Seminar 7 – Recap

Prolog

1. Remove from a list all the elements that occur multiple times. Ex: [1,2,3,2,1] => [3].

This solution is made of 3 predicates:

* exists: checks if an element appears in a list
* removeElem: removes all occurrences of an element from a list
* solution: removes from a list the elements that occur multiple times.

%exists(L:list, E:element) (I, I)

exists([H|T], E):- E = H.

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

Exists([1,2,3,2,1], 2)

True;

True;

false

%removeElem(L:list, E:element, R:list) (I, I, o)

removeElem([], X, []).

removeElem([H|T], X, R):- H = X, removeElem(T, X, R).

removeElem([H|T], X, [H|R]):- removeElem(T, X, R).

removeElem([1, 2, 3, 2, 1], 2, R).

R = [1, 3, 1];

R = [1, 3, 2, 1];

R = [1, 2, 3, 1];

R = [1, 2, 3, 2, 1]

%solution(L:list, R:list) (I, o)

solution([], []).

solution([H|T], S):- exists(T, H), removeElem(T, H, R), !, solution(R, S).

solution([H|T], [H|R]):- not(exists(T, H)), solution(T, S).

[2,3,2,1] 2 is in [3,2,1] => true removeElem([3,2,1], 2, R) => R = [3, 1]

[2,1] 2 is in [1] => false

Is this code correct? Is it going to work correctly?



%comb(L:list, N:integer, R:list) (I, I, o)

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

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

comb(T, N, R).

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

N > 1,

N1 is N – 1,

comb(T, N1, R).

Comb([1,3,2], 2, R).

R = [1,3]

R = [1,2]

R = [3, 2]

Generate only the combinations where elements are in increasing order.

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

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

comb(T, N, R).

comb([H|T], N, [H|[H1 | R]]):-

%H < H1,

N > 1,

N1 is N –1,

comb(T, N1, [H1|R]),

H < H1.

Is this going to work?

Lisp

a.

(setq a ‘(1 2 3 4))

(setq b ‘a)

(setq c ‘(length ‘(1 2 3 4)))

a / (print a) => (1 2 3 4)

b/ (print b) => A

c / (print c) => (length ‘(1 2 3 4))

(eval c) => 4

(eval b) => (1 2 3 4)

(eval a) => Error. 1 is not a function name.

b.

(setq car ‘cdr)

(car ‘(1 2 3 4)) => 1

(eval car ‘(1 2 3 4)) => Error. Too many params for EVAL.

(eval (cons car ‘(1 2 3 4))) => Error. Too many params for CDR

(cdr 1 2 3 4)

(eval (list car ‘(1 2 3 4))) =>Error. 1 is not a function name

(cdr (1 2 3 4))

(eval (list car ‘ ‘(1 2 3 4))) => (2 3 4)

(cdr ‘(1 2 3 4))

Car / (print car) => cdr

(eval car) => Error. Variable CDR has no value.

(apply #’car ‘(1 2 3 4)) <=> (car 1 2 3 4) => Error. Too many params for CAR.

(apply #’car ‘((1 2 3 4))) => 1

(apply car ‘((1 2 3 4))) => (2 3 4)

(apply car ‘(1 2 3 4)) => Error. Too many params for CDR

(funcall #’car ‘((1 2 3 4))) <=> (car ‘((1 2 3 4)) ) => (1 2 3 4)

(funcall car ‘((1 2 3 4))) => NIL

(funcall #’car ‘(1 2 3 4)) => 1

(funcall car ‘(1 2 3 4)) => (2 3 4)

c.

(mapcar #’list ‘(1 2 3 4 5)) <=> (list (list 1) (list 2) (list 3) (list 4) (list 5)) => ((1) (2) (3) (4) (5))

(mapcan #’list ‘(1 2 3 4 5)) <=> (nconc (list 1) (list 2) (list 3) (list 4) (list 5)) => (1 2 3 4 5)

(maplist #’list ‘(1 2 3 4 5)) <=>(list (list ‘(1 2 3 4 5)) (list ‘(2 3 4 5)) (list ‘(3 4 5)) (list ‘(4 5)) (list ‘(5))) => (((1 2 3 4 5)) ((2 3 4 5)) ((3 4 5)) ((4 5)) ((5)))

(mapcon #’list ‘(1 2 3 4 5))<=> (nconc (list ‘(1 2 3 4 5)) (list ‘(2 3 4 5)) (list ‘(3 4 5)) (list ‘(4 5)) (list ‘(5))) => ((1 2 3 4 5) (2 3 4 5) (3 4 5) (4 5 ) (5))

(apply #’append (mapcon #’list ‘(1 2 3 4 5))) =>

(apply #’append ‘((1 2 3 4 5) (2 3 4 5) (3 4 5) (4 5 ) (5))) => (1 2 3 4 5 2 3 4 5 3 4 5 4 5 5)

d. (mapcar #’(lambda (a b) (eval (list a b))) ‘(list max min evenp) ‘(1 2 3 4 5 6))

(lambda list 1) => (list 1) => (1)

(lambda max 2) => (max 2) => 2

(lambda min 3) => (min 3) => 3

(lambda evenp 4) => (evenp 4) => t

=> ((1) 2 3 t)

e.

(setq x ‘(1 2 3 4 5))

(setq y ‘(6 7 8 9 10 11 12))

(mapcar #’(lambda (a b c d) (eval (funcall c d a b)))

X

Y

(mapcar #’(lambda (q) ‘list) y); (list list list list list list list)

(mapcar #’(lambda(v) ‘+) x) ; (+ + + + +)

) => (7 9 11 13 15)