CMPE 260 – PRINCIPLES OF PROGRAMMING LANGUAGES

AFRA AKBAŞ

HASAN FERİT ENİŞER

PROLOG PROJECT

PROGRAMMING PROJECT

22.04.2018

The goal is to write the Prolog predicates that gives some information about the football teams. There are some predicates that already exist in the database. *team(teamName, hometown)* and *match(week, homeTeam, homeTeamScore, awayTeam, awayTeamScore).* The former is the team with its home town. The latter gives us a match that is played in the given week between the given teams. I implemented the predicates described in the project description. Those predicates are *allTeams(L, N), wins(T, W, L, N), losses(T, W, L, N), draws(T, W, L, N), scored(T, W, S), conceded(T, W, S), average(T, W, A), order(L, W)* and *topThree(L, W)*. With the *allTeams* predicate, we gain insight about the all existing teams in the database. *wins* and *losses* and *draws* are the match results. These gives the teams that won the match or are defeated by certain team or did neither of them. *scored* and *conceded* are number of goals up to the given week. *scored* is the number of scored by the given team and *conceded* is the scored by other teams. Goals scored – goals conceded gives the average of a team so *average* is the average score of the team up to given week. order of all teams in the given week can be reached with *order* predicate. It sorts all teams by descending according to their average. Additionally, *topThree* gives the top three teams in the order list.

To execute the program, you must install the SWI-Prolog to your computer. After the installation, run the program and load the source files. It can be more than one file. The program consults the files and uses its clauses. Then you give the goal to system. The system checks the source file and gives the output.

In this project, the program uses the source file which names **predicates.pl.** This prolog file contains the facts and predicates. In prolog, you can give the inputs as variable by instantiating in the goal. Variables begin with uppercase letters. Because prolog is the logic programming, outputs are true and false. If your predicate can be arrived by facts, the output is true. Otherwise, it false. If there are several cases, it gives the possible outputs.

Here is my prolog code;

* *count* is an additional predicate that I need for addition.

/\*

\* count is the summation of all terms in list

\*the former is the list

\*X is the sum of elements

\*adds elements recursively

\*/

**count([], X):- X is 0.**

**count([H|T], N):- count(T, N1), N is N1+H.**

* *allTeams* predicate that I implemented

/\*

\*finds all teams in the database

\*L is the list that contains all teams

\*N is the number of teams

\*findall is the built-in function that finds all teams in the base and append in L1

\*perm takes the permutation of L1

\*length is the built-in function that find the length of L

\*/

**allTeams(L, N):- findall(X, team(X, \_), L1), permutation(L1, L), length(L, N).**

* My additional predicates(*check\_wins, win\_list*) to define *wins* predicate.

/\*

\*win\_list is the predicate that generates the list

\*findall finds the defeated team and appends into L1 and L2

\*appends L1 and L2 into L

\*

\*check\_wins is the predicates that checks the winning conditions

\*W1 is the week between 1 to W

\*match is the predicates in the database

\*/

**check\_wins(T, W, X):- match(W1, T, A, X, B), A>B, between(1, W, W1).**

**win\_list(T, W, L):- findall(X, check\_wins(T, W, X), L1),**

**findall(X1, check\_losses(X1, W, T), L2), append(L1, L2, L).**

* *wins* predicate I implemented

/\*

\*wins finds the teams that are defeated by team T up to week W

\*T and W are constants

\*L is the list of defeated teams

\*N is the length of L

\*/

**wins(T, W, L, N):- win\_list(T, W, L), length(L, N).**

* My additional predicates (*check\_losses, loss\_list*) to define *losses* predicate.

/\*

\*loss\_list is the predicate that generates the list

\*findall finds the winner team and appends into L1 and L2

\*appends L1 and L2 into L

\*

\*check\_losses is the predicates that checks the losing conditions

\*W1 is the week between 1 to W

\*match is the predicates in the database

\*/

**check\_losses(T, W, X):- match(W1, T, A, X, B), A<B, between(1, W, W1).**

**loss\_list(T, W, L):- findall(X, check\_losses(T, W, X), L1),**

**findall(X1, check\_wins(X1, W, T), L2), append(L1, L2, L).**

* *losses* predicate I implemented

/\*

\*losses finds the teams that defeated team T up to week W

\*T and W are constants

\*L is the list of winner teams

\*N is the length of L

\*loss\_list finds list L

\*/

**losses(T, W, L, N):- loss\_list(T, W, L), length(L, N).**

* My additional predicates (*check\_draws, draw\_list*) to define *draws* predicate.

/\*

\*check\_draws is the predicates that checks the draw conditions

\*W1 is the week between 1 to W

\*match is the predicates in the database

\*

\*draw\_list is the predicate that generates the list

\*findall finds the teams and appends into L1 and L2

\*appends L1 and L2 into L

\*/

**check\_draws(T, W, X):- match(W1, T, A, X, B), A is B, between(1, W, W1).**

**draw\_list(T, W, L):- findall(X, check\_draws(T, W, X), L1),**

**findall(X1, check\_draws(X1, W, T), L2), append(L1, L2, L).**

* *draws* predicate I implemented

/\*

\*draws finds the teams that team T could not defeat and did not lose

up to week W

\*T and W are constants

\*L is the list of winner teams

\*N is the length of L

\*/

**draws(T, W, L, N):- draw\_list(T, W, L), length(L, N).**

* My additional predicates to define *scored* predicate.

/\*

\* ts\_inverse is the predicates that checks the matches in the awayhome

\* tscored is the predicates that checks the matches in the hometown

\*/

**tscored(T, W, S):- match(W1, T, S, \_, \_), between(1, W, W1).**

**ts\_inverse(T, W, S):- match(W1, \_, \_, T, S), between(1, W, W1).**

* *scored* predicate I implemented

/\*

\*scored is the predicate that contains T, W, S

\*T and W are constants, T is the team, W is week

\*S is the number of goals scored by team T

\*findall finds the goals and appends into the L1 and L2

\*appends L1, L2 into the L

\*count is the predicate that adds the elements

\*/

**scored(T, W, S):- findall(X, tscored(T, W, X), L1),**

**findall(X, ts\_inverse(T, W, X), L2), append(L1, L2, L), count(L, S).**

* My additional predicates to define *conceded* predicate.

/\*

\* tc\_inverse is the predicates that checks the matches in the awayhome

\* tconceded is the predicates that checks the matches in the hometown

\* T and W are constants, T is the team, W is week

\* W1 is the week between 1 to W

\*/

**tconceded(T, W, C):- match(W1, T, \_, \_, C), between(1, W, W1).**

**tc\_inverse(T, W, C):- match(W1, \_, C, T, \_), between(1, W, W1).**

* *conceded* predicate I implemented

/\*

\*conceded is the predicate that contains T, W, S

\*T and W are constants, T is the team, W is week

\*S is the number of goals conceded by team T

\*findall finds the goals and appends into the L1 and L2

\*appends L1, L2 into the L

\*count is the predicate that adds the elements

\*/

**conceded(T, W, C):- findall(X, tconceded(T, W, X), L1),**

**findall(X, tc\_inverse(T, W, X), L2), append(L1, L2, L), count(L, C).**

* *average* predicate I implemented

/\*

\*average is the predicates that finds the average score

\*T and W is the constant, T is the team, W is week

\*A is the number of scored goals minus the number of conceded goals

\*scored and conceded give the numbers of goals

\*/

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

* My additional predicate to help *q\_sort* predicate

/\*

\*pivot creates left and right list with respect to the pivot element

\*puts the elements left and right list recursively with ascending order

\*/

**pivot(\_, [], [], [], \_).**

**pivot(X, [H|T], [H|L], R, W) :- average(H, W, A), A @=< X,**

**pivot(X, T, L, R, W).**

**pivot(X, [H|T], L, [H|R], W) :- average(H, W, A), A @> X,**

**pivot(X, T, L, R, W).**

* My additional predicate to define *order* predicate
* Sorts list according to quick sort algorithm

/\*

\* q\_sort is sorts with divide and conquer strategy

\* H is the head of list, T is the tail of list

\* X is the updated list

\* W is constant, it is week

\* Sorted is the sorted form of the first list

\* average finds the average score A of H

\* pivot creates L1 and L2 with respect to A

\* sorts L1 and L2 recursively

\*/

**q\_sort([], X, \_, X).**

**q\_sort([H|T], X, W, Sorted):- average(H, W, A), pivot(A, T, L1, L2, W),**

**q\_sort(L1, X, W, Sort1), q\_sort(L2, [H|Sort1], W, Sorted).**

* My additional predicate to define *order* predicate

/\*

\* quick\_sort sorts the List

\* List contains all teams

\* W is constant, it is week

\* Sorted is the sorted list of all teams with ascending order

\*/

**quick\_sort([], \_, \_).**

**quick\_sort(List, W, Sorted):- q\_sort(List, [], W, Sorted).**

* *order* predicate I implemented

/\*

\* order is the predicates that gives the league order in that week

\* L is the order of all teams

\* W is constant and it is week

\* findall finds all teams and appends into L1

\* quick\_sort sorts the L1, according to W, and appends L

\*/

**order(L, W):- findall(X, team(X, \_ ), L1), quick\_sort(L1, W, L).**

* My additional predicate to define *topThree* predicate

/\*

\*takes N elements of list

\* N is the number of elements

\* first is the given list

\* it takes element recursively and puts into the second list

\*/

**take(0, \_, []).**

**take(0, [], []).**

**take(N, [H|X], [H|Y]) :- N1 is N-1, take(N1, X, Y).**

* *topThree* predicate I implemented

/\*

\*topThree is the predicate that takes top three elements in order list

\*L is the list of top three teams

\*W is the constant, it is week

\*order finds the list of teams L1 in week W

\*take takes 3 elements in L1 and appends into L

\*/

**topThree(L, W):- order(L1, W), take(3, L1, L).**

Examples of operations;

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

L = [realmadrid, juventus, galatasaray, kobenhavn, manutd, realsociedad, shaktard, bleverkusen, omarseille|...],

N = 12 ;

L = [realmadrid, juventus, galatasaray, kobenhavn, manutd, realsociedad, shaktard, bleverkusen, omarseille|...],

N = 12 ;

L = [realmadrid, juventus, galatasaray, kobenhavn, manutd, realsociedad, shaktard, bleverkusen, omarseille|...],

N = 12 ;

…

**?- allTeams([realmadrid, juventus, galatasaray, kobenhavn, manutd, realsociedad, shaktard, bleverkusen, omarseille|...], N).**

N=12;

**?- wins(galatasaray, 6, L , N).**

L = [kobenhavn, juventus],

N = 2.

**?- losses(Juventus, 3, L ,N).**

L = [realmadrid],

N = 1.

**?- draws(manutd, 4, L, N).**

L = [],

N = 0.

**?- scored(realmadrid, 3, S).**

S = 12.

**?- conceded(realmadrid, 3, S).**

S = 2.

**?- average(realmadrid, 3, S).**

S = 10.

**?- order(L, 4).**

L = [realmadrid, manutd, arsenal, fcnapoli, bdortmund, shaktard, juventus, bleverkusen, galatasaray|...];

false.

**?- topThree(L, 4).**

L = [realmadrid, manutd, arsenal];

false.