LIST OF PRACTICALS CORE PAPER XIII: ARTIFICIAL INTELLIGENCE

1. Write a prolog program to calculate the sum of two numbers.
2. Write a Prolog program to implement max(X, Y, M) so that M is the maximum of two numbers X and Y.
3. Write a program in PROLOG to implement factorial (N, F) where F represents the factorial of a number N.
4. Write a program in PROLOG to implement generate_fib(N,T) where T represents the Nth term of the fibonacci series.
5. Write a Prolog program to implement GCD of two numbers.
6. Write a Prolog program to implement power (Num,Pow, Ans) : where Num is raised to the power Pow to get Ans.
7. Prolog program to implement multi (N1, N2, R) : where N1 and N2 denotes the numbers to be multiplied and R represents the result.
8. Write a program in PROLOG to implement towerofhanoi (N) where N represents the number of discs
9. Consider a cyclic directed graph [edge (p, q), edge (q, r), edge (q, r), edge (q, s), edge (s,t)] where edge (A,B) is a predicate indicating directed edge in a graph from a node A to a node B. Write a program to check whether there is a route from one node to another node.
10. Write a Prolog program to implement memb(X, L): to check whether X is a member of L or not.
11. Write a Prolog program to implement conc (L1, L2, L3) where L2 is the list to be appended with L1 to get the resulted list L3.
12. Write a Prolog program to implement reverse (L, R) where List L is original and List R is reversed list.
13. Write a program in PROLOG to implement palindrome (L) which checks whether a list L is a palindrome or not.
14. Write a Prolog program to implement sumlist(L, S) so that S is the sum of a given list L.
15. Write a Prolog program to implement two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively
16. Write a Prolog program to implement nth_element (N, L, X) where N is the desired position, L is a list and X represents the Nth element of L.
17. Write a program in PROLOG to implement remove_dup (L, R) where L denotes the list with some duplicates and the list R denotes the list with duplicates removed.
18. Write a Prolog program to implement maxlist(L, M) so that M is the maximum number in the list
19. Write a prolog program to implement insert_nth(I, N, L, R) that inserts an item I into Nth position of list L to generate a list R.
20. Write a Program in PROLOG to implement sublist(S, L) that checks whether the list S is the sublist of list L or not. (Check for sequence or the part in the same order).
21. Write a Prolog program to implement delete_nth (N, L, R) that removes the element on Nth position from a list L to generate a list R.
22. Write a program in PROLOG to implement delete_all (X, L, R) where X denotes the element whose all occurrences has to be deleted from list L to obtain list R.

23. Write a program in PROLOG to implement merge (L1, L2, L3) where L1 is first ordered list and L2 is second ordered list and L3 represents the merged list.
24. Write a PROLOG program that will take grammar rules in the following format:

    NT → (NT | T)*

    Where NT is any nonterminal, T is any terminal and Kleene star (*) signifies any number of repetitions, and generate the corresponding top-down parser, that is:

    sentence → noun-phrase, verb-phrase

    determiner → [the]

    will generate the following:

    sentence (I, O) :- noun-phrase(I,R), verb-phrase (R,O).

    determiner ([the|X], X) :- !.
25. Write a prolog program that implements Semantic Networks (ATN/RTN).

```
/************************************
**************** Q1 ****************
************************************/


sum(X,Y,Z) :-  Z is X+Y.
```

```
/************************************
**************** Q2 ****************
************************************/


max(X,Y,Z) :-  (X>Y -> Z is X; Z is Y).
```

```
/************************************
**************** Q3 ****************
************************************/


fac(0,1).
fac(N,X) :- N > 0, M is N - 1, fac(M,Y), X is Y * N.
```

```
/************************************
**************** Q4 ****************
************************************/



fib(0,0) :- !.
fib(1,1) :- !.
fib(N,T) :-
    N > 1,
    N1 is N-1,
    N2 is N-2,
    fib(N1, T1),
    fib(N2, T2),
    T is T1+T2.
```

```
/************************************
**************** Q5 ****************
************************************/


gcd(0, X, X):- X > 0, !.
gcd(X, Y, Z):- X >= Y, X1 is X-Y, gcd(X1,Y,Z).
gcd(X, Y, Z):- X < Y, X1 is Y-X, gcd(X1,X,Z).
```

```
/************************************
**************** Q6 ****************
************************************/



mulit(K, L, M, S) :- (
            L>1 -> S1 is K*S,
            L1 is L-1,
            mulit(K,L1,M,S1);
                  (
                  L=:=0 -> M is 1;
                  M is S1
                  )
            ).

power(Num, Pow, Ans) :-
            Store is Num,
            mulit(Num, Pow, Ans, Store).
```

```
/************************************
**************** Q7 ****************
************************************/


multi(N1,N2,R) :- R is N1*N2.
```

```
/************************************
**************** Q8 ****************
************************************/



move(1,X,Y,_):- write('Move disk from '),
            write(X),write(' to '),
            write(Y),nl.

move(N,X,Y,Z):- N>1,M is N-1,
            move(M,X,Z,Y),
            move(1,X,Y,_),
            move(M,Z,Y,X).

tower_of_hanoi(N) :- move(N,left,center,right).
```

```
/************************************
**************** Q9 ****************
************************************/



path(A,B) :- walk(A,B,[]).

walk(A,B,V) :-
      edge(A,X),
      not(member(X,V)),
      (
            B = X;
            walk(X,B,[A|V])
      ).

edge(p,'q').
edge(q,r).
edge(q,'s').
edge(s,t).
```

```
/************************************
**************** Q10 ****************
************************************/


memb(X,[H|L]) :- (X=:=H -> write('Is a Member');memb(X,L)).
```

```
/************************************
**************** Q11 ****************
************************************/



conc([],L,L).
conc(L,[],L).
conc([H|T],L2,[H|L3]) :-  conc(T, L2, L3).
conc(X, Y, L3):- conc([X], Y, L3).
conc(X, Y, L3):- conc([X], [Y], L3).
```

```
/************************************
**************** Q12 ****************
************************************/



conc([],L,L).
conc(L,[],L).
conc([H|T],L2,[H|L3]) :-  conc(T, L2, L3).
reverse([],[]).
reverse([H|T],R):- reverse(T,RevT), conc(RevT,[H],R).
```

```
/************************************
**************** Q13 ****************
************************************/



conc([],L,L).
conc(L,[],L).
conc([H|T],L2,[H|L3]) :-  conc(T, L2, L3).

palindrome([]):- write('palindrome').

palindrome([_]):- write('palindrome').

palindrome(L):- conc([H|T], [H], L),
            palindrome(T);
            write('Not a palindrome').
```

```
/*************************************
**************** Q14 *****************
*************************************/



add_el([X],Y,Y2) :- Y is Y2+X.
add_el([H|Tail],Y, Y2) :- Y3 is Y2+H, add_el(Tail,Y,Y3).
sumList([H|Tail],Y):- Y2 is H, add_el(Tail,Y,Y2).
```

```
/************************************
**************** Q15 ****************
************************************/



check_even(L2) :-       L3 is integer(L2/2),
                  L4 is L3*2,
                  L5 is L2-L4,
                  (
                        L5 =:= 0, true;
                        false
                  ).

check_odd(L2) :-  L3 is integer(L2/2),
                  L4 is L3*2,
                  L5 is L2-L4,
                  (
                        L5 =:= 1, true;
                        false
                  ).

check_len([H|T], L, R1) :- L1 is L+1,
                     check_len(T, L1, R1).

check_len([Y], L, R1) :-   (
                        R1 =:= 0 -> L2 is L+1,
                        check_even(L2);
                        L2 is L+1, check_odd(L2)
                     ).

evenlength(X) :- check_len(X, 0, 0).

oddlength(X) :- check_len(X, 0, 1).
```

```
/************************************
**************** Q16 ****************
************************************/



find_f([H|Tail], N, Y, L):- (
                        N=:=L -> Y is H;
                        L1 is L+1,
                        find_f(Tail, N, Y, L1)
                      ).

nth_element(N,L,X) :- find_f(L,N,X, 1).
```

```
/************************************
**************** Q17 ****************
************************************/



conc([],L,L).
conc(L,[],L).
conc([H|T],L2,[H|L3]) :-  conc(T, L2, L3).

find_f([H|Tail], R9, R, K):- (
                        K=:=0 -> conc([H],[],R),
                        find_f(Tail, R9, R);
                        find_f(Tail, R9, R)
                       ).

find_f([H|Tail], R9, R4):-   (
                        member(H,R4) -> find_f(Tail, R9, R4);
                        conc(R4, [], R2),
                        conc(R2,[H], R3),
                        find_f(Tail, R9, R3)
                       ).

find_f([X], R, R9):-         (
                        member(X,R9) -> conc(R9, [], R);
                        conc(R9, [X], R)
                       ).

dup(L,R) :- find_f(L, R, R2, 0).
```

```
/*************************************
**************** Q18 ****************
*************************************/



max_f([H|Tail], Y, Y2):-  (
                        Y2<H -> Y3 is H,
                        max_f(Tail, Y, Y3);
                        Y3 is Y2,
                        max_f(Tail, Y, Y3)
                    ).

max_f([X],Y,Y2) :-        (
                        Y2<X -> Y is X;
                        Y is Y2
                    ).

max_el([H|Tail],Y, Y2) :- (
                        Y2<H -> Y3 is H,
                        max_f(Tail, Y, Y3);
                        Y3 is Y2,
                        max_f(Tail, Y, Y3)
                    ).

max([H|Tail],Y):-       Y2 is H,
                   max_el(Tail,Y,Y2).
```

```
/************************************
**************** Q19 ****************
************************************/


% insert_nth(I,N,L,R)

insert_nth(I,N,[H|L],[H|R]):- N > 1, !,
                                N1 is N - 1, insert_nth(I,N1,L,R).
insert_nth(I,  1, L, [I|L]).
```

```
/************************************
**************** Q20 ****************
************************************/



it_it([H|Tail], [S|Tail2]):- (H=:=S -> it_it(Tail, Tail2);false).
it_it([X], [S|Tail2]):- false.
it_it([X],[S]):- ([X]=:=[S] -> true;false).
it_it([H|Tail],[S]):- (H=:=[S] -> true;false).
find_f([H|Tail], [S|Tail2]):- (
                        H=:=S -> it_it(Tail, Tail2);
                        find_f(Tail, [S|Tail2])
                      ).
find_f([X], [S|Tail2]):- false.
find_f([X], [Tail2]):- ([X]=:=[Tail2] -> true; false).
sublist(S,L) :- find_f(L,S).
```