Een funktioneel programma

Maarten M Fokkinga, 22 juni 1984

(Berztiss and Thatte 1983) beweren, in een overigens aanbevolen verhaal, het volgende.

A really difficult problem for which to write a purely functional program is one in which the nodes of a binary tree are visited under inorder and scaled as follows: Substract 1 at the first node visited, substract 2 at the second node, 3 at the third, and so forth. A procedural program that is still well protected from programmer errors would be much easier to comprehend.

Binnen 10 minuten nadat ik dit gelezen had vond ik beide volgende programma's. Ik begon met methode **1**, liep daarbij vast, deed het toen recht-toe-recht-aan volgens methode **0** in één minuut en probeerde vervolgens de eerste poging nog te voltooiien, hetgeen toen snel lukte.

Methode **0** Recht-toe-recht-aan

Definieer een funktie scale zo dat

(scale n tree) = het paar bestaande uit de geschaalde tree (te beginnen met n af te trekken van de eerste knoop), en een getal n' zo dat n' - n = het aantal knopen in tree, {ofwel: n' moet als eerste aftrek-getal genomen worden bij de scaling van de rest van de boom die tree bevat}. De "truuk" om (scale n t) een paar te laten opleveren is standaard in funktioneel programmeren, 2nivo-grammatica's en attributen grammatica's!

```
scale n ()       =  (), n
scale n (tl, i, tr) = (tl', i'', tr'''), n'''
    where  tl', n'  = scale n tl
           i'', n'' = i - n', n' + 1
           tr''', n''' = scale n'' tr
```

Methode 1 Een poging om slim te zijn

Omdat het probleem als "really difficult" was omschreven begon ik mijn poging met het bedenken van een "slimme" oplossing: lever eerst de lijst op van knopen die met een inorder wandeling bezocht worden, bepaal dan de geschaalde knopen (da's makkelijk) en bouw vervolgens de boom weer

op. Dit laatste lukte me in eerste instantie niet (bevredigend). Maar nadat ik volgens methode 0 gesleegd was, bleek ook dit niet moeilijk. Hier zijn de funkties.

```
inorderlijst () = ()
inorderlijst (tl, i, tr) = inorderlijst tl ++ (i,) ++ inorderlijst tr

scale n () = ()
scale n (x:y) = (x-n): scale (n+1) y

imitate () list = (), list                    ||toelichting volgt!
imitate (tl, i, tr) list = (tl', i'', tr'''), list'''
    where tl', list' = imitate tl list
          i'', list'' = list'
          tr''', list''' = imitate tr list''
```

Dus (imitate tree list) bouwt een boom die de vorm heeft van tree, en knopen uit list, zó dat de inorder opsomming van de knopen van tree juist gelijk is aan (de een initieel beginstuk van) list. De programma-body luidt nu

```
first (imitate tree (scale 1 (inorder tree)))
```

Niet alleen waren de programma's snel geschreven, ze zijn ook makkelijk te begrijpen (omdat o.a. er geen begrippen of namen aan te pas komen die vreemd zijn aan de probleemstelling; bij een imperatief programma zijn er vast wel hulpvariabelen of begrippen die niet direkt in de probleemstelling voorkomen!). Dus Bertiss en Thatte hebben ongelijk.

Vervijzing

Beetiss, A.T., Thatte, S.: Specification and implementation of Abstract data types. Advances in Computers, 23 (1983) pp 295-353. (Academic Press,)