**C.** Write a PROLOG program that generates the list of all subsets with at least N elements such that the value of sum of all elements from each subset is divisible with 3, from a list of integers. Write the mathematical models and flow models for the predicates used. For example, for the list

L=[2,3,4] and **N**=1 ⇒ [[3],[2,4],[2,3,4]] (not necessarily in this order).

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% Mathematical model

% my\_len(l1l2..ln) =

% { 0, if n=0

% { 1 + my\_len(l2...ln) , otherwise

% my\_len(L: list, R: Integer)

% flow model: my\_len(i, o)

my\_len([], 0).

my\_len([\_|T], NewR):-

my\_len(T, R),

NewR is R + 1.

% oneSolution(l1l2..ln,N) =

% { S = subsets(l1l2..ln), where S=S1S2..SK, if K>=N and sum(S)%3=0

% onesolution(L:list, N: Integer, R: list)

% flow model: onesolution(i,i,o)

oneSolution(L, N, R):-

subsets(L, R),

my\_len(R, LengthList),

LengthList >= N,

sum(R, Sum),

Sum mod 3 =:= 0.

% allSolutions(l1l2...ln, N) =

% { findall(oneSolution(l1l2..ln, N))

% allSolutions(L: int input list, N: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(L, N, R):-

findall(X, oneSolution(L, N, X), R).

**C.** Given a list made of integer numbers, generate using PROLOG the list of arrangements with even number of elements, having the sum an odd number. Write mathematical models and flow models for the predicates used. For example, for the list L=[2,3,4] ⇒ [[2,3],[3,2],[3,4],[4,3]] (not necessarily in this order).

% Mathematical model

% Insert an element on every position of a list

% insertOnPos(l1l2..ln, elem) =

% { elem U l1l2..ln

% { l1 U insert(l2...ln, elem)

% insert(E: element, L:List, LRez:list)

% (i, i, o) - nedeterminist

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

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

insertOnPos(T, E, R).

%subsets(l1l2..ln) =

% { [], if n=0

% { insertOnPos(subsets(l2l3..ln), l1), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R1),

insertOnPos(R1,H,R).

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

subsets(T, R).

% Mathematical model

% len(l1l2..ln) =

% { 0, if n=0

% { 1 + length(l2...ln) , otherwise

% len(L: list, R: Integer)

% flow model: len(i, o)

len([], 0).

len([\_|T], NewR):-

len(T, R),

NewR is R + 1.

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

% oneSolution(l1l2..ln) =

% { S, where S = S1S2...Sm = subsets(l1l2..ln), if m%2=0 and sum(S)%2=1.

% oneSolution(L: List, R: List)

% (i, o)

oneSolution(L, R):-

subsets(L, R),

len(R, LengthR),

LengthR mod 2 =:=0,

sum(R, Sum),

Sum mod 2 =:=1.

% allSolutions(l1l2...ln, N) =

% { findall(oneSolution(l1l2..ln))

% allSolutions(L: int input list, R: out list)

% flow model: allSolutions(i, o)

allSolutions(L, R):-

findall(X, oneSolution(L, X), R).

**C.** Write a PROLGO program that generates the list of arrangements of k elements from a list of integer numbers, having the given product P. Write the mathematical models and flow models for the predicates used. For example, for the list [2, 5, 3, 4, 10], k=2 and P=20 ⇒ [[2,10],[10,2],[5,4],[4,5]] (not necessarily in this order).

% Mathematical model

% Insert an element on every position of a list

% insertOnPos(l1l2..ln, elem) =

% { elem U l1l2..ln

% { l1 U insert(l2...ln, elem)

% insert(E: element, L:List, LRez:list)

% (i, i, o) – nedeterminist

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

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

insertOnPos(T, E, R).

%Mathematical model

% arr(l1l2...ln, K) =

% { l1, if k = 1

% { arr(l2...ln, k), if k >= 1

% { insertOnPos(l1, arr(l2...ln, k-1)), if k>1

% arr(L: list, K: integer, R: list)

% arr(i,i,o)

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

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

arr(T, K, R).

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

K>1,

K1 is K-1,

arr(T, K1, R),

insertOnPos(R, H, R1).

% Mathematical model

% product(l1l2..ln) =

% { 1, if n = 0

% { l1 \* product(l2...ln), otherwise

% product(L: list, R: Integer). flow: (i, o)

product([], 1).

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

product(T, R1),

R is H\*R1.

% Mathematical model

% oneSolution(l1l2...ln, K, P) =

% { A, where A = a1a2...am = arr(l1l2..ln, k), if product(A) = P.

% oneSolution(L: List, K: Integer, P: Integer, R: List)

% (i, i, i, o)

oneSolution(L, K, P, R):-

arr(L, K, R),

product(R, AranjP),

AranjP =:= P.

% allSolutions(l1l2...ln, K, P) =

% { findall(oneSolution(l1l2..ln, K, P))

% allSolutions(L: int input list, K: Integer, P: Integer, R: out list)

% flow model: allSolutions% (i, i, i, o)

allSolutions(L, K, P, R):-

findall(X, oneSolution(L, K, P, X), R).

**C.** Write a PROLOG program that generates the list of permutations of the set 1..N, having the property that the absolute value of the difference between 2 consecutive values from the permutation is >=2. Write the mathematical models and flow models for the predicates used. For example, for N=4 ⇒ [[3,1,4,2], [2,4,1,3]] (not necessarily in this order).

% Mathematical model

% Insert an element on every position of a list

% insert(elem, l1l2..ln) =

% { elem U l1l2..ln

% { l1 U insert(elem, l2...ln)

% insert(E: element, L:List, LRez:list)

% (i, i, o) - nedeterminist

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

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

% Mathematical model

% perm(l1l2...ln) =

% { [], if n = 0

% { insert(l1, perm(l2...ln)), otherwise.

% insert(L:List, P:list)

% (i, i, o)

perm([], []).

perm([H|T], P) :-

perm(T, L),

insert(H, L, P).

% Create a ranged list (from k to n)

% range(k, n) =

% [k], k = n.

% k U range(k+1, n), k < n

% range(K: Number, N:number, R:list)

% range(i, i, o)

range(N, N, [N]):-!.

range(K, N, [K|R]) :-

NewK is K + 1,

range(NewK, N, R).

% Create list from 1 to n

% createList(N) = range(1, N) if N > 0

% createList(N: Number, R:List) (i, 0)

createList(N, R):-

N > 0,

range(1, N, R).

% diffAbs(a, b) =

% a - b, a > b

% b - a, a <= b

% diffAbs(A:number, B:number, R:number)

% diffAbs(i, i, o)

diffAbs(A, B, R) :-

A > B, !,

R is A - B.

diffAbs(A, B, R) :-

R is B - A.

% check(l1...ln) =

% true, n = 1

% check(l2...ln), diff(l1, l2) >= 2

% false, otherwise

% check(L:list)

% check(i)

check([\_]):- !.

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

diffAbs(H1, H2, Diff),

Diff >= 2,

check([H2|T]).

% Mathematical model

% oneSolution(l1l2...ln, K, P) =

% { A, where A = a1a2...am = aranj(l1l2..ln, k), if product(A) = P.

% oneSolution(L: List, K: Integer, P: Integer, R: List)

% (i, i, i, o)

oneSolution(N, PermR):-

createList(N, R),

perm(R, PermR),

check(PermR).

% allSolutions(l1l2...ln, K, P) =

% { findall(oneSolution(l1l2..ln))

% allSolutions(L: int input list, K: Integer, P: Integer, R: out list)

% flow model: allSolutions% (i, i, i, o)

allSolutions(N, R):-

findall(X, oneSolution(N, X), R).

**C.** Given a list composed of integer numbers, generate in PROLOG the list of arrangements of N elements ending with an odd value and have the sum S given. Write the mathematical models and flow models for the predicates used. For example, for the list L=[2,7,4,5,3], N=2 and S=7 ⇒ [[2,5], [4,3]] (not necessarily in this order).

% Mathematical model

% Insert an element on every position of a list

% insertOnPos(l1l2..ln, elem) =

% { elem U l1l2..ln

% { l1 U insert(l2...ln, elem)

% insert(E: element, L:List, LRez:list)

% (i, i, o) – nedeterminist

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

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

insertOnPos(T, E, R).

%Mathematical model

% arr(l1l2...ln, K) =

% { l1, if k = 1

% { arr(l2...ln, k), if k >= 1

% { insertOnPos(l1, arr(l2...ln, k-1)), if k>1

% arr(L: list, K: integer, R: list)

% arr(i,i,o)

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

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

arr(T, K, R).

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

K>1,

K1 is K-1,

arr(T, K1, R),

insertOnPos(R, H, R1).

% sum(l1l2..ln)= {0, n=0

% {l1 , n=1

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

% lastValue(l1l2...ln)=

% { false, n = 0

% { l1, n=1

% { lastValue(l2...ln), otherwise

% lastValue(L: List, R: Integer), (i, o)

lastValue([H], H):-!.

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

lastValue(T, R).

% Mathematical model

% oneSolution(l1l2...ln, K, S) =

% { A, where A = a1a2...am = aranj(l1l2..ln, k), if sum(A) = S and lastValue(A) % 2 = 1.

% oneSolution(L: List, K: Integer, S: Integer, R: List)

% (i, i, i, o)

oneSolution(L, K, S, R):-

arr(L, K, R),

sum(R, Sum),

Sum =:= S,

lastValue(R, Val),

Val mod 2 =:= 1.

% allSolutions(l1l2...ln, K, P) =

% { findall(oneSolution(l1l2..ln))

% allSolutions(L: int input list, K: Integer, S: Integer, R: out list)

% flow model: allSolutions(i, i, i, o)

allSolutions(L, K, S, R):-

findall(X, oneSolution(L, K, S, X), R).

**C.** Write a PROLOG program that generates the list of all subsets with values between [a, b] interval with an even numbers of even values and an odd number of odd values from each subset. Write the mathematical models and flow models for the predicates used. For example, for **a**=2 and **b**=4 ⇒ [[2,3,4]].

% Create a ranged list (from A to B)

% range(a, b) =

% [b], a = b.

% a U range(a+1, b), a < b

% range(K: Number, N:number, R:list)

% range(i, i, o)

range(B, B, [B]):- !.

range(A, B, [A|R]) :-

NewA is A + 1,

range(NewA, B, R).

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% Mathematical model

% evenNumbers(l1l2...ln) =

% { 0, if n = 0

% { 1 + evenNumbers(l2...ln), if l1 % 2 == 0

% { evenNumbers(l2...ln), otherwise

evenNumbers([], 0):- !.

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

H mod 2 =:= 0, !,

evenNumbers(T, R1),

R is R1+1.

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

evenNumbers(T, R).

% Mathematical model

% oddNumbers(l1l2...ln) =

% { 0, if n = 0

% { 1 + oddNumbers(l2...ln), if l1 % 2 == 1

% { oddNumbers(l2...ln), otherwise

oddNumbers([], 0):- !.

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

H mod 2 =:= 1, !,

oddNumbers(T, R1),

R is R1+1.

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

oddNumbers(T, R).

% Mathematical model

% oneSolution(a, b) =

% { S, where S = s1s2...sm = subsets(range(a, b)), if evenNumbers(S) % 2 = 0 and oddNumbers(S) % 2 = 1.

% oneSolution(A: Integer, B: Integer, R: List)

% (i, i, o)

oneSolution(A, B, R):-

range(A, B, Range),

subsets(Range, R),

evenNumbers(R, Even),

oddNumbers(R, Odd),

Even mod 2 =:= 0,

Odd mod 2 =:= 1.

% allSolutions(A, B, R) =

% { findall(oneSolution(A, B))

% allSolutions(A: Integer, B: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(A, B, R):-

findall(X, oneSolution(A, B, X), R).

**C.** Write a PROLOG program that generates the list of all subsets of k elements (all elements being odd numbers) in arithmetic progression. Write the mathematical models and flow models for the predicates used. For example, for L=[1,5,2,9,3] and k=3 ⇒ [[1,5,9],[1,3,5]] (not necessarily in this order).

% Mathematical model

% Insert an element on every position of a list

% insertOnPos(l1l2..ln, elem) =

% { elem U l1l2..ln

% { l1 U insert(l2...ln, elem)

% insert(E: element, L:List, LRez:list)

% (i, i, o) – nedeterminist

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

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

insertOnPos(T, E, R).

%Mathematical model

% arr(l1l2...ln, K) =

% { l1, if k = 1

% { arr(l2...ln, k), if k >= 1

% { insertOnPos(l1, arr(l2...ln, k-1)), if k>1

% arr(L: list, K: integer, R: list)

% arr(i,i,o)

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

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

arr(T, K, R).

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

K>1,

K1 is K-1,

arr(T, K1, R),

insertOnPos(R, H, R1).

%checkIfOdd(l1l2..ln)=

% { true, n=0

% { false, l1%2==0

% { checkIfOdd(l2l3..ln), otherwise

checkIfOdd([]).

checkIfOdd([H|T]):-

H mod 2 =:= 1,

checkIfOdd(T).

% arithmeticProgression(l1l2..ln,diff) =

% { true, n=2 and l2-l1=diff

% { false, if l2-l1 !=diff

% { arithmeticProgression(l2..ln,diff), otherwise

% arithmeticProgression(L: list, diff: integer)

% arithmeticProgression(i,i)

arithmeticProgression([H1,H2],D):-

Diff is H2-H1,

Diff =:=D,!.

arithmeticProgression([H1,H2|T],D):-

Diff is H2-H1,

Diff =:=D,

arithmeticProgression([H2|T],D).

% mainAP(l1l2..ln) = arithmeticProgression(l1l2..ln,l2-l1)

% mainAP(L: list) ... flow: (i)

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

Diff is H2-H1,

arithmeticProgression([H1,H2|T],Diff).

% oneSolution(l1l2...ln, K) =

% { A, where A = a1a2...am = arr(l1l2..ln, k), if checkIfOdd(A) = true and mainAP(A) = true.

% oneSolution(L: List, K: Integer, P: Integer, R: List)

% (i, i, i, o)

oneSolution(L,K,R):-

arr(L,K,R),

checkIfOdd(R),

mainAP(R).

% allSolutions(l1l2...ln, N) =

% { findall(oneSolution(l1l2..ln, N))

% allSolutions(L: int input list, N: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(L, K, R):-

findall(X, oneSolution(L, K, X), R).

**C.** Write a PROLOG program that generates the list of all combinations of k elements with numbers from 1 to N, with the property that difference between two consecutive numbers from a combination has an even value. Write the mathematical models and flow models for the predicates used. For example, for the **N**=4, **k**=2 ⇒ [[1,3],[2,4]] (not necessarily in this order).

% comb(l1l2..ln, k) =

% { l1, if k = 1

% { comb(l2...ln, k)

% { l1 U comb(l2...ln, k-1), if k > 1

% comb(L: list, K:integer, C:list)

% (i, i, o) - nedeterminist

comb([H|\_], 1, [H]).

comb([\_|T], K, C) :- comb(T, K, C).

comb([H|T], K, [H|C]) :-

K > 1,

K1 is K-1,

comb(T, K1, C).

% Create a ranged list (from k to n)

% range(k, n) =

% [k], k = n.

% k U range(k+1, n), k < n

% range(K: Number, N:number, R:list)

% range(i, i, o)

range(N, N, [N]):-!.

range(K, N, [K|R]) :-

NewK is K + 1,

range(NewK, N, R).

%checkProperty(l1l2..ln)= { true, if n=2 and (l1-l2)%2 = 0

% { false, if (l1-l2) %2 = 1

% { checkProperty(l2..ln), otherwise

% checkProperty(L: list) ... flow model: checkProperty(i)

checkProperty([H1,H2]):-

Diff is H1-H2,

Diff mod 2 =:= 0, !.

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

Diff is H1-H2,

Diff mod 2 =:= 0,

checkProperty([H2|T]).

% oneSolution(N, K) =

% { C = comb(range(1,N), K), where C=c1c2..cK, checkProperty(C) = true

% onesolution(L:list, N: Integer, R: list)

% flow model: onesolution(i,i,o)

oneSolution(N,K,R):-

range(1,N,Lista),

comb(Lista,K,R),

checkProperty(R).

% allSolutions(N, K) =

% { findall(oneSolution(N, K))

% allSolutions(N: Integer, K: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(N,K,R):-

findall(X, oneSolution(N, K, X), R).

**C.** Write a PROLOG program that generates the list of all subsets of sum **S** given, using the elements of a list, such that the number of even elements from each subset is even. Write the mathematical models and flow models for the predicates used. For example for the list [1, 2, 3, 4, 5, 6, 10] and S=10 ⇒ [[1,2,3,4], [4,6]].

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

% Mathematical model

% evenNumbers(l1l2...ln) =

% { 0, if n = 0

% { 1 + evenNumbers(l2...ln), if l1 % 2 == 0

% { evenNumbers(l2...ln), otherwise

evenNumbers([], 0):- !.

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

H mod 2 =:= 0, !,

evenNumbers(T, R1),

R is R1+1.

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

evenNumbers(T, R).

% oneSolution(l1l2...ln, Sum) =

% { S, where S = s1s2...sm = subsets(L), if sum(S) = Sum and evenNumbers(S) % 2 = 0.

% oneSolution(L: list, Sum: Integer, R: List) ... (i, i, o)

oneSolution(L,S,R):-

subsets(L,R),

sum(R,Sum),

Sum =:= S,

evenNumbers(R,Even),

Even mod 2 =:= 0.

% allSolutions(l1l2..ln, Sum) =

% { findall(oneSolution(l1l2..ln, Sum))

% allSolutions(L: List, S: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(L,S,R):-

findall(X, oneSolution(L, S, X), R).

**C.** Write a PROLOG program that generates the list of all subsets of even sum, using the elements of a list. Write the mathematical models and flow models for the predicates used. For example, for the list L=[2, 3, 4] ⇒ [[],[2],[4],[2,4]] (not necessarily in this order).

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% Mathematical model

% oneSolution(l1l2..ln) =

% { S, where S = s1s2...sm = subsets(l1l2..ln), if sum(S) % 2 = 0.

% oneSolution(L: List, R: List)

% (i, o)

oneSolution(L,R):-

subsets(L,R),

sum(R,Sum),

Sum mod 2 =:= 0.

% allSolutions(l1l2...ln) =

% { findall(oneSolution(l1l2..ln))

% allSolutions(L: List, R: out list)

% flow model: allSolutions(i, o)

allSolutions(L,R):-

findall(X, oneSolution(L,X), R).

**C.** Write a PROLOG program that generates the list of all subsets with N elements, using the elements of a list, such that the sum of elements from a subset is an even number. Write the mathematical models and flow models for the predicates used. For example, for the list L=[1, 3, 4, 2] and N=2 ⇒ [[1,3], [2,4]].

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% Mathematical model

% len(l1l2..ln) =

% { 0, if n=0

% { 1 + length(l2...ln) , otherwise

% len(L: list, R: Integer)

% flow model: len(i, o)

len([], 0).

len([\_|T], NewR):-

len(T, R),

NewR is R + 1.

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

% oneSolution(l1l2..ln, N) =

% { S, where S = s1s2...sm = subsets(l1l2..ln), if sum(S) % 2 = 0 and m=N

% oneSolution(L: List, N: Integer, R: List)

% (i, i, o)

oneSolution(L,N,R):-

subsets(L,R),

len(R,Length),

Length =:= N,

sum(R, Sum),

Sum mod 2 =:= 0.

% allSolutions(l1l2...ln, N) =

% { findall(oneSolution(l1l2..ln, N))

% allSolutions(L: List, N: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(L,N,R):-

findall(X, oneSolution(L,N,X), R).

**C.** Write a PROLOG program that determines from a list made of integer numbers, the list of subsets with at least 2 elements, composed of numbers in strictly increasing order. Write the mathematical models and flow models for the predicates used. For example for the list [1, 8, 6, 4] ⇒ [[1,8],[1,6],[1,4],[6,8],[4,8],[4,6],[1,4,6],[1,4,8],[1,6,8],[4,6,8],[1,4,6,8]] (not necessarily in this order).

% Insert an element on every position of a list

% insertOnPos(l1l2..ln, elem) =

% { elem U l1l2..ln

% { l1 U insert(l2...ln, elem)

% insert(E: element, L:List, LRez:list)

% (i, i, o) - nedeterminist

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

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

insertOnPos(T, E, R).

%subsets(l1l2..ln) =

% { [], if n=0

% { insertOnPos(subsets(l2l3..ln), l1), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R1),

insertOnPos(R1,H,R).

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

subsets(T, R).

% Mathematical model

% len(l1l2..ln) =

% { 0, if n=0

% { 1 + length(l2...ln) , otherwise

% len(L: list, R: Integer)

% flow model: len(i, o)

len([], 0).

len([\_|T], NewR):-

len(T, R),

NewR is R + 1.

% checkProperty(l1l2..ln)=

% { true, if n=2 and (l2-l1) > 0

% { false, if (l2-l1) <= 0

% { checkProperty(l2..ln), otherwise

% checkProperty(L: list) ... flow model: checkProperty(i)

checkProperty([H1,H2]):-

Diff is H2-H1,

Diff > 0, !.

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

Diff is H2-H1,

Diff > 0,

checkProperty([H2|T]).

% oneSolution(l1l2..ln) =

% { S, where S = s1s2...sm = subsets(l1l2..ln), if m>=2 and checkProperty(S) is true

% oneSolution(L: List, R: List)

% (i, o)

oneSolution(L,R):-

subsets(L,R),

len(R, Len),

Len >= 2,

checkProperty(R).

% allSolutions(l1l2...ln) =

% { findall(oneSolution(l1l2..ln))

% allSolutions(L: List, R: out list)

% flow model: allSolutions(i, o)

allSolutions(L,R):-

findall(X, oneSolution(L,X), R).

**C.** Write a PROLOG program that generates the list of all subsets with values between the [a, b] interval such that the sum of elements from each subset is an odd value. Write the mathematical models and flow models for the predicates used. For example, for **a**=2 and **b**=4 ⇒ [[2,3],[3,4],[2,3,4]] (not necessarily in this order).

% Create a ranged list (from A to B)

% range(a, b) =

% [b], a = b.

% a U range(a+1, b), a < b

% range(K: Number, N:number, R:list)

% range(i, i, o)

range(B, B, [B]):- !.

range(A, B, [A|R]) :-

NewA is A + 1,

range(NewA, B, R).

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

% Mathematical model

% oneSolution(a, b) =

% { S, where S = s1s2...sm = subsets(range(a, b)), if sum(S) % 2 = 1.

% oneSolution(A: Integer, B: Integer, R: List)

% (i, i, o)

oneSolution(A, B, R):-

range(A, B, Range),

subsets(Range, R),

sum(R, Sum),

Sum mod 2 =:= 1.

% allSolutions(A, B, R) =

% { findall(oneSolution(A, B))

% allSolutions(A: Integer, B: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(A, B, R):-

findall(X, oneSolution(A, B, X), R).

**C.** Write a PROLOG program that generates the list of all subsets, each subset having an odd sum of elements and also even number of elements. Write the mathematical models and flow models for the predicates used. For example, for [2,3,4] ⇒ [[2,3,4]].

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

% Mathematical model

% my\_len(l1l2..ln) =

% { 0, if n=0

% { 1 + my\_len(l2...ln) , otherwise

% my\_len(L: list, R: Integer)

% flow model: my\_len(i, o)

my\_len([], 0).

my\_len([\_|T], NewR):-

my\_len(T, R),

NewR is R + 1.

% oneSolution(l1l2...ln) =

% { S, where S = s1s2...sm = subsets(l1l2..ln), if m is even and sum(S) % 2 = 1.

% oneSolution(L: List, R: List)

% (i, o)

oneSolution(L, R):-

subsets(L, R),

my\_len(R, LengthList),

LengthList mod 2 =:= 0,

sum(R, Sum),

Sum mod 2 =:= 1.

% allSolutions(l1l2...ln, R) =

% { findall(oneSolution(l1l2...ln))

% allSolutions(L: list, R: out list)

% flow model: allSolutions(i, o)

allSolutions(L, R):-

findall(X, oneSolution(L, X), R).

**C.** Write a PROLOG program that generates the list of all subsets with values between the [a, b] interval such that the sum of elements from each subset is an odd value. Write the mathematical models and flow models for the predicates used. For example, for **a**=2 and **b**=4 ⇒ [[2,3],[3,4],[2,3,4]] (not necessarily in this order).

% Create a ranged list (from A to B)

% range(a, b) =

% [b], a = b.

% a U range(a+1, b), a < b

% range(K: Number, N:number, R:list)

% range(i, i, o)

range(B, B, [B]):- !.

range(A, B, [A|R]) :-

NewA is A + 1,

range(NewA, B, R).

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

% Mathematical model

% oneSolution(a, b) =

% { S, where S = s1s2...sm = subsets(range(a, b)), if sum(S) % 2 = 1.

% oneSolution(A: Integer, B: Integer, R: List)

% (i, i, o)

oneSolution(A, B, R):-

range(A, B, Range),

subsets(Range, R),

sum(R, Sum),

Sum mod 2 =:= 1.

% allSolutions(A, B, R) =

% { findall(oneSolution(A, B))

% allSolutions(A: Integer, B: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(A, B, R):-

findall(X, oneSolution(A, B, X), R).

**C.** Write a PROLOG program that generates the list or all arrangements of **k** elements from a list of integer numbers, for which the product of the elements is less than a value **V** given. Write the mathematical models and flow models for the predicates used. For example, for the list [1, 2, 3], **k**=2 and **V**=7 ⇒ [[1,2],[2,1],[1,3],[3,1],[2,3],[3,2]] (not necessarily in this order).

% Mathematical model

% Insert an element on every position of a list

% insertOnPos(l1l2..ln, elem) =

% { elem U l1l2..ln

% { l1 U insert(l2...ln, elem)

% insert(E: element, L:List, LRez:list)

% (i, i, o) – nedeterminist

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

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

insertOnPos(T, E, R).

%Mathematical model

% arr(l1l2...ln, K) =

% { l1, if k = 1

% { arr(l2...ln, k), if k >= 1

% { insertOnPos(l1, arr(l2...ln, k-1)), if k>1

% arr(L: list, K: integer, R: list)

% arr(i,i,o)

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

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

arr(T, K, R).

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

K>1,

K1 is K-1,

arr(T, K1, R),

insertOnPos(R, H, R1).

% product(l1l2..ln) =

% { 1, if n = 0

% { l1 \* product(l2...ln), otherwise

% product(L: list, R: Integer). flow: (i, o)

product([], 1).

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

product(T, R1),

R is H\*R1.

% oneSolution(l1l2...ln, K, V) =

% { A, where A = a1a2...am = arr(l1l2..ln, K), if product(A)<V.

% oneSolution(L: List, K: Integer, V: Integer, R: List)

% (i, i, i, o)

oneSolution(L,K,V,R):-

arr(L,K,R),

product(R,Product),

Product < P.

% allSolutions(l1l2...ln, K, P) =

% { findall(oneSolution(l1l2..ln, K, P))

% allSolutions(L: int input list, K: Integer, P: Integer, R: out list)

% flow model: allSolutions% (i, i, i, o)

allSolutions(L,K,V,R):-

findall(X, oneSolution(L,K,V,X), R).

C. Write a PROLOG program that generates the list of all subsets of k elements in arithmetic progression. Write the mathematical models and flow models for the predicates used. For example, for L=[1,5,2,9,3] and k=3 ⇒ [[1,2,3],[1,5,9],[1,3,5]] (not necessarily in this order).

% Mathematical model

% Insert an element on every position of a list

% insertOnPos(l1l2..ln, elem) =

% { elem U l1l2..ln

% { l1 U insert(l2...ln, elem)

% insert(E: element, L:List, LRez:list)

% (i, i, o) – nedeterminist

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

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

insertOnPos(T, E, R).

%Mathematical model

% arr(l1l2...ln, K) =

% { l1, if k = 1

% { arr(l2...ln, k), if k >= 1

% { insertOnPos(l1, arr(l2...ln, k-1)), if k>1

% arr(L: list, K: integer, R: list)

% arr(i,i,o)

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

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

arr(T, K, R).

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

K>1,

K1 is K-1,

arr(T, K1, R),

insertOnPos(R, H, R1).

% arithmeticProgression(l1l2..ln,diff) =

% { true, n=2 and l2-l1=diff

% { false, if l2-l1 !=diff

% { arithmeticProgression(l2..ln,diff), otherwise

% arithmeticProgression(L: list, diff: integer)

% arithmeticProgression(i,i)

arithmeticProgression([H1,H2],D):-

Diff is H2-H1,

% H2 > H1,

Diff =:=D,!.

arithmeticProgression([H1,H2|T],D):-

Diff is H2-H1,

% H2 > H1,

Diff =:=D,

arithmeticProgression([H2|T],D).

% mainAP(l1l2..ln) = arithmeticProgression(l1l2..ln,l2-l1)

% mainAP(L: list) ... flow: (i)

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

Diff is H2-H1,

arithmeticProgression([H1,H2|T],Diff).

% function to check if elements are in increasing order

% increasingCheck(l1l2...ln) =

% { true, if l1 < l2 and n = 2

% { false, if l1 > l2 and n = 2

% { false, if l1 > l2

% { increasingCheck(l2...ln), otherwise

% increasingCheck(L:list)...flow: (i)

increasingCheck([H1,H2]):-

H1<H2.

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

H1<H2,

increasingCheck([H2|T]).

% oneSolution(l1l2...ln, K) =

% { A, where A = a1a2...am = arr(l1l2..ln, k), if checkIfOdd(A) = true and mainAP(A) = true.

% oneSolution(L: List, K: Integer, P: Integer, R: List)

% (i, i, i, o)

oneSolution(L,K,R):-

arr(L,K,R),

% increasingCheck(R), ( or you check in arithm)

mainAP(R).

% allSolutions(l1l2...ln, N) =

% { findall(oneSolution(l1l2..ln, N))

% allSolutions(L: int input list, N: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(L, K, R):-

findall(X, oneSolution(L, K, X), R).

**C.** Write a PROLOG program that generates the list of all permutations with the property the absolute value of difference between two consecutive values from each permutation is <=3. Write the mathematical models and flow models for the predicates used. For example, for L=[2,7,5] ⇒ [[2,5,7], [7,5,2]] (not necessarily in this order).

% C. Write a PROLOG program that generates the list of all permutations with the property the

% absolute value of difference between two consecutive values from each permutation is <=3.

% Write the mathematical models and flow models for the predicates used. For example, for

% L=[2,7,5] ⇒ [[2,5,7], [7,5,2]] (not necessarily in this order).

% Insert an element on every position of a list

% insert(elem, l1l2..ln) =

% { elem U l1l2..ln

% { l1 U insert(elem, l2...ln)

% insert(E: element, L:List, LRez:list)

% (i, i, o) - nedeterminist

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

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

% perm(l1l2...ln) =

% { [], if n = 0

% { insert(l1, perm(l2...ln)), otherwise.

% insert(L:List, P:list)

% (i, i, o)

perm([], []).

perm([H|T], P) :-

perm(T, L),

insert(H, L, P).

% myAbsDiff(a,b) =

% { a-b, a>b

% { b-a, otherwise

% myAbsDiff(a: integer, b: integer, rez: list) ... flow(i,i,0)

myAbsDiff(A,B,REZ):-

A>B, !,

REZ is A-B.

myAbsDiff(A,B,REZ):-

REZ is B-A.

%checkCond(l1l2..ln)=

% { true, n=2 and |l1-l2|<=3

% { false, |l1-l2|>3

% { checkCond(l2..ln), otherwise

% checkCond(L:list) ... flow: checkCond(i)

checkCond([H1,H2]):-

myAbsDiff(H1,H2,Diff),

Diff =< 3, !.

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

myAbsDiff(H1,H2,Diff),

Diff =< 3,

checkCond([H2|T]).

% oneSolution(l1l2...ln) =

% { P, where P = p1p2...pm = perm(l1l2..ln, k), if checkCond(P) = true.

% oneSolution(L: List, R:list)

% oneSolution(i, o)

oneSolution(L,R):-

perm(L,R),

checkCond(R).

% allSolutions(l1l2...ln) =

% { findall(oneSolution(l1l2..ln))

% allSolutions(L: int input list, R: out list)

% flow model: allSolutions(i, o)

allSolutions(L,R):-

findall(X, oneSolution(L,X), R).

**C.** Write a PROLOG program that generates the list of all combinations of k elements with the value of sum of each combination even number, from a list of integers. Write the mathematical models and flow models for the predicates used. For example, for the list L[6, 5, 3, 4], **k**=2 ⇒ [[6,4],[5,3]] (not necessarily in this order).

% comb(l1l2..ln, k) =

% { l1, if k = 1

% { comb(l2...ln, k)

% { l1 U comb(l2...ln, k-1), if k > 1

% comb(L: list, K:integer, C:list)

% (i, i, o) - nedeterminist

comb([H|\_], 1, [H]).

comb([\_|T], K, C) :- comb(T, K, C).

comb([H|T], K, [H|C]) :-

K > 1,

K1 is K-1,

comb(T, K1, C).

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

% oneSolution(l1l2...ln, K) =

% { C = comb(l1l2...ln, K), where C=c1c2..cK, sum(C) % 2 = 0

% oneSolution(L:list, K: Integer, R: list)

% flow model: onesolution(i,i,o)

oneSolution(L,K,R):-

comb(L,K,R),

sum(R,Sum),

Sum mod 2 =:= 0.

% allSolutions(l1l2...ln, K) =

% { findall(oneSolution(l1l2...ln, K))

% allSolutions(L: List, K: Integer, R: out list)

% flow model: allSolutions(i, i, o)

allSolutions(L,K,R):-

findall(X, oneSolution(L,K,X), R).

**C.** Write a PROLOG program that generates the list of all arrangements of k elements with the value of sum of all elements from each arrangement equal with a given S, from a list of integers. Write the mathematical models and flow models for the predicates used. For example, for the list [6, 5, 3, 4], **k**=2 and **S**=9⇒ [[6,3],[3,6],[5,4],[4,5]] (not necessarily in this order).

% Mathematical model

% Insert an element on every position of a list

% insertOnPos(l1l2..ln, elem) =

% { elem U l1l2..ln

% { l1 U insert(l2...ln, elem)

% insert(E: element, L:List, LRez:list)

% (i, i, o) – nedeterminist

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

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

insertOnPos(T, E, R).

%Mathematical model

% arr(l1l2...ln, K) =

% { l1, if k = 1

% { arr(l2...ln, k), if k >= 1

% { insertOnPos(l1, arr(l2...ln, k-1)), if k>1

% arr(L: list, K: integer, R: list)

% arr(i,i,o)

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

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

arr(T, K, R).

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

K>1,

K1 is K-1,

arr(T, K1, R),

insertOnPos(R, H, R1).

% sum(l1l2..ln)= {0, n=0

% {l1 + sum(l2..ln), otherwise

% sum(L: List, R: Integer)

% (i, o)

sum([],0).

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

sum(T,R1),

R is H+R1.

% oneSolution(l1l2...ln, K, S) =

% { A = arr(l1l2...ln, K), where A=a1a2..am, sum(A) = S

% oneSolution(L:list, K: Integer, S:Integer, R: list)

% flow model: onesolution(i,i,i,o)

oneSolution(L,K, S, R):-

arr(L,K,R),

sum(R,Sum),

Sum =:= S.

% allSolutions(l1l2...ln, S, K) =

% { findall(oneSolution(l1l2...ln, S, K))

% allSolutions(L: List, K: Integer, R: out list)

% flow model: allSolutions(i, i, i, o)

allSolutions(L,K,S,R):-

findall(X, oneSolution(L,K,S,X), R).

**C.** For a given value N, generate the list of all permutations with elements N, N+1, ..., 2\*N-1 with the property that the absolute value between two consecutive values from the permutation is <=2. Write the mathematical models and flow models for the predicates used.

% Mathematical model

% Insert an element on every position of a list

% insert(elem, l1l2..ln) =

% { elem U l1l2..ln

% { l1 U insert(elem, l2...ln)

% insert(E: element, L:List, LRez:list)

% (i, i, o) - nedeterminist

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

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

% Mathematical model

% perm(l1l2...ln) =

% { [], if n = 0

% { insert(l1, perm(l2...ln)), otherwise.

% insert(L:List, P:list)

% (i, i, o)

perm([], []).

perm([H|T], P) :-

perm(T, L),

insert(H, L, P).

% Create a ranged list (from A to B)

% range(a, b) =

% [b], a = b.

% a U range(a+1, b), a < b

% range(K: Number, N:number, R:list)

% range(i, i, o)

range(B, B, [B]):- !.

range(A, B, [A|R]) :-

NewA is A + 1,

range(NewA, B, R).

% myAbsDiff(a,b) =

% { a-b, a>b

% { b-a, otherwise

% myAbsDiff(a: integer, b: integer, rez: list) ... flow(i,i,0)

myAbsDiff(A,B,REZ):-

A>B, !,

REZ is A-B.

myAbsDiff(A,B,REZ):-

REZ is B-A.

%checkCond(l1l2..ln)=

% { true, n=2 and |l1-l2|<=2

% { false, |l1-l2|>2

% { checkCond(l2..ln), otherwise

% checkCond(L:list) ... flow: checkCond(i)

checkCond([H1,H2]):-

myAbsDiff(H1,H2,Diff),

Diff =< 2, !.

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

myAbsDiff(H1,H2,Diff),

Diff =< 2,

checkCond([H2|T]).

% oneSolution(n) =

% { P, where P = p1p2...pm = perm(range(n, n\*2-1)), if checkCond(P) = true.

% oneSolution(n: Integer, R:list)

% oneSolution(i, o)

oneSolution(N,R):-

Auxiliar is N\*2,

Right is Auxiliar-1,

range(N,Right,GeneratedList),

perm(GeneratedList,R),

checkCond(R).

% allSolutions(n) =

% { findall(oneSolution(n))

% allSolutions(n: integer, R: out list)

% flow model: allSolutions(i, o)

allSolutions(N,R):-

findall(X, oneSolution(N, X), R).

**C.** Given a list made of integer numbers, generate in PROLOG the list of all subsets with even number of elements. Write the mathematical models and flow models for the predicates used. For example, for the list L=[2,3,4] ⇒ [[],[2,3],[2,4],[3,4]] (not ne[[3],[2,4],[2,3,4]]cessarily in this order).

% len(l1l2..ln) =

% { 0, if n=0

% { 1 + length(l2...ln) , otherwise

% len(L: list, R: Integer)

% flow model: len(i, o)

len([], 0).

len([\_|T], NewR):-

len(T, R),

NewR is R + 1.

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% oneSolution(l1l2...ln) =

% { S, where S = s1s2...sm = subsets(l1l2...ln), if sum(S) % 2 = 1.

% oneSolution(A: Integer, B: Integer, R: List)

% (i, i, o)

oneSolution(L,R):-

subsets(L,R),

len(R,Len),

Len mod 2 =:= 0.

% allSolutions(l1l2..ln) =

% { findall(oneSolution(l1l2..ln))

% allSolutions(L: list, R: out list)

% flow model: allSolutions(i, o)

allSolutions(L,R):-

findall(X, oneSolution(L, X), R).

**C.** Write a PROLOG program that generates the list of all subsets with value of sum for each subset odd number and also odd numbers of odd values from each subset. Write the mathematical models and flow models for the predicates used. For example, for [2,3,4] ⇒ [[2,3],[3,4],[2,3,4]] not necessarily in this order).

%subsets(l1l2..ln) =

% { [], if n=0

% { l1 U subsets(l2l3..ln), if n>0

% { subsets(l2l3..ln), if n>0

% subsets (L: int input list, R: result List)

% flow model: subsets (i,o)

subsets([], []).

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

subsets(T, R).

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

subsets(T, R).

% sum(l1l2..ln)= 0, n=0

% { l1+sum(l2l3..ln), otherwise

sum([],0).

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

sum(T,R1),

R is R1+H.

% Mathematical model

% oddNumbers(l1l2...ln) =

% { 0, if n = 0

% { 1 + oddNumbers(l2...ln), if l1 % 2 == 1

% { oddNumbers(l2...ln), otherwise

oddNumbers([], 0):- !.

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

H mod 2 =:= 1, !,

oddNumbers(T, R1),

R is R1+1.

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

oddNumbers(T, R).

% Mathematical model

% oneSolution(l1l2...ln) =

% { S, where S = s1s2...sm = subsets(l1l2...ln), if sum(S) % 2 = 1 and oddNumbers(S)%2=1.

% oneSolution(L: list, R: List)

% (i, o)

oneSolution(L,R):-

subsets(L,R),

sum(R,Sum),

Sum mod 2=:=1,

oddNumbers(R,OddNumber),

OddNumber mod 2 =:=1.

% allSolutions(l1l2..ln) =

% { findall(oneSolution(l1l2..ln))

% allSolutions(L: list, R: out list)

% flow model: allSolutions(i, o)

allSolutions(L,R):-

findall(X, oneSolution(L, X), R).