Laten we het niet erger maken

De waarheid is een kind van de tijd, niet van de autoriteit. (Leonardo Boff)

0 goede onderzoekers zijn bescheiden ...

Een welgekende onderneming, niet (meer) hier ter stede, gebruikt tegenwoordig als slogan: *Let's make things better*. Deze onderneming produceert allerhande artikelen en vele daarvan zijn voor verbetering vatbaar: *Let's make better things* zou meer op zijn plaats geweest zijn.

Iedere goede wetenschappelijk onderzoeker weet dat hij al heel tevreden mag zijn als zijn werk kan varen onder de vlag: *Let's not make things worse*. Lex Bijlsma is zo'n bescheiden onderzoeker.

1 en verdraagzaam

Iedere wetenschappelijk onderzoeker heeft het recht zijn eigen spel te spelen. Hij mag daarbij best de hoop koesteren dat zijn spel de tand des tijds zal weerstaan, maar het besef dat de meeste pogingen, hoe oprecht en zorgvuldig ook, een droeviger lot zal zijn beschoren, verplicht hem zich verdraagzaam op te stellen jegens onderzoekers die een ander spel spelen, of hetzelfde spel anders. Lex Bijlsma is zo'n verdraagzame onderzoeker. Hij heeft zich vele soorten spel eigen gemaakt en op waarde geschat, maar hij heeft nooit (spel of spelers) veroordeeld; iets wat niet van iedereen kan worden gezegd.

Dat Lex Bijlsma ons nu gaat verlaten is voor ons erg genoeg, maar laten we het niet erger maken door hierover te treuren. Laten we in plaats daarvan eens terugblikken en zien wat de laatste (pakweg) 20 jaren ons in de Informatica hebben gebracht. De voorbeeldjes die ik hierbij gebruik zijn niet nieuw maar wel leerzaam.

2 laten we rekenen

Een onmiskenbare, hoewel nog lang niet overal op waarde geschatte, ontwikkeling is de opkomst van een meer calculationele stijl van probleem oplossen, met toepassingen bij het construeren van zowel programma's als wiskundige bewijzen. Hierbij spelen formules en symbolen een veel belangrijker rol dan in de traditionele wiskunde.

De voordelen van een calculationele werkwijze zijn dat:

- een programma en zijn correctheidsbewijs gelijktijdig kunnen worden ontwikkeld, wat (lastige) a posteriori verificatie onnodig maakt;
- de vorm van de formules heuristische aanknopingspunten biedt;
- ontwerpbeslissingen en gebruikte aannamen expliciet worden gemaakt;
- aldus de wiskundige structuur van de redenering wordt verhelderd.

Hier is een eenvoudig voorbeeld. We willen bewijzen dat $\sqrt{p}$ irrationaal is, voor een gegeven, vast priemgetal p. Hiertoe introduceren we een functie f, met als informele betekenis dat $f{\cdot}x$ het aantal keren is dat p als factor voorkomt in x. Deze functie heeft dan (onder meer) de volgende eigenschappen, voor onze vaste p en voor alle positieve natuurlijke getallen x and y:

$$\begin{array}{ll} f{\cdot}p & = \ 1 \\ f{\cdot}(x{*}y) & = \ f{\cdot}x \ + \ f{\cdot}y \end{array}$$

Dit is alles wat we nodig hebben voor de volgende berekening:

$$\begin{array}{ll} & \text{“}\sqrt{p}\text{ is rationaal”} \\ \equiv & \quad \{ \text{ definitie van “rationaal” } \} \\ & (\exists x, y :: \sqrt{p} \ = \ x/y \,) \\ \equiv & \quad \{ \text{ definitie van } \sqrt{\ } \} \\ & (\exists x, y :: p{*}y^2 \ = \ x^2 \,) \\ \Rightarrow & \quad \{ \text{ Leibniz, om } f \text{ in het spel te brengen } \} \\ & (\exists x, y :: f{\cdot}(p{*}y^2) \ = \ f{\cdot}(x^2) \,) \\ \equiv & \quad \{ \text{ eigenschappen van } f \ \} \\ & (\exists x, y :: 1{+}2{*}f{\cdot}y \ = \ 2{*}f{\cdot}x \,) \\ \equiv & \quad \{ \text{ even getallen verschillen van oneven getallen } \} \\ & \mathsf{false} \quad , \end{array}$$

waaruit we concluderen dat $\sqrt{p}$ irrationaal is. In dit bewijs hebben we niet geëist dat x en y geen delers gemeen hebben –wat in andere bewijzen nogal eens gebeurt–, en we hebben niet expliciet gebruik gemaakt van volledige inductie: die zit “verstopt” in de eigenschappen van f.

In dit bewijs heb ik nergens (expliciet) gebruikt dat p een priemgetal is: geldt de stelling *dus* ook voor alle andere getallen? Nee, natuurlijk niet: dat

p priem is zit ook verstopt in de definitie van f: calculationele redeneringen dragen ook bij aan (en lukken ook alleen maar met) een goede modularisering van het bewijs.

3 goede notatie is belangrijk, ...

Om het calculationele spel effectief en efficiënt te kunnen spelen is de keuze van de gebruikte notatie van cruciaal belang. Probeer, bijvoorbeeld, maar eens staartdelingen te doen met Romeinse Cijfers!

Een ander eenvoudig voorbeeld is het gebruik van infixnotatie voor binaire operatoren die associatief zijn. Bij het gebruik van prefixnotatie word ik gedwongen tot een irrelevant keuze; ik moet dan bijvoorbeeld kiezen tussen:

$$append(x, append(y, z))$$

en:

$$append(append(x, y), z) \quad ,$$

en dat wil ik helemaal niet! Met infixnotatie (en weglating van haakjes) kan ik het hele dilemma vermijden:

$$x +\!\!+ y +\!\!+ z \quad ,$$

en de formule op elk moment ontleden zoals het mij dan uitkomt.

4 aangepast aan de toepassing, en ...

Ook belangrijk is dat de notatie is aangepast aan het doel dat zij dient. Logici, bijvoorbeeld, bedrijven metatheorie –"wat is een goede logica?"– en metametatheorie –"wat is een metatheorie?"–; voor dat doel hebben ze notaties nodig waarin voor hetzelfde begrip verschillende symbolen worden gebruikt, op verschillende niveaus van abstractie. (Bovendien zijn er dan regels nodig om die niveaus met elkaar te verbinden en aldus de verschillen te overbruggen.) Aldus ontstaan er formules zoals, bijvoorbeeld, de (zogenaamde) regels voor introductie en eliminatie van de typeconstructor $\rightarrow$:

$$\frac{\Gamma, x:U \ \vdash \ E:V}{\Gamma \ \vdash \ (\lambda x.E) \ : \ U \rightarrow V}$$

$$\frac{\Gamma \ \vdash \ (\lambda x.E) \ : \ U \rightarrow V \quad , \quad \Gamma \ \vdash \ F:U}{\Gamma \ \vdash \ E(x:=F) \ : \ V}$$

Als je geen logica wil bedrijven maar wil programmeren is het met zulke formules niet echt lekker rekenen. Gelukkig hebben programmeurs niet zulke fijne onderscheidingen nodig als logici; zij kunnen zich daarom veroorloven gewoon te formuleren wat ze ook bedoelen:

$$(U \rightarrow V)\cdot(\lambda x.E) \quad \equiv \quad (\forall x :: U\cdot x \Rightarrow V\cdot E) \quad .$$

Het is niet eens zo moeilijk te laten zien dat die twee regels *precies* de twee implicaties van deze equivalentie voorstellen, ook al wordt dat door de gebruikte notatie wat verdoezeld.

5 beknopt

Goede notatie is niet alleen nauwkeurig maar ook beknopt: anders worden de formules onhanteerbaar en de berekeningen zo bewerkelijk dat het hele spel ondoenlijk wordt.

Een prachtig voorbeeld van een beknopt formalisme is de puntvrije relationele calculus. Hierin kan, bijvoorbeeld, de mededeling dat een relatie transitief is zo worden geformuleerd:

$$[\, R ; R \Rightarrow R \,] \quad .$$

De transitieve afsluiting, $*S$, van een relatie S kan worden gedefinieerd als de sterkste oplossing van de vergelijking:

$$X : \quad [\, S \Rightarrow X \,] \;\wedge\; [\, X ; X \Rightarrow X \,] \quad .$$

Als we nu een (gerichte) graaf definiëren als een relatie, S, op een gegeven verzameling (knopen), dan kan de mededeling dat graaf S samenhangend is, worden geformuleerd als:

$$[\, *S \,] \quad ,$$

en dat is veel beknopter en mooier dan de traditionele definitie in termen van (het technisch lastige) begrip “paden”: die paden, en de bijbehorende volledige inductie, zitten nu adequaat “verstopt” in de transitieve afsluiting.

Een andere, minder voor de hand liggende, definitie van samenhang in een graaf is: voor elke kleuring der knopen geldt dat, als er twee knopen zijn van verschillende kleur, dan zijn er twee knopen van verschillende kleur verbonden door een pijl. De relatie “hebben dezelfde kleur” is transitief; noemen we deze relatie R dan kan deze andere karakterisering van samenhang relationeel worden geformuleerd als:

$$[\, S \Rightarrow R \,] \;\Rightarrow\; [\, R \,] \quad .$$

Het bewijs dat dit volgt uit onze eerdere definitie van samenhang is nu een fluitje van een cent:

$$\begin{array}{ll} & [\, S \Rightarrow R \,] \\ \Rightarrow & \quad \{\ R \text{ is transitief } \} \\ & [\, S \Rightarrow R \,] \;\wedge\; [\, R ; R \Rightarrow R \,] \\ \Rightarrow & \quad \{\ \text{definitie van } *S \text{ als de sterkste } \ldots \ \} \\ & [\, *S \Rightarrow R \,] \\ \equiv & \quad \{\ \text{S is samenhangend } \} \\ & [\, \mathsf{true} \Rightarrow R \,] \\ \equiv & \quad \{\ \text{predikatenrekening } \} \\ & [\, R \,] \quad . \end{array}$$

6 de Steen der Wijzen bestaat niet

Een formalisme is een middel tot een doel, en je moet niet de fout maken het middel zelf tot doel te verheffen. Als beknoptheid het doel is, is puntvrijheid hiertoe een middel, zoals in de relationele calculus overtuigend is gebleken. Maar er bestaat geen universeel middel tegen alle kwalen, en dat geldt ook voor puntvrijheid.

Hier is een voorbeeld. We beschouwen functies op (eindige) lijsten. Voor gegeven predikaat Q op lijsten en voor gegeven integer functie L op lijsten, kunnen wij geïnteresseerd zijn in een integer functie F op lijsten, informeel gekarakteriseerd door: $F{\cdot}s$ is de maximale waarde die functie L aanneemt op de verzameling van al die segmenten van s die aan Q voldoen. (Neem je, bijvoorbeeld, voor Q het constante predikaat true en voor L "som", dan krijg je het bekende probleem van de maximale segmentsom; neem je voor Q "is constant" en voor L "lengte" dan krijg je het probleem van de maximale lengte van enig constant segment.)

Onze functie F kan aldus formeel worden gespecificeerd, waarbij de dummies s, x, y en z alle van het type "lijst" zijn:

$$F{\cdot}s \;=\; (\,\mathsf{max}\, x, y, z : x +\!\!+ y +\!\!+ z = s \wedge Q{\cdot}y :\, L{\cdot}y\,) \quad , \text{for all } \; s \quad .$$

Om dit te kunnen begrijpen moet je vertrouwd zijn met lijsten en concatenatie $+\!\!+$, en met kwantificatie en maximum.

Functie F kan ook puntvrij worden gespecificeerd; hier betekent dat: zonder variabelen van het type "lijst". Hier is zo'n specificatie zoals ik die (nog niet zo lang geleden) heb gezien:

$$F \;=\; \mathit{foldr}{\cdot}(\mathsf{max}){\cdot}(-\infty) \circ (L*) \circ \mathit{filter}{\cdot}Q \circ \mathit{segs} \quad ,$$

waarin de functie *segs* een lijst afbeeldt op een lijst die alle segmenten van die lijst bevat, in een of andere volgorde:

$$\mathit{segs} \;=\; \mathit{foldr}{\cdot}(+\!\!+){\cdot}[\,] \circ (\mathit{inits}*) \circ \mathit{tails} \quad .$$

Dit is al ingewikkelder dan de versie met variabelen, en dan ontbreken nog de definities voor de functies *inits* en *tails*! Verder moet je om dit te begrijpen vertrouwd zijn met *foldr*, maximum, $*$ ("map"), *filter* en concatenatie $+\!\!+$.

Hier is de puntvrije specificatie duidelijk minder beknopt. Bovendien is zij overspecifiek op een manier die hoogst onwenselijk is, want er zijn allerlei irrelevante maar niet onschuldige keuzen gemaakt: in plaats van *foldr* had je ook *foldl* kunnen gebruiken; omdat *foldr* op twee plaatsen voorkomt betekent dit al een keuze uit 4 varianten[0]. Evenzo had *segs* met evenveel recht kunnen worden gedefinieerd als:

$$\mathit{segs} \;=\; \mathit{foldr}{\cdot}(+\!\!+){\cdot}[\,] \circ (\mathit{tails}*) \circ \mathit{inits} \quad .$$

Specificaties moeten zo neutraal mogelijk worden geformuleerd, want elke voorbarige keuze is een potentieel struikelblok voor het afleiden van (efficiënte) programma's.

7 specificaties hoeven niet uitvoerbaar te zijn

De puntvrije specificatie uit het laatste voorbeeld is een bijzondere: het is meteen een uitvoerbaar programma, al is het inefficiënt. Het is lange tijd als een voordeel van functioneel programmeren beschouwd dat je in de functionele taal uitvoerbare specificaties kunt opschrijven, en dat je geen afzonderlijk formalisme voor specificaties nodig zou hebben, maar dat is een misvatting.

Programmeren, dat wil zeggen, het maken van efficiënte programma's, komt dan vooral neer op transformeren: van inefficiënt naar efficiënt, waarbij het uitgangspunt dan geacht wordt "obviously correct" te zijn. Het vorige voorbeeld toont al aan dat dit een illusie is: voor de puntvrije versie is helemaal niet "obvious" dat die weergeeft wat er werd bedoeld. Bovendien zijn

[0] hetgeen te vermijden was geweest, door gebruik van het neutrale *fold*.

algoritmen, naar hun aard, altijd overspecifiek en dat is, zoals gezegd, onwenselijk: specificaties moeten helder zijn, omwille van hun valideerbaarheid, en verder zo neutraal mogelijk ten aanzien van de mogelijke oplossingen.

Hier is een ander eenvoudig voorbeeld van een niet-uitvoerbare specificatie waarbij ik me geen heldere en eenvoudige wel-uitvoerbare specificatie kan voorstellen.

In dit voorbeeld staan $\mathcal{L}_2$ en $\mathcal{L}_3$ voor de eindige lijsten met elementen in $\{0,1\}$ en $\{0,1,2\}$, respectievelijk. Verder zijn er functies *v2*, van type $\mathcal{L}_2 \to \mathsf{Nat}$, en *v3*, van type $\mathcal{L}_3 \to \mathsf{Nat}$, gegeven met de volgende definities (met $[\,]$ ("empty") en $\triangleright$ ("cons") voor de lijstconstructors):

$$\begin{array}{lcl@{\qquad\qquad}lcl} \mathit{v2}\cdot[\,] & = & 0 & \mathit{v3}\cdot[\,] & = & 0 \\ \mathit{v2}\cdot(a \triangleright s) & = & a + 2 * \mathit{v2}\cdot s & \mathit{v3}\cdot(b \triangleright t) & = & b + 3 * \mathit{v3}\cdot t \end{array}$$

Functie *v2* interpreteert een rij cijfers als een natuurlijk getal, volgens de regels van het tweetallig stelsel, terwijl *v3* hetzelfde doet maar dan volgens de regels van het drietallig stelsel. De omzetting van tweetallig naar drietallig stelsel kan nu worden gemodelleerd als een functie *c23*, met type $\mathcal{L}_2 \to \mathcal{L}_3$, en met als specificatie:

$$(\forall s :: \mathit{v3}\cdot(\mathit{c23}\cdot s) \;=\; \mathit{v2}\cdot s) \quad .$$

Desgewenst kan *c23* ook beknopter puntvrij worden gespecificeerd, maar veel helpt dat niet en de specificatie blijft niet-uitvoerbaar:

$$\mathit{v3} \circ \mathit{c23} \;=\; \mathit{v2} \quad .$$

We kunnen *c23* ook expliciet specificeren, met behulp van de inverse van *v3*:

$$\mathit{c23} \;=\; \mathit{v3}^{-1} \circ \mathit{v2} \quad ,$$

maar functie-inversie is gewoonlijk geen operator in functionele talen, dus daar wordt de specificatie ook niet uitvoerbaar van.

Er is een hele klasse problemen waarvan de specificaties zulke impliciete vormen aannemen; behalve conversieroutines vallen ook sorteeralgoritmen en compilers –ook een soort conversieroutines– in deze klasse.

8 toegift

Een ander veel gehoord argument waarom functionele programma's geen aparte specificaties zouden behoeven, is dat ze "self-explaining" zouden zijn. Inderdaad, iedereen begrijpt bijvoorbeeld onmiddellijk dat dit een programma is voor het berekenen van de Fibonacci-getallen:

$$
\begin{array}{lcl}
F{\cdot}i & = & G{\cdot}i{\cdot}0{\cdot}1{\cdot}0{\cdot}1 \\
 & & \\
G{\cdot}0{\cdot}b{\cdot}c{\cdot}x{\cdot}y & = & x \\
G{\cdot}(2{*}i){\cdot}b{\cdot}c{\cdot}x{\cdot}y & = & G{\cdot}i{\cdot}(b{*}b{+}c{*}c){\cdot}(2{*}b{*}c{+}c{*}c){\cdot}x{\cdot}y \\
G{\cdot}(i{+}1){\cdot}b{\cdot}c{\cdot}x{\cdot}y & = & G{\cdot}i{\cdot}b{\cdot}c{\cdot}(b{*}x{+}c{*}y){\cdot}(c{*}x{+}b{*}y{+}c{*}y)
\end{array}
$$

9 besluit

Kortom, wij hebben de afgelopen 20 jaren heel wat bijgeleerd, en hieraan heeft Lex Bijlsma zijn steentjes bijgedragen. Waarvoor onze dank!

Eindhoven, 26 februari 1999

Rob R. Hoogerwoord
faculteit der Wiskunde en Informatica
Technische Universiteit Eindhoven
postbus 513
5600 MB Eindhoven