1.

a. Write a predicate to determine the lowest common multiple of a list formed from integer numbers.

Model matematic:

gcd(a, b) = b, if a = 0

a, if b = 0

gcd(a % b, b), if a >= b

gcd(a, b % a), if a < b

gcd(0, B, B) :- !.

gcd(A, 0, A) :- !.

gcd(A, B, R) :-

A >= B,

A1 is A mod B,

gcd(A1, B, R), !.

gcd(A, B, R) :-

A < B,

B1 is B mod A,

gcd(A, B1, R), !.

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

gcd(i, i, o)

Model matematic:

lcm(a, b) = a \* b / gcd(a, b)

lcm(A, B, R) :-

gcd(A, B, RGCD),

R is A \* B / RGCD.

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

lcm(i, i, o)

lcmForList([H], H).

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

lcmForList(T, GCD),

lcm(GCD, H, R).

lcmForList(L:list, R:number)

lcmForList(i, o)

b. Write a predicate to add a value v after 1-st, 2-nd, 4-th, 8-th, … element in a list.

Model matematic:

insert\_pow(l1...ln, v, pos, index) = [], n = 0

l1 + v + insert\_pow(l2...ln, v, pos \* 2, index + 1), index = pos

l1 + insert\_pow(l2...ln, v, pos, index + 1), pos != index

insetPow([], \_, \_, \_, []).

insertPow([H|T], V, P, P, [H,V|R]):-

newP is P \* 2,

newI is P + 1,

inserPow(T, V, newP, newI, R).

insertPow([H|T], V, P, I, [H|R]):-

newI is I + 1,

insertPow(T, V, P, newI, R).

insert\_pow(L:list, V:number, POS:number, INDEX:number, R:list)

insert\_pow(i, i, i, i, o)

insert(L, V, R) :-

insert\_pow(L, V, 1, 1, R).

insert(L:list, V:number, R:list)

insert(i, i, o)

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

2

a. Write a predicate to remove all occurrences of a certain atom from a list.

Model matematic:

remove\_occurences(l1...ln, e) = [], n = 0

remove\_occurences(l2...ln, e), l1 = e

l1 + remove\_occurences(l2...ln, e), l1 <> e

removeOccurences([], \_, []).

removeOccurences([E|T], E, R):-

removeOccurences(T, E, R).

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

removeOccurences(T, E, R).

remove\_occurences(L:list, E:number, R:list)

remove\_occurences(i, i, o)

b. Define a predicate to produce a list of pairs (atom n) from an initial list of atoms.

In this initial list atom has n occurrences.

Eg.: numberatom([1, 2, 1, 2, 1, 3, 1], X) => X = [[1, 4], [2, 2], [3, 1]].

Model matematic:

countOccurences(l1...ln, e) = 0, n = 0

1 + countOccurences(l2...ln, e), e = l1

countOccurences(l2...ln, e), e <> l1

countOccurences([], \_, 0).

countOccurences([E|T], E, R):-

countOccurences(T, E, R1),

R is R1 + 1, !.

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

countOccurences(T, E, R).

countOccurences(L:list, E:number, R:number)

countOccurences(i, i, o)

Model matematic:

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

[l1, count\_occurences(l1...ln, l1)] + numberatom(remove\_occurences(l2...ln, l1))

numberAtom([],[]).

numberAtom([H|T], [[H|COUNT]|R]):-

countOccurences([H|T], H, COUNT),

removeOccurences(T, H, RLIST),

numberAtom(RLIST, R).

numberatom(L:list, R:list)

numberatom(i, o)

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

3.

a. Define a predicate to remove from a list all repetitive elements. Eg.: l=[1,2,1,4,1,3,4] => l=[2,3])

Model matematic:

removeOccurences(l1...ln, e) = [], n = 0

removeOccurences(l2...ln, e), l1 = e

l1 + removeOccurences(l2...ln, e), l1 <> e

removeOccurences([], \_, []).

removeOccurences([E|T], E, R):-

removeOccurences(T, E, R).

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

removeOccurences(T, E, R).

remove\_occurences(L:list, E:number, R:list)

remove\_occurences(i, i, o)

Model matematic:

countOccurences(l1...ln, e) = 0, n = 0

1 + countOccurences(l2...ln, e), e = l1

countOccurences(l2...ln, e), e <> l1

countOccurences([], \_, 0).

countOccurences([E|T], E, R):-

countOccurences(T, E, R1),

R is R1 + 1, !.

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

countOccurences(T, E, R).

countOccurences(L:list, E:number, R:number)

countOccurences(i, i, o)

Model matematic:

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

l1 + removeRepetitive(l2...ln), countOccurences(l1...ln, l1) = 1

removeRepetitive(removeOccurences(l1...ln, l1)), countOccurences(l1...ln, l1) != 1

removeRepetitive([], []).

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

countOccurences([H|T], H, COUNT),

COUNT =\= 1,

removeOccurences([H|T], H, RLIST),

removeRepetitive(RLIST, R).

removeRepetitive(L:list, R:list)

removeRepetitive(i, o)

b. Remove all occurrence of a maximum value from a list on integer numbers.

Model matematic:

maxNumber(a, b) = a, if a >= b

b, if a < b

maxNumber(A, B, A):-

A >= B.

maxNumber(A, B, B):-

A < B.

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

maxNumber(i, i, o)

Model matematic:

maxInList(l1...ln) = l1, n = 1

maxim\_number(l1, maxim\_list(l2...ln))

maxInList([H], H).

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

maxInList(T, MAX),

maxNumber(H, MAX, R).

maxInList(L:list, R:number)

maxInList(i, o)

Model matematic:

removeMax(l1...ln) = [], if n = 0

remove\_occurences(l1...ln, maxim\_list(l1...ln)), otherwise

removeMax([], []).

removeMax(L, R):-

maxInList(L, MAX),

removeOccurences(L, MAX, R).

removeMax(L:list, R:list)

removeMax(i, o)

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

4.

a. Write a predicate to determine the difference of two sets.

Model matematic:

contains(l1..ln, e) = false, n = 0

true, l1 = e

contains(l2...ln, e), otherwise

contains([E|\_], E):- !.

contains([\_|T], E):-

contains(T, E).

contains(L:list, E:number)

contains(i, i)

Model matematic:

diff(a1...an, b1...bn) = [], n = 0

diff(a2...an, b1...bn), contains(b1...bn, a1) = true

a1 + diff(a2...an, b1...bn), otherwise

diff([], \_, []).

diff([H|T], B, R):-

contains(B, H),

diff(T, B, R), !.

diff([H|T], B, [H|R]):-

diff(T, B, R).

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

diff(i, i, o)

b. Write a predicate to add value 1 after every even element from a list.

Model matematic:

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

l1 + 1 + insert1(l2...ln), l1 % 2 = 0

l1 + insert1(l2...ln), otherwise

insertAfterEven([], []).

insertAfterEven([H|T], [H,1|R]):-

H mod 2 =:= 0,

insertAfterEven(T, R).

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

insertAfterEven(T, R).

insertAfterEven(L:list, R:list)

insertAfterEven(i, o)

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

5.

a. Write a predicate to compute the union of two sets.

Model matematic:

removeOccurences(l1...ln, e) = [], n = 0

removeOccurences(l2...ln, e), l1 = e

l1 + removeOccurences(l2...ln, e), l1 <> e

removeOccurences([], \_, []).

removeOccurences([E|T], E, R):-

removeOccurences(T, E, R).

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

removeOccurences(T, E, R).

removeOccurences(L:list, E:number, R:list)

removeOccurences(i, i, o)

Model matematic:

union(a1...an, b1...bm) = [], n = 0 and m = 0

union(b1...bm, []), n = 0

a1 + union(remove\_occurences(a1...an, a1), remove\_occurences(b1...bm, a1)), otherwise

union([], [], []).

union([], B, R):-

union(B, [], R).

union([H|T], B, [H|R]):-

removeOccurences([H|T], H, RA),

removeOccurences(B, H, RB),

union(RA, RB, R).

union(A:list, B:list, R:list)

union(i, i, o)

b. Write a predicate to determine the set of all the pairs of elements in a list.

Eg.: L = [a b c d] => [[a b] [a c] [a d] [b c] [b d] [c d]].

Model matematic:

comb(l1...ln, k) = l1, k = 1, n >= 1

comb(l2...ln, k), k >= 1

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

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

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

comb(T, K, R).

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

K > 1,

K1 is K - 1,

comb(T, K1, R).

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

comb(i, i, o)

Model matematic:

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

findall(comb(l1...ln, 2))

genAllPairs([], []).

genAllPairs(L, R):-

findall(RS, comb(L, 3, RS), R).

genAllPairs(L:list, R:list)

genAllPairs(i, o)

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

6.

a. Write a predicate to test if a list is a set.

Model matematic:

countOccurences(l1...ln, e) = 0, n = 0

1 + countOccurences(l2...ln, e), e = l1

countOccurences(l2...ln, e), e <> l1

countOccurences([], \_, 0).

countOccurences([E|T], E, R):-

countOccurences(T, E, R1),

R is R1 + 1, !.

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

countOccurences(T, E, R).

countOccurences(L:list, E:number, R:number)

countOccurences(i, i, o)

Model matematic:

testIfSet(l1...ln) = true, n = 0

false, count\_occurences(l1...ln, l1) != 1

testIfSet(l2...ln), otherwise

testIfSet([]).

testIfSet([H|T]):-

countOccurences([H|T], H, COUNT),

COUNT =:= 1,

testIfSet(T), !.

testIfSet(L:list)

testIfSet(i)

b. Write a predicate to remove the first three occurrences of an element in a list.

If the element occurs less than three times, all occurrences will be removed.

Model matematic:

removeKOccurences(l1...ln, e, k) = [], n = 0

l1...ln, k = 0

removeKOccurences(l2...ln, e, k - 1), l1 = e

l1 + removeKOccurences(l2...ln, e, k), l1 != e

removeKOccurences([], \_, \_, []):- !.

removeKOccurences(L, \_, 0, L):- !.

removeKOccurences([E|T], E, K, R):-

K1 is K - 1,

removeKOccurences(T, E, K1, R).

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

H =\= E,

removeKOccurences(T, E, K, R).

removeKOccurences(L:list, E:number, K:number, R:list)

removeKOccurences(i, i, i, o)

remove3Occurences(L, E, R) :-

removeKOccurences(L, E, 3, R).

remove\_3\_occurences(L:list, E:number, R:list)

remove\_3\_occurences(i, i, o)

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

7.

a. Write a predicate to compute the intersection of two sets.

Model matematic:

contains(l1..ln, e) = false, n = 0

true, l1 = e

contains(l2...ln, e), otherwise

contains([E|\_], E):- !.

contains([\_|T], E):-

contains(T, E).

contains(L:list, E:number)

contains(i, i)

Model matematic:

intersection(a1...an, b1...bm) = [], n = 0

a1 + intersection(a2...an, b1...bm), contains(b1...bm, a1) = true

intersection(a2...an, b1...bm), otherwise

intersection([], \_, []).

intersection([H|T], B, [H|R]):-

contains(B, H),

intersection(T, B, R), !.

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

intersection(T, B, R).

intersection(A:list, B:list, R:list)

intersection(i, i, o)

b. Write a predicate to create a list (m, ..., n) of all integer numbers from the interval [m, n].

Model matematic:

intervalToList(m, n) = [m], m = n

m + new\_list(m + 1, n)

intervalToList(A, A, [A]).

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

NEXTA is A + 1,

intervalToList(NEXTA, B, R).

intervalToList(A:number, B:number, R:list)

intervalToList(i, i, o)

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

8.

a. Write a predicate to determine if a list has even numbers of elements

without counting the elements from the list.

Model matematic:

evenList(l1...ln) = true, n = 0

false, n = 1

evenList(l3...ln), otherwise

evenList([]).

evenList([\_, \_|T]):-

evenList(T).

even\_list(L:list)

even\_list(i)

b. Write a predicate to delete first occurrence of the minimum number from a list.

Model matematic:

minNumber(a, b) = a, if a <= b

b, if a > b

minNumber(A, B, A):-

A =< B.

minNumber(A, B, B):-

A > B.

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

minNumber(i, i, o)

Model matematic:

minInList(l1...ln) = l1, n = 1

minNmber(l1, minInList(l2...ln))

minInList([H], H).

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

minInList(T, MIN),

minNumber(H, MIN, R).

minInList(L:list, R:number)

minInList(i, o)

Model matematic:

removeKOccurences(l1...ln, e, k) = [], n = 0

l1...ln, k = 0

removeKOccurences(l2...ln, e, k - 1), l1 = e

l1 + removeKOccurences(l2...ln, e, k), l1 != e

removeKOccurences([], \_, \_, []):- !.

removeKOccurences(L, \_, 0, L):- !.

removeKOccurences([E|T], E, K, R):-

K1 is K - 1,

removeKOccurences(T, E, K1, R).

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

H =\= E,

removeKOccurences(T, E, K, R).

removeKOccurences(L:list, E:number, K:number, R:list)

removeKOccurences(i, i, i, o)

remove1Occurences(L, E, R) :-

removeKOccurences(L, E, 1, R).

remove1Occurences(L:list, E:number, R:list)

remove1Occurences(i, i, o)

deleteFirstMin(L, R) :-

minInList(L, MIN),

remove1Occurences(L, MIN, R).

deleteFirstMin(L:list, R:list)

deleteFirstMin(i, o)

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

9.

a. Insert an element on the position n in a list.

Model matematic:

insetOnPos(l1...ln, v, pos) = v + l1...ln, p = 0

l1 + insetOnPos(l2...ln, v, pos-1)

insetOnPos([H|T], V, 0, [V, H|T]).

insetOnPos([H|T], V, P, [H|R]):-

P1 is P -1,

insetOnPos(T, V, P1, R).

insetOnPos(L:list, V:number, P:number, R:list)

insetOnPos(i, i, i, o)

b. Define a predicate to determine the greatest common divisor of all numbers from a list

Model matematic:

gcd(a, b) = b, if a = 0

a, if b = 0

gcd(a % b, b), if a >= b

gcd(a, b % a), if a < b

gcd(0, B, B) :- !.

gcd(A, 0, A) :- !.

gcd(A, B, R) :-

A >= B,

A1 is A mod B,

gcd(A1, B, R), !.

gcd(A, B, R) :-

A < B,

B1 is B mod A,

gcd(A, B1, R), !.

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

gcd(i, i, o)

Model matematic:

gcdForList(l1...ln) = l1, n = 1

gcd(l1, gcdForList(l2...ln)), otherwise

gcdForList([H], H).

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

gcdForList(T, GCD),

gcd(GCD, H, R).

gcdForList(L:list, R:number)

gcdForList(i, o)

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

10.

a. Define a predicate to test if a list of integer elements has a "valley" aspect (a set has a "valley" aspect if elements decreases up to a certain point, and then increases.

eg: 10 8 6 9 11 13 – has a “valley” aspect

Model matematic:

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

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

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

false, otherwise

valley([\_], 1).

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

H1 > H2,

valley([H2|T], 0).

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

H1 < H2,

valley([H2|T], 1).

valley(L:list, F:number)

valley(i, i)

b. Calculate the alternate sum of list’s elements (l1 - l2 + l3 ...).

Model matematic:

alternative\_sum(l1...ln) = 0, n = 0

l1, n = 1

l1 - l2 + alternative\_sum(l3...ln), otherwise

alternativeSum([], 0).

alternativeSum([H], H).

alternativeSum([H1, H2|T], R):-

alternativeSum(T, R1),

R is H1 - H2 + R1.

alternative\_sum(L:list, R:number)

alternative\_sum(i, o)

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

11.

a. Write a predicate to substitute an element from a list with another element in the list.

Model matematic:

subsElem(l1...ln, e1, e2) = [], n = 0

e2 + subsElem(l2...ln, e1, e2), l1 = e1

l1 + subsElem(l2...ln, e1, e2), l1 != e1

subsElem([], \_, \_, []).

subsElem([E1|T], E1, E2, [E2|R]):-

subsElem(T, E1, E2, R).

subsElem([H|T], E1, E2, [H|R]):-

subsElem(T, E1, E2, R).

subsElem(L:list, E1:number, E2:number, R:list)

subsElem(i, i, i, o)

b. Write a predicate to create the sublist (lm, …, ln) from the list (l1,…, lk).

Model matematic:

sublist(l1...lk, m, n, pos) = [], pos > n

l1 + sublist(l2...lk, m, n, pos + 1), m < pos and pos < n

sublist(l2...lk, m, n, pos + 1), otherwise

subList(\_, \_, N, POS, []):-

POS > N.

subList([H|T], M, N, POS, [H|R]):-

M =< POS,

POS =< N,

NEWPOS is POS + 1,

subList(T, M, N, NEWPOS, R).

subList([\_|T], M, N, POS, R):-

NEWPOS is POS + 1,

subList(T, M, N, NEWPOS, R).

subList(L:list, M:number, N:number, POS:number, R:list)

subList(i, i, i, o)

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

12.

a. Write a predicate to substitute in a list a value with all the elements of another list.

Model matematic:

insertList(l1...ln, list) = list, n = 0

l1 + insertList(l2...ln, list), otherwise

insertList([], L, L).

insertList([H|T], L, [H|R]):-

insertList(T, L, R).

insertList(L:list, List:list, R:list)

insertList(i, i, o)

Model matematic:

subsElem(l1...ln, e, L2) = [], n = 0

L2 U subsElem(l2...ln, e, L2), l1 = e1

l1 + subsElem(l2...ln, e, L2), l1 != e1

subsElem([], \_, \_, []).

subsElem([E|T], E, L, R):-

insertList(L, T, RL),

subsElem(RL, E, L, R).

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

subsElem(T, E, L, R).

subsElem(L:list, E1:number, E2:number, R:list)

subsElem(i, i, i, o)

b. Remove the n-th element of a list

Model matematic:

removeElemOnPos(l1...ln, k) = [], n = 0

l2...ln, k = 1

l1 + removeElemOnPos(l2...ln, k -1), otherwise

removeElemOnPos([], \_, []).

removeElemOnPos([\_|T], 1, T).

removeElemOnPos([H|T], POS, [H|R]):-

POS1 is POS - 1,

removeElemOnPos(T, POS1, R).

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

removeElemOnPos(i, i, o)

Model matematic:

removeNthOccurence(l1...ln, e, k) = [], n = 0

l2...ln, k = 0 and l1 = e

l1 + removeNthOccurence(l2...ln, e, k - 1), otherwise

removeNthOccurence([], \_, \_, []).

removeNthOccurence([E|T], E, 0, T).

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

K1 is K - 1,

removeNthOccurence(T, E, K1, R).

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

13.

a. Transform a list in a set, in the order of the last occurrences of elements.

Eg.: [1,2,3,1,2] is transformed in [3,1,2].

Model matematic:

countOccurences(l1...ln, e) = 0, n = 0

1 + countOccurences(l2...ln, e), e = l1

countOccurences(l2...ln, e), e <> l1

countOccurences([], \_, 0).

countOccurences([E|T], E, R):-

countOccurences(T, E, R1),

R is R1 + 1, !.

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

countOccurences(T, E, R).

countOccurences(L:list, E:number, R:number)

countOccurences(i, i, o)

Model matematic:

list\_to\_set(l1...ln) = [], n = 0

l1 + list\_to\_set(l2...ln), count\_occurences(l2...ln, l1) = 0

list\_to\_set(l2...ln), otherwise

listToSet([], []).

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

countOccurences(T, H, OCC),

OCC =:= 0,

listToSet(T, R).

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

listToSet(T, R).

list\_to\_set(L:list, R:list)

list\_to\_set(i, o)

b. Define a predicate to determine the greatest common divisor of all numbers in a list

Model matematic:

gcd(a, b) = b, if a = 0

a, if b = 0

gcd(a % b, b), if a >= b

gcd(a, b % a), if a < b

gcd(0, B, B).

gcd(A, 0, A).

gcd(A, B, R) :-

A >= B,

A1 is A mod B,

gcd(A1, B, R).

gcd(A, B, R) :-

A < B,

B1 is B mod A,

gcd(A, B1, R).

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

gcd(i, i, o)

Model matematic:

gcdForList(l1...ln) = l1, n = 1

gcd(l1, gcdForList(l2...ln)), otherwise

gcdForList([H], H).

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

gcdForList(T, GCD),

gcd(GCD, H, R).

gcdForList(L:list, R:number)

gcdForList(i, o)

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

14.

a. Write a predicate to test the equality of two sets without using the set difference.

Model matematic:

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

1 + myLength(l2...ln), otherwise

myLength([], 0).

myLength([\_|T], LENGTH):-

myLength(T, LENGTH2),

LENGTH is LENGTH2 + 1.

myLength(L -list, N - number)

myLength(i, o)

Model matematic:

contains(l1..ln, e) = false, n = 0

true, l1 = e

contains(l2...ln, e), otherwise

contains([E|\_], E):- !.

contains([\_|T], E):-

contains(T, E).

contains(L:list, E:number)

contains(i, i)

Model matematic:

removeOccurences(l1...ln, e) = [], n = 0

removeOccurences(l2...ln, e), l1 = e

l1 + removeOccurences(l2...ln, e), l1 <> e

removeOccurences([], \_, []).

removeOccurences([E|T], E, R):-

removeOccurences(T, E, R).

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

removeOccurences(T, E, R).

removeOccurences(L:list, E:number, R:list)

removeOccurences(i, i, o)

Model matematic:

set\_equal(a1...an, b1...bm) = true, n = 0 and m = 0

set\_equal(remove\_occurences(a2...an, b1), remove\_occurences(b2...bm, a1)), n = m and contains(a1...an, b1) = true and contains(b1...bm, a1) = true

equalSets([], []).

equalSets([HA|TA], [HB|TB]):-

myLength([HA|TA], LA),

myLength([HB|TB], LB),

LA =:= LB,

contains([HA|TA], HB),

contains([HB|TB], HA),

removeOccurences(TA, HB, RA),

removeOccurences(TB, HA, RB),

equalSets(RA, RB).

equalSets(A:list, B:list)

equalSets(i, i)

b. Write a predicate to select the n-th element of a given list.

Model matematic:

selectNthElement(l1...ln, e, k) = [], n = 0

l1, k = 1

selectNthElement(l2...ln, k - 1), otherwise

selectNthElement([], \_, []).

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

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

K1 is K - 1,

selectNthElement(T, K1, R).

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

15.

a. Write a predicate to transform a list in a set, considering the first occurrence.

Eg: [1,2,3,1,2] is transform in [1,2,3].

Model matematic:

removeOccurences(l1...ln, e) = [], n = 0

removeOccurences(l2...ln, e), l1 = e

l1 + removeOccurences(l2...ln, e), l1 <> e

removeOccurences([], \_, []).

removeOccurences([E|T], E, R):-

removeOccurences(T, E, R).

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

removeOccurences(T, E, R).

removeOccurences(L:list, E:number, R:list)

removeOccurences(i, i, o)

Model matematic:

list\_to\_set(l1...ln) = [], n = 0

l1 + list\_to\_set(removeOccurences(l2...ln, l1)), otherwise

listToSet([], []).

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

removeOccurences(T, H, R1),

listToSet(R1, R).

list\_to\_set(L:list, R:list)

list\_to\_set(i, o)

b. Write a predicate to decompose a list in a list respecting the following:

[list of even numbers list of odd numbers] and also return the number of even numbers and the numbers of odd numbers.

Model matematic:

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

[evenOddDecompose (l2...ln), 1 + nr\_even, l1 + even\_list], l1 % 2 = 0

[evenOddDecompose (l2...ln), 1 + nr\_odd, l1 + odd\_list], l1 % 2 = 1

evenOddDecompose([], [[], [], 0, 0]).

evenOddDecompose([H|T], [[H|RE], RO, NE1, NO]):-

H mod 2 =:= 0,

evenOddDecompose(T, [RE, RO, NE, NO]),

NE1 is NE + 1.

evenOddDecompose([H|T], [RE, [H|RO], NE, NO1]):-

H mod 2 =:= 1,

evenOddDecompose(T, [RE, RO, NE, NO]),

NO1 is NO + 1.

decompose(L:list, R:list)

decompose(i, o)