1.

Write a predicate to generate the list of all subsets with all elements of a given list.

Eg: [2, 3, 4] N=2 => [[2,3],[2,4],[3,4]]

combinations(l1...ln, k) = [], k = 0

l1 + combinations(l2...ln, k - 1), k > 0

combinations(l2...ln, k), k > 0

combinations(\_, 0, []).

combinations([H|T], K, [H|R]) :-

K > 0,

NK is K - 1,

combinations(T, NK, R).

combinations([\_|T], K, R) :-

K > 0,

combinations(T, K, R).

combinations(L:list, K:number, R:list)

combinations(i, o)

allsolutions(L, N, R) :-

findall(RPartial, combinations(L, N, RPartial), R).

allsolutions(L:list, N:number, R:list)

allsolutions(i, i, o)

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2.

Are given n points in a plan (represented using its coordinates).

Write a predicate to determine all subsets of collinear points.

coliniare(x1, y1, x2, y2, z1, z2) = True, x1 == x2 == x3

= True, y1 == y2 == y3

= True, all different & (y2-y1) / (x2-x1) == (y3-y2) /(x3-x2)

collinearPoints(X, \_, X, \_, X, \_).

collinearPoints(\_, Y, \_, Y, \_, Y).

collinearPoints(X1, Y1, X2, Y2, X3, Y3):-

X1 =\= X2, X2 =\= X3, X3 =\= X1,

Y1 =\= Y2, Y2 =\= Y3, Y3 =\= Y1,

P1 is (Y2 - Y1) / (X2 - X1),

P2 is (Y3 - Y2) / (X3 - X2),

P1 =:= P2.

coliniare(x1:number, y1:number, x2:number, y2:number, x3:number, y3:number)

coliniare(i,i,i,i)

subsets([],[]).

subsets([[A1,B1],[A2,B2],[A3,B3]|T], [[[A1,B1],[A2,B2],[A3,B3]]|R]):-

collinearPoints(A1,B1,A2,B2,A3,B3), !,

subsets([[A2,B2], [A3,B3]|T], R).

subsets([\_|T], R):-

subsets(T, R).

subsets(L - list, R - list)

subsets(i, o)

allsolutions(L, R) :-

findall(RPartial, subsets(L, RPartial), R).

allsolutions(L:list, R:list)

allsolutions(i, o)

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.

Write a predicate to determine all decomposition of n (n given, positive), as sum of consecutive natural numbers.

decomposition(n, e) = [], n = 0

e + decomposition(n - e, e + 1), otherwise

decomposition(0, \_, []).

decomposition(N, E, [E|R]):-

N >= E,

NN is N - E,

NE is E + 1,

decomposition(NN, NE, R).

decomposition(N:number, E:number, R:list)

decomposition(i, i, o)

oneSolution(n, e) = decomposition(n, e), e < n

oneSolution(n, e + 1), e < n

oneSolution(N, E, R) :-

E < N,

decomposition(N, E, R).

oneSolution(N, E, R) :-

E < N,

NE is E + 1,

oneSolution(N, NE, R).

oneSolution(N:number, E:number, R:list)

oneSolution(i, i, o)

allsolutions(N, R) :-

findall(RPartial, oneSolution(N, 1, RPartial), R).

allsolutions(N:number, R:list)

allsolutions(i, o)

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4.

The list a1...an is given. Write a predicate to determine all sublists strictly ascending of this list a.

subset(l1..ln) = [], n = 0

l1 + subset(l2..ln)

subset(l2..ln)

subset([], []).

subset([H|T], [H|R]):-

subset(T, R).

subset([\_|T], R):-

subset(T, R).

subset(L:list, R:list)

subset(i, o)

strictlyAsc(l1..ln) = True, n = 0 || n = 1

False, l1 >= l2

strictlyAsc(l2..ln), l2 > l1

strictlyAsc([]).

strictlyAsc([\_]).

strictlyAsc([H1, H2|T]):-

H1 < H2,

strictlyAsc([H2|T]).

strictlyAsc(L:list)

strictlyAsc(i)

oneSol(l1..ln) = [], n = 0

oneSol(l1..ln), ascending(subset(l1..ln))

oneSol([], []).

oneSol(L, LSUBS):-

subset(L, LSUBS),

strictlyAsc(LSUBS).

oneSol(L:list, R:list)

allsolutions(L, R) :-

findall(RPartial, oneSol(L, RPartial), R).

allsolutions(N:number, R:list)

allsolutions(i, o)

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5.

Two integers, n and m are given. Write a predicate to determine all possible sequences of numbers from 1 to n, such that between any two numbers from consecutive positions, the absolute difference to be >= m.

diff(a, b) = b - a, a < b

a - b, a > b

difference(A, B, R):-

A < B,

R is B - A.

difference(A, B, R):-

A > B,

R is A - B.

diff(A:number, B:number, R:number)

diff(i, i, o)

sequence(n, i) = [], i = n + 1

i + sequence(n, i + 1), i <= n

sequence(n, i + 1), i <= n

sequence(N, I, []):-

I =:= N + 1.

sequence(N, I, [I|R]):-

I =< N,

I1 is I + 1,

sequence(N, I1, R).

sequence(N, I, R):-

I =< N,

I1 is I + 1,

sequence(N, I1, R).

sequence(N:number, I:number, R:list)

sequence(i, i, o)

checkProperty(l1...ln, m) = true, diff(l1, l2) >= m and n = 2

checkProperty(l2...ln, m), diff(l1, l2) >= m and n > 2

false, otherwise

checkProperty([H1, H2], M):-

difference(H1, H2, DIF),

DIF >= M.

checkProperty([H1, H2|T], M):-

difference(H1, H2, DIF),

DIF >= M,

checkProperty([H2|T], M).

check(L:list, M:mumber)

check(i, i)

onesolution(N, M, R) :-

sequence(N, 1, R),

checkProperty(R, M).

onesolution(N:number, M:number, R:list)

onesolution(i, i, o)

allsolutions(N, M, R) :-

findall(RPartial, onesolution(N, M, RPartial), R).

allsolutions(N:number, M:number R:list)

allsolutions(i, i, o)

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.

Generate the list of all arrangements of K elements of a given list.

Eg: [2, 3, 4] K=2 => [[2,3], [3,2], [2,4], [4,2], [3,4], [4,3]] (not necessary in this order)

insert an element in a list

insert(E, L, [E|L]).

insert(E, [H|T], [H|R]):-

insert(E, T, R).

arrangements of K elements from a list L

arrangements([E|\_], 1, [E]).

arrangements([\_|T], K, R):-

arrangements(T, K, R).

arrangements([H|T], K, R1):-

K > 1,

K1 is K-1,

arrangements(T, K1, R),

insert(H, R, R1).

find all solutions

allsolutions(L, N, R) :-

findall(RPartial, arrangements(L, N, RPartial), R).

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

7.

A player wants to choose the predictions for 4 games.

The predictions can be 1, X, 2.

Write a predicate to generate all possible variants considering that:

last prediction can’t be 2 and no more than two possible predictions X.

candidate(1).

candidate(3). % X = 3

candidate(2).

all(i, c, p) = c, i = 4

all(i + 1, candidat(x) + c, p \* x), p \* x % 27 = 0

all(4, R, R, \_):- !.

all(I, C, R, P):-

candidate(X),

I1 is I + 1,

P1 is P \* X,

P1 mod 27 =\= 0,

all(I1, [X|C], R, P1).

all(N:number, C:list, R:list, P:number)

all(i, i, o, i)

result(R):-

all(1, [1], R, 1).

result(R):-

all(1, [3], R, 1).

rez(R:list)

rez(o)

allsolutions(R) :-

findall(RPartial, result(RPartial), R).

allsolutions(R:list)

allsolutions(o)

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

8.

Generate all strings of n parenthesis correctly closed.

Eg: n=4 => (()) and () ()

parenthesis('(').

parenthesis(')').

check(l1...ln, c) = check(l2...ln, c + 1), l1 = '('

check(l2...ln, c - 1), l1 = ')' and c > 0

check([], 0).

check([H|T], C):-

H == '(',

C1 is C + 1,

check(T, C1).

check([H|T], C):-

H == ')',

C > 0,

C1 is C - 1,

check(T, C1).

check(L:list, C:number)

check(i, i)

allcombinations(n, i, c) = c, i = n

allcombinations(n, i + 1, paranteza(x) + c), otherwise

allcombinations(N, N, C, C):- !.

allcombinations(N, I, C, R):-

parenthesis(X),

I1 is I + 1,

allcombinations(N, I1, [X|C], R).

allcombinations(N:number, I:number, C:list, R:list)

allcombinations(i, i, i, o)

onesolution(N, R) :-

allcombinations(N, 0, [], R),

check(R, 0).

onesolution(N:number, R:list)

onesolution(i, o)

allsolutions(N, R) :-

findall(RPartial, onesolution(N, RPartial), R).

allsolutions(N:number, R:list)

allsolutions(i, o)

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

9.

Generate all permutation of N (N - given) respecting the property:

for every 2<=i<=n exists an 1<=j<=i, so |v(i)-v(j)|=1.

insert

inserare([], E, [E]).

inserare([H|T], E, [E,H|T]).

inserare([H|T], E, [H|R]) :-

inserare(T, E, R).

permutations

permutari([], []).

permutari([H|T], R) :-

permutari(T, RP),

inserare(RP, H, R).

create a list

createList(0, []).

createList(N, [N|R]) :-

N > 0,

NN is N - 1,

createList(NN, R).

difference of 2 numbers

diff(A, B, R) :-

A > B,

R is A - B.

diff(A, B, R) :-

A =< B,

R is B - A.

check if the property holds throughout a list for a given value

check(l1...ln, e) = true, n = 0

true, diff(l1, e) = 1

check(l2...ln, e), n > 0

false, otherwise

check([], \_).

check([H|\_], X) :-

diff(X, H, R),

R =:= 1, !.

check([\_|T], X) :-

check(T, X).

check if the property holds all values

checkPerm([], \_).

checkPerm([H|T], L) :-

check(L, H),

checkPerm(T, [H|L]).

onesolution(L, R) :-

permutari(L, R),

checkPerm(R, []).

allsolutions(N, R) :-

createList(N, RL),

findall(RP, onesolution(RL, RP), R).

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

10.

For a list a1... an with integer and distinct numbers,

define a predicate to determine all subsets with sum of elements divisible with n.

subset(l1..ln) = [], n = 0

l1 + subset(l2..ln)

subset(l2..ln)

subset([], []).

subset([H|T], [H|R]):-

subset(T, R).

subset([\_|T], R):-

subset(T, R).

sum(l1..ln) = 0, n = 0

l1 + sum(l2..ln), otherwise

sum([], 0).

sum([H|T], S):-

sum(T, S2),

S is S2 + H.

onesolution(L, N, R) :-

subset(L, R),

sum(R, SUM),

SUM mod N =:= 0.

allsolutions(L, N, R) :-

findall(RPartial, onesolution(L, N, RPartial), R).

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

13.

The list a1, ..., an is given and it consists of distinct integers.

Write a predicate to determine all subsets with aspect of "mountain"

(a set has a "mountain" aspect if the elements increase to a certain point and then decrease).

Model matematic:

mountain(l1...ln, f) = true, n = 1 and f = 1

mountain(l2...ln, 0), l1 > l2

mountain(l2...ln, 1), l1 < l2

false, otherwise

mountain([\_], 1).

mountain([H1, H2|T], 0):-

H1 < H2,

mountain([H2|T], 0).

mountain([H1, H2|T], \_):-

H1 > H2,

mountain([H2|T], 1).

check if it is a mountain

isMountain([H1, H2|T]) :-

H1 < H2,

mountain([H1, H2|T] ,0).

subset(l1..ln) = [], n = 0

l1 + subset(l2..ln)

subset(l2..ln)

subset([], []).

subset([H|T], [H|R]):-

subset(T, R).

subset([\_|T], R):-

subset(T, R).

onesolution(L, R) :-

subset(L, R),

isMountain(R).

allsolutions(L, R) :-

findall(RPartial, onesolution(L, RPartial), R).

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

14.

Write a program to generate the list of all subsets of sum S with the elements of a list (S - given).

Eg: [1,2,3,4,5,6,10] si S=10 => [[1,2,3,4], [1,4,5], [2,3,5], [4,6], [10]] (not necessary in this order)

subset(l1..ln) = [], n = 0

l1 + subset(l2..ln)

subset(l2..ln)

subset([], []).

subset([H|T], [H|R]):-

subset(T, R).

subset([\_|T], R):-

subset(T, R).

sum(l1..ln) = 0, n = 0

l1 + sum(l2..ln), otherwise

sum([], 0).

sum([H|T], S):-

sum(T, S2),

S is S2 + H.

onesolution(L, N, R) :-

subset(L, R),

sum(R, SUM),

SUM =:= N.

allsolutions(L, N, R) :-

findall(RPartial, onesolution(L, N, RPartial), R).

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

15.

For a given n, positive, determine all decomposition of n as a sum of consecutive natural numbers.

decomposition(n, e) = [], n = 0

e + decomposition(n - e, e + 1), otherwise

decomposition(0, \_, []).

decomposition(N, E, [E|R]):-

N >= E,

NN is N - E,

NE is E + 1,

decomposition(NN, NE, R).

oneSolution(n, e) = decomposition(n, e), e < n

oneSolution(n, e + 1), e < n

oneSolution(N, E, R) :-

E < N,

decomposition(N, E, R).

oneSolution(N, E, R) :-

E < N,

NE is E + 1,

oneSolution(N, NE, R).

allsolutions(N, R) :-

findall(RPartial, oneSolution(N, 1, RPartial), R).