

Labonat

Reguli

- recursivitate
- acces direct doar la elem de la începutul listei
- nu avem acces direct la lungimea listei
- funcții pure
- adăugare la începutul listei : elem \oplus l₁ l₂ ... l_n

Săpt 8

- test Prolog (45 min) : P₁, P₂(a)
 - temă P₃ → predicat mediterminist
- duloug : trær.apl() methace
 ! ca să afișeze totă lista

Laborator P1 :

15)

- a) Să se scrie un predicat care se va satifica dacă o listă are număr par de elemente și va avea în cauză comtrar, să se numere elementele listei.
- b) Să se elimine prima apariție a elementului minim dintr-o listă de mai înțelegi
 $[1, 2, 3, 4, 5, 6, \cancel{7}] \rightarrow \text{false}$
- c) eliminăm căte 2

```
%a
%listaPar(L:lista)
%(i)- determinist
listaPar([]):- !. → are o singură soluție
listaPar([_,_|T]) :- → spunești acolo
    listaPar(T). → eliminăm căte 2 și restul listei se apelează recursiv
    ! → dacă rămâne [-] dă pur și simplu false
%b
%gaseste(L:lista, E:int)
%(i,o) - determinist
gaseste([E], E):- !. → cauză de bază: lista un singur elem → acela e min
gaseste([H|T], M):-
    gaseste(T, M), → comparăm recursiv fiecare elem. aflat primul în subliste cu minimul
    M=<H, → OPREȘTE BACKTRACKING
    !. → primul devine min dacă nu
gaseste([H|_], H). → primul devine min dacă nu

%del(L:lista, LR: lista)
%(i, o) - determinist
del([], []). → cauză de bază: lista e goală, nu avem ce eliminăm
del([H|T], T) :-
    gaseste([H|T], M), → mi se returnează în M minim
    H is M, → dacă H este M nu eprim
    !.
del([H|T], [H|Rez]) :-
    del(T, Rez).
```

a) $\text{listaPar}(l_1, l_2, l_3, \dots, l_m) = \begin{cases} \text{true, } m=0 \\ \text{false, } m=1 \\ \text{listaPar}(l_2, \dots, l_m), \text{ altfel} \end{cases}$

b) $\text{gaseste}(l_1, \dots, l_m, M) = \begin{cases} l_1, \text{ dacă } m=1 \\ \text{gaseste}(l_2, \dots, l_m, M), \text{ dacă } l_1 > M \text{ și } m \neq 1 \\ \text{gaseste}(l_2, \dots, l_m, l_1), \text{ dacă } l_1 \leq M \text{ și } m \neq 1 \end{cases}$

$\text{del}(l_1, \dots, l_m) = \begin{cases} [], \text{ dacă } m=0 \\ \text{del}(l_2, \dots, l_m), \text{ dacă } l_1 = \text{gaseste}(l_1, \dots, l_m, M), \\ l_1 \oplus \text{del}(l_2, \dots, l_m), \text{ altfel} \end{cases}$

↑
listă initială

```
?-
% c:/Users/Antonia/Downloads/UBB Informatica - Computer Science/Semestrul 3/PLF
- Programare logica si functionala/Laboratoare/p1-15.pl compiled 0.00 sec, 8 cla
uses
?- listaPar([1,2,3,4]).
true.

?- listaPar([1,2,3,4,5]).
false.

?- del([7,6,7,8,6,10],Rez).
Rez = [7, 7, 8, 6, 10].

?- del([3,4,-1,4,5],Rez).
Rez = [3, 4, 4, 5].

?- del([-2,-1,3,4,-2,-1,10],Rez).
Rez = [-1, 3, 4, -2, -1, 10].

?- listaPar([]).
true.

?- listaPar([43]).
false.

?- listaPar([2,3,4,4,5,6,9]).
false.

?- del([-3, -1, -4, -2, -4], Rez).
Rez = [-3, -1, -2, -4].

?- del([7], Rez).
Rez = [].

?- del([], Rez).
Rez = [].
```

Laborator | 2 - P2 pb 6

6. a) Înțe-o listă L să se înlocuiască toate aparitările unui element E cu elementele unei alte liste.

$$\text{înloc}(l_1 \dots l_m, E, L_1 \dots L_m) = \begin{cases} [], \text{ dacă } m=1 \\ L_1 \dots L_m \oplus \text{înloc}(l_2 \dots l_m, E, L_1 \dots L_m), \text{ dacă } l_1 = E \text{ și } m \geq 2 \\ \text{înloc}(l_2 \dots l_m, E, L_1 \dots L_m), \text{ altfel} \end{cases}$$

elementul
↓
înloc(l₁...l_m, E, L₁...L_m)
↑
listă inițială listă cu care se înlocuiește

(ex) $\text{înloc}([1, 2, 1, 3, 1, 4], 1, [10, 11], X) \Rightarrow X = [10, 11, 2, 10, 2, 10, 11, 3, 10, 11, 4]$

$$\text{concatenare}(l_1 \dots l_m, L_1 \dots L_m) = \begin{cases} \emptyset, m=1 \\ l_1 \oplus \text{concatenare}(l_2 \dots l_m, L_1 \dots L_m), \text{ altfel} \end{cases}$$

elementele vor fi concatenate
↑
concatenare(l₁...l_m, L₁...L_m)
↑
în lista lor se concatenează

$[1, 2, 3] + [4, 5, 6] \rightarrow [1] [4, 5, 6], [1] [2] [4, 5, 6], [1, 2, 3] [4, 5, 6]$

← backtracking și apelată concatenare

b) Se dă o listă de numere întregi, formată din numere întregi și liste de numere întregi. În fiecare sublistă să se înlocuiască toate aparitările primului element din sublistă cu o listă dată

$$\text{înlocuireB}(l_1 \dots l_m, L_1 \dots L_m) = \begin{cases} \emptyset, m=1 \\ \text{înlocuireB}(l_2 \dots l_m, L_1 \dots L_m), \text{ dacă } l_1 \text{ nu este listă, } m \geq 2 \\ l_1 \oplus \text{înlocuireB}(l_2 \dots l_m, L_1 \dots L_m), \\ \quad \quad \quad \text{dacă } l_1 \text{ este listă, } m \geq 2 \end{cases}$$

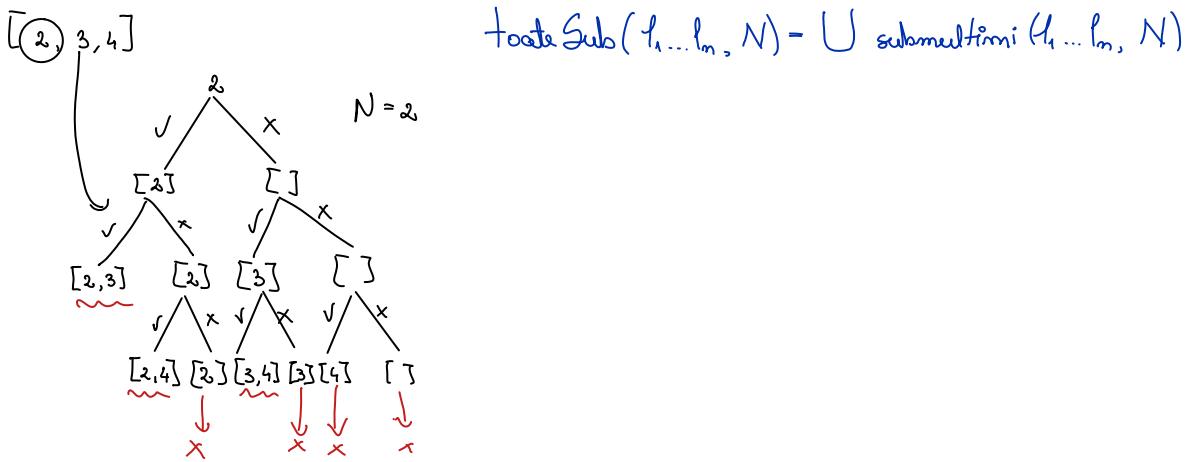
listă inițială listă de înlocuit

$$\text{primul}(l_1 \dots l_m) = \begin{cases} l_1, \text{ orice caz} \end{cases}$$

Laborator P₃

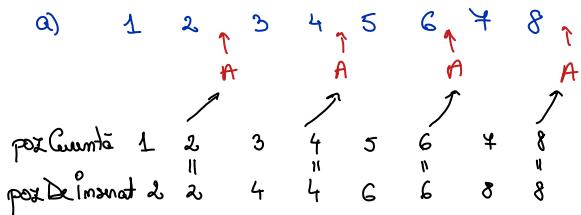
5. Să se genereze lista submultimilor cu m elemente, cu elementele unei liste date
 ex) $[2, 3, 4]$, $N=2 \rightarrow [[2, 3], [2, 4], [3, 4]]$

$$\text{submultimi}(l_1 \dots l_m, N) = \begin{cases} \emptyset, & \text{daca } m=0 \\ \text{submultimi}(l_2 \dots l_m, N-1), & m>0 \\ l_1 \oplus \text{submultimi}(l_2 \dots l_m, N-1), & m>0 \end{cases}$$



Laborator L1

- 1.
- a) Sa se insereze intr-o lista liniara un atom a dat dupa al 2-lea, al 4-lea, al 6-lea,....element.
 - b) Definiti o functie care obtine dintr-o lista data lista tuturor atomilor care apar, pe orice nivel, dar in ordine inversa. De exemplu: (((A B) C) (D E)) --> (E D C B A)
 - c) Definiti o functie care intoarce cel mai mare divizor comun al numerelor dintr-o lista neliniara.
 - d) Sa se scrie o functie care determina numarul de aparitii ale unui atom dat intr-o lista neliniara.



Model matematic:

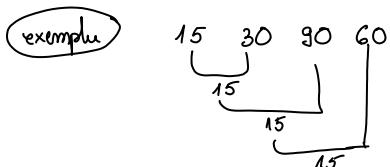
$$\text{inseraza}(t_1 \dots t_m, \text{pozCurunta}, \text{pozDeInserat}, E) = \begin{cases} \emptyset, \text{ dacă } t \text{ vidă} \\ t_1 \oplus \text{inseraza}(t_2 \dots t_m, \text{pozCurunta}+1, \text{pozDeInserat}, E), \text{ dacă } \text{pozCurunta} \neq \text{pozDeInserat} \\ t_1 \oplus E \oplus \text{inseraza}(t_2 \dots t_m, \text{pozCurunta}+1, \text{pozDeInserat}+2, E), \text{ dacă } \text{pozCurunta} = \text{pozDeInserat} \end{cases}$$

$$\text{inserazaMain}(t_1 \dots t_m, E) = \text{inseraza}(t_1 \dots t_m, 1, 2, E)$$

b) $((((A B) C) (D E))) \Rightarrow (E D C B A)$

$$\text{inversaLista}(t_1 \dots t_m) = \begin{cases} \emptyset, \text{ dacă } t \text{ vidă} \\ t_1, \text{ dacă } t_1 \text{ este atom} \\ \text{inversaLista}(t_2 \dots t_m) \oplus \text{inversaLista}(t_1), \text{ altfel} \end{cases}$$

c) $\text{cmmndc}(A, B) = \begin{cases} A, \text{ dacă } B = 0 \\ \text{cmmndc}(B, A \% B), \text{ altfel} \end{cases}$



Algoritmul lui Euclid

cât timp $B \neq 0$

$$R \leftarrow A \bmod B$$

$$A \leftarrow B$$

$$B \leftarrow R$$

$$\text{CMMDC} = A$$

$$\text{cmmndcLista}(t_1 \dots t_m, D) = \begin{cases} D, \text{ dacă } t \text{ vidă} \\ \text{cmmndcLista}(t_2 \dots t_m, \text{cmmndc}(D, t_1)), \text{ altfel} \end{cases}$$

$$\text{cmmndcMain}(t_1 \dots t_m) = \text{cmmndcLista}(t_1 \dots t_m, t_1)$$

d) $\left[1 \ 2 \ (1 \ 2) \ 4 \ 5 \ (1 \ (4 \ (8 \ (1)))) \ 2 \right] \xrightarrow{\text{mn Aparitii } 1} 4$

$$\text{mn Aparitii } (l_1 \dots l_m, E) = \begin{cases} 0, \text{ dacă } l_1 \neq \text{vădă} \\ 1 + \text{mn Aparitii } (l_2 \dots l_m, E), \text{ dacă } l_1 = \text{atom și } l_1 = E \\ \text{mn Aparitii } (l_2 \dots l_m, E), \text{ dacă } l_1 = \text{atom și } l_1 \neq E \\ \text{mn Aparitii } (l_1, E) + \text{mn Aparitii } (l_2 \dots l_m, E), \text{ dacă } l_1 \in \text{listă} \end{cases}$$

Laborator L2

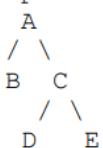
Pentru urmatoarele probleme se cer functii Lisp programate in mod recursiv (eventual folosind functii MAP):

Un arbore binar se memoreaza in urmatoarele doua moduri

(nod nr-subbarbri lista-subarbore-1 lista-subarbore-2 ...) (1)

(nod (lista-subarbore-1) (lista-subarbore-2)) (2)

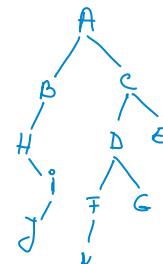
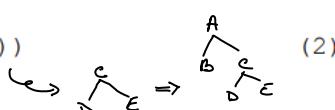
De exemplu arborele



se poate reprezenta astfel in cele doua moduri:

(A 2 B 0 C 2 D 0 E 0) (1)

(A (B) (C (D) (E))) (2)



(A 2 B 1 H 1 I 1 J 0 C 2 D 2 F 1 K 0 G 0 E 0)

Exceptand problemele 6 and 7, nu este permisa conversia intre tipuri - se vor folosi metode directe.

4) Să se convertescă un arbore de tipul (2) la un arbore de tipul (1)

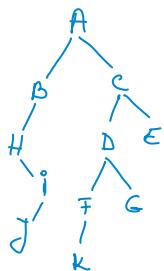
$$(A (B) (C (D) (E))) \xrightarrow{\text{red}} (A \ 2 \ B \ 0 \ C \ 2 \ D \ 0 \ E \ 0)$$

$$\text{conversie Arbore Tip 1 } (l_1 \dots l_m) = \begin{cases} \emptyset, m=0 \\ l_1 \oplus 0, m=1 \\ l_1 \oplus m-1 \oplus \text{conversie Subarbore Tip 1 } (l_2 \dots l_m), \text{ altfel} \end{cases}$$

$$\text{conversie Subarbore Tip 1 } (l_1 \dots l_m) = \begin{cases} \emptyset, \\ \text{conversie Arbore Tip 1 } (l_1) \oplus \text{conversie Subarbore Tip 1 } (l_2 \dots l_m), \text{ altfel} \end{cases}$$

valoarea
modului
mn de
subarbore
reprezentarea
subarborelor

Laborator:

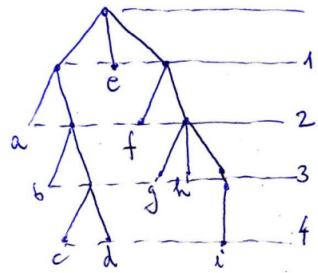


(A (B (H (i(j)))) (C (D (F (K) (G)) (E))))

A 2 B1 H1 i1 j0 C2 D2 F1 K0 G0 E0

Lăboanță L₃ → funcții MAP

1) Să se construiască o funcție care întoarce adâncimea unei liste



$$((a(b(ed)))) \quad e(f(g(h(i))))$$

$$(1 \ 2 \ 3 \ 4) \rightarrow 1$$

$$(1 \ 2 \ (3 \ 4) \ 5 \ (6 \ (7)))))$$

0, dacă $m=1$

$$\text{adâncime}(l_1, l_2, \dots, l_m) = \begin{cases} 1 + \max(\text{adâncime}(l_1), \text{adâncime}(l_2), \dots, \text{adâncime}(l_m)), & \text{altfel} \\ 0, & \text{dacă } m=1 \end{cases}$$

• bonus seminare

2 oct 2024

Seminariu 1 Recursivitate

I Algoritmi recursivi cu numere

① Se dă un număr natural m . Să se verifice dacă m este monosos. Numim un număr monosos dacă este format exclusiv din cifrele 2 și 4.

(ex) 224 \Rightarrow adevărat

2124 \Rightarrow fals

22 \Rightarrow adevărat

0 \Rightarrow fals

$$\text{monosos}(m) = \begin{cases} \text{Adevărat, (dacă) } m=2 \text{ sau } m=4 \quad / \quad m \in \{2, 4\} \\ \text{Fals, } m \% 10 \notin \{2, 4\} \\ \text{monosos}(\lfloor m/10 \rfloor), \text{ altfel} \end{cases}$$

model matematic, soluție formală

soluție Python:

```
def monosos(m):
    if m == 2 or m == 4:
        return True
    if m % 10 != 2 and m % 10 != 4:
        return False
    return monosos(m // 10)
```

② Se dă un număr natural $m > 1$. Se cere să se calculeze suma divizorilor proprii și improprii ai lui m .

(ex) $m = 6 \Rightarrow 1 + 2 + 3 + 6 = 12$

$$\text{sumadiv}(m, k_0) = \begin{cases} 0, \quad k > m \\ k + \text{sumadiv}(m, k+1), \quad k \mid m \\ \text{sumadiv}(m, k+1), \text{ altfel} \end{cases}$$

soluție Python:

```
def sumadiv(m, k):
    if k > m:
        return 0
    if m % k == 0:
        return k + sumadiv(m, k+1)
    return sumadiv(m, k+1)
```

$$\text{suma}(m) = \text{sumadiv}(m, 1)$$

$$\text{sumaDiv2}(m, k, \text{sum}) = \begin{cases} \text{sum}, k > m \\ \text{sumDiv2}(m, k+1, \text{sum} + k), k \mid m \\ \text{sumDiv2}(m, k+1, \text{sum}), \text{altfel} \end{cases}$$

$$\text{suma}(m) = \text{sumaDiv2}(m, 1, 0)$$

- TAD Listă → indexată, conținează ordinea l_1, l_2, \dots, l_m

DA	NU
• $m=0, m=1, m>3$	• $m>k$ (k var), $m \% 2 = 0$
• accesare l_1, l_2, l_3, \dots	• l_k (k var), l_m
• $l_2 \dots l_m$ (sublistă)	• $l_{k+1}, l_{k+2}, \dots, l_m$ (k var)
• $\ell l_1 l_2 \dots l_m$ (înșiruire)	• $l_1 \dots l_m \in$ (înșiruire final / poze k)
$\ell \oplus l_1 l_2 \dots l_m$	
$\ell \cup l_1 l_2 \dots l_m$	

③ Se dă o listă de numere naturale. Se cere să se calculeze produsul elementelor pare.

$$\text{ex } \ell = [51, 2, 13, 4, 5] \Rightarrow 2 * 4 = 8$$

$$\text{produs}(\ell_1 \dots \ell_m) = \begin{cases} 1, m=0 \\ \ell_1 * \text{produs}(\ell_2 \dots \ell_m), m>0 \text{ și } \ell_1 \% 2 = 0 \\ \text{produs}(\ell_2 \dots \ell_m), \text{ altfel} \end{cases}$$

$$\text{eVida}(\ell) \Rightarrow T/F$$

$$\text{prim}(\ell_1 \dots \ell_m) \Rightarrow \ell_1$$

$$\text{sublista}(\ell_1 \dots \ell_m) \Rightarrow \ell_2 \dots \ell_m$$

$$\text{cruaza}() \Rightarrow []$$

$$\text{adauga / incarcat}(\ell, \ell_1 \dots \ell_m) \Rightarrow \ell \ell_1 \dots \ell_m$$

soluție Python

```
def produs(t):
    if eVida(t):
        return 1
    if prim(t) % 2 == 0:
        return prim(t) * produs(sublista(t))
    else:
        return produs(sublista(t))
```

④ Se dă o listă l , o poziție $p \geq 1$ și un element e . Se cere să se inserzeze e în lista l pe poziția p .

(ex) $l = [1, 2, 3, 4, 5]$

$e = 0$

$$p = 3 \Rightarrow [1, 2, 0, 3, 4, 5]$$

$$p = 1 \Rightarrow [0, 1, 2, 3, 4, 5]$$

$$p = 6 \Rightarrow [1, 2, 3, 4, 5, 0]$$

$$p = 10 \Rightarrow [1, 2, 3, 4, 5]$$

$$\text{inserarea}(l, e, p) = \begin{cases} [e], m=0 \wedge p=1 \\ [], m=0 \wedge p>1 \leftarrow \text{listă vidă} \\ [e \ t_1 \dots t_m], m>0 \wedge p=1 \\ t_1 \oplus \text{inserarea}(t_2 \dots t_m, e, p-1), \text{ altfel} \end{cases}$$

$$\begin{aligned} \text{inserarea}([1, 2, 3], 0, 4) &= 1 \oplus \text{inserarea}([2, 3], 0, 3) \\ &= 1 \oplus 2 \oplus \text{inserarea}([3], 0, 2) \\ &= 1 \oplus 2 \oplus 3 \oplus \text{inserarea}([], 0, 1) \\ &= 1 \oplus 2 \oplus 3 \oplus [0] \hookrightarrow [1, 2, 3, 0] \end{aligned}$$

soluție Python

dif inserarea(l , e , p):

if eVida(l) and $p == 1$:

return adaugaInceput(e , creaLista())

elif eVida(l):

return creaLista()

elif $p == 1$:

return adaugaInceput(e , l)

return adaugaInceput(prim(l), inserarea(sublista(l)), e , $p-1$)

⑤ Se dă o listă l , un element e și o valoare $p \geq 1$. Se cere să inserzi e în l din p în p .

(ex) $l = [1, 2, 3, 4, 5, 6]$

$e = 0$

$$p = 3 \Rightarrow [1, 2, 0, 3, 4, 0, 5, 6, 0]$$

$$\text{inserarea}(l_1 \dots l_m, e, p, p_i) = \begin{cases} [e], m=0 \wedge p=1 \\ [], m=0 \wedge p>1 \\ e \oplus \text{inserarea}(l_1 \dots l_m, e, p_i, p_i), m>0 \wedge p=1 \\ l_1 \oplus \text{inserarea}(l_2 \dots l_m, e, p-1, p_i), \text{ altfel} \end{cases}$$

variantă 2 :

$$\text{adaugare}(l_1 \dots l_m, e, p, i) = \begin{cases} [e], m=0 \text{ și } i \% p = 0 \\ [], m=0 \text{ și } i \% p \neq 0 \\ e \oplus \text{adaugare}(l_1 \dots l_m, e, p, i+1), m>0 \text{ și } i \% p = 0 \\ l_i \oplus \text{adaugare}(l_1 \dots l_{i-1}, e, p, i+1), altfel \end{cases}$$

Semimana 2

Liste în Prolog

① Să se scrie predicat care elimină dintr-o listă toate elementele care apar o singură dată.
 (ex) $[1, 2, 1, 4, 1, 3, 4] \Rightarrow [1, 1, 4, 1, 4]$ $[1, 2, \cancel{2}, \cancel{2}, 1, \cancel{2}, 6, 6] \Rightarrow [1, 2, 2, 1, 2, 6, 6]$

Cum determinăm dacă un elem. apare o singură dată?

↪ un predicat care numără de câte ori apare un element într-o listă

$$\text{mn Apariții}(\ell_1 \ \ell_2 \dots \ell_m, e) = \begin{cases} 0, & \text{dacă } m=0 \\ 1 + \text{mn Apariții}(\ell_2 \dots \ell_m, e), & \text{dacă } \ell_1 = e \text{ și } m \neq 0 \\ \text{mn Apariții}(\ell_2 \dots \ell_m, e), & \text{altfel} \end{cases}$$

! funcția trebuie apelată
pt lista originală, nu pt lista din care am tot eliminat elem.

% el = integer
% list = el*

%
% mn Apariții(L: list, E: int, R: integer)
% rezultatul

% model de flex: (i,i,o) sau (i,i,i)

% L: lista în care numărăm ap. elementului E

% E: elementul al cărui apariții le vom număra în lista L

% R: (rezultatul) numărul de apariții ale lui E în lista L

mn Apariții([], _ , 0). ← cazul de bază

prințul elum ↓ rezultatul listei ↓ if

mn Apariții([A | L], E, R) :-

A = E

mn Apariții(L, E, R₁),

R is R₁ + 1.

mn Apariții([A | L], E, R) :-

A \= E,

mn Apariții(L, E, R).

Operatori:

= pt variabile legate, variabile - constantă, constante (comparații simple)

is evaluarea partea dreaptă, dacă în stanga

e constantă compară
e variabilă nelegată → atenție

= : = evaluare

$$\text{elimina}(\ell_1 \dots \ell_m, L_1 L_2 \dots L_m) = \begin{cases} \emptyset, & \text{daca } m=0 \\ \text{elimina}(\ell_2 \dots \ell_m, L_1 L_2 \dots L_m), & \text{daca } m \text{ Aparutii}(L_1 L_2 \dots L_m, \ell_1) = 1 \text{ si } m>0 \\ \ell_1 \cup \text{elimina}(\ell_2 \dots \ell_m, L_1 L_2 \dots L_m), & \text{altfel} \end{cases}$$

$\text{apel}(\ell) = \text{elimina}(\ell, \ell)$

- $\text{elimina}(L: \text{List}, L\Theta: \text{List}, R: \text{List})$
- model de flux: (i, i, o) sau (i, i, i)
- L - lista din care se elimină elem. care aparțin și singură dată
- $L\Theta$ - o copie a listei originale, folosită pentru numărarea aparituirilor
- R - lista rezultată, statutul prim eliminarea elem. care aparțin și singură dată

$\text{elimina}([], -, []).$

$\text{elimina}([H | T], L\Theta, R):-$

 m Aparutii(L\Theta, H, N),

 N > 1,

 elimina(T, L\Theta, R1)

 R = [H | R1]

$\text{elimina}([H | T], L\Theta, R):-$

 m Aparutii(L\Theta, H, N),

 N = 1,

 elimina(T, L\Theta, R)

② Dămdu-se o listă finită numerică să se întoarcă toate secvențele de valori crescătoare.

ex) $[2, 1, 3, 5, 2, 4, 1, 6, 4, 6, 5] \Rightarrow [2, 6, 5]$

$$\text{eliminaGusc}(\ell_1 \dots \ell_m, F) = \begin{cases} [\], m=0 \\ [\ell_1], m=1 \text{ și } F=0 \\ [\] , m=1 \text{ și } F=1 \\ \ell_1 \oplus \text{elimina}(\ell_2 \dots \ell_m, 0), m \geq 2, \ell_1 \geq \ell_2, F=0 \\ \text{elimina}(\ell_2 \dots \ell_m, 1), m \geq 2, \ell_1 < \ell_2 \\ \text{elimina}(\ell_2 \dots \ell_m, 0), m \geq 2, \ell_1 \geq \ell_2, F=1 \end{cases}$$

$$\text{elimP}(\ell) = \text{eliminaGusc}(\ell, 0)$$

% $\text{eliminaGusc}(L: \text{list}, F: \text{int}, R: \text{list})$

% model de flux: $(i, i, o) \xrightarrow{\text{sam}} (i, i, i)$

listă dintr-un singur elem
 $[H] \Leftrightarrow [H | []]$

$\text{eliminaGusc}([\], - , [\]).$

$\text{eliminaGusc}([H | L], 0, [H]).$

$\text{eliminaGusc}([-], 1, [\]).$

$\text{eliminaGusc}([H_1, H_2 | L], 0, [H | R]): -$

$$H >= H_2,$$

$\text{eliminaGusc}([H_2 | L], 0, R).$

$\text{eliminaGusc}([H_1, H_2 | L], - , R): -$

$$H_1 < H_2,$$

$\text{eliminaGusc}([H_2 | L], 1, R).$

$\text{eliminaGusc}([H_1, H_2 | L], 1, R): -$

$$H_1 >= H_2,$$

$\text{eliminaGusc}([H_2 | L], 0, R).$

Semimana 3

Liste heterogene în Prolog

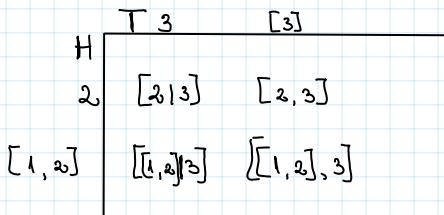
30 oct 2024

[H | T]

poate avea tipuri diferențiale \Rightarrow folosim predicate pentru a verifica

$$\left\{ \begin{array}{l} \text{is_list}(H), \\ \text{number}(H) \\ \text{atom}(H) \end{array} \right| \rightarrow \text{atomic}(H)$$

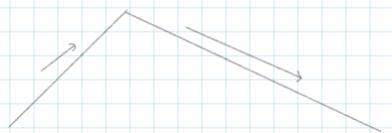
[H | T]



① Se dă o listă heterogenă formată din întregi și liste de întregi. Se cere să se determine numărul de elemente de tip listă având aspect de munte.

Se consideră că o listă are aspect de munte dacă este formată dintr-o secvență strict crescătoare și este urmată de una strict descrescătoare.

Ex. $1, 3, 2, [1, 3, 3, 2], [1, 5, 10], 4, [6, 8, 1], [10, 9, 5, 2], 4, *$ $\Rightarrow 1$



$$\text{verif Munte Cusc } (l_1 \dots l_m) = \begin{cases} \text{fals}, m=1 \\ \text{verif Munte Cusc } (l_2 \dots l_m), l_1 < l_2, m \neq 1 \\ \text{verif Munte Desc } (l_2 \dots l_m), l_1 > l_2, m \neq 1 \\ \text{fals}, l_1 = l_2, m \end{cases}$$

$$\text{verif Munte Desc } (l_1 \dots l_m) = \begin{cases} \text{aduvarat}, m=1 \\ \text{verif Munte Desc } (l_2 \dots l_m), l_1 > l_2, m \neq 1 \\ \text{fals}, l_1 \leq l_2, m \neq 1 \end{cases}$$

$$\text{verif Munte } (l_1 \dots l_m) = \begin{cases} \text{fals}, m < 2 \\ \text{verif Munte Cusc } (l_1 \dots l_m), l_1 < l_2, m > 2 \\ \text{fals}, \text{ altfel} \end{cases}$$

\hookrightarrow prolog returnează implicit fals \Rightarrow nu trebuie implementate limitări

% verif Munte Cusc (Z: lista timpano)
% (?)

verif Munte Cusc ($[H_1, H_2] | T$) :-

$$H_1 < H_2 !,$$

verif Munte Cusc ($[H_2] | T$).

verif Munte Cusc ($[H_1, H_2] | T$) :-

$$H_1 > H_2,$$

verif Munte Desc ($[H_2] | T$).

verif Munte Desc ($[\neg] | T$). [H] dan cu wanming

verif Munte Desc ($[H_1, H_2] | T$) :-

$$H_1 > H_2,$$

verif Munte Desc ($[H_2] | T$)

verif Munte ($[H_1, H_2, H_3] | T$) :-

$$H_1 < H_2,$$

verif Munte Cusc ($[H_1, H_2, H_3] | T$)

varianta 2

$$\text{verif Munte} (l_1 \dots l_m, f) = \begin{cases} \text{fals}, m=0 \\ \text{aduñat}, m=1, f=0 \\ \text{fals}, m=1, f \neq 0 \\ \text{verif Munte} (l_2 \dots l_m, 1), l_1 < l_2, m \geq 2, f = -1 \\ \text{verif Munte} (l_2 \dots l_m, 1), l_1 < l_2, m \geq 2, f = 1 \\ \text{verif Munte} (l_2 \dots l_m, 0), l_1 > l_2, m \geq 2, f = 1 \\ \text{verif Munte} (l_2 \dots l_m, 0), l_1 > l_2, m \geq 2, f = 0 \end{cases}$$

% verif Munte (Z: lista, F: intreg)

% model de flux : (i, i)

verif Munte ($[-], 0$).

verif Munte ($[H_1, H_2] | T$, F) :-

$$H_1 < H_2,$$

$$F \setminus = 0,$$

verif Munte ($[H_2] | T$, 1)

verif Munte ($[H_1, H_2] | T$, F) :-

$$H_1 > H_2,$$

$$F \triangleright -1,$$

verif Munte ($[H_2] | T$, 0).

$$\text{mn Subliste}(\ell_1 \dots \ell_m) = \begin{cases} 0, & m=0 \\ 1 + \text{mn Subliste}(\ell_2 \dots \ell_m), & \text{veziNrunde}(\ell_n, -1), m \geq 1, \ell_1 \text{ este lista} \\ \text{mn Subliste}(\ell_2 \dots \ell_m), & \text{altfel} \end{cases}$$

% mn Subliste(L: lista, R: int)

% modul de flux (i, o)

mn Subliste([], 0).

mn Subliste([H | T], R):-

is_list(H),

vezi(H, -1) !,

mn Subliste(T, R).

R is R + 1

mn Subliste([- | T], R):-

mn Subliste(T, R).

Seminar 4

Backtracking

impar(1)

impar(3)

par(2)

par(4)

par_impar(x,y) :- !, impar(x), !, par(y).

par_impar(x,y) :- !, par(x), !, impar(y).

făță ! $\Rightarrow (1,2), (1,4), (3,2), (3,4), (2,1), (2,3), (4,1), (4,3)$

! $\Rightarrow (1,2), (1,4), (3,2), (3,4)$

! $\Rightarrow (1,2), (1,4)$

! $\Rightarrow (1,2)$

! $\Rightarrow (1,2), (1,4), (3,2), (3,4), (2,1), (2,3), (4,1), (4,3)$

! $\Rightarrow (1,2), (1,4), (3,2), (3,4), (2,1), (2,3)$

! $\Rightarrow (1,2), (1,4), (3,2), (3,4), (2,1)$

① Se dă o listă formată din numere întregi distinse. Se cere să se genereze toate listele cu aspect de vale cu elemente din lista dată

exemplu: $[1, 2, -1, 5, 3] \Rightarrow [1, -1, 5], [1, -1, 3], [3, -1, 1], [5, -1, 3]$ etc



$$\text{candidat}(l_1..l_m) = \begin{cases} l_1, m \neq 0 \\ \text{candidat}(l_2..l_m), m \neq 0 \end{cases}$$

$$\text{candidat}(l_1..l_m) = \begin{cases} (l_1, \emptyset), m \neq 0 \\ (c, l_1 \oplus s), \text{ unde } (c, s) = \text{candidat}(l_2..l_m) \end{cases}$$

% candidat(L: lista de întregi, X: întreg)

% mediterminist

candidat([H|T], H).

candidat([-|T], X) :- candidat(T, X).

de nescis M.M

$$\text{generare } (\mathcal{L}, \mathcal{F}, l_1 \dots l_m) = \left\{ \begin{array}{l} l_1 \dots l_m, \mathcal{F} = -1 \\ \text{generare } (\mathcal{L}_{-1}, \text{cand} \cup l_1 \dots l_m), \text{cand} < \\ \text{generare } (\mathcal{L}, -1, \text{cand} \cup l_1 \dots l_m) \end{array} \right.$$

% generare (\mathcal{L} : list, \mathcal{F} : int, C : list, R : list)

% model de flux (i, i, i, Θ) - mediterimist

generare ($-$, -1 , C , C).

generare (\mathcal{L} , -1 , $[H|C]$, R): - candidat (\mathcal{L} , E),

$E < H$,

generare (\mathcal{L} , 1 , $[E, H|C]$, R).

generare (\mathcal{L} , $-$, $[H|C]$, R): - candidat (\mathcal{L} , E),

\+ candidat ($[H|C]$, E)

$E > H$,

generare (\mathcal{L} , -1 , $[E, H|C]$, R).

apel ($l_1 \dots l_m$) = generare ($l_1 \dots l_m$, 1 , $\underbrace{[E_1, E_2]}_{S_1}$), unde $E_1 = \text{candidat}(l_1 \dots l_m)$, $E_2 = \text{candidat}(l_1 \dots l_m)$, $E_1 < E_2$

$(E_1, S_2) = \text{candidat}(l_1 \dots l_m)$, $(E_1, S_1) = \text{candidat}(S_2)$

% apel (\mathcal{L} : list, R : list)

% model de flux (i, Θ) - mediterimist

apel (\mathcal{L} , R): - candidat (\mathcal{L} , E_1), candidat (\mathcal{L} , E_2 , S_0),

candidat (\mathcal{L} , E_2), candidat (S_0 , E_1 , S_1)

$E_1 < E_2$,

-||-

generare (\mathcal{L} , 1 , $[E_1, E_2]$, R).

de nescis

Seminar 5

Recursivitate în Lisp

→ în Lisp liste sunt strucuri by default

① Se dau 2 liste numerice sortate crescător și formate din elemente distincte.

Să se interclasă cele două liste cu eliminarea dublurilor.

(exemplu) $l = [2, 0, 1, 4]$ $\rightarrow [-2, -1, 0, 1, 2, 4]$
 $k = [-1, 0, 1, 2, 4]$

$\emptyset, m=0, m=0$
 $l_1 < l_m, m=0, m \neq 0$
 $k_1 < k_m, m=0, m \neq 0$
 $l_1 \oplus \text{interclasare}(l_2..l_n, k_2..k_m), l_1 < k_1, m \neq 0, m \neq 0$
 $k_1 \oplus \text{interclasare}(l_1..l_n, k_2..k_m), k_1 < l_1, m \neq 0, m \neq 0$
 $l_1 \oplus \text{interclasare}(l_2..l_n, k_2..k_m), k_1 = l_1, m \neq 0, m \neq 0$

$\text{interclasare}(l_1..l_n, k_1..k_m) =$

(diferența interclasare (l, k)

(cond

((AND (NULL l) (NULL k)) $\rightarrow \emptyset$, false)

((NULL k) l)

((NULL l) k)

((< (car l) (car k)) (cons (car l) (interclasare (cdr l) k)))

((< (car k) (car l)) (cons (car k) (interclasare l (cdr k))))

(T (cons (car l) (interclasare (cdr l) (cdr k))))

)

→ [H | T]

	cons	list	append
'A 'B	(A . B)	(A B)	Error
'A '(B C D)	(A B C D)	(A (B C D))	Error
'(A B C) '(D E F)	((A B C) D E F)	((A B C) (D E F))	(A B C D E F)
'(A B C) 'D	((A B C) . D)	((A B C) D)	(A B C . D) → exception
'A 'B 'C 'D	Error	(A B C D)	Error
'(A B) '(C D) '(E F)	Error	((A B) (C D) (E F))	(A B C D E F)
'(A B) 'C '(E F) 'D	Error	((A B) C (E F) D)	Error
'(A B) '(E F) 'D	Error	((AB) (E F) D)	(A B E F . D)

② Să se elimină toate aparițiile unui atom dat dintr-o listă.

$$\text{ex: } l = [1 \ 2 \ [3 \ 0 \ 4 \ [0 \ 0]] \ [[0 \ 0]]]$$

$$e = 0$$

$$\rightarrow l' = [1 \ 2 \ [3 \ 4 \ []] \ [[\]]]]$$

$$\text{eliminare}(l_1 \dots l_n, e) = \begin{cases} \emptyset, & m=0 \\ \text{eliminare}(l_2 \dots l_m, l_1 \text{ este atom}, m \neq 0, l_1 = e) \\ l_1 \oplus \text{eliminare}(l_2 \dots l_m, l_1 \text{ este atom}, m \neq 0, l_1 \neq e) \\ \underbrace{\text{eliminare}(l_1, e) \oplus \text{eliminare}(l_2 \dots l_m, e)}_{l_1 \text{ este lista}}, & \text{altfel} \end{cases}$$

(defun eliminare (l e)

(cond

((mușc e) NIL)

((equal (car l) e) (eliminare (cdr l) e)))

((atom (car l)) (cons (car l) (eliminare (cdr l) e)))

((T (cons (eliminare (car l) e) (eliminare (cdr l) e)))

vînerei - lumi

acasă: Amca Manu

Silviu ?

Briana ?

David ?

Gaiaști ?

③ Să se determine lista pozitilor elementului numeric minimum dintr-o listă finitară

$$\text{ex: } l = [a \ 1 \ -1 \ 3 \ -1 \ 4] \Rightarrow [3 \ 5]$$

$$\min C = \min l, \text{ col} = \emptyset, \text{ poz} = 1$$

$$\text{listPoz}(l_1 \dots l_n, \min C, \text{col}, \text{poz}) = \begin{cases} \text{col}, & m=0 \\ \text{listPoz}(l_2 \dots l_m, l_1, [\text{poz}], \text{poz}+1), & m \neq 0, \min C = \text{NIL}, l_1 \text{ este numeric} \\ \text{listPoz}(l_2 \dots l_m, l_1, [\text{col} \oplus \text{poz}], \text{poz}+1), & m \neq 0, l_1 = \min C \\ \text{listPoz}(l_2 \dots l_m, l_1, [\text{poz}], \text{poz}+1), & m \neq 0, l_1 < \min C, l_1 \text{ este numeric} \\ \text{listPoz}(l_2 \dots l_m, \min C, \text{col}, \text{poz}+1), & \text{altfel} \end{cases}$$

sx post amâncare

`pozMin (l1 l2 ... ln, minC, pozC, listPoz)`

$$= \begin{cases} \text{listPoz}, & n=0 \\ \text{pozMin}(l_2 \dots l_n, \min C, \text{pozC} + 1, \text{listPoz}), & l_1 \text{ este atom nenumeric} \\ \text{pozMin}(l_2 \dots l_n, l_1, \text{pozC} + 1, (\text{pozC})), & \min C \text{ nu este număr și } l_1 \text{ este număr} \\ \text{pozMin}(l_2 \dots l_n, \min C, \text{pozC} + 1, \text{listPoz} \cup \text{pozC}), & \min C = l_1 \\ \text{pozMin}(l_2 \dots l_n, l_1, \text{pozC} + 1, (\text{pozC})), & l_1 \text{ este număr } l_1 < \min C \\ \text{pozMin}(l_2 \dots l_n, \min C, \text{pozC} + 1, \text{listPoz}), & \text{altfel} \end{cases}$$

```
(defun pozMin (l minC pozC listPoz)
  (cond
    ((null l) listPoz)
    ((not (numberp (car l))) (pozMin (cdr l) minC (+ pozC 1) listPoz))
    ((not (numberp minC)) (pozMin (cdr l) (car l) (+ pozC 1) (list pozC)))
    ((= minC (car l)) (pozMin (cdr l) minC (+ pozC 1) (append listPoz (list pozC))))
    ((< (car l) minC) (pozMin (cdr l) (car l) (+ pozC 1) (list pozC)))
    (t (pozMin (cdr l) minC (+ pozC 1) listPoz)))
  )
)
```

Ne mai trebuie și funcția principală care să facă primul apel la pozMin:

`pozMinMain(l1 ... ln) = pozMin(l1 ... ln, l1, 1, ∅)`

```
(defun pozMinMain (l)
  (pozMin l (car l) 1 nil)
)
```

Semian 6

Funcții MAP

(defun triplexa(x)
 (* x 3))

exemplu '(1 2 3 4) \Rightarrow (3 6 9 12)

(list (triplexa 1) (triplexa 2) (triplexa 3) (triplexa 4))

\implies (mapcar #'triplexa '(1 2 3 4))
 $\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$
 3 6 9 12

(mapcar #'triplexa '(1 A 2 B))

(defun triplexa (x)
 (cond
 ((numberp x) (* x 3))
 ((atom x))
)))
 \implies (3 A 6 B)

(mapcar #'triplexa '(1 (A 2) B)) \Rightarrow (3 (A 6) B)

(defun triplexa (x)
 (cond
 ((numberp x) (* x 3))
 ((atom x))
 ((listp x) (triplexa x)))
)))

\hookrightarrow (triplexa '(1 (A 2) B))
 $\downarrow \quad \downarrow \quad \downarrow$
 (3 (A 6) B)

poate fi lista sau atom

$\begin{cases} 3 * x, \text{ dacă } x \text{ este număr} \\ x \text{ este atom menumerie} \end{cases}$

$\text{triplexa}(x) = \text{triplexa}(x_1) \oplus \dots \oplus [\text{triplexa}(x_m)]$, x este lista, $x = x_1 \dots x_m$

sau

$$\bigcup_{i=1}^m \text{triplexa}(x_i)$$

$$[\text{triplexa}(x_1), \text{triplexa}(x_2), \text{triplexa}(x_3) \dots \text{triplexa}(x_m)]$$

② Să determinăm produsul numerelor dintr-o listă numărătă

(exemplu) $(A \ 1 \ (B \ 3)) \ 4 \ C \Rightarrow 12$

$$\text{produs}(x) = \begin{cases} x, & \text{dacă } x \text{ este nr} \\ 1, & \text{dacă } x \text{ este atom numărătă} \\ \prod_{i=1}^m \text{produs}(x_i), & x \text{ este lista, } x = x_1 \dots x_m \end{cases}$$

(dăjim $\text{produs}(x)$)

(cond)

((numărătăp x) x)

((atom x) 1)

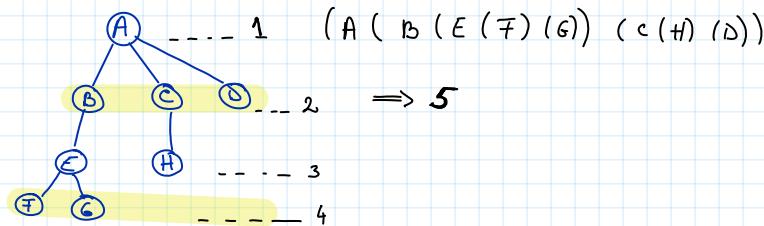
(t(apply '* (mapcar #'produs x)))

))

③ Se dă un arbore m-ar reprezentat subformă $(\text{nod}(\text{subarbore}_1) \dots (\text{subarbore}_m))$

Să se determine numărul total de noduri de pe nivelurile paralele.

Se presupune că nivelul nădejdeanu este 1.



$\text{nivelPan}(x) = \text{nivelPanAux}(x, 0)$

$$\text{nivelPanAux}(x, \text{minv}) = \begin{cases} 1, & x \text{ e atom și } \text{minv} \% 2 = 0 \\ 0, & x \text{ e atom și } \text{minv} \% 2 = 1 \\ \sum_{i=1}^m \text{nivelPanAux}(x_i, \text{minv}+1), & x \text{ lista, } x = x_1 \dots x_m \end{cases}$$

(dăjim $\text{nivelPanAux}(x, \text{minv})$)

(cond)

((and (atom x) (= (mod minv 2) 0)) 1)

((atom x) 0)

(t(apply '+ (mapcar #'nivelPanAux (+ minv 1))))

)) $\hookrightarrow (\lambda(x) (\text{nivelPanAux} (+ 1 minv) x)))$

(4) Se dă o listă de numere. Să se determine numărul sublistelor (inclusiv lista însăși) pe care primul atom numeric (indiferent de nivel) este 5

exemplu: $(A \ 5 \ (B \ (5 \ 1) \ 6 \ C \ (5))) \ (1(5)) \Rightarrow 5$

$$\text{numarau}(x) = \begin{cases} 1 + \sum_{i=1}^m \text{numar}(x_i), & x = x_1 \dots x_m \text{ și } \text{verifica}(x) = T \\ \sum_{i=1}^m \text{numar}(x_i), & x = x_1 \dots x_m \text{ și } \text{verifica}(x) = F \\ 0, & \text{daca } x \text{ e atom} \end{cases}$$

(defun numarau (x)

(cond

((AND (listp x) (verifica x)) (+ 1 (apply '+ (mapcar #'numarau x))))

((listp x) (apply '+ (mapcar #'numarau x))))

((atom x) 0)

))

MAPCAR — list

MAPCAN — merge & append

$$\text{limNum}(x) = \begin{cases} [x], & x \text{ numeric} \\ \bigcup_{i=1}^m \text{limNum}(x_i), & x = x_1 \dots x_m \text{ atom numeric} \end{cases}$$

(defun limNum (x)

(cond

((numberp x) (list x))

((atom x) null)

(t (mapcan #'limNum x))

))

$$\text{verifica}(\varphi) = \begin{cases} A, & \text{limNum}(\varphi) = [5 \dots] \text{ sau } \text{limNum}(\varphi) = x_1 \dots x_n, \text{ unde } x_i = 5 \\ F, & \text{altfel} \end{cases}$$

Seminar 7

Recapitulare

① Se dă o listă binară numerică. Să se steargă toate elementele care se repetă.

$$\text{Ex: } [1, 3, 2, 1, 2, 4, 1, 4] \Rightarrow [3]$$

- există
- sterge Elém
- rezolvă

$$\text{există } (l_1..l_n, e) = \begin{cases} \text{fals}, m=0 \\ \text{adevărat}, l_1 = e, m > 0 \\ \text{există } (l_2..l_n, e), \text{ altfel} \end{cases}$$

% există (L: lista, E: întreg)

% model de flux (i,i) - determinist

există ([H|T], H) :- !

există ([H|T], E) :-

$$\text{există } (T, E)$$

$$\text{sterge Elém } (l_1..l_n, E) = \begin{cases} \emptyset, m=0 \\ \text{sterge Elém } (l_2..l_n, E), m > 0, l_1 = E \\ l_1 \oplus \text{sterge Elém } (l_2..l_n, E), \text{ altfel} \end{cases}$$

% sterge Elém (L: lista, E: întreg, R_L: lista)

% model de flux (i,:,:) :-

sterge Elém ([], _, []) :- !

sterge Elém ([E|T], E, R_E) :-

 sterge Elém (T, E, R_E) !

sterge Elém ([H|T], E, [H|R_E]) :-

 sterge Elém (T, E, R_E).

$$\text{rezolvă } (l_1..l_n) = \begin{cases} \emptyset, m=0 \\ \text{rezolvă } (\text{sterge Elém } (l_2..l_n, l_1)), m > 0, \text{există } (l_2..l_n, l_1) \\ l_1 \oplus \text{rezolvă } (l_2..l_n), \text{ altfel} \end{cases}$$

% rezolva (L : lista, R : lista)
% model de flux (i, Θ)

rezolva ([], []) :- !.

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

 exista (T, H),

 stergeElem (T, H, S)

 rezolva (S, R)

② Se da Θ lista binară numerică. Să se elementele pe poz 1, 3, 4, 15 etc

(ex) $[1, 2, 3, 4, 5, 6, *, 8] \Rightarrow [2, 4, 5, 6, 8]$

$$\text{elimina}([], m=0) \\ \text{elimina}([L_1 .. L_m, pos+1, posN], m \neq posN) \quad ; \\ \text{elimina}([L_1 .. L_m, pos+1, posN * 2 + 1], pos = posN) \quad ; \\ \text{elimina}([L_1 .. L_m, pos+1, posN], m > posN)$$

% elimina (L : lista, Pos : int, $PosN$: int, R : lista)

% model de flux: ($i, i, ;, \Theta$) - determinist

elimina ([], -, -, []).

elimina ([H | T], Pos, PosN, [H | R]) :-

 Pos \neq PosN

 PosNou is Pos + 1

 elimina ($T, PosNou, PosN, R$).

elimina ([- | T], Pos, Pos, R)

 PosNou is Pos + 1

 NextPos is $2^x Pos + 1$

 elimina ($T, PosNou, NextPos, R$)