Liste adânci

1 Obiective

În lucrarea de față vom prezenta o generalizare a tipului listă. Elementele listei pot să fie la rândul lor tot liste. Astfel o listă adâncă poate să aibă elemente la diverse nivele de imbricare.

Exemple:

```
L1 = [1,2,3,[4]]
L2 = [[1],[2],[3],[4,5]]
L3 = [[],2,3,4,[5,[6]],[7]]
L4 = [[[[1]]],1,[1]]
L5 = [1,[2],[[3]],[[[4]]],[5,[6,[7,[8,[9],10],11],12],13]]
L6 = [alpha,2,[beta],[gamma,[8]]]
```

Testați următoarele întrebări:

```
?- member(2,L5).
?- member([2], L5).
?- member(X, L5).
?- append(L1,R,L2).
?- append(L4,L5,R).
?- delete(1, L4,R).
?- delete(13,L5,R).
```

2 Considerații teoretice

2.1 Predicatul „atomic”

Vom folosi predicatul *atomic(X)* pentru a verifica dacă X este un element simplu (număr, simbol) sau este o structură (ex: o altă listă).

Testați următoarele întrebări:

```
? – atomic(apple).
? – atomic(4).
? – atomic(X).
? – atomic(apple(2)).
? – atomic([1,2,3]).
? – atomic([]).
```

2.2 Predicatul „depth”

Acest predicat calculează nivelul maxim de imbricare al unei liste adânci. Iterează peste elementele listei și verifică natura lor. Dacă elementul nu este atomic atunci se apelează recursiv pe acel element. Adâncimea listei este egală cu adâncimea maximă a unui element plus unu.

```
depth([],1).
depth([H|T],R):- atomic(H), !, depth(T,R).
depth([H|T],R):- depth(H,R1), depth(T,R2), R3 is R1+1, max(R3,R2,R).
```

Testați predicatul pentru listele L1-L6 (ex: ?-depth(L1, R).)

2.3 Predicatul „flatten”

Acest predicat aplatizează o listă adâncă. Lista rezultat va conține numai elemente atomice. Și în acest caz trebuie să se verifice natura elementului. Dacă elementul este la rândul lui o listă atunci va trebui aplatizată.

```
flatten([],[]).
flatten([H|T], [H|R]):- atomic(H), !, flatten(T,R).
flatten([H|T], R):- flatten(H,R1), flatten(T,R2), append(R1,R2,R).
```

Testați predicatul pentru listele L1-L6 (ex: ?-flatten(L1, R).)

2.4 Predicatul „heads”

Acest predicat extrage toate elementele atomice de la capul fiecărei liste. Ne vom folosi de un al treilea parametru pentru a ști dacă am extras capul listei curente sau nu.

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

% dacă flag=1 atunci suntem la început de lista și putem extrage
capul listei; în apelul recursiv setam flag=0
heads([H|T],[H|R],1):- atomic(H), !, heads(T,R,0).

% dacă flag=0 atunci nu suntem la primul element atomic și atunci
continuam cu restul elementelor
heads([H|T],R,0):- atomic(H), !, heads(T,R,0).

% dacă am ajuns la aceasta clauza înseamnă că primul element nu este
atomic și atunci trebuie să apelam recursiv și pe acest element
heads([H|T],R,_):- heads(H,R1,1), heads(T,R2,0), append(R1,R2,R).
heads_pretty(L,R):- heads(L, R, 1).
```

Testați predicatul pentru listele L1-L6 (ex: ?- heads_pretty(L1, R).)

2.5 Predicatul „member”

Dacă vrem să căutam un element la alt nivel de imbricare va trebui să modificăm predicatul *member* astfel:

```
% Varianta 1
member1(H,[H|_]).
member1(X,[H|_]):-member1(X,H).
member1(X,[_|T]):-member1(X,T).
```

```
% Varianta 2
member2(X,L):- flatten(L,L1), member(X,L1).
```

Testați următoarele întrebări:

```
? – member1(1,L1).
? – member1(4,L2).
? – member1([5,[6]], L3).
? – member1(X,L4).
? – member1(X,L6).
? – member1(14,L5).
```

3 Exerciții

1. Scrieți predicatul *count_atomic(L,R)* care calculează numărul de elemente atomice din lista *L* (toate elementele atomice).
2. Scrieți predicatul *sum_atomic(L,R)* care calculează suma elementelor atomice din lista *L* (toate elementele atomice).
3. Modifică predicatul *member* pentru liste adânci astfel încât să fie determinist.
4. Scrieți predicatul *lasts(L,R)* care extrage elementele atomice de pe ultima poziție din fiecare sublistă din *L*.

```
?- lasts([1,2,[3],[4,5],[6,[7]]], R).
R = [3,5,7] ;
false
```

5. Scrieți predicatul *replace(X,Y,L,R)* care înlocuiește pe *X* cu *Y* în lista adâncă *L* (la orice nivel de imbricare) și pune rezultatul în *R.*
6. (*) Scrieți un predicat care să sorteze o listă adâncă în funcție de adâncimea fiecărui element. Dacă două elemente au aceeași adâncime atunci se vor compara în funcție de elementele atomice pe care le conțin.

```
?- sort_depth([[[[1]]], 2, [5,[4],7], [[5],4], [5,[0,9]]], R).
R = [2, [5,[0,9]], [[5],4], [5,[4],7], [[[1]]]] ;
false
```