

Seminar 2 - Liste în Prolog

1. Să se scrie un predicat care elimină dintr-o listă toate elementele care apar o singură dată. De exemplu pentru lista [1,2,1,4,1,3,4] rezultatul va fi [1,1,4,1,4].

Soluție:

Metoda 1

Pentru a determina dacă un element apare o singură dată putem scrie un predicat care să numere de câte ori apare un element într-o listă. Pentru a obține rezultatul corect, predicatul nrApariții trebuie apelat pentru lista originală, nu pentru lista din care am tot eliminat elemente pe parcursul apelurilor recursive. Altfel, de fiecare dată când ajungem la ultima apariție a unui element în listă, numărul de apariții ale elementului în restul listei va fi 0.

$$nrAparitii(l_1 l_2 \dots l_n, e) = \begin{cases} 0 & , \text{dacă } n = 0 \\ nrAparitii(l_2 l_3 \dots l_n, e) + 1 & , \text{dacă } l_1 = e \\ nrAparitii(l_2 l_3 \dots l_n, e) & \text{altfel} \end{cases}$$

```
% el = integer
% list = el*
%
% nrAparitii(L:list, E:el, S:integer)
% model de flux: (i, i, o) sau (i, i, i)
% L - lista în care numaram aparițiile elementului E
% E - elementul al cărui apariții le vom număra în lista L
% S - rezultatul, numărul de apariții ale lui E în lista L

nrAparitii([], _, 0).
nrAparitii([H|T], E, S):-
    H = E,
    nrAparitii(T, E, S1),
    S is S1 + 1.
nrAparitii([H|T], E, S):-
    H \= E,
    nrAparitii(T, E, S).
```

$$elimina(l_1 l_2 \dots l_n, L_1 \dots L_m) = \begin{cases} \emptyset & , \text{dacă } n = 0 \\ elimina(l_2 l_3 \dots l_n, L_1 \dots L_m) & , \text{dacă } nrAparitii(L_1 \dots L_m, l_1) = 1 \\ l_1 \cup elimina(l_2 l_3 \dots l_n, L_1 \dots L_m) & \text{altfel} \end{cases}$$

% elimina(L: List, LO:List, R>List)
% model de flux: (i, i, o) sau (i, i, i)
% L - lista din care eliminam elementele care apar o singură dată
% LO - o copie a listei originale, folosită pentru numărarea aparițiilor
% R - lista rezultat, obținută prin eliminarea elementelor care apar o singură
dată

```
elimina([], _, []).  

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

    nrAparitii(LO, H, S),  

    S = 1,  

    elimina(T, LO, R).  

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

    nrAparitii(LO, H, S),  

    S > 1,  

    elimina(T, LO, R).
```

Varianta alternativă de scriere a predicatului `elimina` folosind predicatul “!” (“cut”):

```
elimina([], _, []).  

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

    nrAparitii(LO, H, S),  

    S = 1, !,  

    elimina(T, LO, R).  

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

    elimina(T, LO, R).
```

La primul apel al funcției `elimină` trebuie să inițializăm lista `LO` cu lista inițială. Pentru acest lucru vom scrie încă o funcție (respectiv un predicat în implementare).

$eliminaMain(l_1 \dots l_n) = elimina(l_1 \dots l_n, l_1 \dots l_n)$

```
% eliminaMain(L: List, R: List)  

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

% L - lista originală din care eliminam elementele care se repeta
```

```
% R - lista rezultat, obținută prin eliminarea elementelor care apar o singură
dată
eliminaMain(L, R):-elimina(L,L,R).
```

Metoda 2

În rezolvarea precedentă lista rezultat era construită la revenirea din recursivitate. Există o metodă alternativă de construire a listelor, folosind o variabilă colectoare, care este de fapt un parametru auxiliar în care colectăm rezultatul. În acest parametru (care în exemplul nostru va fi o listă) se construiește rezultatul element cu element.

Atunci când folosim o variabilă colectoare și adăugăm elemente la începutul variabilei colectoare, **elementele listei vor fi inversate**. Acest lucru este perfect dacă ne dorim să inversăm o listă (sau ordinea elementelor în rezultat este irelevantă (ex. operăm cu mulțimi)), dar dacă ne dorim elementele în ordinea originală, atunci trebuie să le adăugăm la sfârșitul listei colectoare.

$$adaugaSf(l_1 l_2 \dots l_n, e) = \begin{cases} (e) & , \text{dacă } n = 0 \\ l_1 \cup adaugaSf(l_2 \dots l_n, e) & \text{altfel} \end{cases}$$

```
% adaugaSf(L: list, E: el, R:list)
% model de flux: (i, i, o), (o, i, i), (i,o,i), (i, i, i), (o, o, i)
% L - lista la care vrem sa adaugam un elementul E la sfarsit
% E - elementul de adaugat la sfarsitul listei L
% R - lista rezultat, obținută prin adăugarea elementului E la finalul
listei L
```

```
adaugaSf([], E, [E]).
adaugaSf([H|T], E, [H|R]) :-
    adaugaSf(T, E, R).
```

- Predicatul nrAparitii ne este necesar în continuare; folosim implementarea de mai sus.

$$elimina2(l_1 \dots l_n, L_1 \dots L_m, C_1 \dots C_k) = \begin{cases} C_1 \dots C_k & , \text{dacă } n = 0 \\ elimina2(l_2 \dots l_n, L_1 \dots L_m, C_1 \dots C_k) & , \text{dacă} \\ nrAparitii(L_1 \dots L_m, l_1) = 1 & \\ elimina2(l_2 \dots l_n, L_1 \dots L_m, adaugaSf(C_1 \dots C_k, l_1)) & \text{altfel} \end{cases}$$

```
% elimina2(L:list, LO:list, Col:list, R:list)
% model de flux: (i,i,i,o) sau (i,i,i,i)
% L - lista din care eliminam elementele care apar o singura data
% LO - lista originala, folosita pentru a numară aparițiile
% Col - lista colectoare, în care colectăm elementele care apar de mai multe
% ori
% R - lista rezultat, obtinută prin eliminarea elementelor care apar o singură
% dată
elimina2([], _, Col, Col).
elimina2([H|T], LO, Col, R):-
    nrAparitii(LO, H, S),
    S = 1,
    elimina2(T, LO, Col, R).
elimina2([H|T], LO, Col, R):-
    nrAparitii(LO, H, S),
    S > 1,
    adaugaSf(Col, H, Col1),
    elimina2(T, LO, Col1, R).
```

Lista LO trebuie initializată cu lista originală, iar variabila colectoare trebuie să fie lista vidă la început. Deci este necesară, din nou, încă o funcție.

$$elimina2Main(l_1 \dots l_n) = elimina2(l_1 \dots l_n, l_1 \dots l_n, \emptyset)$$

```
% elimina2Main(L:list, R:list)
% model de flux: (i,o), (i,i)
% L - lista din care eliminam elementele care apar o singura data
% R - lista rezultat
elimina2Main(L, R):- elimina2(L, L, [], R).
```

Metoda 3

O altă idee posibilă de rezolvare este să verificăm pentru fiecare element dacă există în restul listei sau a fost întâlnit până acum. În caz afirmativ, elementul apare în listă de mai multe ori.

Pentru aceasta, implementăm un predicat auxiliar care verifică existența unui element într-o listă și memorăm într-o variabilă auxiliară elementele care au fost întâlnite până în acel moment. Întrucât ordinea în care memorăm aceste elemente în variabila auxiliară este irelevantă, vom adăuga la începutul variabilei colectoare.

$$\text{exista}(e, l_1 l_2 \dots l_n) = \begin{cases} \text{fals} & , \text{dacă } n = 0 \\ \text{adevărat} & , \text{dacă } l_1 = e \\ \text{exista}(e, l_2 l_3 \dots l_n) & \text{altfel} \end{cases}$$

```
% exista(E: el, L: list)
% model de flux: (i, i), (o, i)
% E - elementul pe care îl căutăm
% L - lista în care căutăm elementul
exista(E,[E|_]). 
exista(E,[H|T]):- 
    H \= E,
    exista(E,T).
```

$$\text{elimina3}(l_1 \dots l_n, Viz) = \begin{cases} \emptyset & , \text{dacă } n = 0 \\ l_1 \cup \text{elimina3}(l_2 \dots l_n, Viz) & , \text{dacă } \text{exista}(l_1, Viz) = \text{adevarat} \\ l_1 \cup \text{elimina3}(l_2 \dots l_n, l_1 \cup Viz) & , \text{dacă } \text{exista}(l_1, l_2 \dots l_n) = \text{adevarat} \\ \text{elimina3}(l_2 \dots l_n, Viz) & \text{altfel} \end{cases}$$

```
% elimina3(L: list, Viz: list, R: list)
% model de flux: (i, i, o), (i, i, i)
% L - lista din care eliminăm elementele care apar o singură dată
% Viz - lista elementelor vizitate până acum
% R - lista rezultat
```

```
elimina3([],_,[]).
elimina3([H|T],Viz,[H|R]) :-
    exista(H,Viz), !,
    elimina3(T,Viz,R).
elimina3([H|T],Viz,[H|R]) :-
    exista(H,T), !,
    elimina3(T,[H|Viz],R).
elimina3([_|T],Viz,R) :-
    elimina3(T,Viz,R).
```

$\text{elimina3Main}(l_1 \dots l_n) = \text{elimina3}(l_1 \dots l_n, \emptyset)$

```
% L - lista din care eliminam elementele care apar o singura data
% R - lista rezultat
elimina3Main(L, Rez) :- elimina3(L, [], Rez).
```

2. Dându-se o listă liniară numerică, să se șteargă toate secvențele de valori crescătoare. Ex. $\text{șterg}([1,2,4,6,5,7,8,2,1]) \Rightarrow [2, 1]$.

Soluție:

Metoda 1

Este important de observat că nu este suficient să verificăm dacă primele 2 elemente sunt în ordine crescătoare și să le eliminăm în caz afirmativ. Dacă procedăm astfel, vom avea probleme în cazul secvențelor crescătoare de lungime 3 (și de lungime impară, în general), pentru că după ce am eliminat primele 2 elemente, nu mai avem cu ce să comparăm al 3-lea element din secvență. O primă metodă de rezolvare este să verificăm mereu câte 3 elemente.

$$\text{eliminaCresc}(l_1 \dots l_n) = \begin{cases} \emptyset & , \text{dacă } n = 0 \text{ sau } (n = 2 \text{ și } l_1 < l_2) \\ (l_1) & , \text{dacă } n = 1 \\ \text{eliminaCresc}(l_2 \dots l_n) & , \text{dacă } l_1 < l_2 < l_3 \\ \text{eliminaCresc}(l_3 \dots l_n) & , \text{dacă } l_1 < l_2 \geq l_3 \\ l_1 \cup \text{eliminaCresc}(l_2 \dots l_n) & \text{altfel} \end{cases}$$

```
% eliminaCresc(L:list, R:list)
% model de flux: (i,o) sau (i,i)
% L - lista din care eliminam secvențele de elemente crescătoare
% R - lista rezultat
eliminaCresc([], []).
eliminaCresc([H], [H]). 
eliminaCresc([H1,H2], []) :- H1 < H2.
eliminaCresc([H1,H2,H3|T], R) :-
    H1 < H2,
    H2 < H3,
    eliminaCresc([H2,H3|T], R).
eliminaCresc([H1,H2,H3|T], R) :-
    H1 < H2,
    H2 >= H3,
    eliminaCresc([H3|T], R).
```

```

eliminaCresc([H1,H2|T], [H1|R]) :-
    H1 >= H2,
    eliminaCresc([H2|T], R).

```

Metoda 2

O soluție alternativă presupune lucrul cu primele 2 elemente, dar în acest caz avem nevoie de încă un parametru care să arate dacă suntem sau nu într-o secvență crescătoare. Vom considera un parametru care are valoarea 0 dacă nu suntem într-o secvență sau 1 dacă suntem într-o secvență crescătoare. În funcție de relația dintre primele 2 elemente și valoarea acestui parametru vom decide care elemente vor fi păstrate în listă.

$$eliminaCresc2(l_1 \dots l_n, f) = \begin{cases} \emptyset & , \text{dacă } n = 0 \text{ sau } (n = 1 \text{ și } f = 1) \\ (l_1) & , \text{dacă } n = 1 \text{ și } f = 0 \\ eliminaCresc2(l_2 \dots l_n, 1) & , \text{dacă } l_1 < l_2 \\ eliminaCresc2(l_2 \dots l_n, 0) & , \text{dacă } l_1 \geq l_2 \text{ și } f = 1 \\ l_1 \cup eliminaCresc2(l_2 \dots l_n, 0) & \text{altfel } (l_1 \geq l_2 \text{ și } f = 0) \end{cases}$$

```

% eliminaCresc2(L:list, F:integer, R>List)
% model de flux: (i,i,o) sau (i,i,i)
% L - lista din care eliminam secvențele de elemente crescătoare
% F - variabila care arată dacă suntem într-o secvență crescătoare
% R - lista rezultat
eliminaCresc2([], _, []).
eliminaCresc2([], 1, []).
eliminaCresc2([H], 0, [H]).
eliminaCresc2([H1,H2|T], _, R):-
    H1 < H2,
    eliminaCresc2([H2|T], 1, R).
eliminaCresc2([H1,H2|T], 1, R):-
    H1 >= H2,
    eliminaCresc2([H2|T], 0, R).
eliminaCresc2([H1,H2|T], 0, [H1|R]):-
    H1 >= H2,
    eliminaCresc2([H2|T], 0, R).

```

Pentru că am adăugat un parametru în plus, este necesar un predicat (wrapper) care să

efectueze primul apel, cu parametrul adițional inițializat corespunzător.

$\text{eliminaCresc2Main}(l_1 \dots l_n) = \text{eliminaCresc2}(l_1 \dots l_n, 0)$

```
% eliminaCresc2Main(L:list, R:list)
% model de flux: (i,o) sau (i,i)
% L - lista din care eliminam secvențele de elemente crescătoare
% R - lista rezultat
eliminaCresc2Main(L, R):- eliminaCresc2(L, 0, R).
```