SUBJECT 7

B.

%Dându-se o listă formată din numere întregi, să se genereze ȋn PROLOG lista permutărilor având proprietatea că valoarea

%absolută a diferenţei dintre două valori consecutive din permutare este <=3.

%Se vor scrie modelele matematice și modelele de flux pentru predicatele folosite.

%Exemplu- pentru lista L=[2,7,5] ⇒ [[2,5,7], [7,5,2]] (nu neapărat în această ordine)

%insert

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

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

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

insert(T, E, R).

%permutations

permutations([], []).

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

permutations(T, RP),

insert(RP, H, R).

%difference

difference(A, B, R):-

A < B,

R is B - A.

difference(A, B, R):-

A > B,

R is A - B.

checkProperty([\_]).

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

difference(H1, H2, DIF),

DIF =< 3,

checkProperty([H2|T]).

onesolution(L, R) :-

permutations(L, R),

checkProperty(R).

allsolutions(L, R) :-

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

C.

;Un arbore n-ar se reprezintă în LISP astfel (nod subarbore1 subarbore2 .....). Se cere să se verifice dacă un nod x apare pe

;un nivel par în arbore. Nivelul rădăcinii se consideră a fi 0. Se va folosi o funcție MAP.

;Exemplu pentru arborele (a (b (g)) (c (d (e)) (f)))

;a) x=g => T b) x=h => NIL

(defun evenLevelNode (tree level x)

(cond

((and (atom tree) (eq tree x) (eq (mod level 2) 0)) (list T))

((and (atom tree) (eq tree x) (eq (mod level 2) 1)) (list nil))

((and (atom tree) (not (eq tree x))) (list nil))

(t (list (apply #'append(mapcar #'(lambda (s) (evenLevelNode s (+ level 1) x)) tree))))

)

)

(defun wrapper (l x)

(car (evenLevelNode l -1 x))

)

(print (wrapper '(a (b (g)) (c (d (e)) (f))) 'g))

(defun orFunc (l)

(cond

((null l) NIL)

(t (OR (CAR l) (orFunc (cdr l))))

)

)

(defun main (l x)

(orFunc(wrapper l x))

)

(print (main '(a (b (g)) (c (d (e)) (f))) 'g))

;---------------------------------------------------

(defun searchNode (L N)

(cond

((and (atom L) (equal L N)) 1)

((listp L)( apply '+ (mapcar #' (lambda (S)(searchNode S N)) L)))

(t 0)

)

)

(print (searchNode '(A (b (g)) (c (d (e)) (f))) 'g ))

(defun is\_on\_evenLvl(L X lvl)

(SETQ A (searchNode L X))

(cond

((null L) nil)

((= 0 A) nil)

((= 0 (mod lvl 2)) T)

((= 1 (mod lvl 2)) nil)

(t (mapcar #'(lambda (S) (is\_on\_evenLvl S X (+ 1 lvl))) L))

)

)

(print (is\_on\_evenLvl '(A (b (g)) (c (d (e)) (f))) 'b 0))

SUBJECT 8

B.

%Scrieţi un program PROLOG care determină dintr-o listă formată din numere întregi lista subşirurilor cu cel puțin 2 elemente,

%formate din elemente în ordine strict crescătoare. Se vor scrie modelele matematice și modelele de flux pentru predicatele

%folosite.

%Exemplu- pentru lista [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]] (nu

%neapărat în această ordine)

%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

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),

insert(H, R, R1).

%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)

%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)

%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)

%oneSolution of the problem

oneSolution(\_, 0, \_).

oneSolution(L, K, AK):-

arr(L, K, AK),

myLength(AK, LENGTH),

LENGTH >= 2,

strictlyAsc(AK).

%allSolutions of the problem

allsolutions(\_, 0, \_).

allsolutions(L, K, RR) :-

findall(RPartial, oneSolution(L, K, RPartial), R1),

K1 is K - 1,

insertList(R2, R1, RR),

allsolutions(L, K1, R2).

%wrapper function for allSolutions

wrapperAllSolutions(L, R):-

myLength(L, LENGTH),

allsolutions(L, LENGTH, R).

C.

;Un arbore n-ar se reprezintă în LISP astfel (nod subarbore1 subarbore2 .....). Se cere să se determine numărul de noduri de

;pe nivelul k. Nivelul rădăcinii se consideră 0. Se va folosi o funcție MAP.

;Exemplu pentru arborele (a (b (g)) (c (d (e)) (f)))

;a) k=2 => nr=3 (g d f) b) k=4 => nr=0 ()

(defun countNodesOnLevel (L LEVEL K)

(cond

((and (atom L) (equal LEVEL K)) 1)

((and (atom L) (not(equal LEVEL K))) 0)

(t( apply '+ (mapcar #' (lambda (S)(countNodesOnLevel S (+ 1 LEVEL) K)) L)))

)

)

(print (countNodesOnLevel '(a (b (g)) (c (d (e)) (f))) -1 4))

SUBJECT 9

B.

%Scrieţi un program PROLOG care determină dintr-o listă formată din numere întregi lista subşirurilor cu cel puțin 2 elemente,

%formate din elemente în ordine strict crescătoare. Se vor scrie modelele matematice și modelele de flux pentru predicatele

%folosite.

%Exemplu- pentru lista [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]] (nu

%neapărat în această ordine)

%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

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),

insert(H, R, R1).

%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)

%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)

%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)

%oneSolution of the problem

oneSolution(\_, 0, \_).

oneSolution(L, K, AK):-

arr(L, K, AK),

myLength(AK, LENGTH),

LENGTH >= 2,

strictlyAsc(AK).

%allSolutions of the problem

allsolutions(\_, 0, \_).

allsolutions(L, K, RR) :-

findall(RPartial, oneSolution(L, K, RPartial), R1),

K1 is K - 1,

insertList(R2, R1, RR),

allsolutions(L, K1, R2).

%wrapper function for allSolutions

wrapperAllSolutions(L, R):-

myLength(L, LENGTH),

allsolutions(L, LENGTH, R).

C.

;Se consideră o listă neliniară. Să se scrie o funcţie LISP care să aibă ca rezultat lista iniţială în care toate apariţiile unui

;element e au fost înlocuite cu o valoare e1. Se va folosi o funcție MAP.

;Exemplu

;a) dacă lista este (1 (2 A (3 A)) (A)) e este A şi e1 este B => (1 (2 B (3 B)) (B))

;b) dacă lista este (1 (2 (3))) şi e este A => (1 (2 (3)))

(defun replaceElements (L E E1)

(cond

((and (atom L) (eq L E) (list E1)))

((and (atom L) (not(eq L E)) (list L)))

(t (list (apply #'append (mapcar #'(lambda (S) (replaceElements S E E1)) L))))

)

)

(print (replaceElements '(1 (2 A (3 A)) (A)) 'A 'B))

SUBJECT 10

B.

%Să se scrie un program PROLOG care generează lista aranjamentelor de k elemente dintr-o listă de numere întregi, având o

%sumă S dată. Se vor scrie modelele matematice și modelele de flux pentru predicatele folosite.

%Exemplu- pentru lista [6, 5, 3, 4], k=2 şi S=9⇒ [[6,3],[3,6],[5,4],[4,5]] (nu neapărat în această ordine)

%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).

%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, K, S, ARR) :-

arrangements(L, K, ARR),

sum(ARR, SUM),

SUM =:= S.

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

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

C.

Un arbore n-ar se reprezintă în LISP astfel ( nod subarbore1 subarbore2 .....). Se cere să se determine înălțimea unui nod în

;arbore. Se va folosi o funcție MAP.

;Exemplu pentru arborele (a (b (g)) (c (d (e)) (f)))

;a) nod=e => înălțimea e 0 b) nod=v => înălțimea e -1 c) nod=c => înălțimea e 2

(defun maxLevel (L LEVEL)

(cond

((atom L) LEVEL)

(t (apply #' max (mapcar #' (lambda (S) (maxLevel S (+ 1 LEVEL))) L)))

)

)

(defun findHeight(L MAXLEVEL E)

(cond

((and (atom L) (eq L E)) MAXLEVEL)

((and (atom L) (not(eq L E))) -1)

(t (apply '\*(mapcar #' (lambda (S) (findHeight S (- MAXLEVEL 1) E)) L)))

)

)

(defun main(L E)

(findHeight L (maxLevel L 0) E)

)

(print (main '(a (b (g)) (c (d (e)) (f))) 'v ))

SUBJECT 11

B.

%Să se scrie un program PROLOG care generează lista combinărilor de k elemente cu numere de la 1 la N, având diferența

%între două numere consecutive din combinare număr par. Se vor scrie modelele matematice și modelele de flux pentru

%predicatele folosite. Exemplu- pentru N=4, k=2 ⇒ [[1,3],[2,4]] (nu neapărat în această ordine)

%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)

%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)

%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]):-

difference(H1, H2, DIF),

DIF mod 2 =:= 0.

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

difference(H1, H2, DIF),

DIF mod 2 =:= 0,

checkProperty([H2|T]).

generateListFrom1ToN(0, \_, []).

generateListFrom1ToN(N, N, [N]).

generateListFrom1ToN(N, K, [K|R]):-

K1 is K + 1,

generateListFrom1ToN(N, K1, R).

onesolution(L, K, COMB) :-

combinations(L, K, COMB),

checkProperty(COMB).

allsolutions(N, K, R) :-

generateListFrom1ToN(N, 1, GENLIST),

findall(RPartial, onesolution(GENLIST, K, RPartial), R).

C.

;Se dă o listă neliniară și se cere înlocuirea valorilor numerice pare cu numărul natural succesor. Se va folosi o funcție MAP.

;Exemplu pentru lista (1 s 4 (2 f (7))) va rezulta (1 s 5 (3 f (7))).

(defun replaceWithSuccessor (L)

(cond

((and (numberp L) (eq (mod L 2) 0)) (list (+ 1 L)))

((atom L) (list L))

(t (list (apply #'append (mapcar #'(lambda (S) (replaceWithSuccessor S)) L))))

)

)

(print (replaceWithSuccessor '(1 s 4 (2 f (7))) ))

SUBJECT 12

B.

%Să se scrie un program PROLOG care generează lista submulţimilor de sumă S dată, cu elementele unei liste, astfel încât

%numărul elementelor pare din submulțime să fie par. Exemplu- pentru lista [1, 2, 3, 4, 5, 6, 10] şi S=10 ⇒ [[1,2,3,4], [4,6]].

%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)

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

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

sum([], 0).

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

sum(T, S2),

S is S2 + H.

checkCond(L, S):-

sum(L, SUM),

SUM =:= S.

countEvenNumbers([], 0).

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

H mod 2 =:= 0,

countEvenNumbers(T, C1),

C is C1 + 1, !.

countEvenNumbers([\_|T], C):-

countEvenNumbers(T, C).

onesolution(L, S, SUBS) :-

subset(L, SUBS),

checkCond(SUBS, S),

countEvenNumbers(SUBS, COUNTER),

COUNTER mod 2 =:= 0.

allsolutions(L, S, R) :-

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

C.

;Un arbore n-ar se reprezintă în LISP astfel (nod subarbore1 subarbore2 .....). Se cere să se determine lista nodurilor de pe

;nivelul k. Nivelul rădăcinii se consideră 0. Se va folosi o funcție MAP.

;Exemplu pentru arborele (a (b (g)) (c (d (e)) (f)))

;a) k=2 => (g d f)

;b) k=5 => ()

(defun my\_append (l k)

(if (null l)

k

(cons (car l) (my\_append (cdr l) k))

)

)

(defun NodesFormKLevel (L K LEVEL)

(cond

((and (atom L) (eq K LEVEL)) (list L))

((atom L) nil)

(t (list (apply #' append(mapcar #' (lambda(S) (NodesFormKLevel S K (+ 1 LEVEL))) L))))

)

)

(defun main (L K)

(car (NodesFormKLevel L K -1))

)

(print (main '(a (b (g)) (c (d (e)) (f))) 2))

SUBJECT 13