Algoritmen
abstract bezien

Jaap van Oosten

Department Wiskunde, Universiteit Utrecht

Gastcollege bij "Programmeren in de Wiskunde", 6 april 2017

Een algoritme is een rekenvoorschrift dat op elk moment van de berekening de volgende stap vastlegt, en ook bepaalt wanneer de berekening 'af' is.
Andere woorden: procédé, methode.
Algoritmen zijn zo oud als de wiskunde zelf:

- Mesopotamische kleitabletten geven methode voor: "delen door 7" (ca. 2500 v. Chr.)
- Euclidisch algoritme voor het bepalen van de grootste gemene deler
- Meetkundige algoritmen voor constructies met passer en liniaal
- Staartdelen

De opgave: “vind een algoritme voor...”:

Het “Delische probleem”

Kwadratuur van de cirkel

trisectie van een hoek

een formule voor de wortels van een 7e graads vergelijking

Maar...wat *is* een algoritme precies?
Kun je bewijzen dat er geen algoritme mogelijk is, voor een gegeven probleem? Geen algoritme, van welke soort dan ook?
Vragen uit het begin van de 20e eeuw.

Filosofische vraag: wanneer kan ik een uitspraak doen van de vorm: *voor elke x is er een y zodat ...*?
(Bijvoorbeeld: *elke veelterm van oneven graad heeft een nulpunt in* $\mathbb{R}$)
Volgens sommige wiskundigen: zo'n uitspraak kun je alleen doen als je een *algoritme* hebt om, gegeven x, zo'n y te vinden.

David Hilbert presenteerde in 1900 een lijst van belangrijke problemen voor de twintigste eeuw. Zijn Tiende Probleem luidde:

Vind een procédé om, gegeven een veelterm $P(X_1, \ldots, X_n)$ in gehele coëfficiënten en variabelen $X_1, \ldots, X_n$, in een eindig aantal stappen uit te maken of de vergelijking

$$P(X_1, \ldots, X_n) = 0$$

oplossingen heeft in gehele getallen.
Bijvoorbeeld: de vergelijking van Fermat:

$$(X_1^2 + X_2^2 + X_3^2 + X_4^2 + 1)^7 + (Y_1^2 + Y_2^2 + Y_3^2 + Y_4^2 + 1)^7 = Z^7$$

Je ziet: Hilbert vraagt om een algoritme *van welke soort dan ook*: je mag passer en liniaal gebruiken, of een telraam, of iets anders.

In de jaren 1920–1940 dacht een aantal mensen na over een definitie van wat het betekent, dat een berekening volgens een algoritme wordt uitgevoerd.
Een van die mensen was Alan Turing, die in 1936 een artikel schreef, *On Computable Numbers*[...]. Turing stelt zich een mens voor, die volgens een vast voorschrift rekent. Hij noemt zo'n mens een "computer". De rekenaar leest symbolen die in een schrift staan genoteerd, en schrijft ook symbolen.

"The behaviour of the computer at any moment is determined by the symbols which he is observing, and his "state of mind" at that moment. We may suppose that that there is a bound B to the number of symbols or squares which the computer can observe at one moment.[...] We will also suppose that the number of states of mind which need to be taken into account is finite"

Op grond van deze overwegingen komt Turing tot de definitie van een “machine”, later “Turing machine” genoemd. Ik bespreek nu een variant op dit idee, later door o.a. Minsky bedacht: de *registermachine*.
De registermachine heeft een oneindig aantal registers, of geheugenplaatsen, die elk een geheel getal ≥ 0 kunnen bevatten. Als we $R_1, R_2, \ldots$ schrijven voor de registers, geven we met r_i het getal aan dat in register R_i zit.

Een *programma* voor de registermachine is een eindige, genummerde lijst instructies van de volgende vorm:
$r_i^+ \Rightarrow k$ betekent: hoog de inhoud van register R_i met één op, en ga naar instructie k
$r_i^- \Rightarrow k, l$ betekent: als $r_i > 0$, trek één af van de inhoud van R_i en ga naar instructie k; als $r_i = 0$, doe niets en ga naar instructie l
Het kan zijn, dat een instructie ons opdraagt om naar "instructie k" te gaan terwijl er geen "instructie k" is; de machine stopt dan.

Voorbeelden:

$1.r_1^+ \Rightarrow 1$

Dit programma blijft aldoor de inhoud van R_1 met één ophogen, en stopt nooit.

$1.r_1^- \Rightarrow 2, 4$
$2.r_2^+ \Rightarrow 3$
$3.r_3^+ \Rightarrow 1$

Dit programma hevelt de inhoud van R_1 tegelijkertijd over in zowel R_2 als R_3, en stopt dan.

$1.r_1^- \Rightarrow 3, 2$
$2.r_1^+ \Rightarrow 2$
$3.r_1^+ \Rightarrow 4$

Dit programma stopt precies dan, wanneer $r_1 \neq 0$.

Een *berekening* van de registermachine met programma P en beginsituatie ("input")
$r_1 = a_1, \ldots, r_k = a_m$
is een rij van "machinetoestanden"

$$(k, r_1, \ldots, r_m)$$

waarbij k het nummer is van de op dit moment uit te voeren instructie, en $r_1, \ldots, r_m$ de registerinhouden op dit moment zijn. Hierbij moet elke volgende machinetoestand volgen uit zijn voorganger door instructie k uit te voeren.
En de eerste machinetoestand is natuurlijk $(1, a_1, \ldots, a_m)$
Als er geen instructie k is, dan stopt de berekening. Let wel: niet elke berekening stopt!

Definitie: we noemen een functie $F : \mathbb{N}^k \to \mathbb{N}$ van k variabelen *berekenbaar*, als er een programma P is zodat voor elk k-tal argumenten $a_1, \ldots, a_k$, de berekening van de registermachine met programma P en input $a_1, \ldots, a_k$ stopt, en in de laatste machinetoestand is $r_1 = F(a_1, \ldots, a_k)$.

De *These van Church-Turing* zegt: als een functie door een algoritme is uit te rekenen, is hij berekenbaar.

Deze these heeft de tand des tijds doorstaan!

Er zijn goed gedefinieerde functies, die *niet* berekenbaar zijn!

Nummer alle programma's: $P_0, P_1, \ldots$ en definieer de volgende functie van twee variabelen:
$f(n, m) = 0$ als de berekening met programma P_n en input m stopt
$f(n, m) = 1$ anders.

Turing's *Stopprobleem* (Halting Problem): bovenstaande functie is **niet** berekenbaar!
We zeggen ook wel: het stopprobleem is *niet oplosbaar*. We komen hier later op terug.

Terug naar Hilbert's Tiende Probleem.

De Rus Yuri Matiyasevich bewees in 1970 dat, als er een algoritme was om uit te maken of een willekeurige veelterm met coëfficiënten in $\mathbb{Z}$ een nulpunt heeft in gehele getallen, er ook een algoritme moet zijn dat het stopprobleem oplost.

Het door Hilbert gewenste algoritme kan dus niet bestaan.

Voor elk programma P en elk natuurlijk getal $k \geq 1$ is er een "partiële functie" F_P van k variabelen:

$F_P(a_1, \ldots, a_k) = m$ als de berekening van de registermachine met programma P en input $a_1, \ldots, a_k$ stopt, en in de laatste machinetoestand is $r_1 = m$
$F_P(a_1, \ldots, a_k)$ is *ongedefinieerd* als bovenstaande berekening niet stopt.
Eigenlijk is F_P dus een functie $U \to \mathbb{N}$, waar $U \subseteq \mathbb{N}^k$.
Dit soort functies heet *partieel recursief*.

Als we de programma's netjes nummeren, hebben we ook een nummering

$$F_0, F_1, \ldots$$

van de partieel recursieve functies.
Merk op, dat elke partieel recursieve functie oneindig vaak voorkomt in deze opsomming. Als de partieel recursieve functie F gelijk is aan F_n, heet het getal n een *index* van F. Elke partieel recursieve F heeft oneindig veel indices.
Er geldt, als gevolg van het "netjes nummeren", dat de partiële functie

$$(n, m) \mapsto F_n(m)$$

ook partieel recursief is. Deze functie wordt uitgerekend door een "universeel programma" U.

De Recursiestelling

Stel F is een partieel recursieve functie van $k+1$ variabelen. Dan is er een index e zo, dat voor elk k-tal $a_1, \ldots, a_k \in \mathbb{N}$ geldt:

$$F_e(a_1, \ldots, a_k) \simeq F(a_1, \ldots, a_k, e)$$

Hier betekent het symbool $\simeq$: òf beide kanten zijn ongedefinieerd, òf beide zijn gedefinieerd, en gelijk aan elkaar.
Voorbeeld 1. Laat $k = 1$; de functie $(a, e) \mapsto e$ is zeker berekenbaar. Volgens de recursiestelling is er een index e zodat $F_e(a) = e$. Dat wil zeggen: er is een programma P dat bij elke input zijn eigen rangnummer (uit de opsomming $P_0, P_1, \ldots$) geeft.

Voorbeeld 2. We laten zien dat het stopprobleem niet oplosbaar is.
Laat G een berekenbare functie zijn, zo dat $G(x)$ precies dan gedefinieerd is, als $x \neq 0$.
Neem aan dat de functie $f(n, m)$ van het stopprobleem berekenbaar is. Dan geldt dat ook voor de samengestelde functie

$$H(m, n) \simeq G(f(n, m))$$

Volgens de recursiestelling is er een index e zodat

$$F_e(m) \simeq H(m, e) \simeq G(f(e, m))$$

Maar nu krijgen we:
$F_e(m)$ is gedefinieerd $\Leftrightarrow$ $G(f(e, m))$ is gedefinieerd $\Leftrightarrow$
$f(e, m) = 1$ (vanwege de keuze van G) $\Leftrightarrow$ $F_e(m)$ is ongedefinieerd (vanwege de definitie van f).
Een duidelijke tegenspraak.

Meer weten?
Er is een mastercursus “Computability Theory”.