Programmacorrectheid en het college Inleiding Informatica

R.P. van de Riet

Faculteit Wiskunde en Informatica
Vrije Universiteit, Amsterdam

Februari 1989

1. Inleiding

Wanneer het verzoek bij je komt om een bijdrage te leveren voor het liber amicorum van Jaco de Bakker en je bent, zoals ik, studie- en jaargenoot van hem geweest, je hebt samen een tijd in de rekenafdeling van het toenmalige Mathematisch Centrum doorgebracht en je bent beide hoogleraar informatica aan dezelfde universiteit dan kun je dit verzoek niet weigeren.
Het onderwerp dat ik koos lag voor de hand: mijn ervaringen bij het geven van onderwijs, met name in het college Inleiding Informatica, op het gebied van programmacorrectheid. Het gaf me de mogelijkheid iets van de ervaringen van de afgelopen 15 jaar op een belangrijk gebied waar de invloed van De Bakker intensief is geweest te vertellen.
Dat ik daarbij de kans laat lopen over de ervaringen van de eerste tien jaren te vertellen neem ik maar voor lief; het bespaart me mijn ongenoegen te uiten over het feit dat, hoewel ik hem in vele functies vóór was (medewerker MC, sous-chef rekenafdeling, hoogleraar VU) hij het van mij gewonnen heeft met het vieren van dit vijf en twintig-jarig jubileum.
Ik kan hem daarmee van harte feliciteren en wil hem ook bedanken voor de vele jaren van voortreffelijke samenwerking die geweest zijn en die ongetwijfeld nog zullen komen.

De lezer vraag ik dit verhaal te lezen tegen de achtergrond van het gelegenheidsaspect ervan en rekening te houden met het feit dat ik geen deskundige ben op het gebied van de programmacorrectheid.

2. Het college Inleiding Informatica in de beginjaren

In 1970, toen ik bij de VU begon, startten Jim van Keulen en ik de cursus Inleiding Informatica dat uit een college bestond en een practicum. Voor het college werd een diktaat gebruikt dat ook bij het MC werd gebruikt voor het "Oriënterend Colloquium Informatica", een cursus voor leraren wiskunde. Het diktaat bevat vier hoofdstukken: "Inleiding", "algoritmen", "De werking van een Computer", waarin de Verschrikkelijk Eenvoudige RekenAutomaat (VERA een bijzondere uitvinding van mezelf; Wolbers gebruikte in die tijd dezelfde naam voor een Delftse uitvinding) en de MIX computer van Knuth [Knu] werden beschreven, en "De werking van een assembler geschreven in ALGOL 60". Het is duidelijk waar in die tijd mijn interesse naar uitging: programmeren en computers; correctheid kwam nog niet voor.

6 maart 1989

Het Koude Start Programma (KSP) van de MIX was van grotere interesse. Voor de liefhebbers: het bootstrapprogramma voor onze MIX implementatie (een ALGOL 60 programma van negen en een halve pagina's) las de inhoud van één kaart op een zekere plaats in het geheugen. Het KSP was bedoeld om gelezen te worden van die kaart zó dat de MIX computer daarna een willekeurig programma en data op veel meer kaarten kon lezen en verwerken. Curiositeitshalve volgen hier de 80 symbolen op die kaart:

0E02.0705A0100,0705U0D05A0100,0D05U001G=0015A0115H000050015U0100.0704¬0123456789

Het was een probleem bij het maken van de KSP dat een noodzakelijke instructie niet was voor te stellen door symbolen op de ponskaart, simpelweg omdat de interne code ervoor niet overeenkwam met een (MC-ALGOL-RESYM) code. De oplossing was de instructie te "bakken", d.w.z. eerst als data te laten uitrekenen. Dat deze oplossing vanuit het oogpunt van programmacorrectheid een erg slechte was wisten we toen niet. Hetzelfde idee werd (en wordt nog steeds) in LISP toegepast. In een latere versie van het collegediktaat Inleiding Informatica heb ik dat idee toegepast voor een computer die slechts één soort instructie kent, de "Een Instructie Computer" (EIC). Feitelijk werden in die computer vrijwel alle instructies eerst "gebakken". Er kwamen instructies in voor die "zichzelf" veranderen (een thema waarover ik het in de laatste paragraaf nog wil hebben). Uiteraard is het idee al van Turing afkomstig, die immers de Turing Machine voorziet van een tape waarop eerst data wordt gezet die daarna als instructies wordt gelezen.

De reden dat ik zo uitweid over die eerste cursus is om aan te geven waar toen bij mij de grootste interesse lag: problemen overwinnen die gesteld werden door bijzondere eigenschappen van de hardware. Ik kon me de luxe veroorloven die problemen in ALGOL 60 te bekijken, hoewel ik als verantwoordelijke voor het functioneren van de EL X8 computer ook exercities uitvoerde op het niveau van machine-instructies, en dat was bepaald veel lastiger.

Dat mijn aandacht momenteel veel meer gericht is op problemen die door de toepassingen worden gesteld, met name toepassingen op het gebied van informatiesystemen, is een tegenstelling die ik graag onderstreep. Het heeft ook veel te maken met een verschuiving van het **hoe** naar het **wat**, van imperatief naar declaratief denken, van programmeren naar specificeren. Dit laatste heeft dan weer nauw verband met programmacorrectheid.

Het eerste VU collegediktaat Inleiding Informatica is van 1972. Het gaat weer over algoritmen (als voorbeeld wordt een algoritme in mini-ALGOL gegeven die gebruikt kon worden om zich te begeven naar een protestdemonstratie, een actueel onderwerp in die tijd!), over de

programmeertaal ALGOL 60, representatie van informatie, booleans, integers en reals, machine code (VERA), talen en vertalen (waarin ik kon verwijzen naar eigen werk [vdR]), het operating systeem en de "Computer en Maatschappij" (een onderwerp dat nog steeds even actueel is als toen). Geen Programmacorrectheid.
In het diktaat van 1973 ontbreekt het hoofdstuk "Computer en Maatschappij", maar de eerlijkheid gebiedt te zeggen dat het eerder genoemde stuk slechts uit een halve pagina literatuurverwijzingen bestaat. Opmerkelijk is hier dat het werken met magnetische tapes wordt uitgelegd aan de hand van een programma om een straat in een stad te zoeken. Verder wordt uitvoerig ingegaan op compiler-compilers, inderdaad een interessant onderwerp, maar nog steeds geen programmacorrectheid.

3. Het college Inleiding Informatica in de midden periode

In 1973 werd De Bakker bij de VU benoemd om les te geven op het gebied van de Programmeertheorie. Blijkens het collegediktaat "Inleiding Programmeren" uit 1976 is de programmeertheorie de

"studie van theoretische grondslagen voor algoritmen".

Hier wordt de programmacorrectheid niet expliciet genoemd, maar inmiddels is de betekenis van dit begrip wel doorgedrongen in het collegediktaat Inleiding Informatica. In dat jaar verschijnt dit diktaat namelijk voor de eerste keer met een hoofdstuk over programmeermethoden, waarin ruime plaats aan het begrip programmacorrectheid wordt gegeven.
Het college krijgt dan een vorm die redelijk aan de intentles voldoet, namelijk een overzicht van het gehele gebied van de informatica zodat studenten bij het begin van de informaticastudie, toen nog ingebed in de wiskundestudie, een redelijk overzicht kregen van het geheel van de informatica. Zodat ook de docenten, en inmiddels waren er drie: Tanenbaum, de Bakker en ikzelf, hun colleges konden geven die enigszins herkenbaar pasten in een groter geheel. We achtten het van belang dat de studenten de samenhang met andere vakken reeds kenden alvorens een bepaald vak dieper gingen bestuderen. Aldus zouden ze ook beter gemotiveerd het betreffende college volgen. Met name vond en vind ik dit van belang voor de theoretische vakken, aangezien er een behoorlijk gevaar dreigt dat met name deze vakken door de studenten niet als echt relevant herkend worden en als "wiskundige ballast" gezien worden. Een feit is dat het tentamen Programmeertheorie door veel studenten wordt uitgesteld tot het laatste moment.

6 maart 1989

programmacorrectheid

De manier waarop het onderwerp programmacorrectheid werd en nog steeds wordt behandeld is als volgt. Eerst wordt het Software Engineering probleem geïntroduceerd: dat het maken en onderhouden van software een dure aangelegenheid aan het worden is; dat dit met gestructureerd programmeren te maken heeft en met programmacorrectheid. Het sorteren van een array met reële getallen wordt als voorbeeld genomen. Het betreft een bepaald soort schuiven, zodat we het i.h.v. met "Schuifsort" zullen aanduiden. Er wordt een stroomschema (dit is de enige plaats in vermoedelijk de hele studie waar onze studenten het begrip stroomschema tegenkomen) opgesteld en aan de hand daarvan een Pascal programma (momenteel wordt Modula 2 gebruikt, want we gaan met de tijd meel). Dit programma heeft een zeer ondoorzichtige structuur vanwege de goto's. (In Modula is er een complicatie omdat geen goto's gebruikt kunnen worden, zodat je ook niet kunt laten zien hoe slecht ze zijn). Dit wordt aangeduid als "ongestructureerd" programmeren. Daarna wordt hetzelfde programma opgezet met de bekende trits methoden: sequentie, conditioneel, repetitie en er wordt gedemonstreerd hoe veel beter leesbaar het resultaat is geworden: een "gestructureerd programma". De idee van successieve verfijning wordt vervolgens geïllustreerd. Daarna komt de programmacorrectheid aan bod.
Een natuurlijke vraag die gesteld en beantwoord wordt is: "je beweert dat Schuifsort dat array sorteert, maar kun je dat ook bewijzen?"

Voor mij, opgeleid als wiskundige, is dit altijd weer een boeiende vraag. Je moet uitleggen wat een bewijs is. In de eerste jaren deed ik dat niet goed. Later kwam John-Jules Meyer, medewerker van De Bakker, met uitbreidingen in de vorm van bewijsregels, correctheidsformules, enz. waardoor het diktaat aanzienlijk verbeterde. De eerste keer dat ik er over schreef meende ik te kunnen volstaan met de bekende Floyd axioma's, die ik nu als volgt noteer:
voor de sequentiële structuur:

$$\frac{\{P\}\ S\ \{Q\}\ ,\ \{Q\}\ T\ \{R\}}{\{P\}\ S\ ;T\ \{R\}}$$

voor de if-then-else structuur:

$$\frac{\{P\wedge B\}\ S\ \{Q\},\{P\wedge\neg B\}\ T\ \{Q\}}{\{P\}\ \text{if B then S else T}\ \{Q\}}$$

6 maart 1989

voor de repetitiestructuur:

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \text{while } B \text{ do } S\ \{P \wedge \neg B\}}$$

Bovendien voerde ik als axioma voor de assignment statement in:
"als de uitspraak B(a) waar is vóór de assignment "b:= a" dat dan, in principe, de uitspraak P(b) waar is na de assignment". (Let op: "in principe") Met deze hulpmiddelen werden "bewijzen van correctheid" gegeven voor de delingsalgoritme, de algoritme van Euclides (uiteraard dat moet), twee sorteringsalgoritmes, Maxsort (gebaseerd op het vinden van maxima) en Schuifsort en het werd toegepast bij de behandeling van Quicksort. Bij Euclides gebruikte ik als definitie voor grootste gemene deler de bekende eisen van deelbaarheid. Als loop-invarianten nam ik er twee: een die aangeeft dat delers van de uitgangsgetallen n_0 en m_0 ook delers zijn van de variabelen n en m en een die aangeeft dat n_0 en m_0 steeds lineaire combinaties zijn van n en m. Het simpele feit dat aan het eind van de algoritme de rest bij de laatste deling van m door n nul oplevert geeft het bewijs dat de gevonden n ook een deler is van n_0 en m_0. Helaas is na verhitte discussies dit, in mijn ogen fraaie, bewijs vervangen door het bekende bewijs dat gebaseerd is op de stelling: ggd(m,n)= ggd(n, m mod n). Dat mijn bewijs gebaseerd was op de primaire eigenschappen van de ggd woog in de ogen van anderen helaas niet op tegen de veel eenvoudiger loop invariant.
Belangrijker dan dit detail was echter dat ik aan drie gevallen, waarvan de complexiteit niet al te simpel was, namelijk Maxsort, Schuifsort en Quicksort, kon laten zien dat het werken met correctheidsformules in de praktijk te doen was. Het is in dit verband relevant te vermelden dat Schuifsort een programma is dat uit een stuk of tien korte regeltjes bestaat en dat het "bewijs van correctheid" vier (ruim) gedrukte pagina's beslaat. We laten een gedeelte van die tekst als illustratie zien:

"

Het schuiven hebben we in fig. 5.3 in beeld gebracht:

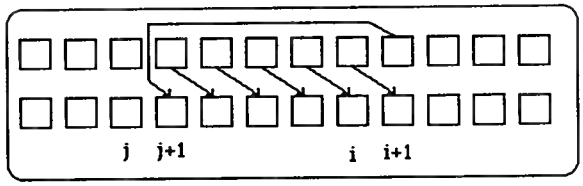

Fig. 5.3. Het schuiven in *schuifsort*.

Na afloop van het schuiven moet het volgende waar zijn (dat echter nog wel bewezen moet worden):

$$S\colon \exists j\colon\ 0\le j<\mathbf{i}$$
$$\wedge\ (\forall k{:}\, j+1\le k\le \mathbf{i} \Rightarrow A_1[k]=\mathbf{A}[k+1])$$
$$\wedge\ (\forall k{:}\, 1\le k\le j \vee \mathbf{i}+2\le k\le \mathbf{n} \Rightarrow A_1[k]=\mathbf{A}[k])$$
$$\wedge\ (\mathbf{A}[j+1]=\mathbf{A}\mathbf{i} \wedge \mathbf{A}[j+2]>\mathbf{A}\mathbf{i}) \wedge (j>0 \Rightarrow \mathbf{A}[j]\le \mathbf{A}\mathbf{i})\ .$$

[.....]

Rest ons nu nog de uitspraak S te bewijzen die na de algoritme voor *schuif* moet gelden. Deze algoritme luidt:

```
schuif:
11:   j:=i-1;
      while (j>0) and (A[j]>Ai) do
      12:   A[j+1]:=A[j];
      13:   j:=j-1
      end;
14:   A[j+1]:=Ai;
15:
```

Te bewijzen is dat als vooraf geldt:

$$(\forall k\colon\ 1\le k\le \mathbf{n} \wedge k\ne \mathbf{i}+1 \Rightarrow A_1[k]=\mathbf{A}[k])$$
$$\wedge\ perm(A_0, A_1)$$
$$\wedge\ A_1[\mathbf{i}] > A_1[\mathbf{i}+1]$$
$$\wedge\ \mathbf{A}\mathbf{i} = A_1[\mathbf{i}+1]$$
$$\wedge\ \mathbf{A}[\mathbf{i}+1] = A_1[\mathbf{i}]$$

dat dan na de schuifoperatie de uitspraak S geldt.
Als loop-invariant kiezen we een iets gewijzigde S:

Q:

$$\forall k\colon \mathbf{j}+1 \le k \le \mathbf{i} \Rightarrow A_1[k] = \mathbf{A}[k+1] \qquad (1)$$

$$\wedge\ \forall k\colon (1 \le k \le \mathbf{j} \vee \mathbf{i}+2 \le k \le \mathbf{n}) \Rightarrow A_1[k] = \mathbf{A}[k] \qquad (2)$$

$$\wedge\ \mathbf{A}[\mathbf{j}+2] > \mathbf{A}\mathbf{i}\ . \qquad (3)$$

Initiëel geldt met $\mathbf{j}=\mathbf{i}-1$:

$$A_1[\mathbf{i}] = \mathbf{A}[\mathbf{i}+1],$$

dus (1) geldt;

$$\forall k\colon (1 \le k \le \mathbf{n} \wedge k \ne \mathbf{i}+1) \Rightarrow A_1[k] = \mathbf{A}[k],$$

dus ook (2) geldt;

$$\mathbf{A}[\mathbf{i}+1] = A_1[\mathbf{i}] > A_1[\mathbf{i}+1] = \mathbf{A}\mathbf{i},$$

dus (3) geldt ook.

Als we de invariantie van Q bewezen heben, dan volgt uit het niet voldoen aan de conditie in het while-statement en uit $\mathbf{A}[\mathbf{j}+1]:=\mathbf{A}\mathbf{i}$ de gevraagde bewering S, immers: $\mathbf{A}[\mathbf{j}+1]=\mathbf{A}\mathbf{i}$ volgt uit het zojuist genoemde assignment statement, $\mathbf{j}>0 \Rightarrow \mathbf{A}[\mathbf{j}] \le \mathbf{A}\mathbf{i}$ volgt uit het niet voldoen aan de conditie en het bestaan van $\mathbf{j}$ met $0 \le \mathbf{j} < \mathbf{i}$ is ook verzekerd.

Tenslotte tonen we nu de invariantie van Q aan:
Aangenomen dat Q geldt bij 12. Bij 12 geldt bovendien:

$$\mathbf{j}>0 \wedge \mathbf{A}[\mathbf{j}] > \mathbf{A}\mathbf{i}\ .$$

Voor $k=\mathbf{j}$ geldt op grond van (2): $A_1[\mathbf{j}] = \mathbf{A}[\mathbf{j}]$. Na de assignment $\mathbf{A}[\mathbf{j}+1]:=\mathbf{A}[\mathbf{j}]$ geldt dus in 13: $A_1[\mathbf{j}]=\mathbf{A}[\mathbf{j}+1]$, zodat

$$\forall k\colon (\mathbf{j} \le k \le \mathbf{i} \Rightarrow A_1[k] = \mathbf{A}[k+1] \wedge (1 \le k \le \mathbf{j}-1 \vee \mathbf{i}+2 \le k \le \mathbf{n}) \Rightarrow A_1[k] = \mathbf{A}[k]\ ).$$

Uit $\mathbf{A}[\mathbf{j}] > \mathbf{A}\mathbf{i}$ volgt $\mathbf{A}[\mathbf{j}+1] > \mathbf{A}\mathbf{i}$.
Maar dit betekent, na substitutie van $\mathbf{j}$-1 door $\mathbf{j}$, dat we Q weer terug hebben, waarmee de invariantie van Q is aangetoond.
"

Met opzet is dit stuk van het diktaat gekozen omdat het een aantal zaken goed laat zien:

• de formules zijn door hun veelheid complex;

• de redeneringen worden opgehangen aan plaatsen in de tekst van het programma: als een of andere uitspraak bij 12 geldt dan volgt, na enige redenaties, dat een andere uitspraak bij 13 geldt;

• het gebruik van verschillende fonts is niet geheel consistent: moet je nu j+1 of j+1 schrijven in een logische uitdrukking, waar j een variabele is die in het programma voorkomt; de tekst van een programma wordt met *schuif* aangeduid;

• het bewijs van correctheid loopt mee met de volgorde van de tekst: van links naar rechts. Dit geldt ook voor assignment statements, zoals blijkt uit de laatste aangehaalde zin.

Hiermee kom ik op een moeilijk punt: mijn gebruik van het assignment statement axioma was niet correct maar het werkte prima in de bewijsvoering.

4. Het college Inleiding Informatica in de laatste jaren

In het collegediktaat van 1984 wordt het assignment statement als volgt ingevoerd:

Als men van een uitspraak R wil bewijzen dat deze geldt ná het assignment statement:

v:= expr

dan is het voldoende als aangetoond wordt dat de uitspraak R[expr/v] geldt vóór dat statement. Dit is de regel:

{P[expr/v]} v:= expr {P}

die ook in het bekende boek van Alagi´c en Arbib [A&A] wordt gebruikt.

In dit collegediktaat worden de axioma's op wat meer formelere wijze ingevoerd, waarin de hand van eerder genoemde John-Jules is te zien. Eenvoudiger voorbeelden worden nu behandeld om de begrippen te verduidelijken. Alles van rechts naar links in verband met de werking van het assignment statement. Voor die eenvoudige voorbeelden werkt dit ook prima, zo wordt

$\{x=x_0 \wedge y=y_0\}$

h:=x; x:=y; y:=h

$\{x=y_0 \wedge y=x_0\}$

6 maart 1989

recht toe recht aan bewezen door uit te gaan van de laatste uitspraak. In het geval van een programma dat het maximum berekent van een array wordt de methode ook toegepast, maar het resultaat van de redenering van twee pagina's is toch weer zo complex dat er behoefte is het als volgt samen te vatten:

"

m:=1; i:=2;
{ $m=1 \wedge i=2$ }
while i ≠ n+1 do
{ P: $1 \le m < i \wedge \forall j: 1 \le j < i \Rightarrow A[m] \ge A[j]$ }
if A[m]<A[i] then
{ $i \ge 1 \wedge \forall j: 1 \le j \le i \Rightarrow A[i] \ge A[j]$ }
m:=i
else ε end;
{ $1 \le m \le i \wedge \forall j: 1 \le j \le i \Rightarrow A[m] \ge A[j]$ }
i:=i+1
{ P: $1 \le m < i \wedge \forall j: 1 \le j < i \Rightarrow A[m] \ge A[j]$ }
end
{ $1 \le m \le n \wedge \forall j: 1 \le j \le n \Rightarrow A[m] \ge A[j]$ } .

"

Het probleem met die samenvatting is dat het geen echt bewijs is. Je kunt ook niet meer zien of je van links naar rechts gegaan bent of omgekeerd. Toch vragen we van onze studenten op het tentamen een bewijs à la deze samenvatting te geven want het verhaal dat er bij hoort is te lang. Overigens blijkt het onderwerp correctheidsbewijs voor de student het moeilijkste onderdeel van het tentamen te zijn.

Op mijn verzoek aan de "theoretische groep" de bewijzen van correctheid voor Maxsort, Schuifsort en Quicksort te veranderen zó dat ze wel van rechts naar links lopen is, begrijpelijkerwijs, tot nu toe niet ingegaan. Zelf zie ik de noodzaak om de vier pagina's tekst om te werken tot een veelvoud daarvan ook niet zo erg zitten.
Nu zou het probleem opgelost worden als ik naast het bovengegeven axioma voor de assignment statement de volgende axioma kies, uit het bekende boek van de Bakker [deB]: (p. 38)

"{P} x:= s { $\exists y$[P[y/x] $\wedge$ x = s[y/x]]},

where y is some variable such that y ≠x, and y ∉ ivar(P,s)"

6 maart 1989

Dit axioma werkt namelijk van links naar rechts. Het zal echter een ieder duidelijk zijn dat je met zo'n axioma niet moet komen bij eerste jaars informaticastudenten.
Hier blijft dus sprake van een dilemma.

5. Evaluatie

Zoals uit het voorgaande blijkt is de laatste jaren in de vakgroep met vereende inspanning gewerkt aan een goede introductie van het onderwerp programmacorrectheid. Helaas zijn we er nog niet in geslaagd de studenten een zodanige vorm van werken met correctheidsformules te presenteren die ook geschikt is voor grotere programma's. Zelfs de korte sorteerprogramma's vergen erg veel formules, en eerlijk gezegd kon ik het ook niet opbrengen alle details van die bewijzen aan de studenten voor te zetten. Na de behandeling van Maxsort doorgeworsteld te hebben maak ik me er in het college vanaf door te zeggen dat het bewijs van correctheid van Schuifsort op soortgelijke wijze gaat, zij het dat de formules nog wat langer zijn: "zie diktaat".

Een veel ernstiger probleem dook echter op: de gebruikte axioma's voor de assignment statement zijn niet goed. Ze gelden alleen als je eenvoudige variabelen hebt, maar ze gelden niet meer als je met arrayelementen werkt en al mijn "echte" voorbeelden betreffen sorteeralgoritmen werkend op arrays! De Bakker vertelde me van zijn vondst; het was 1980 of daaromtrent. Ik was zo onder de indruk dat ik een week later collega van Oorschot, die toen nog Directeur bij de PTT was, er over vertelde. Vanaf die tijd staat het fenomeen ook in het collegediktaat. Het betreffende stuk tekst luidt als volgt:

"
Overigens is een waarschuwend woord hier op zijn plaats: de gebruikte regels (uitgevonden door Floyd en verbeterd door Hoare) werken alleen goed voor eenvoudige "normale" gevallen. Het was De Bakker die aantoonde dat het axioma voor het assignment statement niet voldoende is om adequaat voor alle mogelijke ingewikkelde assignment statements de betekenis vast te leggen. Beschouw de volgende schijnbaar juiste correctheidsformule:

$$\{\ A[1]=1 \wedge A[2]=1\ \}$$
$$A[A[1]]:=2;$$
$$\{\ A[A[1]]=2\ \}$$

Inderdaad, als we in de uitspraak $A[A[1]]=2$ de uitdrukking $A[A[1]]$ letterlijk substitueren door 2 dan ontstaat $2=2$, hetgeen waar is. Echter als $A[1] = A[2] = 1$ bij het begin van het

assignment statement dan is de waarde van **A[A[1]]** na afloop gelijk aan **1**, hetgeen in tegenspraak is met bovenstaande correctheidsformule!
Omdat we van bewijsregels verwachten dat ze *altijd* juist zijn en niet alleen maar *meestal*, (we wilden immers weten dat ons programma *altijd* werkt) moeten we nog één extra afspraak maken:

> In correctheidsformules gebruiken we arrays alleen in de vorm:
>
> **A[*u*]**
>
> waarin *u* een uitdrukking is waarin gehele getallen en integer variabelen voor mogen komen, maar *geen* arrays.

In zijn boek [deB] geeft De Bakker een ander, meer ingewikkeld, axioma voor het assignment statement. Daarmee kunnen correctheidsformules bewezen worden waarin een array-index een willekeurige expressie mag zijn. We gaan er hier (uiteraard) niet verder op in.
"

Inderdaad heeft de Bakker in dit boek een vijftiental pagina's nodig om de subscripted variabelen te introduceren en de betekenis van assignment in de vorm van van vier definities en drie stellingen. Het is uitgesloten zulke zaken te behandelen voor eerste jaars.

Voor mij, als liefhebber van Hofstadter's "Gödel, Escher, Bach"[Hof], zat er in de statement

A[A[1]]:=2

wel een mooie aanleiding het een en ander over zelfreferentie te zeggen. Immers hier hebben we het geval van een statement die zijn eigen betekenis wijzigt (ik weet niet of je mag zeggen "die zichzelf wijzigt"): was de betekenis eerst:

A[1]:=2

na de uitvoering ervan is de betekenis gewijzigd in:

A[2]:=2

Het gaf me de mogelijkheid om het een en ander te zeggen over programma's die zichzelf als input hebben (denk aan compilers, maar ook aan Turing's halting probleem en aan Gödel's onvolledigheidsstelling), over recursieve procedures, over vorm en inhoud en de verschillende niveaus van machinearchitectuur, over Bach die zijn eigen naam als thema gebruikt, over Escher's zelfportretten, kortom een bonte lijst van interessante zaken.
Instructies die, zoals in het "Koude Start Programma" van de eerste versie van het collegediktaat, eerst "gebakken" moeten worden, zijn verdwenen uit het diktaat; daarvoor in de plaats zijn statements gekomen die zichzelf wijzigen, wat natuurlijk veel interessanter is.

Nog enkele opmerkingen over het collegediktaat Inleiding Informatica: Het hoofdstuk over

machinearchitectuur is nu zo gewijzigd dat het begint met de elementen van eerste orde logica en in no time overgaat naar Boole'se Algebra, sequentiële circuits, poorten en de electronica van een chip, inclusief optellers en geheugen; dit alles als prelude op het boek van Tanenbaum [Tan]. Overigens, was er een goed boek van Pohl en Shaw [P&S] dat we een groot aantal jaren gebruikten met name voor dit onderwerp. Het hoofdstuk "Computers in de Samenleving" beslaat nu drie pagina's, dat is niet overdreven veel. Als compensatie merk ik op dat de laatste twee uren van het college normaliter aan dit onderwerp besteed worden met allerlei gadgets, dia's, audio (uit de laatste fuga van de Kunst der Fugen waarin Bach naar zichzelf refereert) en de laatste tijd ook video (over robots enzo).

Tenslotte, we zien in de laatste jaren een veel grotere interesse van informatici voor specificaties. De reden is dat de systemen zo complex worden dat er steeds hoger niveau geprogrammeerd moet worden met machtige hulpmiddelen als grafische tools (denk aan op plaatjes gebaseerde Software Engineering tools) en logische programmeertalen als Prolog (zie [C&M]).
De bedoeling van Prolog was een taal te zijn waar de specificatie samengaat met het programma: het programma is de specificatie. Helemaal gelukt is dat niet: je moet nog teveel imperatief programmeren, nog te veel het hoe aangeven. Toch is het een stap in de goede richting gezet.
Een goed begrip van deze specificatietalen en het belang ervan krijg je door een goed inzicht te hebben in de achtergronden van programmacorrectheid. Daarom denk ik dat het onderwerp nog lang in het diktaat Inleiding Informatica opgenomen zal blijven, ondanks de gesignaleerde problemen.

Literatuur

[A&A] S. Alagi´c, M.A. Arbib: The design of Well Structured and Correct Programs, Springer, 1978.

[deB] J.W. de Bakker: Mathematical Theory of Program Correctness, Prentice Hall 1980.

[C&M] W.F. Clocksin, C.S. Mellish: Programming in Prolog, Springer, 1881.

[Hof] D.R. Hofstadter: Gödel, Escher, Bach: an Eternal Golden Braid, Basic Books, New York 1979.

[Knu] D.E. Knuth: The Art of Computer Programming Vol.3: Sorting and Searching,

Addison-Wesley, 1973.

[P&S] Ira Pohl, Alan Shaw: The Nature of Computation: an Introduction to Computer Science, Computer Science Press, 1981, ISBN 0-914894-12-9.

[vdR] R.P. van de Riet: ABC ALGOL, a portable language for formula manipulation systems, part 1 and part 2, Mathematical Centre tracts 46 en 47, Mathematisch Centrum, Amsterdam, 1973.

[Tan] A.S. Tanenbaum: Structured Computer Organization, Second Edition, Prentice Hall 1984.