Structuri incomplete

1 Obiective

În lucrarea de față vom prezenta un caz special de structuri. Structurile incomplete nu se termină într-o constantă (ex: [] pentru liste sau *nil* pentru arbori), ci se termină într-o variabilă liberă „_”.

2 Considerații teoretice

2.1 Reprezentare

Structurile incomplete (structuri instanțiate parțial) oferă posibilitatea modificării variabilei de la final, ceea ce permite de noi elemente în aceeași structură. Iterarea peste elementele structurii incomplete diferă față de structurile complete prin condiția de terminare. Pentru structurile incomplete, în condiția de terminare, trebuie să se verifice dacă s-a ajuns la o variabilă neinițializată (cu predicatul *var*). Această clauză trebuie să fie prima din cadrul predicatul astfel încât să nu se facă o unificare nedorită.

Exemple:

```
?- L = [a, b, c|_].
?- L = [1, 2, 3|T], T = [4, 5|U].
?- T = t(7, t(5, t(3, _, _), _), t(11, _, _)).
?- T = t(7, t(5, t(3, A, B), C), t(11, D, E)), D = t(9, F, G).
```

2.2 Liste incomplete

2.2.1 Predicatul „member”

Înainte de a scrie noul predicat, urmăriți execuția la vechiul *member*:

```
?- L = [1, 2, 3|_], member1(3, L).
?- L = [1, 2, 3|_], member1(4, L).
?- L = [1, 2, 3|_], member1(X, L).
```

Noul predicat rezolvă problemele de la vechiul predicat.

```
% trebuie testat explicit faptul ca am ajuns la sfârșitul listei
% și nu am găsit elementul căutat
member_il(_, L):-var(L), !, fail.
% celelalte 2 clauze sunt la fel ca în trecut
member_il(X, [X|_]):-!.
member_il(X, [_|T]):-member_il(X, T).
```

Urmărește execuția la noul predicat pe aceleași întrebări.

2.2.2 Predicatul „insert”

După cum ați observat, adăugare unui element la sfârșitul unei liste se poate realiza pe lista de intrare și nu mai este nevoie de un parametru nou pentru lista rezultat.

```
% am ajuns la finalul listei atunci putem adăuga elementul
insert_il(X, L):-var(L), !, L=[X|_].
insert_il(X, [X|_]):-!. % elementul există deja
insert_il(X, [_|T]):- insert_il(X, T).
```

Funcționalitatea primei clauze poate fi realizată și de a doua clauză. Dacă elementul căutat nu a fost găsit înseamnă că am ajuns la finalul listei. Caz în care variabila din coada se va unifica cu [X|_], ceea ce înseamnă că se inserează elementul căutat la sfârșitul listei.

Urmărește execuția la:

```
?- L = [1, 2, 3|_], insert_il(3, L).
?- L = [1, 2, 3|_], insert_il(4, L).
?- L = [1, 2, 3|_], insert_il(X, L).
```

2.2.3 Predicatul „delete”

În predicatul *delete* vom păstra aceeași variabilă neinițializată de la final și pentru lista de intrare și pentru lista rezultat. În rest predicatul este la fel cu varianta de la liste complete.

```
delete_il(_, L, L):-var(L), !. % am ajuns la finalul listei
delete_il(X, [X|T], T):-!. % ștergem prima apariție și ne oprim
delete_il(X, [H|T], [H|R]):-delete_il(X, T, R).
```

Urmărește execuția la:

```
?- L = [1, 2, 3|_], delete_il(2, L, R).
?- L = [1, 2, 3|_], delete_il(4, L, R).
?- L = [1, 2, 3|_], delete_il(X, L, R).
```

2.3 Arbori incompleți

2.3.1 Predicatul „search”

La fel ca și la predicatul *member* de la liste incomplete trebuie să verificăm explicit că am ajuns la o variabila neinițializată.

```
search_it(_, T):-var(T),!,fail.
search_it(Key, t(Key, _, _)):-!.
search_it(Key, t(K, L, _)):-Key<K, !, search_it(Key, L).
search_it(Key, t(_, _, R)):-search_it(Key, R).
```

Urmărește execuția la:

```
?- T=t(7, t(5, t(3,_,_), t(6,_,_)), t(11,_,_)), search_it(6, T).
?- T=t(7, t(5, t(3,_,_), _), t(11,_,_)), search_it(9, T).
```

2.3.2 Predicatul „insert”

Deoarece inserarea unui nod nou se realizează în frunze, ne putem folosi de același arbore de intrare și pentru rezultat. Ceea ce înseamnă că nu mai avem nevoie de încă un parametru pentru arborele rezultat.

```
% inserează sau verifică dacă elementul există deja în arbore
insert_it(Key, t(Key, _, _)):-!.
insert_it(Key, t(K, L, R)):-Key<K, !, insert_it(Key, L).
insert_it(Key, t(_, _, R)):-insert_it(Key, R).
```

Urmărește execuția la:

```
?- T=t(7, t(5, t(3,_,_), t(6,_,_)), t(11,_,_)), insert_it(6, T).
?- T=t(7, t(5, t(3,_,_), _), t(11,_,_)), insert_it(9, T).
```

2.3.3 Predicatul „delete”

La fel ca în cazul predicatului *delete* de la liste incomplete vom păstra aceeași variabilă neinițializată din arborele de intrare și pe arborele rezultat.

```
delete_it(Key, T, T):-var(T),!.
delete_it(Key, t(Key, L, R), L):-var(R),!.
delete_it(Key, t(Key, L, R), R):-var(L),!.
delete_it(Key, t(Key, L, R), t(Pred,NL,R)):-!,get_pred(L,Pred,NL).
delete_it(Key, t(K,L,R), t(K,NL,R)):-Key<K,!,delete_it(Key,L,NL).
delete_it(Key, t(K,L,R), t(K,L,NR)):-delete_it(Key,R,NR).

% caută nodul predecesor
get_pred(t(Pred, L, R), Pred, L):-var(R),!.
get_pred(t(Key, L, R), Pred, t(Key, L, NR)):-get_pred(R, Pred, NR).
```

Urmărește execuția la:

```
?- T=t(7, t(5, t(3,_,_), t(6,_,_)), t(11,_,_)), delete_it(6, T, R).
?- T=t(7, t(5, t(3,_,_), _), t(11,_,_)), delete_it(9, T, R).
```

3 Exerciții

Scrieți un predicat care:

1. Concatenează 2 liste incomplete (rezultatul este tot o listă incompletă).
2. Inversează o listă incompletă (rezultatul este tot o listă incompletă).
3. Convertește o listă incompletă într-o listă completă și viceversa.
4. Traversează un arbore incomplet în pre-ordine (cheile se adaugă într-o listă incompletă).
5. Calculează înălțimea unui arbore incomplet.
6. Convertește un arbore incomplet într-un arbore complet și viceversa.
7. Aplatizează o listă adâncă incompletă (rezultatul este o listă simplă incompletă).

```
?- flat_il([[1|_], 2, [3, [4, 5|_]|_]|_], R).
R = [1, 2, 3, 4, 5|_] ? ;
false
```

8. Calculează diametrul unui arbore incomplet.

$$diam(T) = \max\{diam(T.left), diam(T.right), height(T.left) + height(T.right) + 1\}$$

9. (*) Determină dacă o listă incompletă este o sub-listă într-o altă listă incompletă.

```
?- subl_il([1, 1, 2|_], [1, 2, 3, 1, 1, 3, 1, 1, 1, 2|_]).
true

?- subl_il([1, 1, 2|_], [1, 2, 3, 1, 1, 3, 1, 1, 1, 3, 2|_]).
false
```