A paper with text on it

Description automatically generated

1.

(defun f (g l)

(lambda(x)

(cond

((null l) nil)

((> x 0) (cons x (f (cdr l))))

(t x)

)

(funcall g l)

)

)

JUSTIFICATION:

In order to have only one recursive call (funcall g l) we use a lambda function which will encapsulate the logic and ensure that the recursive call won t be executed multiple times avoiding redundant recursive calls. The function lambda will have as parameter (funcall g l) and will keep the result , making in reausable along the code.

A close up of a paper

Description automatically generated

f([],-1).

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

f(T,S1),

f\_aux(H,S1,S).

f\_aux(H,S1,S):-

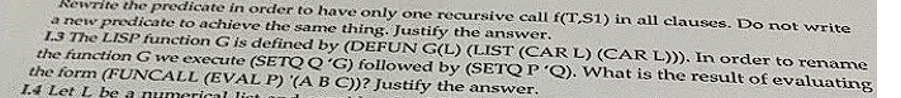
S1 > 0,

!,

S is S1 +H.

f\_aux(\_ , S1,S1).

The auxiliary function declared earlier takes over the decision logic. In order to have only one recursive call f(T,S1) in all clauses , I will use an auxiliary function which will separate the decision logic from the rest of the code. In this way , the structure of the code will be simplified and more clear.



The result of evaluating the form (funcall (eval p) ‘(A B C)) is (A A).The function funcall executed the function(eval p) having as parameters the list (A B C).(Eval p) will evaluate the symbol ro resolve its value which will result in the symbol g.The previous SETQ maps p to the symbol of q which is mapped also to the symbol of q , not the function itself.In the execution of the function a list containing the first element of the list twice will be returned , in our case (A A).

A close up of a paper

Description automatically generated

The result of the following goal will be [] because of the !(cut operator) placed in the second branch.

The cut operator tells Prolog to stop recursing after finding a match. For our example , every element is matched until we get to the [] , which will match the first branch and return []. Because of the cut operator , the last branch won’ t be matched so there will be no elements added in the result list.

A close up of a text

Description automatically generated

%Generating a list with elements from N to 2\*N-1

%generateListWithElementsN( A- the starting value , B – ending value , R – resulted list)

%flow model(i,i,o)

%mathematical model

%generateListWithElementsN(a,b) = { [] , a=b

% a U generateListWithElementsN(a+1,b) , otherwise

% }

generateListWithElementsN(B, B, []) %base case : got to the final element of the range 🡺 return [].

generateListWithElementsN(A, B ,[A|R]):- %appending the element to the resulted list

A1 is A+1, %add 1 to go the next element of the range

generateListWithElementsN(A1,B) %go recursively over all range

%insert an element in a list

%insert(L-list , E – element , R-result list)

%E – element to be inserted in the list

%L -input list

%flow model(i,i,o)

%mathematical model

%insert(l1l2…ln , e) = { e U l1l2..ln

% l1 U insert(l2l3…ln , e)

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

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

insert(E,T,R).

%Permutations of a linear list

%perm(L-list , R-resulted list)

%L – input list

%R – resulted permutation

%flow model(I,o)

%mathematical model

%perm(l1l2…ln) = { [] , n = 0

% insert(l1 , perm(l2l3…ln)) , otherwise

perm([],[]).

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

perm(T,R1),

insert(R1,H,R).

%Determining the solution for our problem

%onesol(N-given number, R-resulted permutation)

% N – given number for generating input list containing all the elements from N to 2\*N -1

%flow model(I,o)

onesol(N,R):-

B is 2\*N,

B1 is B-1,

generateListWithElementsN(N, B1, R),

verifAbsValue(R,1).

%Verify if absolute value between two consecutive values is <=2

%verifAbsValue(L-input List, R-result)

%L-input list

%R – result (1 if the list is valid , 0 otherwise)

%flow model(I,o)

%mathematical model

%verifAbsValue(l1l2…ln) = { 0 , abs(l1-l2) > 2

% 1 , n = 1

% verifAbsValue(l2l3…ln) , otherwise}

verifAbsValue([\_],1).

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

abs(H1-H2) =< 2,

!,

verifAbsValue([H2|T],R).

verifAbsValue([H1,H2|\_],0):-

abs(H1-H2) > 2.

%Determinating all the solutions in a result list

%allsols(N-input number,R-resulted list)

%N – number given for generating permutations between a range containing N

%R – resulted list containing all permutations which are valid

%flow model(I,o)

allsols(N,R):-

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

A close-up of a paper

Description automatically generated

%mathematical model

%multiplyNumbersWithLevel(l , level) = { l\* level , if l is an number

% (union from i=1 ton)multiplyNumbersWithLevel(l,I,level+1) if l is list

% l otherwise

% }

(defun multiplyNumbersWithLevel(l level)

(cond

((numberp l) (+ level l))

((listp l) (mapcar (lambda (x) (multiplyNumbersWithLevel x (+ 1 level)) l)))

(t l)

)

)

A piece of paper with text on it

Description automatically generated

(defun F(L)

(lambda(x)

(cond

((null l )nil)

((listp (car l)) (append x (f (cdr l)) (car x)))

(t (list(car l)))

)

(f (car l))

)

)

To avoid calling the recursive call (f (car l)) multiple times along the code , we are using the lambda function which will take as a parameter exactly (f (car l)) .In this way we encapsulate the logic , ensuring the computation is made only once and there are no redundant recursive calls and also saving the result of the computation and making it reusable along the code.

A close up of a book

Description automatically generated

f(0,0):-!.

f(I,Y):- J is I -1 , j(J,V) , f\_aux(I,V,Y).

f\_aux(I,,V,Y):- V > 1, ! , Y is I-2.

f\_aux(\_,,V,Y):- Y is V+1.

In order to have only one recursive call I am using an auxlilary function. In this way we are separating the decision logic from the rest , making the code more clear and simplified. The auxiliary function takes 3 parameters and performs the necessary logic based on them. In conclusion we avoided the multiple recursive calls by using the auxiliary function.

A close up of a paper

Description automatically generated

The result of evaluating the form (append (F ‘(1 2)) (F ‘(3 4) ‘(5 6))) will return the list (2 3 5 6).

The first executed part of the form is f ’(1 2) which will return (2) because the function gets only one parameter which will be matched with x , so Y will be null and (cdr x) will be returned. For the second function call x will be (3 4) and y will be (5 6). This matches to the t branch which will return a list which concatenates the following elements: (car x) which will be 3 and y which is (5 6). Lastly the append will have to concatenate the list (2) with (3 5 6) resulting the list (2 3 5 6) as output.

A close up of a book

Description automatically generated

The input goal will run infinitely. In function p we case 2 cases. If N is 100 we will return true , otherwise the function keeps printing the number, substracting 1 and then recursively calling the function again. For our input data 0 , the function will never reach the base case p(100) because the number keeps going down and further from 100 and this will cause the recursion to continue indefinitely , printing descending numbers.

A close up of a text

Description automatically generated

%Arrangements of K elements from a list L

%arr(L-list , K – number elements , R – resulted list)

%flow model(I,I,o)

%mathematical model

%arr(l1l2l3…ln , k) = { l1 , if k =1

% arr(l2…ln , k) , if k >= 1

% insert (l1 , arr(l2..ln , k-1)) otherwise

arr([H|\_] , 1 , [H]),

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

arr(T,K,R).

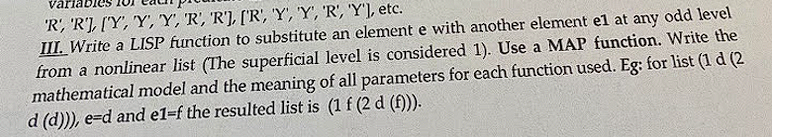
arr([H|T],K,R):-

K > 0,

K1 is K-1,

arr(T,K1,R1),

insert(H,R1,R).



(defun substituteAtOddLevel (l e e1 level)

(cond

((and (atom l) (equal l e) (oddp level)) e1)

((atom l) l)

(t (mapcar #’(lambda(x) (substituteAtOddLevel x e e1 (+ 1 level))) l))

)

)

;%mathematical model

;substituteAtOddLevel(l e e1 level) = { e1 , l = e and level is odd

; l , if l is atom

; union(f(l1l2…ln) , e , e1 , (+ 1 level)) , if l is list

A hand holding a piece of paper

Description automatically generated

%Insert an element into a list

%insert(L-list , E- element , R-resulted list)

%flow model(I,I,o)

%mathematical model

%insert (l1l2..ln, e) = { e U l1l2…ln,

% l1 U insert(l2l3…ln , e)

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

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

insert(T,E,R).

%Arrangements of k elements from a list L

%arr(L-list , K -number of elements , R-resulted list)

%flow model(I,I,o)

%mathematical model

%arr(l1l2..ln , k) = { [l1] , k =1

% arr(l2l3..ln,k) , k >=1

% insert(l1 , arr(l2l3..ln , k-1) , otherwise)

%}

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

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

arr(T,K,R).

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

K > 1,

K1 is K-1,

arr(T,K1,R1),

insert(H,R1,R).

%Verifying if a product is less than a value V

%verifyProduct(L-input list, C – current value of product , V – given product, R – result )

%flow model(I,I,I,o)

%mathematical mode

%verifyProduct(l1l2..ln , c , v) = { 0 , n = 0 and c >= v

% 1 , n = 0 and c < v

% verifyProduct(l2l3..ln , c\*l1,v) , otherwise }

verifyProduct([],C,V,0):-

C >= V,

!.

verifyProduct([],C,V,1):\_

C < V,

!.

verifyProduct([H|T],C,P,R):-

C1 is H \* C,

verifyProduct(T,C1,P,R).

%Determine one solution for our problem

%onesol(L-input list , K-number of elements , V -maximum product , R – resulted list)

%onesol(L,K,V,R):-

arr(L,K,R),

verifyProduct(R,1,V,1).

%Determine all valid arrangements

%allsols(L-list , K- number elements , V – maximum product , R -resulted list)

%flow model(I,I,I,o)

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

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

A close up of a paper

Description automatically generated

;mathematical model

; replaceEvenValuesWithNext(l1l2..ln) = { l+1 , l is number and even

; l if l is atom

; union(from i=1 to n)(f (l1) ) where l = l1l2..ln otherwise

;}

(defun replaceEvenValuesWithNext(l)

(cond

((and (numberp l) (= 0 (mod l 2))) (+ 1 l))

((atom(l)) l)

(t (mapcar #’ (lambda(x) (replaceEvenValuesWithNext x)) l))

)

)