Materiale suplimentare de Prolog

Logică matematică și computațională

Arbori binari

Arbori binari – intuitiv

Reamintim că, în Prolog, listele erau de două feluri:

- lista vidă `[]`;
- listă nevidă `[H|T]` formată din capul `H` (care putea fi orice) și coada `T` (care era o listă).

Similar, arborii binari vor fi de două feluri:

- arborele vid;
- arbore nevid format dintr-o rădăcină și doi subarbori.

Arborii binari nu sunt o funcționalitate inerentă a Prolog-ului, de aceea pentru ei se vor folosi simboluri de funcție *ad hoc*, lucru permis de limbaj.

Arbori binari – definire și parcurgere

Pentru arborele vid, vom folosi atomul `vid`. Pentru arborele care are rădăcina `R` și cei doi subarbori `T` și `U`, vom folosi termenul `arb(R,T,U)`.

Interogați:

```
X = arb(3,arb(2,vid,vid),vid).
```

Definirea predicatului `srd/2` folosit la parcurgerea unui arbore binar în inordine:

```
srd(vid,[]).
srd(arb(R,T,U),L) :- srd(T,L1), srd(U,L2),
append(L1,[R|L2],L).
```

Testați acest predicat!

Exercițiul 1: arbori binari de căutare
Definiți următoarele predicate:

- `bt_ins/3` care inserează numărul natural din primul argument în arborele binar de căutare din al doilea argument;
- `bt_list/2` care transformă lista din primul argument într-un arbore binar de căutare;
- `bt_sort/2` care sortează lista din primul argument trecând prin arborele binar de căutare asociat ei.

Liste de diferențe

Inversarea listelor

Predicatul următor, `listN/2`, va fi util pentru generarea unei liste de lungime dată:

```
listN([],0).
listN([a|T], N) :- N > 0, M is N - 1, listN(T,M).
```

Introducem și metapredicatul `listing/1`, care afișează toate clauzele corespunzătoare unui predicat. Interogați:

```
?- listing(listN).
```

Reamintim, acum, din soluțiile Laboratorului 3, definirea predicatului `rev/2` de inversare a listelor:

```
rev([],[]).
rev([H|T],L) :- rev(T,N), append(N,[H],L).
```

Soluția dată nu este prea eficientă, având o complexitate pătratică.

Acumulatori

Soluția următoare o îmbunătățește pe cea precedentă, adăugând un predicat auxiliar, care are un parametru în plus, care joacă rol de acumulator. Complexitatea devine liniară (testați pentru liste de lungime 1000-10000):

```
reva(L,R) :- revah(L,[],R).
revah([], R, R).
revah([H|T], S, N) :- revah(T,[H|S],N).
```

Contemplați adevărul următoarei afirmații: pentru orice `A`, `B`, `C`, avem că `revah(A,B,C)` dacă și numai dacă, notând cu `M` inversa listei `A`, avem că `append(M,B,C)`.

În continuare, ținând cont de această afirmație, vom rescrie soluția de mai sus, permițând generalizarea ei la alte probleme.

Difference lists

Reamintim că afirmația era: pentru orice `A`, `B`, `C`, avem că `revah(A,B,C)` dacă și numai dacă, notând cu `M` inversa listei `A`, avem că `append(M,B,C)`. Altfel spus, inversa lui `A` este „`C` fără `B`".

Vom reprezenta expresia „`C` fără `B`" sub forma unei perechi `(C,B)` și o vom numi *difference list* sau **difflist**.

Definiția anterioară devine:

```
revd(L,R) :- revdh(L,(R,[])).
revdh([],(R,R)).
revdh([H|T],(N,S)) :- revdh(T,(N,[H|S])).
```

Practică

Exercițiul 2
Definiți un predicat `flatten/2` care aplatizează structura unei liste.

Exemplu:

```
?- flatten([1,2,[3,a],[[7],2],5],L).
L = [1, 2, 3, a, 7, 2, 5]
```

Dați o soluție care folosește `append/3` și una care folosește difflist-uri.

Indiciu: Folosiți metapredicatul `is_list/1`.

Exercițiul 3
Reamintim, tot din soluțiile Laboratorului 3, definirea predicatului `quicksort/2`:

```
quicksort([],[]).
quicksort([H|T],L) :- split(H,T,A,B), quicksort(A,M),
                      quicksort(B,N), append(M,[H|N],L).
split(_,[],[],[]).
split(X,[H|T],[H|A],B) :- H < X, split(X,T,A,B).
split(X,[H|T],A,[H|B]) :- H >= X, split(X,T,A,B).
```

Rescrieți această definiție folosind difflist-uri (fără a mai folosi `append/3`).

Gramatici

Gramatici independente de context

Un instrument matematic util pentru procesarea limbajului natural în stilul programării logice este reprezentat de gramaticile independente de context (*context-free grammars*, CFGs). O definiție riguroasă a conceptului va apărea la cursul de Limbaje formale și automate, aici doar îl vom ilustra prin exemple.

Considerăm gramatica următoare:

$$SENT \to NP\ VP,$$

$$NP \to Det\ N, \quad VP \to TV\ NP, \quad VP \to V,$$

$$Det \to the, \quad Det \to a, \quad Det \to every,$$

$$N \to teacher, \quad N \to doctor, \quad N \to park,$$

$$TV \to likes, \quad V \to walks.$$

Atunci *the teacher likes every doctor* este validă pentru ea. (De ce?) O mulțime de șiruri peste un alfabet (nu neapărat descrisă de o gramatică) se numește **limbaj formal**.

Codificare cu append/3

Putem codifica gramatica ca pe un program Prolog în felul următor:

```
sent(R) :- np(A), vp(B), append(A,B,R).
np(R) :- dete(A), n(B), append(A,B,R).
vp(R) :- tv(A), np(B), append(A,B,R).
vp(R) :- v(R).
dete([the]).  dete([a]).  dete([every]).
n([teacher]).  n([doctor]).  n([park]).
tv([likes]).  v([walks]).
```

Testați:

```
?- sent(R).
```

pentru a vedea limbajul formal descris de această gramatică.

Codificare cu difflist-uri

Pentru a nu mai apela append/3, putem codifica gramatica folosind difflist-uri:

```
sentd(R) :- sentdh((R,[])).
sentdh((R,S)) :- npdh((R,Z)), vpdh((Z,S)).
npdh((R,S)) :- detedh((R,Z)), ndh((Z,S)).
vpdh((R,S)) :- tvdh((R,Z)), npdh((Z,S)).
vpdh((R,S)) :- vdh((R,S)).
detedh(([the|S],S)). detedh(([a|S],S)).
detedh(([every|S],S)).
ndh(([teacher|S],S)). ndh(([doctor|S],S)).
ndh(([park|S],S)).
tvdh(([likes|S],S)). vdh(([walks|S],S)).
```

Firește, nu avem nevoie, de fapt, de parantezele în plus. Testați:

```
?- sentd(R).
```

Codificare cu DCG-uri

Prolog-ul dispune de o notație specială pentru asemenea programe, denumită gramatici de clauze definite (*definite clause grammars*, DCGs).

Programul anterior devine:

```
sentgh --> np, vp.
np --> dete, n.
vp --> tv, np.
vp --> v.
dete --> [the].  dete --> [a].  dete --> [every].
n --> [teacher].  n --> [doctor].  n --> [park].
tv --> [likes].  v --> [walks].
```

Cum se traduc DCG-urile

Dacă vom interoga:

```
?- listing(sentgh).
?- listing(vp).
```

vom vedea că, de fapt, Prolog-ul implementează un program asemănător cu cel anterior care folosește difflist-uri (fără parantezele în plus).

Putem, așadar, testa:

```
?- sentgh(R,[]).
```

sau, încă,

```
?- phrase(sentgh,R).
```

Practică

Exercițiul 4
Definiți o gramatică care definește limbajul formal format din șirurile de a-uri și b-uri în care numărul de a-uri este par.

Testați:

```
?- parA([a,b,a],[]).
?- parA([a,b,b],[]).
?- parA(X,[]).
?- length(X,N), parA(X,[]).
?- length(X,_), parA(X,[]).
```

Practică

Exercițiul 5
Definiți o gramatică care definește limbajul formal

$$\{a^n b^n \mid n \in \mathbb{N}\}.$$