

3. Funcția G calculează suma primelor 2 elemente ale unei liste.

(Sefg H 1 F)  $\rightarrow$  H se evaluatează la F

(Set H 1 G)  $\rightarrow$  F se evaluatează la G

H se evaluatează în continuare la F

Evaluarea Riemer: ( $F'((2\ 3\ 4\ 5\ 6))$ ) va da eroare, cînd neconsecvînd cu funcție cu numerele F (F se evaluatează la G, însă și G-ul trebuie evaluat pentru a apela funcție). Un mod prim care ar putea fi rezolvat situația, este evaluarea Riemer: (functie F ' $((2\ 3\ 4\ 5\ 6))$ ), F fiind că data evaluat de funcție la G, iar apoi acesta fiind evaluat la funcție atunci cînd se apără funcția functie.

2. Lista aranjamenteelor de k elemente dintr-o listă de nr. întregi, cînd să sumă să fie.

Idea algoritmului:

1. un predicat ce determină aranjamentele astfel:

- dacă primul el. de listă este egal cu suma și aranjamentul curent care nu -1 elemente  $\Rightarrow$  se poate forma un aranjament valid, așa că se referință listă de primul elem.
- există și cauză cînd pur și simplu se scurge peste elementul curent (cînd ca este singura opțiune sau nu)
- se poate adăuga elementul curent în aranjament dacă mai este pos și dacă nu se depășește suma

2. un predicat ce returnează ca există componență

totale aranjamentele valide, generate de predicatai  
anterioare

3. predicat ce permite inserarea unui element pe  
toate pozițiile unde este

Modele matematice:

inserare( $e_1 \dots e_m, e$ ) =

1.  $e \oplus e_1 \dots e_m$

2.  $e_1 \oplus \text{inserare}(e_2 \dots e_m, e)$

aranjamente( $e_1 \dots e_m, R, S$ ) =

1. ( $e_1$ ), dacă  $m \geq 1$ ,  $R_1 = \perp$  și  $S = e_1$

2. aranjamente( $e_2 \dots e_m, R_2, S$ )

3. inserare(aranjamente( $e_2 \dots e_m, R_{-1}, S - e_1$ ),  $e_1$ ),  
 $R_2 > \perp$ ,  $S - R_1 > 0$

model( $e, R, S$ ) =  $\bigcup \text{permute}(e, R, S)$

• inserare( $L - \text{lista}, E - \text{Element}, R - \text{Lista}$ )

• model de flux: ( $i, l, \alpha$ )  $\rightarrow$  model determinist

inserare( $L, E, [E] \cup S$ ). % cazul în care elem. este

inserat pe prima poziție

inserare( $\Sigma H(TS), E, R$ ): -

inserare( $T, E, R$ ),

$$R = [H|R, T]$$

- % aranjamente ( $L$ -lăstă,  $K$ -Nr,  $S$ -Nr,  $R$ -Lista)
- %  $K \rightarrow$  nr. de elemente ale aranjamentului
- %  $\rightarrow$  pe pozitii indică nr. de elem. care mai trebuie adăugate
- %  $S \rightarrow$  suma elem. din aranjament
- % model de flux:  $(i, i, i, \alpha) \rightarrow$  determinist

aranjamente ( $[H|T], L, S, R$ )

- % cazul în care se construiește întreg aranjamentul

aranjamente ( $[ - | T ], L, S, R$ ): -

- % se sărăcă peste elementele curent

aranjamente ( $T, K, S, R$ ).

aranjamente ( $[H|T], K, S, R$ ): -

$$K > L,$$

$$S > H,$$

- % se inseră elementul curent pe toate pozi. (pe final)

- % ale aranjamentului curent

$$K_1 \text{ is } K - 1,$$

$$S_1 \text{ is } S - H,$$

aranjamente ( $T, K_1, S_1, R_1$ ),

inserarea ( $R_1, H, R$ ).

- % main( $L$ -Lista,  $K$ -Nr,  $S$ -Nr,  $R$ -Lista de liste)
- % returnază lista cu toate aranjamentele valide
- % model de flux:  $(i, i, i, \alpha) \rightarrow$  determinist

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

Rindare (R), aranjamente (L, K, S, R), R).

5. Nr. de subliste de la coacă mîvel pt. care ultimul atom numeric este impar.

Model matematic.

ultimul ( $e_1 \dots e_m, c\in\mathbb{C}$ ) =

$\begin{cases} c\in\mathbb{C}, dacă m=0 (c=c_1\dots c_m) \\ ultimul (e_2 \dots e_m, p_1 \oplus c\in\mathbb{C}), dacă e_1 \in \text{numere} \\ ultimul (e_2 \dots e_m, c\in\mathbb{C}), dacă e_1 \in \text{atom numeric} \\ ultimul (e_2 \dots e_m, (\text{ultimul}(e_i, c\in\mathbb{C}))), altfel \end{cases}$

subliste (L) =  $\begin{cases} 0, dacă L \in \text{atom} \\ 1 + \sum_{i=1}^n \text{subliste}(e_i), dacă L \in \text{listă și} \\ \quad \text{ultim}(L, i) \in \text{impar} \\ \sum_{i=1}^n \text{subliste}(e_i), altfel \end{cases}$

; ultimul (L - lista  $\rightarrow$  c<sub>i</sub> - lista)

; returnarea ultimul atom numeric de la coacă mîvel  
(de la un ultimul L c<sub>i</sub>)

(com)  
; întotdeauna ultimul nr. va fi primul în colectare

((mîvel e) (car c<sub>i</sub>)) (aceasta se construiește în ordine inversă)

; dacă el împreună cu următorul sunt în colectare

((numerele (car e)) (ultimul (cdr e)))  
(cons (car e) c<sub>i</sub>)))

((atom (car e)) (ultimul (cdr e) c<sub>i</sub>)))

; dacă primul element este listă

; se calculează ultimul nr. din primul el.

; și există pozitie din acesta devine nula  
; corectare pt. există curentă (utile pt. cănd  
; există nr. în primul element, doar nu și  
; în restul listei)

( + (ultimul (cod e) (list (ultimul (car e) col))))

)

)

; succiste (e - lista / Atorn)  
; numără succisele valide  
(de laun succiste (e))

Când

((atorn e) o)

; dacă elem. curent e supălistă validă, se adaugă  
; la suma și se continuă verificarea  
; succisele sau

((and (listp e) (not (mem (mem (ultimul e) n) e)))  
(addp (ultimul e) n)))

(+ L (appey #' + (mapcar #' succiste e)) ))  
; atunci, dacă se parcurg succisele

(+ (appey #' + (mapcar #' succiste e)) ))

)