Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

N.W.P. van Diepen

Integer-square-root
Een voorbeeld van en inleiding tot programmatransformaties

Afdeling Informatica Notitie CS-N8501 Februari

Bibliotheek
Centrum voor Wiskunde en Informatica
Amsterdam

Integer - Square - Root

An example of an introduction to program transformations

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Copyright © Stichting Mathematisch Centrum, Amsterdam

INTEGER-SQUARE-ROOT
Een voorbeeld van en inleiding tot programmatransformaties

N.W.P. van Diepen
Centrum voor Wiskunde en Informatica
Amsterdam

SAMENVATTING

Dit artikel is bedoeld als intuïtieve inleiding tot het gebied van transformationeel programmeren. Er wordt een korte inleidende beschrijving van partiële correctheidsbewijzen in Hoare Logica gegeven. Hierna volgen twee uitgewerkte voorbeelden van correctheidbehoudende programmatransformaties. Beide voorbeelden vangen aan met een evident correcte oplossing van het Integer-Square-Root-probleem (het berekenen van $\lfloor \sqrt{n} \rfloor$ van een gegeven natuurlijk getal n). Daarna worden deze programma's geoptimaliseerd met behulp van programmatransformaties.

ABSTRACT

This paper is an intuitive introduction to the field of transformational programming. A short introductory description of partial correctness proofs in Hoare Logic is given. Next two exhaustive examples of correctness preserving program transformations are given. Both examples start with an evidently correct solution to the problem of computing the Integer-Square-Root ($\lfloor \sqrt{n} \rfloor$) of a given positive integer n. These programs are then optimized using program transformations. (This paper is in Dutch.)

69 K12

Classificatie: CR I.2.2 1980 AMS 68B10
Keywords: program transformation, program verification

1. INLEIDING

Bij het schrijven van een programma heeft een programmeur meestal twee doelstellingen. Allereerst moet het programma werken, correct zijn. En daarnaast moet het liefst snel werken, efficiënt zijn. Vaak is het geen eenvoudige zaak van een efficiënt programma te bewijzen dat het correct is, want zo'n programma is meestal met trucs of ingewikkelde constructies opgebouwd. Dit laatste maakt een bewijs natuurlijk extra gewenst.

Een idee om dit probleem te omzeilen is om allereerst een gemakkelijk correct te bewijzen versie van een programma te schrijven. Daarna wordt dit programma getransformeerd naar een efficiëntere versie met stappen die de correctheid behouden. Dit laatste programma is dan natuurlijk ook correct.

Om te kunnen praten over de correctheid van een programma is het prettig van een standaardformalisme gebruik te kunnen maken. In §2 zal daarom het formalisme van de Hoare Logica worden geïntroduceerd.

In de volgende paragrafen worden een aantal programmatransformaties in dit formalisme beschreven en gedemonstreerd aan de hand van een eenvoudig probleem. Een drietal geoptimaliseerde programma's worden afgeleid uit twee programma's die evident correct zijn.

Als afsluiting wordt een korte beschouwing over de mogelijkheden van de transformationele programmeermethode gegeven.

Notitie CS-N8501
Centrum voor Wiskunde en Informatica
Postbus 4079, 1009 AB Amsterdam

2. HOARE-LOGICA

Van een programma wordt verwacht, dat het iets bepaalds doet: een rij sorteren, een functie uitrekenen, enzovoorts. Dit stelt eisen aan input en output. De input van een programma dat wortel trekt, moet een niet-negatief getal zijn. En kwadrateren van de (weer niet-negatieve) output moet (althans bij benadering) de input teruggeven.

Als de inputvariabele x is en de outputvariabele r, dan zijn deze eisen als volgt te formuleren:

input: $x \geqslant 0$

output: $r \geqslant 0 \ \wedge \ r^2 = x$.

Voldoet het programma $r := sqrt(x)$ aan deze eisen, dan wordt dit in navolging van Tony Hoare als volgt genoteerd:

$$\{x \geqslant 0\} \ \ r := sqrt(x) \ \ \{r \geqslant 0 \ \wedge \ r^2 = x\}.$$

In woorden: als $x \geqslant 0$ dan geldt na uitvoering van $r := sqrt(x)$ dat $r \geqslant 0$ en $r^2 = x$.

Tussen accolades staan uitspraken over de variabelen in het programma, de zogenaamde *asserties*. De eerste assertie wordt ook wel de *preconditie* of *pre-assertie* genoemd, en de laatste assertie van een programma heet wel de *postconditie* of *post-assertie*. Deze asserties maken het mogelijk uitspraken te doen over de samenhang tussen de variabelen in een programma (bijvoorbeeld $r^2 = x$) en daar logisch mee te redeneren. In 1969 is daartoe een systeem van logische afleidingsregels ontwikkeld, dat naar zijn ontwerper de Hoare-Logica heet.

Allereerst zijn er enige *axioma's:*

$$\{P\} \ \textbf{skip} \ \{P\} \qquad \text{(Leeg)}$$

$$\{P[t/x]\} \ \ x := t \ \ \{P\} \qquad \text{(Assignatie)}$$

waarin P een assertie is en $P[t/x]$ de assertie P is, met daarin de variabele x vervangen door de term t. De assignatieregel zegt, dat na uitvoering van $x := t$ nu in de assertie P x mag worden geschreven waar voordien t stond.

Deze beide axioma's zijn triviaal in betekenis: doe niets ($\equiv$ voer het statement **skip** uit) en er verandert niets aan de assertie. En na het toekennen van een bepaalde waarde t aan x mag nu x geschreven worden waar voordien de term t stond.

Verder zijn er een aantal *bewijsregels.* Een paar volgen hieronder. De gebruikte notatie is van de vorm:

$$\frac{\textit{gestelde}}{\textit{conclusie}}$$

dat wil zeggen: de *conclusie* is bewijsbaar indien het *gestelde* bewijsbaar is.

In de volgende regels worden de logische symbolen $\wedge$ (en), $\rightarrow$ (implicatie; als het eerste **waar** is, dan ook het tweede) en $\neg$ (niet) gebruikt. P, Q en R zijn asserties, S, S_1 en S_2 zijn statements en B is een expressie die **waar** of **onwaar** oplevert.

$$\frac{\{P\} \ S_1 \ \{Q\}, \quad \{Q\} \ S_2 \ \{R\}}{\{P\} \ S_1; S_2 \ \{R\}} \qquad \text{(Schakeling)}$$

$$\frac{\{P \wedge B\} \ S_1 \ \{Q\}, \quad \{P \wedge \neg B\} \ S_2 \ \{Q\}}{\{P\} \ \textbf{if} \ B \ \textbf{then} \ S_1 \ \textbf{else} \ S_2 \ \textbf{fi} \ \{Q\}} \qquad \text{(If)}$$

$$\frac{\{P \wedge B\}\ S\ \{P\},\quad P \wedge \neg B \rightarrow Q}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{Q\}} \qquad \text{(Loop)}$$

$$\frac{P' \rightarrow P,\quad \{P\}\ S\ \{Q\},\quad Q \rightarrow Q'}{\{P'\}\ S\ \{Q'\}} \qquad \text{(Implicatie)}$$

De schakelregel zegt, dat twee statements achter elkaar mogen worden gezet als de postconditie van de eerste gelijk is aan de preconditie van de tweede. Meestal worden alleen de voor een redenering relevante asserties en de pre- en postcondities opgeschreven zodat in de conclusie van deze regel bijvoorbeeld ook $\{P\}\ S_1;\ \{Q\}\ S_2\ \{R\}$ had mogen staan. Met de implicatieregel moet soms de postconditie van het eerste statement worden verzwakt. De if-regel spreekt voor zich. Indien beneden het **then**-gedeelte wordt gevolgd was $P \wedge B$ waar en na uitvoering van S_1 dus Q. Voor de **else**-tak geldt eenzelfde redenering. De assertie P in de loop-regel wordt de *loop-invariant* genoemd. Zolang de loop wordt uitgevoerd blijft P immers waar.

Hoare Logica is een methode om de zogenaamde *partiële correctheid* van een programma te bewijzen: als het programma termineert, dan is de uitkomst correct. Een programma dat partiël correct is en termineert heet *totaal correct.* Dat partiële correctheid alleen niet voldoende is, is eenvoudig in te zien met het volgende programma:

$\{n \in \mathbf{Z}\}$
while $n \neq 0$ **do** $n := n+1$ **od**
$\{n \in \mathbf{Z} \wedge \neg n \neq 0\} \quad \Rightarrow \quad \{n = 0\}$

Is $n > 0$ bij de aanvang van dit programma, dan blijft de loop-invariant geldig, en ook de postconditie is correct, maar helaas wordt het eind van het programma nooit bereikt. Er is meer nodig om terminatie te bewijzen, maar de benodigde techniek voert te ver voor deze inleiding. In het voorbeeldprogramma is het mogelijk een terminatiebewijs te leveren als de preconditie wordt verscherpt:

$\{n \in \mathbf{Z} \wedge n \leqslant 0\}$
while $n \neq 0$ **do** $n := n+1$ **od**
$\{n \in \mathbf{Z} \wedge n \leqslant 0 \wedge \neg n \neq 0\} \quad \Rightarrow \quad \{n = 0\}$

Nu kan worden opgemerkt dat $-n \in \mathbf{N}$ en dat de waarde van $-n$ na elke uitvoering van de loop kleiner wordt. Omdat $-n$ niet kleiner dan 0 kan worden moet het programma termineren. Dus dit programma is totaal correct met deze sterkere preconditie.

3. INTEGER-SQUARE-ROOT

Om de techniek van transformationeel programmeren te illustreren gaan we in op het eenvoudige probleem van de Integer-Square-Root: gegeven een $n \in \mathbf{N}$; gevraagd een $x \in \mathbf{N}$ zodat $x^2 \leqslant n < (x+1)^2$, dus $x = \lfloor \sqrt{n} \rfloor$. In termen van programmeren komt dit probleem neer op het vinden van een programma S zodat:

$$\{n \geqslant 0\}\ S\ \{x^2 \leqslant n < (x+1)^2\}$$

geldt. Een programma schrijven dat hieraan voldoet is uiteraard niet moeilijk. In het vervolg worden twee eenvoudige programma's gepresenteerd en vervolgens in efficiëntere versies omgezet.

4. LINEAIR ZOEKEN

Een prettige eigenschap van het Integer-Square-Root-probleem is, dat het mogelijk is één voor één alle potentiële oplossingen te proberen, en te stoppen bij de echte oplossing. Dit leidt altijd tot succes. Dit gebeurt in het programma:

L1: $x:=0;$
while not $x^2 \leqslant n < (x+1)^2$
do $x:=x+1$ **od**

Een correctheidsbewijs in de stijl van Hoare (§2) vinden is natuurlijk niet moeilijk. Voeg onderstaande asserties toe:

L1: $\{n \geqslant 0\}$
$x:=0;\ \{x^2 \leqslant n\}$
while not $x^2 \leqslant n < (x+1)^2$
do $\{(x+1)^2 \leqslant n\}\ x:=x+1\ \{x^2 \leqslant n\}$ **od**
$\{x^2 \leqslant n < (x+1)^2\}$

De correctheid van de assertie binnen de loop volgt nu uit:

$$x^2 \leqslant n \ \wedge \neg[x^2 \leqslant n \leqslant (x+1)^2] \quad \rightarrow \quad (x+1)^2 \leqslant n.$$

Terminatie is ook niet moeilijk te bewijzen, want we weten dat de gezochte oplossing x voldoet aan $x \leqslant \sqrt{n} \leqslant n$. De oplossing ligt dus in de verzameling $\{0, 1, \ldots, n\}$. Deze verzameling wordt door x te beginnen bij 0 doorlopen, totdat de oplossing gevonden is. Dit kost hoogstens $n+1$ maal het doorlopen van de loop.

Nu ligt het voor de hand dat L1 inefficiënt is, zelfs voor een lineair algoritme. In de test wordt er bij elke doorgang van de loop tweemaal gekwadrateerd. Maar op het moment dat we testen **not** $x^2 \leqslant n < (x+1)^2$ weten we uit de loop-invariant dat $x^2 \leqslant n$. Deze test is dus equivalent met de test **not** $n < (x+1)^2$. Maar dan kunnen we deze test in het programma L1 zetten! Een eerste programmatransformatie geeft ons het programma:

L2: $\{n \geqslant 0\}$
$x:=0;\ \{x^2 \leqslant n\}$
while not $n < (x+1)^2$
do $\{(x+1)^2 \leqslant n\}\ x:=x+1\ \{x^2 \leqslant n\}$ **od**
$\{x^2 \leqslant n < (x+1)^2\}$

Schematisch hebben we het volgende gedaan:

$$\frac{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \{P\}\ \textbf{od}\ \{P \wedge \neg B\},\quad P \rightarrow [B \leftrightarrow B']}{\{P\}\ \textbf{while}\ B'\ \textbf{do}\ S\ \{P\}\ \textbf{od}\ \{P \wedge \neg B'\}} \tag{4.1}$$

Dit is een programmatransformatieregel. Een hierop lijkende regel is:

$$\frac{\{P\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{Q\},\quad P \rightarrow [B \leftrightarrow B']}{\{P\}\ \textbf{if}\ B'\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{Q\}} \tag{4.2}$$

De geldigheid van deze regels moet natuurlijk bewezen worden. Dit kan niet rechtstreeks met de bewijsregels van Hoare, omdat het niet mogelijk is in die regels te beschrijven, dat er een afleiding van de gestelde voorwaarde bestaat. In deze geldigheidsbewijzen wordt het bestaan van een afleiding van de voorwaarde gebruikt om het bestaan van een afleiding van de conclusie aan te tonen.

Eerst een geldigheidsbewijs van (4.1): neem aan dat pre-assertie P geldt. Stel nu dat B onwaar is. Omdat $P \rightarrow [B' \rightarrow B]$ is dan ook B' onwaar. Beide loops termineren dus, beneden met post-

assertie $P \wedge \neg B'$. Stel anderszijds dat B waar is, dus ook B' is waar. Zowel boven als beneden wordt S uitgevoerd. Boven geldt $\{P \wedge B\}\ S\ \{P\}$. Beneden hebben we als pre-assertie (voor S) $P \wedge B'$. Omdat $P \wedge B' \rightarrow P \wedge B$ mogen we concluderen dat beneden $\{P \wedge B'\}\ S\ \{P\}$ geldt. Dus (4.1) is geldig.

Ook bij (4.2) zijn er twee gevallen te onderscheiden. Als B waar is, dan hebben we $P \wedge B$ als pre-assertie voor de bovenste **if**-constructie, dus S_1 wordt uitgevoerd en Q is de post-assertie. Omdat $P \rightarrow [B \rightarrow B']$ geldt $P \wedge B \rightarrow P \wedge B'$. In de onderste **if**-constructie wordt dus ook S_1 uitgevoerd en Q resulteert dan als post-assertie. Het tweede geval, B is onwaar, gaat analoog, dus (4.2) is geldig.

Voor de volgende stap hebben we het begrip *fantoomvariabele* nodig. Binnen een programma P definiëren we een *verzameling fantoomvariabelen* AV (auxiliary variables) door:

> voor alle $x \in$AV geldt: x komt alleen voor in statements van de vorm $x := t$ [er mag aan x een waarde toegekend worden], of van de vorm $y := t(x)$ met $y \in$AV [de waarde van x mag alleen op de waarden van fantoomvariabelen van invloed zijn].

Merk op dat we de verzameling fantoomvariabelen op maat kunnen kiezen. Het toevoegen van een variabele kan het verplicht toevoegen van andere variabelen tot gevolg hebben, maar verder zijn we tamelijk vrij. Fantoomvariabelen mogen alleen geen invloed hebben op de structuur van het programma. Hierdoor heeft weglaten van alle fantoomvariabelen of het toevoegen van nieuwe fantoomvariabelen geen invloed op het gedrag van de andere variabelen in een programma.

Nu kunnen we het volgende lemma formuleren:

> **Lemma** (4.3): Laten P en Q predicaten zijn en S en S' programma's, met AV een verzameling fantoomvariabelen van S zodat S' het programma S is, met daarin de statements van de vorm $y := t$ met $y \in$AV vervangen door **skip** (het dummy-statement) en $[\mathrm{FV}(P) \cup \mathrm{FV}(Q)] \cap \mathrm{AV} = \varnothing$ (hierin zijn FV(P) en FV(Q) de verzamelingen vrije (van buiten komende) variabelen van P respectievelijk Q, dus dit wil zeggen dat de asserties P en Q geen fantoomvariabelen bevatten), dan:
>
> $$\{P\}\ S\ \{Q\} \quad \Leftrightarrow \quad \{P\}\ S'\ \{Q\}$$
>
> **Bewijs:** weggelaten.

Dit lemma maakt het mogelijk fantoomvariabelen naar wens in te voeren en weer weg te laten.

In programma L2 wordt nog steeds gekwadrateerd. Nu is het vaak sneller om op te tellen dan om te vermenigvuldigen. Een goed idee is dus te proberen de tweede kwadratering door een optelling te vervangen. Dat kan. Eerst voeren we met lemma (4.3) twee fantoomvariabelen a en b met een aardige eigenschap in:

```
L3:   {n ≥ 0}
      x := 0; a := 1; b := 3; {x² ≤ n ∧ a = (x+1)² ∧ b = (x+2)² − (x+1)²}
      while not n < (x+1)²
      do    {(x+1)² ≤ n ∧ a = (x+1)² ∧ b = (x+2)² − (x+1)²}
            x := x+1; a := a+b; b := b+2
            {x² ≤ n ∧ a = (x+1)² ∧ b = (x+2)² − (x+1)²}
      od
      {x² ≤ n < (x+1)²}
```

En nu volgt weer uit de loop-invariant dat **not** $n < (x+1)^2 \leftrightarrow$ **not** $n < a$ want $a = (x+1)^2$. Toepassing van transformatieregel (4.1) geeft nu het programma:

L4: $\{n \geqslant 0\}$
$x:=0;\ a:=1;\ b:=3;$
while not $n < a$
do $x:=x+1;\ a:=a+b;\ b:=b+2$ **od**
$\{x^2 \leqslant n < (x+1)^2\}$

Dit programma doet per stap slechts drie optellingen in plaats van de twee optellingen en één vermenigvuldiging van L2. L4 is echter niet meer "triviaal correct". Voor een correctheidsbewijs in de stijl van Hoare moeten de ingewikkelde loop-asserties van programma L3 worden afgeleid. Ze liggen echter tamelijk voor de hand als we in programma L2 de term $(x+1)^2$ door fantoomvariabele a proberen te vervangen. Dan moet a immers elke ophoging van x worden verhoogd met de waarde $(x+2)^2 - (x+1)^2$. Noem dit de fantoomvariabele b en het blijkt dat:

$$b \;=\; (x+2)^2 - (x+1)^2 \;=\; 2x + 3$$

De beginwaarde van b is dus 3 en bij elke verhoging van x met 1 moet b met 2 worden verhoogd om aan de gewenste eis te voldoen. De asserties die we bij L3 en L4 nodig hebben komen bijna vanzelf bij deze afleiding te voorschijn!

De techniek van het invoeren van a en b zullen we meer tegenkomen. Eerst worden variabelen met een bepaalde eigenschap verzonnen. Die variabelen worden dan als fantoomvariabelen ingevoerd. Waar nodig kunnen nu de asserties worden uitgebreid. Daarna nemen ze een deel van de taak van de oude variabelen over, dus dan zijn het geen fantoomvariabelen meer. Dit kan zelfs zo ver gaan, dat de oude variabelen fantoomvariabelen kunnen worden.

5. BINAIR ZOEKEN

Hoewel programma L4 behoorlijk snel is, is het nog steeds een lineair programma, dat wil zeggen, L4 doet evenveel stappen als de oplossing $\lfloor \sqrt{n} \rfloor$ groot is, eventueel op een constante en een vermenigvuldigingsfactor na. Anders gezegd: L4 kost orde $\lfloor \sqrt{n} \rfloor$ tijd. Nu zijn de natuurlijke getallen geordend, en het is mogelijk om aan een getal te zien of de oplossing groter dan wel kleiner moet zijn. Het is dus mogelijk binair te zoeken.

Hiervoor hebben we wel een bovengrens nodig, want voor binair zoeken moet er een interval met de oplossing zijn. Die bovengrens moet dan aan de eis dat z'n kwadraat groter dan n is voldoen. In het algemeen kunnen we hieraan voldoen door middel van de *onbepaalde assignatie*:

$$y \;:=\; y' \mid P(y')$$

waarin y een variabele is, y' een verse constante en P een predikaat dat y niet bevat.

Hiermee kunnen we het archetype van de binaire zoek-procedure opschrijven ($\div$ is de integerdeling zonder rest):

B: $\{n \geqslant 0\}$
$y := y' \mid (y')^2 > n;\ \{n \geqslant 0 \ \wedge\ y^2 > n\}$
$x := 0;\ \{x^2 \leqslant n < y^2\} \ \equiv\ \{P\}$
while $x+1 \neq y$
do $a := (x+y) \div 2;\ \{P \ \wedge\ a = (x+y) \div 2\}$
if $a^2 \leqslant n$
then $x := a\ \{P\}$
else $y := a\ \{P\}$
fi
od
$\{P \ \wedge\ x+1 = y\} \ \Rightarrow\ \{x^2 \leqslant n < (x+1)^2\}$

Natuurlijk is B correct, ook terminatie is niet moeilijk te bewijzen. Bovendien kost B orde $\log \lfloor \sqrt{n} \rfloor$ tijd, en dus is B sneller (voor n groot genoeg) dan programma L4. Helaas is B nog geen echt programma. Dat is te verhelpen door de observatie dat $n \geqslant \sqrt{n}$ voor $n \in \mathbf{N}$. Nu kunnen we de onbepaalde assignatie in programma B nader invullen:

BA: $\{n \geqslant 0\}$
$y := n+1;\ \{n \geqslant 0 \ \wedge\ y^2 > n\}$
$x := 0;\ \{P\}$
while $x+1 \neq y$
do $a := (x+y) \div 2;\ \{P \ \wedge\ a = (x+y) \div 2\}$
if $a^2 \leqslant n$
then $x := a\ \{P\}$
else $y := a\ \{P\}$
fi
od
$\{x^2 \leqslant n < (x+1)^2\}$

De asserties uit programma B kunnen eenvoudig worden overgeschreven, dus ook BA is volledig correct.

Pogingen BA te optimaliseren stuiten op problemen. De beste kandidaat voor optimalisatie als in de vorige paragraaf is de variabele a, want deze wordt bij elke uitvoering van de loop gekwadrateerd. Maar in de bepaling van de waarde van a in $a := (x+y) \div 2$ zit een gevalsonderscheiding verstopt. Is $x+y$ even, dan wordt er geen rest weggegooid, is $x+y$ oneven, dan wel. Dit bemoeilijkt het vinden van een eenvoudige regelmaat in de waarden van a.

Dan moeten we ervoor zorgen dat bij het delen door twee geen rest ontstaat. Met andere woorden, we moeten het verschil tussen x en y een macht van 2 houden. Dat kan. Noem dit verschil z, dus $y = x+z$, dan hoeven we ook y niet meer bij te houden. Als de nieuwe integer-deling zonder rest wordt voorgesteld door /2 om hem te onderscheiden van de oude, denk hierbij aan bitshifts, dan is het niet moeilijk in te zien dat programma B kan worden omgezet in:

BB1: $\{n \geqslant 0\}$
$z := 1;$
while $z^2 \leqslant n$ **do** $z := 2z$ **od**; $\{n \geqslant 0 \wedge z^2 > n\}$
$x := 0$; $\{x^2 \leqslant n < (x+z)^2\} \equiv \{P_1\}$
while $z > 1$
do $z := z/2;$
if $(x+z)^2 \leqslant n$
then $x := x+z$
fi $\{P_1\}$
od
$\{P_1 \wedge z = 1\} \Rightarrow \{x^2 \leqslant n < (x+1)^2\}$

Eigenlijk gebeurt deze transformatie in een paar stappen. Eerst wordt fantoomvariabele z ingevoerd. Dan wordt programmatransformatie (4.1) toegepast. Hierna is de variabele y een fantoomvariabele geworden, en dus kunnen we y weglaten. Bij de tweede stap worden de asserties met de variabele y vervangen door equivalente asserties me de variabele z.

In beide loops wordt nu de waarde van z^2 uitgerekend. Een goed idee is het effect van het bijhouden van deze waarde te bekijken; dit doen we door fantoomvariabele q in te voeren en daarna regel (4.1) tweemaal op het nieuwe programma toe te passen. Dan krijgen we het programma:

BB2: $\{n \geqslant 0\}$
$z := 1;\ q := 1;$
while $q \leqslant n$ **do** $z := 2z;\ q := 4q$ **od**; $\{n \geqslant 0 \wedge z^2 > n \wedge q = z^2\}$
$x := 0$; $\{x^2 \leqslant n < (x+z)^2 \wedge q = z^2\} \equiv \{P_2\}$
while $z > 1$
do $z := z/2;\ q := q/4;$
if $x^2 + 2xz + q \leqslant n$
then $x := x+z$
fi $\{P_2\}$
od
$\{P_2 \wedge z = 1\} \Rightarrow \{x^2 \leqslant n < (x+1)^2\}$

Volgens regel (4.1) kunnen we nu de test $z > 1$ door $q > 1$ vervangen. Om in de test $x^2 + 2xz + q \leqslant n$ de vermenigvuldigingen van variabelen kwijt te raken worden nu de fantoomvariabelen $y = n - x^2$ en $p = xz$ ingevoerd en met programmatransformatie (4.2) wordt deze test omgezet in $2p + q \leqslant y$. Dan krijgen we het programma:

BB3: $\{n \geqslant 0\}$
$z := 1;\ q := 1;$
while $q \leqslant n$ **do** $z := 2z;\ q := 4q$ **od**; $\{n \geqslant 0 \wedge z^2 > n \wedge q = z^2\}$
$x := 0;\ y := n;\ p := 0;$
$\{x^2 \leqslant n < (x+z)^2 \wedge q = z^2 \wedge y = n - x^2 \wedge p = xz\} \equiv \{P_3\}$
while $q > 1$
do $z := z/2;\ q := q/4;\ p := p/2;$
if $2p + q \leqslant y$
then $x := x+z;\ y := y - 2p - q;\ p := p+q$
fi $\{P_3\}$
od
$\{P_3 \wedge q = 1\} \Rightarrow \{x^2 \leqslant n < (x+1)^2\}$

De assertie $P_3 \wedge q = 1$ heeft naast de conclusie dat $x = \lfloor \sqrt{n} \rfloor$ ook de conclusie $p = \lfloor \sqrt{n} \rfloor$ in zich verborgen. Immers $p = xz$ en $z^2 = q = 1$, dus $p = x$, want $z \in \mathbf{N}$. De oplossing wordt dus tweemaal berekend! Dan zou x wel eens overbodig kunnen zijn, want samen met z gedraagt hij zich als fantoomvariabele. Alleen moeten we dan x en z uit de asserties verwijderen.

De eerste assertie waarin een van beiden voorkomt is:

$$n \geqslant 0 \ \wedge \ z^2 > n \ \wedge \ q = z^2$$

Die kunnen we herleiden tot:

$$n \geqslant 0 \ \wedge \ q > n$$

Iets lastiger is assertie P_3:

$$x^2 \leqslant n < (x+z)^2 \ \wedge \ q = z^2 \ \wedge \ y = n - x^2 \ \wedge \ p = xz \ \Leftrightarrow$$
$$x^2z^2 \leqslant nz^2 < (xz+z^2)^2 \ \wedge \ q = z^2 \ \wedge \ yz^2 = nz^2 - x^2z^2 \ \wedge \ p = xz \ \Leftrightarrow$$
$$p^2 \leqslant nq < (p+q)^2 \ \wedge \ q = z^2 \ \wedge \ y = n - p^2/q \ \wedge \ p = xz$$

Merk op dat de waarden van x en z hierin geen wezenlijke rol spelen. Ze weglaten levert:

$$p^2 \leqslant nq < (p+q)^2 \ \wedge \ y = n - p^2/q$$

Als hierin $q = 1$ wordt ingevuld levert p het gewenste resultaat op. We kunnen programma BB3 met asserties dus herformuleren:

```
BB3:  {n≥0}
      z:=1; q:=1;
      while q ≤ n do z:=2z; q:=4q od; {n≥0 ∧ q>n}
      x:=0; y:=n; p:=0;
      {p² ≤ nq < (p+q)² ∧ y = n − p²/q} ≡ {P4}
      while q > 1
      do    z:=z/2; q:=q/4; p:=p/2;
            if 2p+q ≤ y
            then  x:=x+z; y:=y−2p−q; p:=p+q
            fi {P4}
      od
      {P4 ∧ q=1} ⇒ {p²≤n<(p+1)²}
```

Maar nu kunnen we AV $= \{x, z\}$ nemen. Toepassing van lemma (4.3) resulteert in het programma:

```
BB4:  {n≥0}
      q:=1;
      while q ≤ n do q:=4q od; {n≥0 ∧ q>n}
      y:=n; p:=0; {P4}
      while q > 1
      do    q:=q/4; p:=p/2;
            if 2p+q ≤ y
            then  y:=y−2p−q; p:=p+q
            fi {P4}
      od
      {P4 ∧ q=1} ⇒ {p²≤n<(p+1)²}
```

Een laatste verbetering is nu nog aan te brengen door de observatie dat binnen de laatste loop de variabele p eerst door 2 wordt gedeeld en daarna één- of tweemaal met 2 wordt vermenigvuldigd. Het is sneller dat slechts waar nodig te doen. Hiervoor hebben we nog twee transformatieregels nodig.

$$\frac{\{P\}\ v:=t\ \{Q\},\quad P \rightarrow t = t'}{\{P\}\ v:=t'\ \{Q\}} \tag{5.1}$$

$$\frac{\{P\}\ S;\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{Q\},\quad \{B'\}\ S\ \{B\},\quad \{\neg B'\}\ S\ \{\neg B\}}{\{P\}\ \textbf{if}\ B'\ \textbf{then}\ S; S_1\ \textbf{else}\ S; S_2\ \textbf{fi}\ \{Q\}} \tag{5.2}$$

Regel (5.1) volgt direct uit de assignatieregel van Hoare.

Bij regel (5.2) hebben we twee gevallen. Als B' in de aanname waar is reduceert het programma tot

$$\{P \wedge B'\}\ S\ \{R \wedge B\}\ S_1\ \{Q\}$$

voor een tussenliggende assertie R, zodat $\{R \wedge B\}\ S_1\ \{Q\}$. In de conclusie hebben we dezelfde afleiding. Het geval dat B' in de aanname onwaar is, gaat analoog. Dus beide regels zijn geldig.

Voor de toepassing van deze regels is het voldoende dat we het binnenste gedeelte van de tweede loop beschouwen na de assignatie $q := q/4$. De transformaties gebeuren daar en buiten de loop heeft dat geen effect. Beschouw dus het programma:

S_1: $\{p^2 \leqslant 4nq < (p+4q)^2 \wedge y = n - p^2/4q \wedge 4q > 1\} \equiv \{P'\}$
$p := p/2;$
if $2p+q \leqslant y$
then $y := y-2p-q;\ p := p+q$
fi
$\{p^2 \leqslant nq < (p+q)^2 \wedge y = n - p^2/q\} \equiv \{P''\}$

Nu voeren we de fantoomvariabele a in, die de waarde van p bevriest:

S_2: $\{P'\}$
$a := p;\ \{P' \wedge a = p\}$
$p := p/2;$
if $2p+q \leqslant y$
then $y := y-2p-q;\ p := p+q$
fi
$\{P''\}$

Hierna kunnen de regels (4.2) en (5.1) worden toegepast:

S_3: $\{P'\}$
$a := p;\ \{P' \wedge a = p\}$
$p := p/2;$
if $a+q \leqslant y$
then $y := y-a-q;\ p := p+q$
fi
$\{P''\}$

De test $a+q \leqslant y$ is niet afhankelijk van de waarde van p, dus we kunnen regel (5.2) toepassen:

S_4: $\{P'\}$
$a:=p;\ \{P' \wedge a=p\}$
if $a+q \leqslant y$
then $p:=p/2;\ y:=y-a-q;\ p:=p+q$
else $p:=p/2$
fi
$\{P''\}$

Met twee regels, die een direct gevolg zijn van de assignatieregel van Hoare, kan het **then**-gedeelte nog een beetje opgepoetst worden. Deze regels zijn:

$$\frac{\{P\}\ v:=t;\ v':=t'\ \{Q\},\quad v' \notin \mathrm{FV}(t)}{\{P\}\ v':=t'[t/v];\ v:=t\ \{Q\}} \tag{5.3}$$

$$\frac{\{P\}\ v:=t(v);\ v:=t'(v)\ \{Q\}}{\{P\}\ v:=t'(t(v))\ \{Q\}} \tag{5.4}$$

Hierin betekent $t'[t/v]$ de term t' met daarin de variabele v vervangen door de term t.

Nu kunnen we de assignaties $p := p/2$ en $y := y-a-q$ in S_4 omwisselen wegens regel (5.3) en daarna kunnen we de assignaties $p := p/2$ en $p := p+q$ samenvoegen wegens regel (5.4). Verwijderen van de fantoomvariabele a resulteert in het programmaonderdeel dat we hebben willen. Op de juiste plaats in BB4 invoegen levert:

BB5: $\{n \geqslant 0\}$
$q:=1;$
while $q \leqslant n$ **do** $q:=4q$ **od**; $\{n \geqslant 0 \wedge q>n\}$
$y:=n;\ p:=0;\ \{P_4\}$
while $q > 1$
do $q:=q/4$
if $p+q \leqslant y$
then $y:=y-p-q;\ p:=p/2+q$
else $p:=p/2$
fi $\{P_4\}$
od
$\{P_4 \wedge q=1\} \Rightarrow \{p^2 \leqslant n < (p+1)^2\}$

Dit programma (afkomstig van J.O. Dahl) is snel, maar ook tamelijk ondoorzichtig, zelfs met de juiste asserties. Toch is er niets wezenlijks veranderd in programma BB5 ten opzichte van programma B, elke stap was te overzien. Dus mogen we concluderen dat BB5 volledig correct is.

Oefeningen:

1. Vul de stappen tussen de programma's B en BB1 in.
2. Bewijs de geldigheid van regels (5.3) en (5.4).
3. Voeg aan S_4 de tussenliggende asserties toe en transformeer S_4 naar het equivalente deel van BB5.

6. CONCLUSIES EN LITERATUUR

Het systeem om de partiële correctheid van programma's te kunnen bewijzen dat Tony Hoare in 1969 beschreef [H] is goed toepasbaar op zeer korte programma's. Deze techniek heeft inmiddels vaste voet in de literatuur verworven. Een adequate inleiding is bijvoorbeeld te vinden in het boek van C. Livercy [L]. Bij wat langere programma's (50 regels) worden de benodigde asserties voor een bewijs van correctheid in de Hoare-Logica echter zo complex, dat zo'n bewijs ondoenlijk wordt. Om dit probleem op te lossen zijn er een aantal technieken ontwikkeld.

Vaak goed toepasbaar en prettig werkend is de techniek van *Data-Abstractie.* Hierbij wordt een data-structuur beschreven aan de hand van de gewenste eigenschappen. Er wordt bijvoorbeeld gesproken over een verzameling waar elementen aan toegevoegd en uit weggelaten kunnen worden, en waarbij kan worden getest of een object in een verzameling zit. Vervolgens wordt een programma geschreven en van een correctheidsbewijs voorzien met deze verzameling als data-structuur. Dan wordt er een concretere implementatie van de data-structuur gegeven (voor de verzameling bijvoorbeeld een binaire zoekboom, een stack, of een hash-tabel) met een implementatie van de bijbehorende gewenste mogelijkheden om de structuur te manipuleren. Als laatste stap wordt onafhankelijk van de rest van het programma bewezen dat de beoogde implementatie aan de gestelde wensen beantwoordt. Een goede inleiding in dit gebied is gegeven door Jones [J].

De invoering van *fantoomvariabelen* is oorspronkelijk gebeurd om het verband tussen twee of meer gewone variabelen eenvoudiger te kunnen omschrijven. Vooral bij het geven van terminatiebewijzen bleek deze techniek nuttig. Een inleiding in de techniek van het bewijzen van terminatie en toepassing van het gebruik van fantoomvariabelen hiervoor is te vinden in het boek van Livercy [L]. Door Van Diepen en De Roever [DR] wordt een fantoomstructuur opgebouwd om het verband aan te geven tussen twee data-structuren.

Programmatransformatie is als techniek nog niet tot volle wasdom gekomen. Darlington [D] geeft aan dat het mogelijk is door middel van transformationeel programmeren verscheidene sorteeralgoritmen te ontwikkelen. Van Diepen en De Roever [DR] gebruiken deze techniek om aan te tonen, dat het mogelijk is om in samenspel met de twee voorgaande technieken zeer complexe - maar al bekende - algorithmen af te leiden. De daarbij gebruikte strategie heeft verwantschap met de strategie van Top-Down programmeren.

Het is redelijk te verwachten, dat intrinsiek complexe algoritmen met een combinatie van deze drie technieken eenvoudiger en met zekerheid over hun correctheid kunnen worden ontwikkeld. Ervaring ontbreekt echter.

Gebruik van de strategie van transformationeel programmeren zal niet noodzakelijk leiden tot het best denkbare algoritme. In dit artikel zijn drie afleidingswegen te vinden, namelijk van programma L1 naar L4, van B naar BA en van B naar BB5. Beginnend met programma L1 is er geen rechtstreekse methode om in BB5 terecht te komen. Om naast de L-serie programma B te bedenken vereist kennelijk inzicht van een hoger niveau. Ook is het blijkbaar mogelijk op meer dan een "optimaal" algoritme (BA en BB5) vanuit een beginalgoritme (B) terecht te komen. Met transformationeel programmeren is het mogelijk betere algoritmen te ontwikkelen vanuit een bestaand algoritme, maar het is zo niet mogelijk een essentiëel nieuw algoritme te verkrijgen.

Het voorbeeld van de Integer-Square-Root algoritmen werd genomen uit een rapport door A. Blikle [B].

Literatuur

[B] Blikle, A. "Specified programming", ICS PAS Reports 333, Warschau (1978).

[D] Darlington, J. "A synthesis of several sorting algorithms", Acta Informatica, deel 11, blz. 1-30 (1978).

[DR] Van Diepen, N.W.P. & De Roever, W.P. "Program derivation through transformations: the evolution of list-copying algorithms", Te verschijnen (1985).

[H] Hoare, C.A.R. "The axiomatic basis of computer programming", Communications of the ACM, deel 12, no. 10, blz. 576-583 (1969).

[J] Jones, C.B. *Software development: a rigorous approach.* Prentice-Hall Series in Computer Science, Londen (1980).

[L] Livercy, C. *Théorie des Programmes. Schémas, Preuves, Sémantique.* Parijs (1978).