Hilbertkrommen tekenen

Maarten Fokkinga, 11 febr 1987

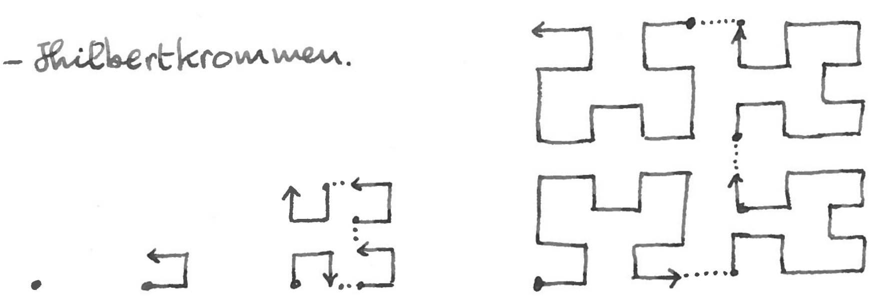
Aan de orde komt

- modulariteit (enige malen) (o.a. abstract data types)
- probleemanalyse: recurrente betrekkingen/rec. vergelijkingen
- representatiekeuze (enige malen)
- aansturing (rand)apparatuur
- opslagruimte inruilen tegen berekenigstijd.
- berekeningstijd inruilen tegen extra code

Modulaire opzet (1).

Het bepalen/definieren van de tekening en het afdrukken ervan op een plotter, laser printer of beeldscherm worden in afzonderlijke modules (functies) ondergebracht. (Voordelen: eenvoudiger analyse/constructie, eenvoudiger aanpassing aan gewijzigde probleemstellingen, etc.)

Probleemanalyse – Hilbertkrommen.



H_0 H_1 H_2 H_3

H_n = "aaneenschakeling dmv ···· van": sv.rot H_{n-1}, H_{n-1}, H_{n-1}, sh.rot H_{n-1}

sh/sv = spiegeling om hor./vert. as; rot = rotatie kwartslag ↺.

Representatie (1)

Tekening := lijst van punten in het vlak, ieder punt gerepresenteerd door cartesische (of polaire!) coördinaten.

Modulariteit (2)

Algemene teken-manipulaties onafhankelijk van Hilbert krommen.
Het volgende Miranda script spreekt --hoop ik-- voor zich.

```
pad == [(num, num)]                    || paden non-empty!
rot d p = map r p  where r (x,y) = (x*cos d, y*sin d)
join2 p p' = p ++ p'
join = concat
cat2 p p' = p ++ [(last p, first p')] ++ p'
cat [p] = p
cat (p:ps) = cat2 p (cat ps),   ps ≠ []
scale (sx, sy) p = map s p  where  s (x,y) = (x*sx, y*sy)
shift (sx, sy) p = map s p  where  s (x,y) = (x+sx, y+sy)
mkpad (x,y) = [(x,y)]
```

Als we willen afdwingen, syntactisch d.m.v. type-checking, dat alléén bovenstaande functies op paden manipuleren, dan voegen we toe:

```
abstype pad
   with rot :: num -> pad -> pad;
        join2, cat2 :: pad -> pad -> pad;
        join, cat :: [pad] -> pad;
        scale, shift :: (num, num) -> pad -> pad;
        mkpad :: (num, num) -> pad;
        yield :: pad -> [(num, num)]
```

yield p = p

De toevoeging van yield is nodig om een geconstrueerd pad later als puntenlijst te kunnen gebruiken.

<u>Definitie</u> Hilbertkrommen bij representatie (1).

```
H 0 = nilpad (0,0)
H n = cat [sv.rot1 H', shift (b'+1, 0) H', shift (b'+1, b'+1) H', shift (b', 2b'+1) (sh.rot1 H')]
      where sv = scale (-1, 1);  sh = scale (+1,-1)
            H'   = H (n-1)
            b'   = 2^(n -1) - 1       || = breedte van H'
            rot1 = rot (pi/4)
```

Opm. De (herhaalde!) machtsverheffing bij b' kan ingeruild worden tegen extra opslagruimte en halvering, door H een extra parameter b (= 2^n) mee te geven: b' = b <u>div</u> 2 - 1 .

<u>Inruil ruimte tegen tijd</u>

Lengte van H n neemt exponentioneel toe met n, en H' is gedurende (minstens driekwart van) de berekening van H n ergens opgeslagen. Dit vormt een ernstige belemmering. Liever zien we dat H' <u>niet</u> wordt opgeslagen en onthouden, maar drie/viermaal opnieuw wordt berekend. (De berekeningstijd is voor Hilbertkrommen tóch rechtevenredig met de lengte van de kromme -- gelóóf ik dd. 11 febr 87, 12:00 -- zodat deze inruil de grootte-orde vd berekeningstijd niet aantast.)
Dus: schrijf H(n-1) in plaats van H' in de def. van H n (en verbied de evaluator common subexpressions te elimineren!).

Representatie (2)

Tekening := lijst van relatieve verplaatsingen in het vlak.
Wij kiezen voorts: verplaatsing is een stap (N, O, Z, W).

stap ::= N | O | Z | W
pad == [stap]
rot p = map r p <u>where</u> r N = W; r W = Z; r Z = O; r O = N
sh p = map s p <u>where</u> s N = Z; s Z = N; s x = x, N ≠ x ≠ Z
sv p = map s p <u>where</u> s O = W; s W = O; s x = x, O ≠ x ≠ Z

Nu krijgen we
H 0 = []
H n = sv.rot H' ++ [O] ++ H' ++ [N] ++ H' ++ [W] ++ sh.rot H'
    <u>where</u> H' = H (n-1)

Ook nu weer is inruil van opslagruimte tegen berekeningsduur gewenst. Bovendien:

Inruil berekeningstijd tegen extra code

We geven afzonderlijke procedures voor geroteerde en gespiegelde versies van H n; dan zijn de bewerkingen sh.rot en sv.rot niet meer nodig - zij zijn af in de (extra) code verdisconteerd. We benoemen de kromme met de richting van begin naar eindpunt; dus H n = HN n en HO, HW, HZ worden nu extra gedefinieerd: HO n = sv.rot (H n) etc.

HN 0 = []; HO 0 = []; HW 0 = []; HZ 0 = []
HN n = HO (n-1) ++ [O] ++ HN (n-1) ++ [N] ++ HN (n-1) ++ [W] ++ HW (n-1)

```
HO n = HN (n-1) ++ [N] ++ HO (n-1) ++ [O] ++ HO (n-1) ++ [Z] ++ HZ (n-1)
HZ n = HW (n-1) ++ [W] ++ HW (n-1) ++ [W] ++ HW (n-1) ++ [O] ++ HO (n-1)
HW n = HZ (n-1) ++ [Z] ++ HW (n-1) ++ [W] ++ HW (n-1) ++ [N] ++ HN (n-1)
```

Productie van de tekening

Hier geven we een POSTSCRIPT programma voor de Apple LaserWriter.

draw :: [(num, num)] → [char]

draw (lijst van punten in cart. coörd) ↦ POSTSCRIPT programma.

We definieren

```
draw pl = prelude ++ pad ++ postlude
          where
          prelude = "newpath␣" ++ string . hd pl ++ "moveto␣"
          pad = { string p ++ "lineto␣" | p ← tl pl }
          postlude = "0␣setlinewidth␣stroke␣showpage"
          string (x,y) = show x ++ "␣" ++ show y ++ "␣"
          || show converts numbers to strings, amongst others
```

Willen we afdrukken op A4 formaat, dan moeten alle coördinaten in argument pl van draw liggen in 0..500 (= breedte A4 in LaserWriter_points gemeten).

POSTSCRIPT progr := draw . yield . shift (+50, +50) . H 8

Oefeningen

1. Geef conversie functie van stappenlijst naar puntenlijst
2. Kan het omgekeerde ook?
3. Definieer picture == [pad]; geef nu rot, join, cat, shift, scale, mkpict voor pictures ipv paden

4. Beschouw de definitie van H volgens representatie (2). Geef H als extra parameter mee een lijst (tuple) van vier richtingen N,O,Z,W (in zekere permutatie) die gebruikt worden voor de oriëntatie van de kromme. (De bewerking sw.rot op H' wordt nu op dat viertal van H' toegepast).

0. Definieer bij representatie (1) een functie rot1 = rot (pi/4) zonder van sin en cos gebruik te maken.

5. Behandel op gelijke wijze de Peano-, Sierpinsky- en andere krommen. (Wenk. Bij de Sièrpinsky krommen is er geen "natuurlijke" recurrente betrekking tussen S(n) en S(n-1), maar wel tussen (de) vier delen waaruit S(n) is opgebouwd en die van S(n-1).)

T ——

Literatuur

M.Fokkinga, Functioneel programmeren in een vogelvlucht. INFORMATIE vol 27, Nr 10 (1985) pp. 862-873.

Cole, A.J., A note on space filling curves. Software Practice and Experience, Vol 13 (1983) pp.1181-1189.

Wirth, N., Algorithms + Data Structures = Programs. Prentice-Hall, (1976) pp. 130-137 (= §3.3).

T 6. Representeer Hilbertkrommen niet als platte lijsten maar als hiërarchische structuren (bomen, geneste lijsten). Werk nu alles weer uit. Heeft dit enige voordelen? (Ik denk van niet, MF.)