De convexe omhullende geprogrammeerd in Bird's stijl

Maarten Fokkinga, 12 april 1987

Ter illustratie van de beoogde moeilijkheidsgraad van de programmeerproblemen aan het einde van het college "Programmeren I nieuwe stijl" stelt [AJWD 1987] voor: het bepalen van de convexe omhullende. Ik geef in dit verhaal de manier waarop ik dat uitgeprogrammeerd heb, en waarbij ik de stijl van Bird's boek heb proberen te volgen.

Probleemstelling

Ik veronderstel het begrippenarsenaal over het platte vlak bekend. Alles speelt zich af in het platte vlak, en dat zal ik verder niet meer expliciet aangeven. Een polygoon is een gesloten, zichzelf niet doorsnijdende kromme die uit eindig veel (minstens 2) eindig lange (lijn)segmenten bestaat. De (hoek)punten van een polygoon zijn de uiterste punten van de segmenten. Een polygoon verdeelt het vlak in drie disjuncte gebieden: een binnengebied (met eindige oppervlakte), een buitengebied (met oneindige oppervlakte) en de rand (de kromme zelf). Een polygoon heet convex als met ieder tweetal punten p en q in het binnengebied ook alle punten van het segment pq in het

binnengebied liggen.

Gevraagd wordt een functie hull zo dat voor willekeurige eindige puntenverzameling ps geldt: (hull ps) = een convex polygoon waarvan de punten tot ps behoren en die geen punten van ps in z'n buitengebied heeft.

Oplossing 1

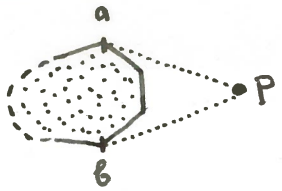
Mijn idee is een reeks van omhullenden $h_0, h_1, h_2, \ldots$ te definiëren die van een steeds groter deel van ps de omhullende zijn: h_{i+1} ontstaat uit h_i door één "nieuw" punt van ps erbij te betrekken. Daartoe (en ook om het einde van de rij te bepalen) beschouw ik tweetallen: $(ps_0, h_0), (ps_1, h_1), (ps_2, h_2), \ldots$. Zodra voor zekere i geldt $ps_i = \emptyset$ dan is h_i = hull ps = de gevraagde omhullende. Initieel nemen we twee willekeurige punten p en p' en vormen daaruit $h_0 = [(p, p'), (p', p)]$. Dus we definiëren:

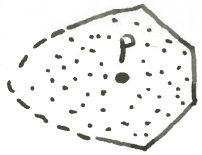
```
point == (num, num)
segment == (point, point)
polygoon == [segment]
```

We gebruiken p voor point, ps voor points ≈ [point], s voor segment, ss voor segments ≈ [segment] == polygoon.

```
hull (p:p':ps) = (snd2 . head . dropwhile nonempty . iterate f) init
                 where  init = (ps, [(p,p'), (p',p)])
                        f (p:ps, ss) = (ps, p bij ss)
                        nonempty = ([] ≠) . fst2
```

Er rest ons nog om de operator bij te definiëren die de convexe omhullende oplevert van een punt p en een convex polygoon ss. Beschouw daartoe de volgende twee gevallen:





In de rechterfiguur ligt p binnen de polygoon, en kan de polygoon onveranderd opgeleverd worden. In de linkerfiguur ligt p erbuiten. Punten a en b verdelen de rand van de polygoon in twee delen: één deel in wiens binnengebied p ligt, en één deel in wiens strikte buitengebied p ligt. (We zeggen dat p in het strikte buitengebied van een deel van een polygoon ligt als het buiten ieder van de segmenten van dat deel ligt.) Deze bewering vergt op zich nog wel een wiskundig bewijs -- waarbij de convexiteits-eigenschap een cruciale rol moet spelen -- maar ten tijde van dit schrijven kan ik het bewijs nog niet geven. De nieuwe polygoon ontstaat nu uit de gegevene

door het ene deel tussen a en b te vervangen door de twee segmenten ap en pb.

De ligging van een punt p = (x,y) ten opzichte van een lijn door ((x1, y1), (x2, y2)) kan men bepalen aan de hand van het teken van (y-y1)*(x2-x1) - (x-x1)*(y2-y1). We associeren een negatief teken met "binnen", positief teken met "buiten" en gelijkheid ~~met~~ aan met "erop".

```
(on), (inside), (outside) :: point → segment → bool
p on s = p wrt s = 0
p inside s = p wrt s < 0
p outside s = p wrt s > 0
(x,y) wrt ((x1,y1),(x2,y2)) = (y-y1)*(x2-x1) - (x-x1)*(y2-y1)
```

In de definitie van bij wordt "het deel a-b waarbinnen" p ligt bepaald als een maximaal deel van segmenten binnen ieder van welke p ligt; wat er dan overblijft is "het deel a-b in wiens strikte buitengebied" p ligt.

```
p bij ss = (replace • split • head • dropwhile out • iterate rotate) ss
           where
           rotate (s:ss) = ss ++ [s]
           out (s:ss) = p outside s
           split ss = (ss1, ss--ss1)
                      where ss1 = takewhile (p inside) ss
```

```
replace (ss1, ss2) = ss1,                          if ss2 = []
                   = ss1 ++ [(a, p), (p, b)],      if ss2 ≠ []
                     where
                     a = (fst2 . head) ss2
                     b = (snd2 . last) ss2
```

Hiermee is oplossing 1 voltooid. Merk op dat in de definities van hull en bij een deel staat dat met recht een "idioom" genoemd kan worden:

$$\text{head} \cdot \text{dropwhile } p \cdot \text{iterate } f$$

Dit betekent zoveel als "de eerste (de beste) van de rij voortgebracht door f, die voldoet aan $\sim p$". Dit idioom heeft als broertje/zusje: "last . takewhile p . iterate f".

Oplossing 2

Deze oplossing is in wezen hetzelfde als de vorige; alleen de formulering verschilt ietwat. De formulering schoot me te binnen toen ik de vorige oplossing netjes wilde omschrijven! We kunnen véél directer, dan met iterate, uitdrukken dat de hull verkregen wordt door één voor één de punten toe te

voegen aan de omhullende van de al beschouwde punten. Kijk maar:

$$\begin{aligned} &hull\ [p, p', p_1, \ldots, p_n] \\ &= p_1 \ \underline{bij}\ (p_2 \ \underline{bij}\ \ldots\ \underline{bij}\ (p_n \ \underline{bij}\ [(p, p'), (p', p)]) \ldots) \\ &= (\ldots((\ [(p,p'),(p',p)] \ \underline{erbij}\ p_1)\ \underline{erbij}\ p_2)\ \ldots)\ \underline{erbij}\ p_n \end{aligned}$$

Operator $\underline{bij}$ is dezelfde als in de vorige oplossing, en $\underline{erbij}$ kunnen we definiëren door: $ss\ \underline{erbij}\ p = p\ \underline{bij}\ ss$. De definitie(s) voor hull luiden dan:

$$\begin{aligned} hull\ (p : p' : ps) &= foldr\ (\underline{bij})\ [(p, p'), (p', p)]\ ps \\ &= foldl\ (\underline{erbij})\ [(p,p'), (p',p)]\ ps \end{aligned}$$

(De gelijkwaardigheid van deze twee alternatieven is inderdaad een gevolg van de Tweede Dualiteitsstelling. In Hoofdstuk 5 van Bird's boek zal aangegeven worden dat --in dit geval-- de definitie mbv foldl qua efficiëntie te verliezen is.)

Conclusie

Ik heb géén correctheidsbewijs van het programma: er zit nog een duidelijke lacune in de argu-

mentatie, zie de beschrijving onder 'Oplossing 1'. Toch vind ik niet dat ik onzorgvuldig of onwishundig te werk ben gegaan. In tegendeel. Ik ben zó "wishundig" te werk gegaan dat ik ook precies weet wáár een essentiële fout zou kunnen zitten. (En ik zou nu dus de computer kunnen gebruiken om mijn vermoeden uit te testen.)

De eenvoud van het programma heeft mij volkomen verrast. Ik had tevoren niet gedacht dat het zo eenvoudig zou zijn / zou kunnen. Temeer niet omdat ik wel eens de tekst van drs. Van Berne, professor Duijvestijn en dr. Van der Hoeven [BDH '85] had gezien en het daar gepresenteerde programma nou niet bepaald vond uitmunten in eenvoudig- en doorzichtigheid. Deze constatering, en mijn voorgaande ervaringen met "Bird's stijl", brengen mij tot de volgende conclusie:

> Het programmeren op functie-nivo is eenvoudiger, óók en juíst voor on-ingewijden (1ste jaars!), dan het programmeren op objectnivo omdat er veel minder clericale details (representatiekeuzen, expliciète recursie) uitge-

druk hoeven worden. De samenstellingen van functies tot een heel programma sluiten precies aan bij de informele beschrijving van de constructie van de gewenste uitkomst.

Het probleem van de convexe omhullende lijkt mij inderdaad geschikt voor eerste jaars: hetzij als extra voorbeeld op het hoorcollege (colstructie), hetzij als praktikumopgave. Voorwaarde daarbij is wel dat "de wiskunde van polygonen etc" duidelijk wordt uitgelegd (of opgefrist - is het VWO-stof?).

Literatuur

[AJWD 1987] Duijvestijn, A.J.W.; Doel van Programmeren I, Notitie voor de commissie HOP. (AJWD - 28-3-87).

[BDH 85] Van Berne, Duijvestijn, Van der Hoeven: Functionele Talen. Informatie Vol 27, Nr 10, 1985, pp 852 - 861.