. Opening prolog environment:
   * A user can start prolog interpreter by entering the command "prolog" after installing swi-prolog to unix based systems. Please make sure to start prolog where the knowledge base is located to directly call it by its name rather than it whole path. .
   * If you installed swi-prolog to windows there should be direct executable and gui rather than console itself.
2. Preparing your knowledge base:
   * In src file there is a file called predicates.pl you need to open that file and copy the rules and facts inside it to your own knowledge base.
   * Then you need to import your knowledge base. Please be informed that if the prolog hasn't started in the directory your knowledge base is in, you need to explicitly state the path of your knowledge base. Refer here for more: <http://lpn.swi-prolog.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse4>
   * The predicates that you can write to your knowledge base are as follows:

team(teamName, hometown).

match(week, homeTeam, homeTeamScore, awayTeam,awayTeamScore).

* + The following is a portion of a sample database illustrating the two relations that are defined:

team(realmadrid, madrid).

team(juventus, torinp).

team(galatasaray, istanbul).

team(kobenhavn, kopenag).

match(1, galatasaray, 1, realmadrid, 6).

match(1, kobenhavn, 1, juventus, 1).

match(2, juventus, 2, galatasaray, 2).

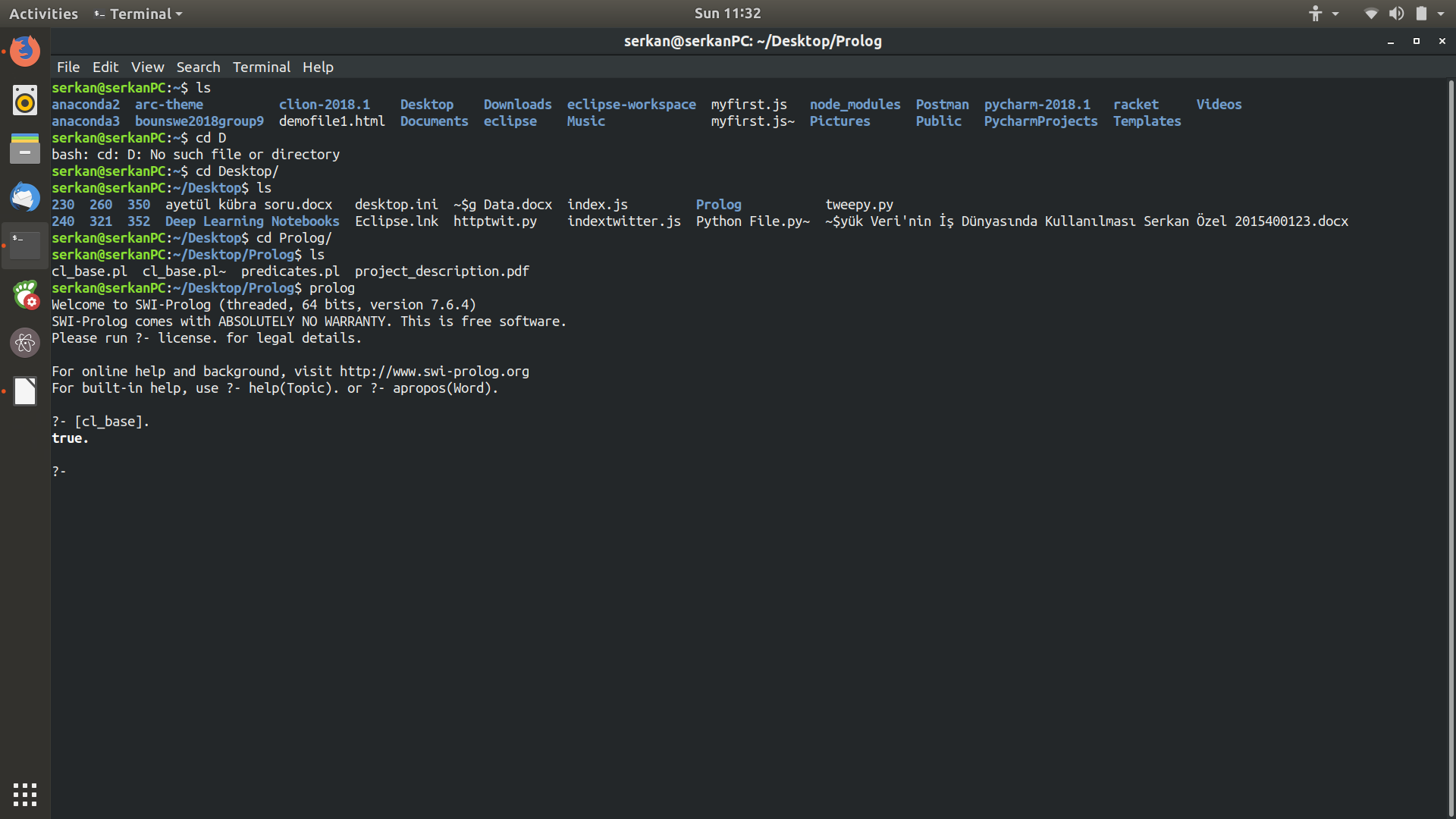
match(2, realmadrid, 4, kobenhavn, 0).

1. Program termination:
   * You can terminate the prolog interpreter in windows by pressing close button top right.
   * You can terminate the prolog interpreter in unix by pressing Control+C. If it is not killed, -you can check this by calling the "ps" command- use "kill –9 <p-id>" command where <p-id> is the process id which can be learned from calling the "ps" command.

Program Execution:

Functionality: The functionality of this prolog program is to extract useful information from raw data.

Execution and Usage: When you open prolog interpreter and imported the knowledge base you are ready to pose some queries to get an answer.

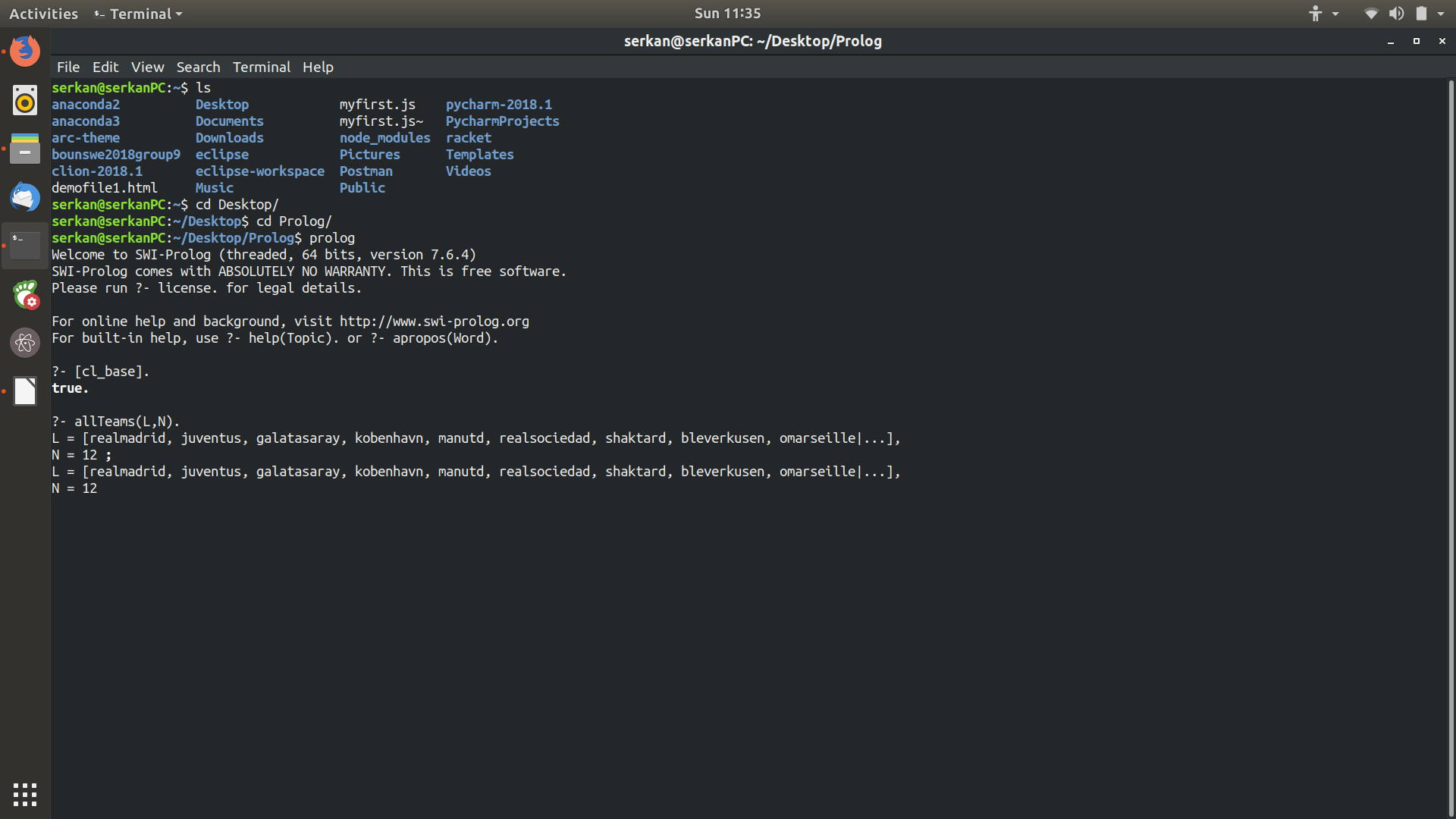


After showing one result that fits the query prolog will suspend. Please press ; if you want to get more results.

You can press a to abort.

You can press h for help.

You can press w to see all the result in full written form.



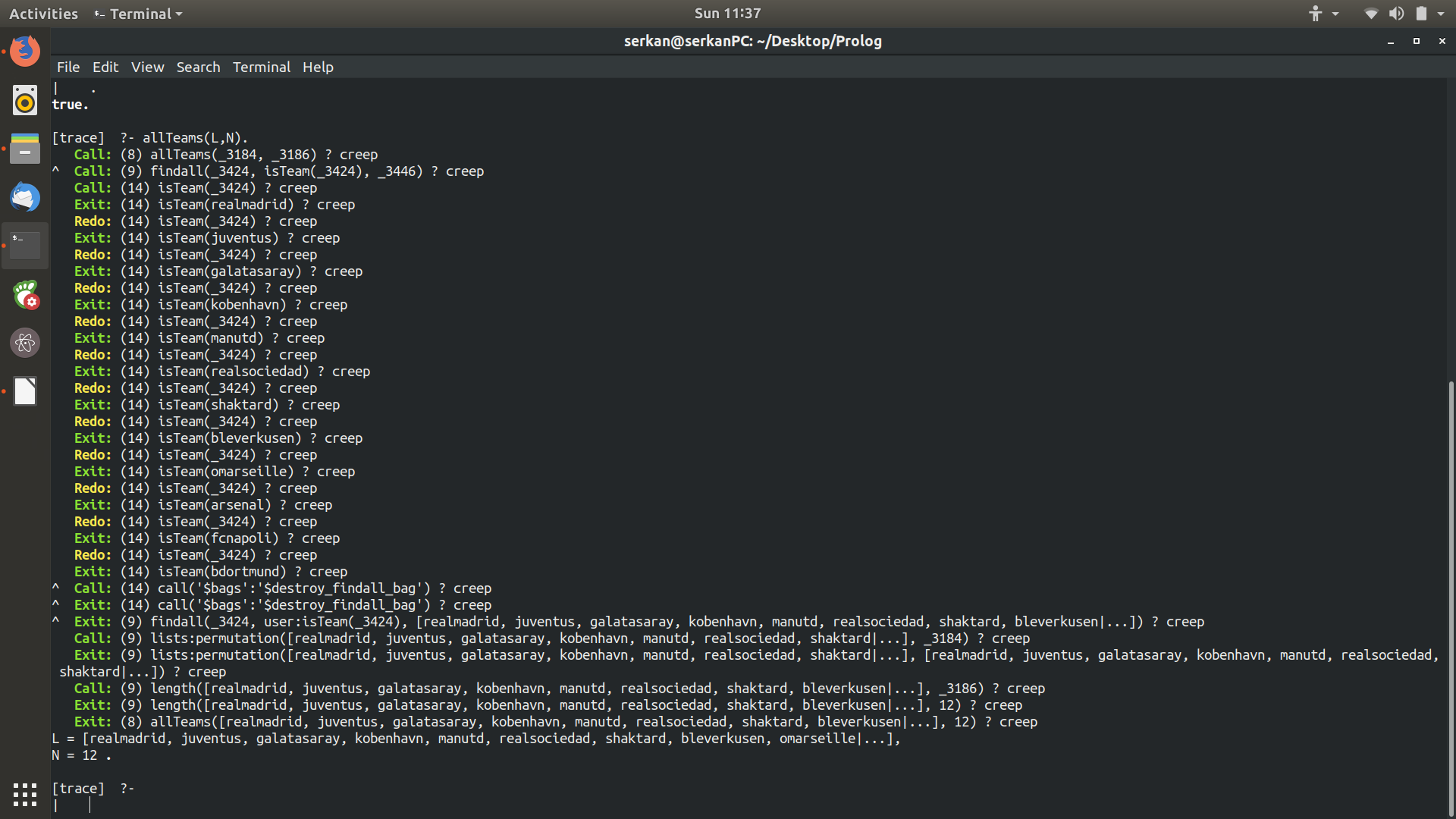
Tracing(like debugging):

If you want you can trace the predicates called by typing "trace" to the prolog console and then entering queries. If you want to undo type "notrace".

Call means prolog is entering a fact or rule.

Exit means prolog finished a fact or rule evaluation and backtracking.

Redo means prolog try to find another result that fits the query.



Input & Output:

Input:

These queries can be posed by the user:

allTeams(List,Number).

This is for finding all permutations of all teams.

Number is the number of all teams. List holds all teams in a list.

wins(Team,Week,List,Number).

This query is for finding the teams that team Team wins.

Team should be given as constant.

Week should be given as constant.

List is the list of the teams that satisfy the query and can be a variable also, Number is the number of elements in List and can be a variable also.

losses(Team,Week,List,Number).

This query is for finding the teams that team Team loses against.

Team should be given as constant.

Week should be given as constant.

List is the list of the teams that satisfy the query and can be a variable also, Number is the number of elements in List and can be a variable also.

draws(Team,Week,List,Number).

This query is for finding the teams that team Team draws with.

Team should be given as constant.

Week should be given as constant.

List is the list of the teams that satisfy the query and can be a variable also, Number is the number of elements in List and can be a variable also.

scored(Team,Week,Score).

Score is the total number of scores scored by the Team up to week Week and Score can be a variable.

Team should be given as constant.

Week should be given as constant.

conceded(Team,Week,Score).

Score is the total number of scores conceded by the Team up to week Week and Score can be a variable.

Team should be given as constant.

Week should be given as constant.

average(Team,Week,Average).

This computes average of a team up to a week. Average means goals scored minus goal conceded and Average can be a variable.

Team should be given as constant.

Week should be given as constant.

order(List,Week).

This predicate shows the ordered status of the teams in week Week. List is the returned ordered list.

Week should be given as constant.

topThree([T1,T2,T3],Week).

This predicate finds top three teams in week Week. Week is given as a constant.

T1 is the first, T2 is the second, T3 is the third team. These can be variables also.

Note: If you don't pose a query including a variable, prolog checks if the meaning of the predicate is true or not and return true or false. Whereas if you include variables in your query prolog will try to instantiate the variables such that the your query's meaning is true.

Program Structure:

The predicates that I wrote are works as follows.

The built in functions used:

findall/3 <http://www.swi-prolog.org/pldoc/man?predicate=findall/3>

permutation/2 <http://www.swi-prolog.org/pldoc/man?predicate=permutation/2>

keysort/2 <http://www.swi-prolog.org/pldoc/man?predicate=keysort/2>

reverse/2<http://www.swi-prolog.org/pldoc/man?predicate=reverse/2>

length/2 <http://www.swi-prolog.org/pldoc/man?predicate=length/2>

append/3 <http://www.swi-prolog.org/pldoc/man?predicate=append/3>

allTeams(List,Number) :- findall(Team,isTeam(Team),AllTeams), permutation(AllTeams,List), length(AllTeams,Number).

* This is for finding all permutations of all teams.
* First I construct a list AllTeams containing all teams in the given order in predicates.
* Then I find all permutations of AllTeams list into List.
* I compute the length of AllTeams to find Number.

wins(Team,Week,List,Number) :- findall(OtherTeam,(match(CurWeek,Team,Score1,OtherTeam,Score2),Score1>Score2,CurWeek=<Week),List1),findall(OtherTeam2,(match(CurWeek2,OtherTeam2,Score3,Team,Score4),Score4>Score3,CurWeek2=<Week),List2),append(List1,List2,List),length(List,Number).

* Here I find all match predicates and extract OtherTeam variable's value.
* OtherTeam holds the team drawed with Team, i.e given team.
* Conditions in findall:
* 1-Compare scores and find matches Team wins.
* 2-Also compare CurWeek with Week to find matches within (Week's value) weeks.
* Find home matches and away matches seperately and append the results.
* Finally, compute Number with length predicate.

losses(Team,Week,List,Number) :- findall(OtherTeam,(match(CurWeek,Team,Score1,OtherTeam,Score2),Score1<Score2,CurWeek=<Week),List1),findall(OtherTeam2,(match(CurWeek2,OtherTeam2,Score3,Team,Score4),Score4<Score3,CurWeek2=<Week),List2),append(List1,List2,List),length(List,Number).

* Here I find all match predicates and extract OtherTeam variable's value.
* OtherTeam holds the team drawed with Team, i.e given team.
* Conditions in findall:
* 1-Compare scores and find matches Team loses.
* 2-Also compare CurWeek with Week to find matches within (Week's value) weeks.
* Find home matches and away matches seperately and append the results.
* Finally, compute Number with length predicate.

draws(Team,Week,List,Number) :- findall(OtherTeam,(match(CurWeek,Team,Score1,OtherTeam,Score2),Score1=:=Score2,CurWeek=<Week),List1),findall(OtherTeam2,(match(CurWeek2,OtherTeam2,Score3,Team,Score4),Score4=:=Score3,CurWeek2=<Week),List2),append(List1,List2,List),length(List,Number).

* Here I find all match predicates and extract OtherTeam variable's value.
* OtherTeam holds the team drawed with Team, i.e given team.
* Conditions in findall:
* 1-Compare scores and find matches Team draws.
* 2-Also compare CurWeek with Week to find matches within (Week's value) weeks.
* Find home matches and away matches seperately and append the results.
* Finally, compute Number with length predicate.

scored(Team,Week,Score):- findall(Score1,(match(CurWeek,Team,Score1,\_,\_),CurWeek=<Week),List1),findall(Score2,(match(CurWeek,\_,\_,Team,Score2),CurWeek=<Week),List2),append(List1,List2,List),listsum(List,Score).

* Here we only extract score of the Team from match predicates
* Only extra condition in findall is week comparison
* Find home matches and away matches seperately and append the results.
* Finally, compute Score with listsum predicate that I wrote before.

conceded(Team,Week,Score):- findall(Score1,(match(CurWeek,Team,\_,\_,Score1),CurWeek=<Week),List1),findall(Score2,(match(CurWeek,\_,Score2,Team,\_),CurWeek=<Week),List2),append(List1,List2,List),listsum(List,Score).

* Here we only extract score of the OtherTeam from match predicates
* Only extra condition in findall is week comparison
* Find home matches and away matches seperately and append the results.
* Finally, compute Score with listsum predicate that I wrote before.

average(Team,Week,Average):- scored(Team,Week,Scored), conceded(Team,Week,Conceded), Average is Scored-Conceded.

* I used last 2 predicates to find scored and conceded goals of a team then subtract them to get the average.

order(List,Week):- findall(Average-Team,(isTeam(Team),average(Team,Week,Average)),AllTeams),keysort(AllTeams,ReverseSorted),reverse(Sorted,ReverseSorted),findall(Team2,member(\_-Team2,Sorted),List).

* First I form key value pairs. Average is averages of the teams and it is the key whereas Team is team name and value.
* AllTeams holds key value pairs. Then I call keysort which sorts according to the keys.
* This sorts but in reverse order, so I reverse ReverseSorted into Sorted.
* Sorted includes key value pairs I need only teams. Therefore, I extract team names from Sorted using findall predicate.
* I used member predicate with findall function to transfer all the members into List.

topThree([T1,T2,T3],Week):- order([T1,T2,T3|\_],Week).

* I basically take the first three elements of the list returned by order predicate.

Listsum sums elements of a list.

listsum([], 0). % base case empty list sum is 0

listsum([Head | Tail], Total) :- listsum(Tail, Sum), Total is Head + Sum.

% This predicate sums all the element in a list.

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).

L = [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]

?- order([realmadrid, manutd, bdortmund, arsenal, fcnapoli, shaktard, juventus, bleverkusen,

galatasaray, realsociedad, kobenhavn, omarseille], W).

W = 3

Improvement and Extensions:

This program can include more rules to extract more useful information like the last three team etc.

Also this project is week based, a system can be added to make it day based and each day some of the matches are played some of them aren't.

Also it will be good if the program should be able to connect to internet and get the real data from a server.

Difficulties Encountered:

First, I struggled a little bit due to the nature of the prolog programming. Data storing via a variable is quite hard compared to imperative languages.

Other difficulties were getting used to the recursion and finding which built In functions to use and how they are used.

Conclusion:

I think this project can be used as a program if the user have match and teams data. There is a lot of better analytics programs out there but this is different that it is implemented using prolog.

Also this project is good to teach some students prolog because it contains most of the concepts in it.

Appendices:

Program source codes:

------------------------------------------------------------START------------------------------------

Example knowledge base:

team(realmadrid, madrid).

team(juventus, torino).

team(galatasaray, istanbul).

team(kobenhavn, copenhagen).

team(manutd, manchester).

team(realsociedad, sansebastian).

team(shaktard, donetsk).

team(bleverkusen, leverkusen).

team(omarseille, marseille).

team(arsenal, london).

team(fcnapoli, napoli).

team(bdortmund, dortmund).

match(1, galatasaray, 1, realmadrid, 6).

match(1, kobenhavn, 1, juventus, 1).

match(1, manutd, 4, bleverkusen, 2).

match(1, realsociedad, 0, shaktard, 2).

match(1, omarseille, 1, arsenal, 2).

match(1, fcnapoli, 2, bdortmund, 1).

match(2, juventus, 2, galatasaray, 2).

match(2, realmadrid, 4, kobenhavn, 0).

match(2, shaktard, 2, manutd, 3).

match(2, bleverkusen, 1, realsociedad, 1).

match(2, bdortmund, 3, omarseille, 0).

match(2, arsenal, 2, fcnapoli, 0).

match(3, galatasaray, 3, kobenhavn, 1).

match(3, realmadrid, 2, juventus, 1).

match(3, manutd, 1, realsociedad, 0).

match(3, bleverkusen, 4, shaktard, 0).

match(3, omarseille, 1, fcnapoli, 2).

match(3, arsenal, 1, bdortmund, 2).

match(4, kobenhavn, 1, galatasaray, 0).

match(4, juventus, 2, realmadrid, 2).

match(4, bleverkusen, 0, manutd, 5).

match(4, shaktard, 4, realsociedad, 0).

match(4, fcnapoli, 4, omarseille, 2).

match(4, bdortmund, 0, arsenal, 1).

match(5, realmadrid, 4, galatasaray, 1).

match(5, juventus, 3, kobenhavn, 1).

match(5, realsociedad, 0, manutd, 0).

match(5, shaktard, 0, bleverkusen, 0).

match(5, bdortmund, 3, fcnapoli, 1).

match(5, arsenal, 2, omarseille, 0).

match(6, galatasaray, 1, juventus, 0).

match(6, kobenhavn, 0, realmadrid, 2).

match(6, manutd, 1, shaktard, 0).

match(6, realsociedad, 2, bleverkusen, 0).

match(6, omarseille, 1, bdortmund, 2).

match(6, fcnapoli, 2, arsenal, 0).

------------------------------------------------------------END------------------------------------

predicates.pl

Note that % used in the source code means single line comment.

------------------------------------------------------------START------------------------------------

isTeam(Team):- team(Team,\_).

% This is used in my predicates. Extraction of team names from team(teamName,Location) is done here.

allTeams(List,Number) :- findall(Team,isTeam(Team),AllTeams), permutation(AllTeams,List), length(AllTeams,Number).

% This is for finding all permutations of all teams.

% Number is the number of all teams. List holds all teams in a list.

% First I construct a list AllTeams containing all teams in the given order in predicates.

% Then I find all permutations of AllTeams list into List.

% I compute the length of AllTeams to find Number.

wins(Team,Week,List,Number) :- findall(OtherTeam,(match(CurWeek,Team,Score1,OtherTeam,Score2),Score1>Score2,CurWeek=<Week),List1),findall(OtherTeam2,(match(CurWeek2,OtherTeam2,Score3,Team,Score4),Score4>Score3,CurWeek2=<Week),List2),append(List1,List2,List),length(List,Number).

% WINS:

% Team(constant), Week(constant),List- list of the teams that satisfy the query,Number - Number of elements in List

% \* Here I find all match predicates and extract OtherTeam variable's value.

% \* OtherTeam holds the team defeated by Team, i.e given team.

% \* Conditions in findall:

% 1-Compare scores and find matches winned by the Team.

% 2-Also compare CurWeek with Week to find matches within (Week's value) weeks.

% \* Find home matches and away matches seperately and append the results.

% \* Finally, compute Number with length predicate.

losses(Team,Week,List,Number) :- findall(OtherTeam,(match(CurWeek,Team,Score1,OtherTeam,Score2),Score1<Score2,CurWeek=<Week),List1),findall(OtherTeam2,(match(CurWeek2,OtherTeam2,Score3,Team,Score4),Score4<Score3,CurWeek2=<Week),List2),append(List1,List2,List),length(List,Number).

% LOSSES:

% Team(constant), Week(constant),List- list of the teams that satisfy the query,Number - Number of elements in List

% \* Here I find all match predicates and extract OtherTeam variable's value.

% \* OtherTeam holds the team wins Team, i.e given team.

% \* Conditions in findall:

% 1-Compare scores and find matches defeated by the Team.

% 2-Also compare CurWeek with Week to find matches within (Week's value) weeks.

% \* Find home matches and away matches seperately and append the results.

% \* Finally, compute Number with length predicate.

% \* Note that only difference with wins is comparison of the scores.

draws(Team,Week,List,Number) :- findall(OtherTeam,(match(CurWeek,Team,Score1,OtherTeam,Score2),Score1=:=Score2,CurWeek=<Week),List1),findall(OtherTeam2,(match(CurWeek2,OtherTeam2,Score3,Team,Score4),Score4=:=Score3,CurWeek2=<Week),List2),append(List1,List2,List),length(List,Number).

% DRAWS:

% Team(constant), Week(constant),List- list of the teams that satisfy the query,Number - Number of elements in List

% \* Here I find all match predicates and extract OtherTeam variable's value.

% \* OtherTeam holds the team drawed with Team, i.e given team.

% \* Conditions in findall:

% 1-Compare scores and find matches Team draws.

% 2-Also compare CurWeek with Week to find matches within (Week's value) weeks.

% \* Find home matches and away matches seperately and append the results.

% \* Finally, compute Number with length predicate.

% \* Note that only difference with wins or losses is comparison of the scores.

scored(Team,Week,Score):- findall(Score1,(match(CurWeek,Team,Score1,\_,\_),CurWeek=<Week),List1),findall(Score2,(match(CurWeek,\_,\_,Team,Score2),CurWeek=<Week),List2),append(List1,List2,List),listsum(List,Score).

% SCORED:

% \* Score is the total number of scores scored by the Team up to week Week.

% \* Here we only extract score of the Team from match predicates

% \* Only extra condition in findall is week comparison

% \* Find home matches and away matches seperately and append the results.

% \* Finally, compute Score with listsum predicate that I wrote before.

conceded(Team,Week,Score):- findall(Score1,(match(CurWeek,Team,\_,\_,Score1),CurWeek=<Week),List1),findall(Score2,(match(CurWeek,\_,Score2,Team,\_),CurWeek=<Week),List2),append(List1,List2,List),listsum(List,Score).

% CONCEDED:

% \* Score is the total number of scores conceded by the Team up to week Week.

% \* Here we only extract score of the OtherTeam from match predicates

% \* Only extra condition in findall is week comparison

% \* Find home matches and away matches seperately and append the results.

% \* Finally, compute Score with listsum predicate that I wrote before.

average(Team,Week,Average):- scored(Team,Week,Scored), conceded(Team,Week,Conceded), Average is Scored-Conceded.

% This computes average of a team up to a week. Average means goals scored minus goal conceded.

% I used last 2 predicates to find scored and conceded goals of a team then subtract them to get the average.

order(List,Week):- findall(Average-Team,(isTeam(Team),average(Team,Week,Average)),AllTeams),keysort(AllTeams,ReverseSorted),reverse(Sorted,ReverseSorted),findall(Team2,member(\_-Team2,Sorted),List).

% This predicate shows the ordered status of the teams in week Week. List is the returned ordered list.

% First I form key value pairs. Average is averages of the teams and it is the key whereas Team is team name and value.

% AllTeams holds key value pairs. Then I call keysort which sorts according to the keys.

% This sorts but in reverse order, so I reverse ReverseSorted into Sorted.

% Sorted includes key value pairs I need only teams. Therefore, I extract team names from Sorted using findall predicate.

% I used member predicate with findall function to transfer all the members into List.

topThree([T1,T2,T3],Week):- order([T1,T2,T3|\_],Week).

% This predicate finds top three teams in week Week.

% T1 is the first, T2 is the second, T3 is the third team.

% I basically take the first three elements of the list returned by order predicate.

listsum([], 0).

% base case

listsum([Head | Tail], Total) :-

listsum(Tail, Sum),

Total is Head + Sum.

% This predicate sums all the element in a list.

% empty list sum is 0

------------------------------------------------------------END------------------------------------