10. Saraksts – populārākais struktūras veids

Nodaļas saturs

Nodaļā ievietotie attēli, programmu piemēri un tabulas

Saraksts ir populārākā datu struktūra valodā Prolog. Saraksts (*list*) valodā Prolog ir termu (vērtību, mainīgo, struktūru) virkne, kas līdzinās datu struktūrai – stekam (*stack*). Steks ir tāda lineāri (t.i., pēc kārtas) sakārtota datu struktūra, kur gan elementu pievienošana, gan izņemšana notiek no vienas puses. Tai pat laikā sarakstu daudzos gadījumos no apstrādes viedokļa var salīdzināt ar simbolu virkni.

Saraksta lielākā ērtība tādējādi ir spēja to definēt un apstrādāt rekursīvi, kas atbilst Prolog vispārējai datu apstrādes koncepcijai.

10.1. Saraksta uzdošana

Saraksts ir datu struktūra, kas var būt tukša:

```
[]
```

vai sastāvēt no vairākiem, ar komatiem atdalītiem objektiem:

```
[a,b,c]
```

Objekti var būt vērtības, mainīgie, citi saraksti utt. Visiem saraksta elementiem nav jābūt ar vienādu tipu.

Kā jau tas ir pierasts ar citām datu struktūrām – nedrīkst jaukt jēdzienus: *saraksta elements* un *saraksts garumā viens*:

```
a ≠ [a]
```

Sevišķi svarīgi to ir apzināties un saprast rekursīvas saraksta reprezentācijas un apstrādes gadījumā.

Rekursīvi definējot, saraksts sastāv no galvas un astes.

Galva (*head*) ir saraksta pirmais elements, bet **aste** (*tail*) – atlikusī daļa. Svarīgi atcerēties, ka galvas datu tips ir elements (nevis saraksts ar garumu 1), bet astes datu tips ir saraksts. Datu apstrādē šīs nianses neievērošana rada daudz neuzmanības kļūdu.

Ja sarakstu definē rekursīvi, tad starp galu un asti ir nevis komats, bet vertikālā svītra "|":

```
[X|Y]=[a,b,c]
```

```
X = a
Y = [b,c]
```

Ievērojiet, ka X ir elements, bet Y saraksts.

Paplašinātajā gadījumā gan galvas (t.i., kreisajā) pusē no vertikālās svītras var būt arī ar komatiem atdalīts elementu uzskaitījums (nevis tikai viens elements):

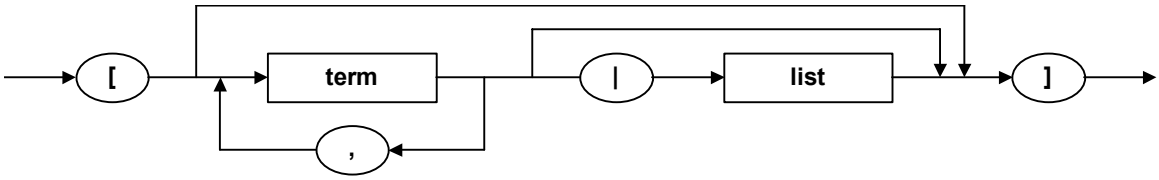
```
[X,Y|Z]=[a,b,c].

X = a
Y = b
Z = [c]
```

Tādējādi, pat tik vienkāršu sarakstu, kā *[a,b,c]*, var uzdot vairākos veidos:

```
[a,b,c]
[a|[b,c]]
[a|[b|[c]]]
[a|[b|[c|[]]]]
[a|[b,c|[]]]
[a,b|[c]]
[a,b|[c|[]]]
[a,b,c|[]]
```

Visas iepriekš apskatītās iespējas apkopo saraksta sintaktiskā definīcija:

Att. 10-1. Sintakse: saraksts (*list*).



10.2. Triviāla sarakstu apstrāde

Vienkāršākie predikātu piemēri saistībā ar sarakstu apstrādi aprakstāmi, izmantojot faktus (nevis noteikumus) un apraksta pozicionālas likumsakarības saraksta sākumā.

Predikāts *listoftwo/1*, kas pārbauda, vai saraksts sastāv tieši no diviem elementiem.

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

```
?- listoftwo([a,b]).

yes
?- listoftwo([a,b,c]).

no
?- listoftwo(X).

X = [_,_]

yes
```

Predikāts *firstelement/2*, kas otrajā pozīcijā atgriež saraksta pirmo elementu.

```
firstelement([A|_],A).
```

```
?- firstelement([a,b,c],X).

X = a

yes
```

Predikāts *firsttwoequal/1*, kas pasaka, vai saraksta pirmie divi elementi ir vienādi.

```
firsttwoequal([A,A|_]).
```

```
?- firsttwoequal([a,b,c]).

no
?- firsttwoequal([x,x,l]).

yes
```

Predikāts *removefirst/3*, kas 3. pozīcijā atgriež saraksta, kurš iegūts, no sākotnējā saraksta noņemot pirmo elementu, kas savukārt tiek atgriezts 2. pozīcijā.

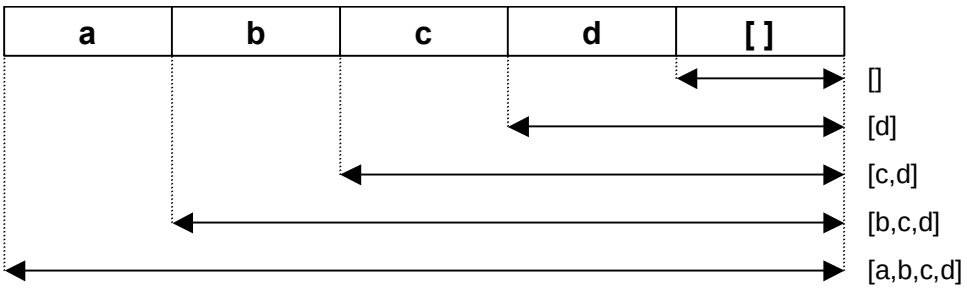
```
removefirst([A|B],A,B).
```

```
?- removefirst([a,b,c],X,Y).

X = a
Y = [b,c]

yes
?- removefirst(W,s,[t,u,v]).

W = [s,t,u,v]

yes
```

10.3. Rekursīva sarakstu apstrāde

Netriviāla sarakstu apstrāde nav iespējama bez rekursijas, jo apstrādājamo sarakstu garums nav fiksēts. Lai vieglāk uztvertu rekursīvu saraksta apstrādi, būtu jāizprot rekursīvā saraksta reprezentācija. Saraksts sastāv no galvas un astes, un aste ir saraksts – tātad, saraksts bez pirmā elementa arī ir saraksts un līdz ar to vienā sarakstā mēs varam ieraudzīt kopā N+1 sarakstus (katrs no tiem ir "lielā" saraksta beigu daļa), kur N – saraksta elementu skaits.

Att. 10-2. Sarakstā [a,b,c,d] ietvertie pieci apakšsaraksti.



10.3.1. Piederība pie saraksta

Viens no vienkāršākajiem rekursīvajiem predikātiem ir piederība pie saraksta: *contains/2*, kur pirmajā pozīcijā ir saraksts, bet otrajā – tam piederošs elements.

```
A1.   contains([A|_],A).
A2.   contains([_|C],A):-contains(C,A).
```

```
?- contains([a,b,c,d],b).

true ? ;

no
?- contains([a,b,c,d],X).

X = a ? a

X = b

X = c

X = d

no
```

Izvedums (*contains([a,b,c,d],X)***).**.

```
S1. contains([a,b,c,d],X?).
  A1. X=a true
  X = a
  A2. C=[b,c,d], contains([b,c,d],X?)
    S2. contains([b,c,d] ,X?)
      A1. X=b true
      X = b
      A2. C=[c,d], contains([c,d],X?)
        S3. contains([c,d],X?)
          A1. X=c true
          X = c
          A2. C=[d], contains([d],X?)
            S4. contains([d] ,X?)
              A1. X=d true
              X = d
              A2. C=[], contains([],X?)
                S5. contains([],X?)
                no
```

Šī predikāta nepilnība ir, ka pārbaudot uz konkrētu elementu (piemēram, *contains([a,b,c,d],b)*), atbilde nav uzreiz *yes*, vēl sliktāk, ja sarakstā ir vairāk nekā viens atbilstošs elements, tik reizes tiek dota atbilde *true*.

Šo problēmu var risināt, pielietojot (!) operatoru, tiesa gan tāds predikāts vairs nebūs derīgs elementu uzskaitei (*contains([a,b,c,d],X)*). *contains2/2*:

```
contains2([A|_],A):-!.
```

```
contains2([_|C],A):-contains2(C,A).

?- contains2([a,b,c,a],a).

yes
```

10.3.2. Divu sarakstu konkatenācija

Pirmais Prolog predikāts, kuru varētu uzskatīt par "galvas laušana" ir divu sarakstu konkatenācija, kam atbilst GNU-Prolog iebūvētais predikāts *append/3*, kas jau tika apskatīts 1. nodaļā, iepazīstoties ar Prolog darbības principiem.

```
?- append([a,b],[c,d,e],X).

X = [a,b,c,d,e]

yes
```

Mēģināsim to uzprogrammēt paši, definējot predikātu *conc/3*, kam būtu jāstrādā tieši tāpat, kā strādā *append/3* – divu sarakstu konkatenācija jeb saķēdēšana. Izrādās, ka attiecīgā Prolog programma ir negaidīti īsa:

```
A1.   conc([],Z,Z).
A2.   conc([X|Y],W,[X|Z]):-conc(Y,W,Z).
```

Tikai saprast, kāpēc tas strādā, no pirmā acu uzmetiena un "neapgriežot smadzenes par 180 grādiem", praktiski nav iespējams, bet tas strādā:

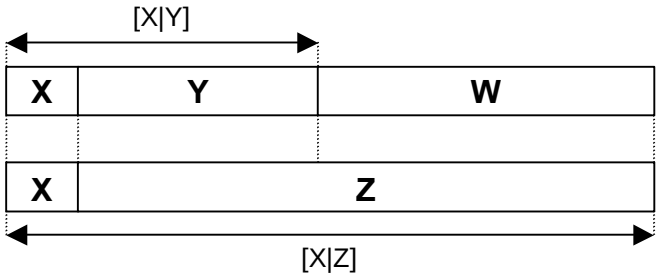
```
?- conc([a,b],[c,d,e],Z).

Z = [a,b,c,d,e]

yes
```

Vispirms mēģināsim saprast, kā "deklarēts" predikāts *conc/3*:

- Pirmais noteikums nozīmē, ka ja tukšam sarakstam pievieno galā kādu sarakstu, tad sanāk šis pats saraksts.
- Sarežģītāks ir otrais noteikums, bet pirmais, kam tajā būtu jāpievērš uzmanība, ir mainīgā X parādīšanās noteikuma galvā 2 reizes, tādējādi ar šo noteikumu ir pateikts, ka 1. saraksts, sakonkatenēts ar 2. sarakstu, veido trešo tad, ja pirmā saraksta un trešā saraksta pirmie elementi sakrīt un, pirmo sarakstu bez pirmā elementa sakonkatenējot ar otro sarakstu, sanāk trešais saraksts bez pirmā elementa.

Grafiski otrais noteikums izskatās šādi:

Att. 10-3. Predikāta *conc/3* noteikuma #2 grafiska reprezentācija.



Tālāk sekos dažu vaicājumu izvedumi predikātam *conc/3*.

Izvedums (*conc([a,b],[c,d,e],Z)***).** *conc/3* noteikumus numurē kā A1 un A2.

```
S1. conc([a,b],[c,d,e],Z).
  A2. X=a, Y=[b], W=[c,d,e], [X|Z?]=S1.Z, conc([b],[c,d,e],Z?)
    S2. conc([b],[c,d,e],Z?)
      A2. X=b, Y=[], W=[c,d,e], [X|Z?]=S2.Z, conc([],[c,d,e],Z?)
        S3. conc([],[c,d,e],Z?)
          A1. Z=[c,d,e], Z=S3.Z true
          [c,d,e]=S3.Z, [b,c,d,e]=[S2.A2.X|S3.Z]=S2.Z,
                        [a,b,c,d,e]=[S1.A2.X|S2.Z]=S1.Z
          Z = [a,b,c,d,e]
          yes
```

Kopējā programma sarakstu apstrādei, kas ietver visus šajā nodaļā definētos predikātus, dota sekojošajā attēlā:

Att. 10-4. Prolog programma sarakstu apstrādei (*list.pl*).

```
listoftwo([_,_]).
firstelement([A|_],A).
firsttwoequal([A,A|_]).
removefirst([A|B],A,B).
contains([A|_],A).
contains([_|C],A):-contains(C,A).
contains2([A|_],A):-!.
contains2([_|C],A):-contains2(C,A).
conc([],Z,Z).
conc([X|Y],W,[X|Z]):-conc(Y,W,Z).
```

Predikātu *conc/3* var izmantot arī citos veidos, piemēram, lai noskaidrotu, kādi divi saraksti jāsakonkatenē, lai iegūtu doto sarakstu.

```
?- conc(X,Y,[a,b,c]).

X = []
Y = [a,b,c] ? a

X = [a]
Y = [b,c]

X = [a,b]
Y = [c]

X = [a,b,c]
```

```
Y = []

no
```

Izvedums (*conc/3* noteikumus numurē kā A1 un A2).

```
S1. conc(X?,Y?,[a,b,c]).
  A1. []=X, Z=Y, Z=[a,b,c] true
  X=[]
  Y=[a,b,c]
  A2. X=a, Z=[b,c], conc(Y?,W?,[b,c]), [a|Y?]= S1.X?
    S2. conc(Y?,W?,[b,c])
      A1. []=Y, Z=W, Z=[b,c] true
      [b,c]=W, [a|[]]=S1.X, [b,c]=S1.Y
      X=[a]
      Y=[b,c]
      A2. X=b, Z=[c], conc(Y?,W?,[c]), [b|Y?]= S2.Y?
        S3. conc(Y?,W?,[c])
          A1. []=Y, Z=W, Z=[c] true
          [c]=W, [b|[]]=S2.Y, [a|S2.Y]=S1.X=[a,b], [c]=S2.W=S1.Y
          X=[a,b]
          Y=[c]
          A2. X=c, Z=[], conc(Y?,W?,[]), [c|Y?]= S3.Y?
            S4. conc(Y?,W?,[])
              A1. []=Y, Z=W, Z=[] true
              []=W, [c|[]]=S3.Y, [b|S3.Y]=S2.Y=[b,c],
                          [a|S2.Y]=S1.X=[a,b,c],[]=S3.W=S2.W=S1.Y
              X=[a,b,c]
              Y=[]
              no
```

10.4. Vingrinājumi

Vingrinājums #1.

Definēt predikātu *listofatleasttwo/1*, kas pasaka par sarakstu, vai tajā ir vismaz divi elementi.

```
?- listofatleasttwo([a]).

no
?- listofatleasttwo([a,b,c]).

yes
```

Vingrinājums #2.

Definēt predikātu *swapfirsttwo/1*, kas otrajā pozīcijā atgriež sarakstu, kurā salīdzinot ar pirmo samainīti vietām divi pirmie elementi.

```
?- swapfirsttwo([a,b,c,d],X).

X = [b,a,c,d]

yes
```

Vingrinājums #3.

Izveidot izpildes izvedumu vaicājumam:

```
?- contains2([a,b,c,d],c).

yes
```

kur

```
A1.   contains2([A|_],A):-!.
A2.   contains2([_|C],A):-contains2(C,A).
```

Vingrinājums #4.

Izveidot izpildes izvedumu vaicājumam:

```
?- conc([a,b],Y,[a,b,c]).

Y = [c]

yes
```

kur

```
A1.   conc([],Z,Z).
A2.   conc([X|Y],W,[X|Z]):-conc(Y,W,Z).
```

Vingrinājums #5.

Definēt predikātu *lastelement/2*, kas otrajā pozīcijā atgriež saraksta pēdējo elementu.

```
?- lastelement([a,b,c,d],X).

X = d

yes
```