Boğaziçi University

Project 1

CMPE260 – Principles of Programming Languages

Fatih iver
4-22-2018

OUTLINE

INTRODUCTION

This project aim is to write a program which infer results from a given knowledge base for queries made by user. The program uses a logic programming language, Prolog.

Prolog is a *declarative* programming language. This means that in prolog, you do not write out what the computer should do line by line, as in *procedural* languages such as C and Java. The general idea behind declarative languages is that you describe a situation. Based on this code, the interpreter or compiler will tell you a solution. In the case of prolog, it will tell you whether a prolog sentence is true or not and, if it contains variables, what the values of the variables need to be.

PROGRAM INTERFACE

This project uses SWI-Prolog. To be able to make queries SWI-Prolog must be installed. After installation, by default, SWI-Prolog is installed as `swipl'. The command line arguments of SWI-Prolog itself and its utility programs are documented using standard Unix **man** pages. SWI-Prolog is normally operated as an interactive application simply by starting the program:

```
$ swipl
Welcome to SWI-Prolog ...
...

1 ?-
```

"?-" means that SWI-Prolog is ready to accept commands. First command should be related to inclusion of a knowledge base which user will going to create queries on it and get results.

Before call a command such that mentioned right above, the knowledge base must be in your current directory so that SWI-Prolog finds it.

After starting Prolog, one normally loads a program into it using consult/1, which may be abbreviated by putting the name of the program file between square brackets. The following goal loads the file likes.pl containing clauses for the predicates likes/2:

```
?- [likes].
true.

?-
```

User also need to add program definition file so that he/she makes program queries to Prolog.

PROGRAM EXECUTION & EXAMPLES

All Teams

Thee predicate allTeams(L, N). L lists containing all the teams in the database where N is the number of elements in the list.

```
?- 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
```

Match Results

Consists of following predicates:
_ wins(T, W, L, N) implies that L involves the teams defeated by team T when we are in week W and N is the number of elements in L.
_ losses(T, W, L, N) implies that L involves the teams that defeated team T when we are in week W and N is the number of elements in L.

_ draws(T, W, L, N) is very similar but now L involves the teams that team T could not defeat also did not lose to.

```
?- 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
```

Goals Scored and Conceded

Predicate scored(T, W, S) where S is the total number of goals scored by team T up to (and including) week W.
Predicate conceded(T, W, C) where C is the total number of goals conceded by team T up to week W.

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

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

Predicate average(T, W, A) where A is the average (goals scored { goals conceded) of a team T gathered up to (and including) week W.

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

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

Order and Top Three

Predicate order(L, W) where W (week) is given as constant and league order in that week will be retrieved in L. Additionally, predicate topThree([T1, T2, T3], W) where T1 T2 and T3 are the top teams when we are in the given week W.

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

?-topThree(L, 6).
L = [realmadrid, manutd, bdortmund]
```

INPUTS AND OUTPUTS

This program infer result of a given query on a knowledge base. This knowledge base should be consist team/2 and match/4 predicates. An example knowledge base:

```
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).
```

The first predicate shows the teams currently existing in the database. For example team(galatasaray, istanbul). means that galatasaray is a team that plays home matches in istanbul.
The second predicate implies that there has been a match between the two given teams in the given week. Score is also given in the predicate. To illustrate, match(2, realmadrid, 4, kobenhavn, 0). implies that \In the 2nd week, realmadrid defeated kobenhavn with the score of 4-0.

PROGRAM STRUCTURE

```
% Calculates sum of elements in a list either integer or real
sum([], 0). % Base Condition
sum([H|T], S):- sum(T, SS), S is H + SS. % Recursive Part

% Check whether an element is in a list
contains([H|_], H). % Base Conditon
contains([_|T], X):- contains(T, X). % Recursive Part

% Check whether list L excludes X
excludes(L, X):- \+ contains(L, X).

% Concatenate two lists
append([],X,X). % Base Condition
append([X|Y],Z,[X|W]) :- append(Y,Z,W). % Recursive Condition

% Find size of a list
size([],0). % Base Condition
size([_|T], S):- size(T, SR), S is SR + 1. % Recursive Condition
```

```
% Find all teams by using built-in function findall
findallteam(L):- findall(Team,team(Team,_),L).

% Get number of teams in the league by calculating size of result of findall
numberofteams(N):- findallteam(L), size(L, N).

% Handmake implementation to find all teams in the league without using built in findall
allteams(L, N):- getallteams(L, [], N).
% allteams(L, N):- findall(Team,team(Team,_),L), length(L, N).

% Get all teams in the league without using built-in findall
getallteams([H|T], I, N):- team(H, _), excludes(I, H), append([H], I, Z), getallteams(T, Z, NS), N
is NS + 1.
getallteams([], I, 0):- numberofteams(N), size(I, S), N =:= S.

% Get total score of a team by calculating its goals in the home and away matches
scored(T, W, S):- scoredHome(T, W, SH), scoredAway(T, W, SA), S is SH + SA.
scoredHome(T, W, SH):- findall(S, (match(UW,T,S,_,_), W>=UW), SL), sum(SL, SH).
scoredAway(T, W, SA):- findall(S, (match(UW,_,_,T,S), W>=UW), SL), sum(SL, SA).

% Get total concedes of a team by calculating its concedes in the home and away matches
conceded(T, W, C):- concededHome(T, W, CH), concededAway(T, W, CA), C is CH + CA.
concededHome(T, W, CH):- findall(C, (match(UW,T,_,_,C), W>=UW), CL), sum(CL, CH).
concededAway(T, W, CA):- findall(C, (match(UW,_,C,T,_), W>=UW), CL), sum(CL, CA).

% Get average of a team by substracting scores from concedes
average(T, W, A):- scored(T, W, S), conceded(T, W, C), A is S - C.

% Get average for the given team, by using previous average function
getAverages([], _, []).
getAverages([H|T], W , [AH|AT]):- average(H, W, AH), getAverages(T, W, AT).

% Get all teams and find their corresponding averages for sorting purposes
getAllAverages(L, W, A):- findallteam(L), getAverages(L, W, A).

% Get list of defeated teams by a team W, by looking its matches in home and away
wins(T, W, L, N):- winsHome(T, W, WH, NH), winsAway(T, W, WA, NA), append(WH, WA, L),
N is NH + NA.
winsHome(T, W, L, N):- findall(AT, (match(UW,T,S,AT,C), W>=UW, S>C), L), size(L, N).
winsAway(T, W, L, N):- findall(HT, (match(UW,HT,C,T,S), W>=UW, S>C), L), size(L, N).

% Get list of teams that defeat a team W, by looking its matches in home and away
losses(T, W, L, N):- lossesHome(T, W, LH, NH), lossesAway(T, W, LA, NA), append(LH, LA, L), N
is NH + NA.
lossesHome(T, W, L, N):- findall(AT, (match(UW,T,S,AT,C), W>=UW, C>S), L), size(L, N).
lossesAway(T, W, L, N):- findall(HT, (match(UW,HT,C,T,S), W>=UW, C>S), L), size(L, N).
```

```
% Get list of teams that draws with a team W, by looking its average
draws(T, W, L, N):- drawsHome(T, W, DH, NH), drawsAway(T, W, DA, NA), append(DH, DA, L),
N is NH + NA.
drawsHome(T, W, L, N):- findall(AT, (match(UW,T,S,AT,C), W>=UW, S=:=C), L), size(L, N).
drawsAway(T, W, L, N):- findall(HT, (match(UW,HT,C,T,S), W>=UW, C=:=S), L), size(L, N).

% Get all averages sorted by using built-in sort and reverse functions
getAveragesSorted(W, AS):- getAllAverages(_, W, A), sort(A, ASR), reverse(ASR, AS).

% Make pairs teams with their averages for sorting purposes
teamAveragePairs(W, P):- getAllAverages(L, W, A), pairs_keys_values(P, A, L).

% Sort pairs by using built-in function keysort and reverse, and pairs_values
order(L, W):- teamAveragePairs(W, P), keysort(P, RL), reverse(RL, IL), pairs_values(IL, L).

% Get top three teams after sorting
topthree([F, S, T], W):- order([F, S, T | _], W).
```

IMPROVEMENTS AND EXTENSIONS

Knowledge base should be taken from a web service by using an application programming interfaces so that user will make queries on up to date information. Even program itself might be used as an application programming interface to serve users who want to benefit from this.

DIFFICULTIES ENCOUNTERED

Prolog is logic programming language. It means that it is a completely different language from languages that we have been seeing up to now. The language itself is simple. It just consists of rules and facts. It is also a tricky language. To implement what you want, you need to use language mechanisms in a tricky and probably in a recursive manner. When these things come together, it gets harder to implement functions in a recursive manner, and writing a base case by making use of language mechanisms.

Another difficulty that I have encountered is that there is no enough documentation even in the official page of the SWI-Prolog. Normally, you search for what you are looking for and it is probably already answered in stackoverflow.com. You scroll down, look at the first answer, copy the code and make some changes, then it is ready for your need. It was not the case in Prolog at least for me. Even I couldn't find enough examples on the official website.

CONCLUSION

Firstly, it was so satisfying to experience a different language. Before starting implementation, I read the documentation line by line, do the given examples. It taught me a lot, but as you know, when it comes to real problem, you really need to involve and you need to solve the given problem. When you are stuck, the solution is not immediately available. In addition to that, if you are JAVA, C++ or a Python person, I will call it like that, your brain tries to think in a way that produce solutions for that language. You force your brain to think in another manner, in a logic manner. After several annoying tries, you see that there is something up. Briefly, it was a great experience to learn and use Prolog for this project.