**Group Activity 5; CS 3060**

**Names** of students in your group: Nicholas W, Dylan S, Ethan F, Nicholas B, Tyson S, Alex L

Points: 10

**Task 1**: (4 points) (a) Say there are four baseball teams (tiger, lion, panther, bear), and each pair of teams play two matches (home and away). Say, as the game scheduler, you need the list of all of these matches. Test the following Prolog code to serve your purpose.

*team(tiger). team(lion). team(panther). team(bear).*

*match(X, Y) :- team(X), team(Y), \+(X=Y).*

Show the result (i.e. list of all matches) in README. How many matches are there?

(b) Now say tournament rules change and you need to schedule ONLY one match between any two teams. Test the following Prolog code to serve your purpose.

*team(tiger). team(lion). team(panther). team(bear).*

*match(X, Y) :- team(X), team(Y), X @< Y.*

Hint: string1 @< string2 compares two strings and gives true/false.

Show the result (i.e. list of all matches) in README. How many matches are there?

(c) Briefly explain why we get all 2-size *permutation*s in version 1 and all 2-size *combination*s in version 2.

In version one we get every variation because were looking for all possible matches when comparing X and Y.

In version 2 we get every combination only once because when we compare X and Y it's only true when X comes before Y alphabetically.

(d) Also, test the following code and show results. What extra does it do? Add to README file.

*team(tiger). team(lion). team(panther). team(bear).*

*match(X, Y) :- team(X), team(Y), dif(X,Y).*

*read\_filename(X) :-*

*write('give a filename'), nl,*

*read(X).*

*writeToFile :-*

*read\_filename(F), open(F, write, Stream),*

*forall(match(X, Y), write(Stream, (X, Y, ' '))),*

*close(Stream).*

**Task 2**: (3 points). Given a bunch of father-child relational facts of a family (over multiple generations), write a prolog rule (*pathFromRootTo(X)*) to print the path of X from the root (i.e. *ggrandpa*) of the family. Test the following code.

*father(ggrandpa, sr). /\* ggrandpa is the father is sr \*/*

*father(sr, jr1). /\* sr is the father is j1r \*/*

*father(sr, jr2). /\* sr is the father is jr2 \*/*

*father(jr1, little). /\* jr1 is the father is little \*/*

*pathFromRootTo(ggrandpa) :- write(ggrandpa).*

*pathFromRootTo(X) :-*

*father(Y, X), pathFromRootTo(Y), write(' -- '), write(X)*.

Test the above code to get the path from root “*ggrandpa”* to “*little”*. Add results to README.

**Task 3**: (1.5 points). Given a list of items, write a prolog rule (*different(List)*) to check whether all items in the list are differrent. Test the following code, which is from lecture ppt3.

*different([]).*

*different([H|T]):-*

*\+member(H,T), different(T).*

Test the above code to check whether [3,4,5,7,4,9] has all different items. Add results to README.

**Task 4**: (1.5 points). Given a list of items, write a prolog rule (*inDomain(List, ValList)*) to check whether all items in List are within the values given in ValList. Test the following code, which is from lecture ppt3.

*inDomain([ ], \_). /\* base case \*/*

*inDomain([H|T], ValList) :-*

*member(H, ValList), inDomain(T, ValList).*

*/\* Note: ValList represents the list of values which are in the domain \*/*

Test the above code to check whether [3,4,5,7,4,9] has all items within the range 1-9. Add results to README.

**Submission**: Submit one copy (per group) via to Canvas.