Programare Logică – Laboratorul 5

Backtracking, Predicatul cut !

Isabela Drămnesc

November 9, 2010

1 Întrebări din curs

- Ce înţelegeţi prin acumulatori? Când se folosesc şi de ce?
- Care este diferenţa între programe scrise cu acumulatori şi fără?
- Scrieţi predicatul pentru calculul lungimii unei liste în:
  - varianta fără acumulator
  - varianta cu acumulator
- Simulaţi trace pentru varianta cu acumulator.

2 Concepte

- Operaţii pe liste
- Acumulatori
- Diferenţa dintre programe cu acumulatori şi fără
- Backtracking (comportament la revenire).
- Predicatul cut !
  - tăietură verde
  - tăietură roşie

3 Exerciţii

Pentru fiecare din exerciţiile de mai jos vedeţi care este diferenţa dacă folosim predicatul cut sau nu, folosiţi comanda trace şi încercaţi să aflaţi răspunsul la întrebarea **“când folosim predicatul cut şi de ce??”**

1. Modificaţi predicatul membru astfel încât la interogarea

$? - membru(1, [2, 1, 1, 3, 1, 4, 1])$. să afişeze o singură soluţie.

```
membru(X, [X|_]).
membru(X, [A|B]):-membru(X,B).
```

2. Un exemplu din curs:

```
suma_la(1,1).
suma_la(N,Res):- N1 is N-1, suma_la(N1,Res1),
                 Res is Res1+N.

 ?- suma_la(5,X).
 X=15.
 ERROR: Out of local stack

/* ramane fara memorie, de ce? */

/* folosind cut ! */
csuma_la(1,1):-!.    /* sistemul e comnpromis la
                        conditia la limita  */
                     /* nu va mai face backtracking */
csuma_la(N,Res):- N1 is N-1, csuma_la(N1,Res1),
                  Res is Res1+N.

 ?-csuma_la(5,X).
 X=15.

/*  dar */
 ?-csuma_la(-3,Res).
 ERROR: Out of local stack

/* solutia este: */

ssuma_la(N,1):-N =< 1, !.
ssuma_la(N,Res):- N1 is N-1, ssuma_la(N1,Res1),
                  Res is Res1+N.
```

3. Verifică dacă o listă e mulţime sau nu (predicatul not)

Modificaţi ca sa funcţioneze!

```
multime([]).
multime([X|T]):-not membru(X,T), multime(T).
```

4. Predicatul care returnează intersecţia a două mulţimi

```
inters([],X,[]).
inters([X|T],Y,[X|Z]):-membru(X,Y),inters(T,Y,Z),!.
inters([X|T],Y,Z):-inters(T,Y,Z).
```

5. Ştergere duplicate şi afişare listă inversă (folosire acumulator şi predicatul cut)

```
sdup(L,X):-dupacc(L,[],X).
dupacc([],A,A).
```

```
dupacc([H|T],A,L):-membru(H,A),!,dupacc(T,A,L).
dupacc([H|T],A,L):-dupacc(T,[H|A],L).
```

Modificaţi predicatul astfel încât să şteargă duplicatele din listă, dar să returneze lista (nu inversa ei) (folosiţi acumulator şi cut !):

```
?-deldup2([1,1,1,2,2,3,4,5,4],L).
L=[1,2,3,4,5].
```

6. Minimul a două numere folosind cut

```
min1(X, Y, X) :- X =< Y, !. /* cut verde */
min1(X, Y, Y) :- X > Y.

min2(X, Y, X) :- X =< Y, !. /* cut rosu */
min2(X, Y, Y).

Explicati de ce cut verde si de ce cut rosu?
```

7. Modificaţi astfel încât sortarea prin generare şi testare să afişeze o singură soluţie

```
permutari([], []).
permutari(Lista, [Prim | Rest]) :- eliminare(Prim, Lista, L),
                                   permutari(L, Rest).

eliminare(Elem, [Elem | Rest], Rest).
eliminare(Elem, [Prim | Rest], [Prim | L]) :-
                                   eliminare(Elem, Rest, L).

relatie(X, Y) :- X =< Y.

ordonata([ _ ]).
ordonata([Prim, Secund | Rest]) :- relatie(Prim, Secund),
                                   ordonata([Secund | Rest]).

sortare(Lista, ListaSortata) :- permutari(Lista, ListaSortata),
                                ordonata(ListaSortata).
```

8. Alte probleme

a) Scrieţi un predicat pentru reuniunea a două mulţimi.

b) Fiind dată o listă, se împarte lista în două - o parte conţine elementele mai mici decât un element al listei (pivot), iar cealaltă conţine elementele mai mari. Cele două parţi se sortează, iar variantele sortate se concatenează. (Quicksort)

c) Fiind dată o listă, se găseşte elementul minim, se plasează pe prima poziţie şi se repetă procesul pentru coada listei. (Selection Sort)

4 Tema:

Efectuaţi diferite operaţii pe cozi cu liste cu diferenţe.

Termen de realizare: laboratorul următor.