STRUCTUUR van PROGRAMMEERTALEN

en consequenties voor het

INLEIDEND PROGRAMMEERONDERWIJS

Maarten Fokkinga

18 november 1986

1. Inleiding - algemeen

Programmeeronderwijs:

WAT wil je bereiken, en HOE?

Programmeren - wat is dat?

1. onderkennen/begrijpen/formuleren van probleem/vraag
2. probleem oplossen
3. wiskundig of anderzins geformuleerde oplossing;
   ↑ bv. functioneel ↑ bv. imperatief
   met aandacht voor (tijd-, ruimte-) efficientie
4. instructie v.d. machine ter berekening vd oplossing;
   met aandacht voor
   robuustheid, leesbaarheid, gebruiksgemak etc
5. programmaverwerking
   (syntaxis, fouten opsporen, gebruik v. computer)

Programmeren - wie doen dat?

De rol van **FP** anno 1986:

- redelijk voor 3 (en ook, steeds beter, voor 4)
- onvoldoende voor 1 en 2
- handig bij uittesten specificaties & ideëen

Aantekeningen bij de overheadplaatjes

Ad 1 In de rest van de voordracht ga ik er van uit dat de doelgroep is "zij die later programmeer-experts worden".
Derhalve gelden mijn conclusies niet noodzakelijk voor het Inleidend Programmeer- Service-onderwijs.

Ad 2 Het artikel "Functioneel Programmeren in een Vogelvlucht" maakt géén vergelijking FP <-> IP (wat wij hier dus wél doen), maar presenteert een groot aantal kleine functionele programma's waarmee de lezer zich een beeld kan vormen van de stijl en het karakter van functioneel programmeren.

Ad 2B "Lazy evaluation" ≈ niet van links naar rechts evalueren (en van binnen naar buiten), zoals in Pascal, maar steeds die evaluatiestap doen die, volgens een bepaalde strategie, strikt noodzakelijk is om tot de uitkomst te komen. Bijgevolg vinden de evaluatiestappen kris-kras en haast onvoorspelbaar plaats.

"∞ datastructuren": bijvoorbeeld oneindige lijsten.
Om de eerste 20 priemgetallen te berekenen is het gemakkelijker eerst de oneindige rij van alle priemgetallen te beschrijven, dan om in het generatie proces zelf rekening te houden met het gewenste aantal priemgetallen.

"coroutine" ≈ subroutine die ~~halverwege~~ enige malen tijdens de executie van zijn romp alvast met een resultaat kan terugkeren en dan later weer doorgestart kan worden. De opgeleverde tussenresultaten te samen vormen in feite één grote datastructuur (die dus op deze manier beetje-bij-beetje verwerkt wordt en niet in zijn geheel aanwezig is). Aldus kunnen gescheiden taken ook gescheiden in de tekst geprogrammeerd worden terwijl de opgeroepen berekeningen toch verweven plaats vinden: een modulaire manier van programmastructurering.

"hogere orde functies" ≈ functies die als resultaat en/of als parameter weer functies hebben.

"zelfsturende evaluatie" ≈ je hoeft geen aanwijzingen aan te brengen in je programmatekst (zgn. control structuren) waarmee je de evaluator door je programmatekst stuurt; de evaluator zoekt zelf zijn weg.

"impliciete typering" ≈ je hoeft geen typen expliciet in de programmatekst te vermelden, terwijl er toch type-controle plaats vindt.

"polymorf" ≈ "veelvormig": een functie/procedure die op argumenten van velerlei typen toepasbaar is.

Ad 3 vb2 : Let wel, de engelse vertaling van het latijnse proza van Cardan gebruikt de hedendaagse, zeer precieze, terminologie. De oorspronkelijke tekst is dus nog veel waziger, onduidelijker, en voor misverstanden vatbaarder.
Bedenk eens hoever we met wishunde gekomen zouden zijn als we wishunde nog à la Cardan zouden bedrijven!

Ad 3 vb3 : Legenda: d = dutch, m = municipality, g = gigantic, r = registration, i = inhabitant, o = oldest, a = age; l = local, m = Methuselah.

2. FUNCTIONEEL VERSUS IMPERATIEF

"Functioneel" =

- gebaseerd op wiskundige begrip 'functie'
- Lambda-calculus + syntactisch suiker
- herschrijfsysteem waarbij
  volgorde van 'evaluatie' niet terzake doet

Beter: "Descriptief" $\overset{\text{not}}{\longleftrightarrow}$ Imperatief

A. Op meta-nivo (voor de taaltheoreticus)

B. Voor- en nadelen van FP (t.o.v. Pascal)

C. Imperatieve taalconcepten

M. Fokkinga:
Functioneel Programmeren in een Vogelvlucht
INFORMATIE Vol 27 (1985) Nr 10, pp 862-873

2.A. FP vs IP: op meta-nivo

voor de taaltheoreticus ≠ voor de programmeur

1. Geen assigment
   - geen (neven)effecten, geen "toestand"
   - eenvoudiger semantiek
   - eenvoudiger & gedegener theorievorming
2. Sluit meer aan bij traditionele wiskunde
   - in feite FP ⊆ Wiskunde
   - i.h.b. van invloed op correctheidbewijzen (nml. "substitutiviteit van gelijkheid")
3. Lambda-abstractie ("vorm $(\lambda x.e)$ uit e en x")
   - het technische hulpmiddel voor ABSTRACTIE
   - het middel waarmee alle naamgeving verklaard kan worden.
   - zonder beperkingen mogelijk in FP

2.B. FP vs IP: voordelen FP op gebruikersnivo

1. Geschiktheid voor LAZY EVALUATION
   * ∞ datastructuren
   * coroutine-effect, modulariteit

   (samen: bevordert BEKNOPTHEID)
   * kosteloos hogere orde functies e.d.
   * zelfsturende evaluatie

FP zonder LAZY EVALUATION is achterhaald!

2. Geschiktheid voor hogere orde functies
   * bevordert BEKNOPTHEID
   * juiste abstractie-nivo bereikbaar
3. Geschiktheid voor polymorfe impliciete typering
   * bevordert BEKNOPTHEID
4. Geschiktheid voor orthogonaliteit
   = alle datasoorten staan op voet van gelijkheid
   * vergemakkelijkt programmeren en wijzigingen
5. Geschiktheid voor (parameter-) outleding / matching
   * voorkomt introductie van "irrelevante" namen
   * bevordert BEKNOPTHEID en duidelijkheid

2B FP vs IP: nadelen FP op gebruikersnivo

1. soms te groot ruimtebeslag (bij lazy evaluation)

   [hier wordt aan gewerkt]
2. soms waslijst van parameters nodig

   [persoonlijk onvermogen?]
3. garbage (detection en) collection nodig
4. mentale simulatie v.d. berekeningsstappen onmogelijk / ondoenlijk
   * nodig bij opsporen van fouten [?]
   * nodig bij schatten v.d. complexiteit [?]

2C FP vs IP : imperatieve taalconcepten

In feite: doorschemeren v.h. onderliggende machine/implementatie - model

- Geven de mogelijkheid machine-eigen-aard-igheden uit te buiten, mn. "a[i]:= ..."
- Niet nodig om algoritmen (= executeerbare probleemoplossingen) uit te drukken.

1. Assignment
   dus: sequentie (en, helaas, overspecificatie)
2. Exception handling constructs
3. Coroutines (versus Subroutines)
4. Pointers, gehetende opslagstructuren
5. LIFO_regime voor beheer over opslagruimte
   dus: geen functies als functieresultaten
   (dus geen "procedurele datastructuren")
6. Sprongopdrachten
7. parametermechanismen (val, var, in, out, ref, ...)

3 HET BELANG VAN BEKNOPTHEID

Niet zozeer BEKNOPTHEID als wel GESCHIKTHEID VOOR ALGEBRAISCHE MANIPULATIE

L. Meertens (CWI):
Algorithmics - towards Programming as a mathematical activety *)

Nodig om programmeren net zo te kunnen bedrijven als wiskundigen wiskunde bedrijven.

Huidige wiskunde onhaalbaar

Zonder de huidige algebraische notatie.

*) In: Proc. CWI Symp on Mathematics and Computer Science, CWI Monographs Vol.1 (eds. JW de Bakker et al), pag 289-334, North-Holland, 1986

3. BEKNOPTHEID - VOORBEELD 1

Bereken $1+2+3+\cdots+n$

```
input n;
s, i := 0, 1;
for i from 1 to n do
    s, t := s+t, t+1
endfor;
output s.
```

```
input n;
output n x (n+1) / 2.
```

Wiskundige formulering + correctheidsbewijs:

$$\sum_{i=1}^{n} i = \tfrac{1}{2}\left(\left(\sum_{i=1}^{n} i\right)+\left(\sum_{i=1}^{n} i\right)\right)$$

$$= \tfrac{1}{2}\left(\left(\sum_{i=1}^{n} i\right)+\left(\sum_{i=1}^{n} n+1-i\right)\right)$$

$$= \tfrac{1}{2}\left(\sum_{i=1}^{n} i + n+1-i\right)$$

$$= \tfrac{1}{2}\, n\,(n+1)$$

3 BEKNOPTHEID - voorbeeld 2

Vind oplossing x voor $x^3 + px = q$:

Cardan (in Ars Magna, 1545):

> Raise the third part of the coefficient of the unknown to the cube, to which you add the square of half the coefficient of the equation, & take the root of the sum, namely the square one, and this you will copy, and to one [copy] you add the half of the coefficient that you have just multiplied by itself, from another [copy] you substract the same half, and you will have the Binomium with its Apotome, next, when the cube root of the Apotome is substracted from the cube root of ~~the~~ it: Binomium, the remainder that is left from this, is the determined value of the unknown.

In hedendaagse notatie:

Zij $c=\sqrt{d}$, $d=\left(\frac{p}{3}\right)^3+\left(\frac{q}{2}\right)^2$, en zij $b=c+\frac{q}{2}$, $a=c-\frac{q}{2}$

Dan $x=\sqrt[3]{b}-\sqrt[3]{a}$.

Correctheidsbewijs:

Vier regels eenvoudige manipulatie.

(Geen uittesten!!)

3. BEKNOPTHEID - voorbeeld 3

Vind de oudste inwoner van Nederland.

```
input dm, mr;
gdb := ∅;
for m ∈ dm do
   gdb := gdb , mr[m]
endfor;
aoi := -∞;
for i ∈ gdb do
  if i.age > aoi then
     oi, aoi := i, i.age
  endif
endfor;
output oi.
```

```
input dm, mr;
slm := ∅;
for m ∈ dm do
   alm := -∞;
   for i ∈ mr[m] do
      if i.age > alm then
         lm, alm := i, i.age
      endif
   endfor;
   slm := slm ∪ {lm}
endfor;
aoi := -∞;
for i ∈ slm do
   if i.age > aoi then
      oi, aoi := i, i.age
   endif
endfor;
output oi.
```

NL hetzelfde in Meertens' notatie

$\uparrow_{age}/\ +/\ ..r\bullet$ | $\uparrow_{age}/\ (\uparrow_{age}/\ mr)\bullet$

En het "correctheids" bewijs

$$\uparrow_{age}/\ +/\ mr\bullet \underset{\{law\ 3\}}{=} \uparrow_{age}/\ \uparrow_{age}/\bullet\ mr\bullet \underset{\{law\ 1\}}{=} \uparrow_{age}/\ (\uparrow_{age}/\ mr)\bullet$$

3 BEKNOPTHEID - voorbeeld 4

Uit een mijner speelgoedprogramma's:

$$t\ w = 0\downarrow\ \downarrow/\ \#\bullet\ (1+)\bullet\ r'_w : \alpha\ tl\ w$$

r.h.s. te ontleden als:

$$0\downarrow\ (\downarrow/\ (\#\bullet\circ((1+)\bullet\circ(r'_w : (\alpha\circ(tl\circ w)))))$$

(?) Extra reden voor het streven naar BEKNOPTHEID

$$\text{productiviteit} = \#\text{regels}/_{\text{tijdsduur}} = \text{constant}_{\text{programmeur}}$$

4 OH, WAAR GAAT DAT HEEN ?

ALGORITMIEK :

* wiskundige activiteit

  dus wiskundige vaardigheden vereist.

  Vergt **TRAINING** en gevoel voor elegantie

* notatie - heel afwijkend van huidige
  (qua beknoptheid vergelijkbaar met APL)

* NIET formaliseerbare taal
  (wel: geïmplementeerd deel)

  Zelf verzinnen ad-hoc symbolen en notatie
  soms nodig!

* Geleidelijke overgang van spec → implementatie

[en.... waarom zou
zou Assignment niet in Algoritmiek passen?...]

4 OH, WAAR GAAT DAT HEEN ?

Functionele Talen anno 1986

- nog te beperkt en nog niet beknopt genoeg
  deel van Algoritmiek-notatie

- schieten te kort wanneer zij als enig
  middel bij Spec ⇝ Impl gebruikt worden

  (Er zijn meer (executeerbare!) notaties/concepten nodig

Vuistregels voor omzetting FP → Pascal

die altijd uitkomst bieden, zijn er niet!

Funct. Progr.'s kunnen wel als inspiratie-
bron dienen.

5 CONSEQUENTIES VOOR INLg PROGR. ONDERWIJS

1. Het ideaal is nog lang niet bereikt,
   maar pas in zicht aan de horizon.

2. FP is 1ste stap op weg naar Algoritmiek
   - redelijke beknoptheid
   - goede abstractie m.b.t. machine-aspecten

ONDERWIJS IN FP IS NUTTIG EN NODIG !

Bij eenvoudige programma's: FP net zo moeilijk als het formuleren v.d. invariant van een herhaling

5 CONSEQUENTIES - vervolg

WAARSCHUWING 1

Zoals BASIC het vermogen tot Abstractie ruïneert, zo ruïneert FP het gevoel voor "von Neumann efficientie"

Parallelle invoering van FP en IP lijkt mij goed mogelijk, en onderwijskundig/didactisch niet bezwaarlijk

WAARSCHUWING 2

FP is "moeilijker" dan IP:
er is een groter vermogen tot wiskundig formuleren vereist.

Maar ... wie dat vermogen niet heeft, zal toch geen groot probleemoplosser (programmeur?) worden

Bij eenvoudige programma's: FP net zo moeilijk als

6 NOG ENIGE STELLINGNAMEN

Lisp: té imperatief om als FP beschouwd te worden. (SETQ, RPLACA, PROG; evaluatievolgorde ook in Puur Lisp belangrijk; geen lazy evaluation; soms: dynamic scope)

Lisp: wel verdienstelijk vanwege "selfreflexiveness"

APL & Iverson: goed voorbeeld van hoe het wel moet en hoe het niet moet.

Miranda:

- nog beknopter dan Twentel
- nog meer "wiskundige stijl"
- heeft dus mijn voorkeur

(bovendien: gebruikersvriendelijke progr. omgeving, faciliteiten voor Abstracte Datatypen, etc.)