Dr Ir W. L. van der Poel

TALEN EN KUNSTTALEN

UITGEVERIJ WALTMAN – HIPPOLYTUSBUURT 4 – DELFT

TALEN EN KUNSTTALEN

REDE

UITGESPROKEN BIJ DE AANVAARDING VAN HET AMBT VAN BIJZONDER HOOGLERAAR IN DE TOEGEPASTE LOGICA ALS GRONDSLAG VAN STRUCTUUR EN GEBRUIK VAN REKENAUTOMATEN AAN DE TECHNISCHE HOGESCHOOL TE DELFT, OP WOENSDAG 12 DECEMBER 1962.

DOOR

Dr Ir W. L. VAN DER POEL

UITGEVERIJ WALTMAN – HIPPOLYTUSBUURT 4 – DELFT

Mijne Heren Leden van het Bestuur van het Delfts Hogeschoolfonds,
Mijne Heren Curatoren van deze bijzondere leerstoel,
Mijne Heren Hoogleraren,
Dames en Heren Lectoren, Docenten, Instructeurs en Assistenten,
Dames en Heren Studenten,
en voorts gij allen, die door Uw aanwezigheid van Uw belangstelling blijk geeft,

Zeer gewaardeerde Toehoorders,

De titel die ik voor deze oratie gekozen heb („Talen en kunsttalen") zal U misschien wat vreemd in de oren klinken. U zou hieruit kunnen opmaken dat ik met U wilde spreken over de natuurlijke talen en over kunsttalen als Esperanto, terwijl U toch van mij verwacht dat ik zal spreken over een onderwerp uit de techniek, in het bijzonder over de techniek van rekenautomaten. Mijn leeropdracht luidt immers onderwijs te geven in „De toegepaste logica als grondslag van structuur en gebruik van rekenautomaten". Welnu, ik zal trachten U het verband in de loop van dit verhaal duidelijk te maken. Maar laat ik dan bij het begin beginnen.

U heeft natuurlijk allen gehoord van de moderne elektronische rekenmachines, deze wonderlijke apparaten die ons tegenwoordig in staat stellen berekeningen uit te voeren met snelheden die vele malen uitgaan boven de menselijke snelheden. Wat belangrijker is dan de snelheid is de mate van automatisme van deze machines. Zij kunnen niet slechts één bewerking doch een geheel probleem automatisch oplossen. Een machine die alleen één enkele vermenigvuldiging in 1/20000 seconde kan uitvoeren en daarna op de mens moet wachten om hem de volgende gegevens toe te dienen is waardeloos; daarentegen kan een langzame maar automatische machine in sommige gevallen wel nuttig zijn.

De geschiedenis van deze machines is niet zo jong als U misschien wel zou denken. Reeds in de 17e eeuw construeerden PASCAL en LEIBNIZ hulpmiddelen om de rekenkundige hoofd-

bewerkingen optellen, aftrekken en vermenigvuldigen machinaal te verrichten. Omstreeks 1840 kwam CHARLES BABBAGE, later Lucasian Professor in Cambridge met plannen voor de zogenaamde Analytical Engine, een mechanische machine die automatisch een gehele formule moest uitrekenen.[1]) Door financiële en technische moeilijkheden zijn slechts enkele stukken van deze machine gebouwd. Toch kunnen wij bij BABBAGE al vrijwel alle essentiële ideeën terugvinden waarop onze tegenwoordige machines berusten. We zullen aan de ontwikkeling van de tafelrekenmachines in dit verhaal voorbijgaan. In de historie van de automatische rekenmachines moeten wij na BABBAGE ineens ca. een eeuw verder gaan voor er weer nieuwe ideëen opduiken.
In 1936 schreef A. M. TURING zijn beroemde artikel „On Computable Numbers".[2]) Hierin wordt een ontwerp van een automatische rekenmachine ten tonele gevoerd, echter niet om ooit het ding te gaan maken maar uitsluitend als gedachtenexperiment om hiermee te bewijzen hoever we met machines kunnen komen en welke problemen principieel niet opgelost kunnen worden.

Gedurende de tweede wereldoorlog werden al in Amerika rekenmachines werkend met elektromechanische relais gemaakt, hoofdzakelijk voor militaire doeleinden, zoals het berekenen van geschutstabellen.[3]) Ook in Duitsland werkte ZUSE als pionier onder moeilijke omstandigheden (merkwaardigerwijze niet voor militaire doeleinden) aan een ontwerp.[4]) Pas na de oorlog kwam de eerste elektronische rekenmachine klaar en van dat moment af begon een uiterst snelle ontwikkeling.[5]) Nu is een rekenmachine geen apparaat meer, waarvan alleen enkele geavanceerde laboratoria met veel moeite een exemplaar gebouwd hebben, maar zijn zij allerwegen in opkomst, vaak in administratieve toepassingen en zijn zij bij wijze van spreken „over de toonbank" te koop. Toch is de ontwikkeling nog in volle gang en het is wel te verwachten dat in de komende jaren nog een geweldige omwenteling in de wijze waarop deze machines „aangesproken" worden, te zien zal zijn.

We gaan nu eens nader een kijkje nemen in deze machines. Paradoxaal genoeg zijn de elementen waaruit ze zijn samengesteld van zeer eenvoudige aard. De elementen zijn eigenlijk alle vergelijkbaar met gewone lichtschakelaars. Ze kunnen af staan, of aan staan, meer niet. Daarmee kunnen ze twee waarden

representeren: 0 of 1. Uit vele van deze eenvoudige elementen kan men een hele machine opbouwen. Ik kan U onmogelijk in deze beperkte tijd enig idee geven van hoe zo'n machine in elkaar zit. Maar laat ik volstaan met te zeggen dat met slechts enkele duizenden elementen al een heel bruikbare machine gebouwd kan worden. De ingewikkeldheid van het samenstel en de organisatie van de schakelelementen staat in scherpe tegenstelling tot de eenvoud van de elementen zelf. Het vak dat zich met deze opbouw bezig houdt noemt men de schakeltechniek.

Wat ons betreft zijn we nu pas gekomen aan het begin van ons verhaal. Van nu af aan nemen we het aanwezig zijn van zo'n machine aan als een fysisch gegeven. Ook zullen we ons in het vervolg op het standpunt stellen dat er door de machine in technisch opzicht geen fouten worden gemaakt. Dit is wel een ideaal beeld maar bij de huidige stand van de techniek benaderen we dit ideaal al vrij aardig.

We zijn dus nu gesteld voor het probleem hoe men deze machine het werk laat verrichten waarvoor hij bedoeld is. Natuurlijk doet een machine een probleem niet „vanzelf". Het moet hem verteld en geleerd worden. Hiertoe wordt aan de machine toegevoerd een program, bestaande uit een groot aantal elementaire instructies die netjes in volgorde uitgevoerd een probleem stap voor stap oplossen. Dit program wordt door de mens opgesteld, althans zeker in de begintijd van het gebruik van een machine. Eigenlijk is het onjuist te spreken over het oplossen van een probleem door de machine; het probleem is al opgelost door het opstellen van het program, door het aangeven van de weg. Het moet alleen nog uitgevoerd worden.

Het ontleden van een probleem in zijn elementaire opdrachten, rekenstappen of handelingen vertoont wat betreft de methodiek grote gelijkenis met de structuur van de machine zelf. Op zich zelf verricht iedere instructie maar een heel eenvoudige handeling, (b.v. één enkele optelling) en er zijn ook maar een zeer beperkt aantal soorten handelingen (net als soorten bouwstenen bij de constructie). Toch kan een groot program zeer moeilijk worden door de veelheid en de gecompliceerdheid van de interrelaties tussen de instructies.

Wat is namelijk het geval? In de moderne machines bevinden zich de instructies in hetzelfde geheugen als de getallen. Men

kan dus rekenen met instructies en de machine kan zelf instructies omvormen. Dit geeft een complicatie maar ook een ongekende flexibiliteit aan het programmeren.

Wanneer men eenmaal een poosje met een rekenmachine gewerkt heeft dan is als spoedig het omvormen van de probleemstelling (of liever de omvorming van de oplosmethode) tot een reeks instructies een sleur geworden; zelfs een vervelende sleur. En wat nog erger is, doordat elke keer alle handelingen weer opgeschreven moeten worden kan men elke keer weer schrijffouten maken. De bedoeling van automatisering in de informatieverwerking is het ontlasten van de mens van sleurhandelingen en het voorkomen van menselijke fouten. De gebruikers hebben dit al spoedig in de gaten gehad en hebben een programmeertechniek gevonden die het mogelijk maakt om een vaak voorkomende gecompliceerde handeling (zoals b.v. het bepalen van een logaritme) tot een eenheid te maken die men subprogram noemt en die met slechts één instructie „aangeroepen" wordt. Deze instructie is dan te beschouwen als een nieuw type instructie. Het repertoire van standaardhandelingen is hiermee dus vergroot, niet door iets toe te voegen aan de technische uitrusting van de machine, maar door een subprogram te schrijven. In tegenstelling tot de technische uitrusting die men in de engelse taal wel „hardware" noemt heeft men deze uitbreiding van faciliteiten door programs wel „software" genoemd.

Heeft men eenmaal een groot aantal van deze subprograms, dan is het duidelijk dat men zich hiermee het werk aanzienlijk kan verlichten maar het zal ook duidelijk zijn dat men deze programs niet alle tegelijk meer in het geheugen van de machine kan opslaan omdat dit hiervoor te klein is. Men zal dus een methode moeten hebben om deze programs buiten de machine op te slaan op het invoermedium van de machine (b.v. ponskaarten). Verder heeft men een organisatieprogram nodig dat deze programs weer in de machine kan brengen, ze op de juiste manier aan elkaar kan verbinden en ze een plaats in het geheugen kan toewijzen. Men noemt dit een assemblageprogram. Hier hebben we dus al te doen met een program dat niet meer dient om het directe rekenprobleem op te lossen maar om het programmeerprobleem op te lossen. En welk apparaat is geschikter om te helpen bij het organiseren van dergelijke problemen dan de machine zelf!

Nog steeds zal het U niet duidelijk zijn waarom ik in de titel van deze voordracht talen als onderwerp heb gekozen. Laten we daarom eens een parallel maken tussen programmeren en talen. Het geven van instructies aan een machine is in zekere zin het mededelen van het probleem aan de machine, het toespreken van de machine, zij het dan in een kunsttaal. Men zou de instructies de werkwoorden kunnen noemen en de getallen waarmee gerekend wordt de objecten. De objecten worden in de machinetalen niet aangeduid door hun naam maar door het nummer van de plaats waar ze in de machine staan, het zgn. adres. Het operatiegedeelte van de opdracht geeft het werkwoord aan. Elke opdracht is een imperatieve zin van een korte standaardstructuur, nl. doe de handeling a met het object b. Het gehele boek van zinnen vormt het program. In het bovenstaande zagen we al het gebruik van subprograms. Men kan ze beschouwen als bepalende bijzinnen, die niet elke keer meer herhaald hoeven te worden. Het omvormen van en rekenen met de ininstructies zelf zou men het naamwoordelijk gebruik van werkwoorden kunnen noemen.

Het gebruik van assemblage-programs om in de goede volgorde subprograms uit een subprogrambibliotheek te lezen en te leren heeft de stimulans gegeven om ook de per probleem zelf te maken stukken van het program in een iets hanteerbaarder vorm te schrijven. Was het in het stenen tijdperk van de programmering nog nodig om zelf aan elk object een plaats toe te kennen, nu kan dit evengoed door de machine georganiseerd worden, mits men het assemblageprogram een adresboek laat bijhouden. Zo'n adresboek is niets anders dan een lijst waarin de machine bijhoudt welk vrij adres in het geheugen toegekend wordt aan een naam. De naam-plaats correspondentie geeft de programmeur de vrijheid om de namen voor zijn objecten zelf te kiezen en zich niet meer te bekommeren om de plaats. Ook wil men de werkwoorden van de zin liever met een gemakkelijk te onthouden naam voor de operatie aanduiden dan met een codecombinatie. Een assemblageprogram verzorgt ook deze vertaling gemakkelijk met behulp van een lijst. En hier hebben we het werkwoordelijk gebruik van naamwoorden als equivalent.

Een assemblageprogram heeft naast de in zgn. symbolische code geschreven opdrachten, de imperatieve zinnen, ook nog aanwijzingen nodig over hoe de stukken aan elkaar gezet moeten

worden, over het reserveren van ruimte in het geheugen enz. Dit laatste betekent dat men voor sommige namen additionele informatie moet geven om de vertaling mogelijk te maken. De instructies hiervoor worden door de assembler begrepen en gebruikt maar behoren niet tot het uiteindelijke program. Men zou het kunnen vergelijken met de aanwijzingen aan de boekbinder, die men ook niet in het latere boek terugvindt. Deze zinnen zijn niet van imperatieve aard maar van declaratieve aard.

Nog steeds bevinden we ons op het niveau van programmeren waarbij eerst door de mens het probleem ontleed moet worden in afzonderlijke opdrachten, zij het dan dat de namen die in die opdrachten voorkomen prettiger gekozen kunnen worden. Gesteld dat ons de in de machine geboden opdrachtcode niet geheel past voor het soort probleem dat wij moeten oplossen. Dan kunnen wij een nieuwe codetaal scheppen door een program te maken dat elk van de nieuwe opdrachten vertaalt en meteen uitvoert. Men noemt zo'n program een interpreteerprogram. Een interpreteerprogram speelt dus een andere machine (of soms de machine zelf) binnen de machine. We bevinden ons hier op een niveau dat een graad hoger ligt dan het programmeren in de machinetaal. Merkwaardige consequenties verkrijgt men als men een interpreteerprogram maakt dat de machinecode zelf interpreteert. Zulke programs worden wel gebruikt om fouten op te zoeken in andere programs. Inplaats van het te controleren program op hoge snelheid, onzichtbaar, echt uit te voeren kan men de interpretator dezelfde handelingen laten doen maar tevens een volledige verslaglegging laten verrichten. Denkt men zich nu eens het geval dat dit interpreteerprogram zichzelf moet interpreteren. Dan komt er een moment dat de interpretator een instructie tegenkomt die hij moet interpreteren en waarvoor hij die instructie zelf moet veranderen. Bedoelde moeilijkheid wordt zeer aardig weergegeven in het volgende gedichtje van Trijntje Fop

Een schaap ontraadt, althans aan schapen,
het schapentellen voor het slapen.
Want meestal, na een schaap of elf,
– zo zegt dat schaap – passeert men zelf.
En 't brein, waarin de slaapzin zetelt,
ligt dan te tobben of dat meetelt.

Het lag voor de hand, dat de programmeurs niet gestopt zijn voor de barrière van de machinetaal maar gepoogd hebben de taal welke door mensen gebruikt wordt direct aan de machine toe te voeren. Reeds in 1951 publiceerde RUTISHAUSER een monografie over het vertalen van normale wiskundige formules in machinetaal.[6)7)] In 1958 werd door PERLIS en SAMELSON een internationale taal ALGOL (*Alg*orithmic *L*anguage) gepubliceerd als eerste proeve van een programmeertaal die zoveel mogelijk moest lijken op het normale gebruikelijke wiskundeschrift.[8)] Later werd door 13 auteurs een verbeterde versie uitgegeven, genaamd ALGOL 60.[9)] Voor administratieve doeleinden is een dergelijke programmeertaal COBOL (*C*ommon *B*usiness *O*riented *L*anguage) gepubliceerd.[10)]

Het procédé dat in de machine gevolgd wordt om deze talen te vertalen ligt hierbij in de regel iets anders dan bij de interpretatieve talen. Men vertaalt eerst het gehele program en daarna wordt het pas uitgevoerd. Men noemt dit compileren. Een compilator is een program dat vertaalt, assembleert en soms ook stukjes program genereert. De beschrijving en definitie van de genoemde programmeertalen moet met de uiterste strengheid en nauwgezetheid gebeuren. Het is duidelijk gebleken dat onze gebruikelijke wiskundige formuletaal ten behoeve van het beschrijven van procédés voor het oplossen van problemen nog veel te kort schiet. Wiskundige formules drukken statische waarheden uit tussen ideale variabelen; een program moet dynamisch aangeven hoe uit onnauwkeurige getallen (slechts een eindig aantal decimalen) met onnauwkeurige bewerkingen (zoals afgeronde vermenigvuldiging) de gewenste resultaten verkregen worden. Voor het definiëren van een kunsttaal gebruikt men soms weer een tussentaal, een metataal. Er worden evenals in de normale taalwetenschap regels van syntaxis en semantiek gegeven om de formele structuur van de tekencombinaties vast te leggen en aan sommige structuren een betekenis te verbinden.

Eigenlijk is de mooiste definitie van de taal de vertaler zelf. Het is wonderlijk dat het mogelijk blijkt dat men deze vertalers in de taal zelf kan beschrijven op een zeer klein gedeelte na. Het is alsof men iemand een grammatica van het hottentots geeft, geschreven in het hottentots. Als hij dan bovendien het algemene vermogen om een taal te leren heeft en laten we zeggen een moeder om hem de moedertaal te leren dan kan hij na een

summiere training in de beginselen door zelfstudie verder komen en tot ongekende hoogte stijgen. Hetzelfde zien we met de kunsttalen. De machine heeft het vermogen tot het leren van de taal door zijn geheugen. Bovendien kent hij een oertaal, zijn eigen machinecode. Met een kleine vertaler kan hem een kunsttaal geleerd worden waarna hem in de kunsttaal zelf uitbreidingen worden geleerd enz. In het engels noemt men dit ,,bootstrapping''. De studie van deze self-compilers kan er veel toe bijdragen het maken van vertalers in de toekomst te vergemakkelijken. Op het ogenblik is het maken van een compilator voor een automatische programmeertaal nog een enorm omvangrijk en gecompliceerd karwei. Bevatte niet een van de eerste compilatoren voor FORTRAN (*For*mula *Trans*lator) een aantal van 25000 instructies?[11]) Ook sommige COBOL vertalers bestaan uit meer dan 40000 instructies. Inmiddels is wel gebleken dat ook al zeer bruikbare ALGOL vertalers geschreven kunnen worden in minder dan 4000 instructies.

Een belangrijk bestanddeel van al deze ,,processors'' is het werken met lijsten, tabellen en stapels. Men heeft zelfs speciale programmeertalen geconstrueerd om handig met lijsten te kunnen werken, b.v. LISP (*List P*rocessor)[12])[13]), IPL V (Information Processing Language V)[14]) en het COMIT-systeem (Compiler-Interpreter voor het vertalen van natuurlijke talen)[15]). Speciaal bij behandeling van niet-numerieke problemen kunnen deze programmeertalen hun diensten bewijzen, b.v. in de taalkunde bij het vertalen van de ene natuurlijke taal naar de andere en voor het maken van concordanties en programs die een woord in het zinsverband laten zien.

De organisatie van lijsten, tabellen en stapels kan in het machinegeheugen op verschillende manieren gebeuren. De meest eenvoudige wijze is om de gegevens op opeenvolgende plaatsen in het geheugen te zetten. Deze methode ligt voor de hand als men met verzamelingen gegevens werkt die in volgorde behandeld worden zoals de elementen van een matrix of de instructies van een program. Een kleine variant is de stapel. Zo noemt men een geheugen waarvan men alleen de plaats waar men het laatst iets opgeborgen heeft onthoudt. Het laatste element is weer als eerste bereikbaar net als het bovenste bordje op een stapel bordjes. Dit is dan volgens het principe :,,wie het laatst komt, het eerst maalt''. Zulke stapels spelen een grote rol bij het vertalen

van algebraische uitdrukkingen. B.v. a+b× (uitdrukking) kan vertaald worden in een machineprogram: reken uitdrukking tussen haakjes uit, vermenigvuldig met b, tel a op. Maar de machine las eerst a, daarna +, daarna b en daarna × en zag toen dat maal voor plus ging. De a, de plus, de b en de maal gaan alle op een stapel en vervolgens wordt het haakje gelezen enz. waardoor het duidelijk is dat eerst de haakjesuitdrukking vertaald moet worden. Daarna worden juist in de goede volgorde ,,vermenigvuldiging'' en b van de stapel gehaald en tenslotte ,,tel op'' en a.[16])

Als er meer dan één stapel in een geheugen geplaatst moet worden komt men al gauw in moeilijkheden als de top van de ene stapel de bodem van de andere raakt. Dan moet men stapels gaan verzetten wat nogal omslachtig is. Maar men kan de stapels ook op een geheel andere wijze organiseren door de opeenvolgende elementen ieder te voorzien van een adres dat naar de plaats van het volgende element verwijst. Die plaatsen kunnen dan kris kras door het geheugen verspreid liggen en vormen zo een keten. Het aardige van ketens is dat men heel eenvoudig een stukje in het midden kan toevoegen of wegnemen door alleen de verwijzingen te veranderen en de elementen zelf op zijn plaats te laten. Het bijbehorende bezwaar is dat men niet meer op eenvoudige wijze de plaats van het n^{de} element te weten kan komen als men weet waar het eerste staat. Men zal dan de keten moeten volgen. Dit ketenprincipe speelt een grote rol bij vele organisatieprograms in een machine.

Als deze ketens vertakt zijn worden ze aangeduid met de naam boom. Boomstructuren vindt men eigenlijk overal. Maar laat ik hier volstaan met een voorbeeld uit de taal. Als ik zeg: ,,De oude man steunt op een stok'', dan kan ik als wortel van de boom het werkwoord ,,is'' opvatten. Er zijn dan twee takken: ,,de oude man'' en ,,op een stok'', die ieder weer vertakt kunnen worden in ,,de'' en ,,oude man'' en ,,op'' en ,,een stok''. Ik zou het kunnen schrijven als een formule met haakjes:

((de (oude man)) steunt (op (een stok))).

U ziet uit deze zgn. geneste structuur meteen het verband met formuleuitdrukkingen met haakjes. De organisatie-moeilijkheid binnen de machine berust op het feit dat onze begrippen zich in meerdimensionale structuren zoals bomen laten denken maar dat

ze in een machine op een eendimensionaal geheugen afgebeeld moeten worden.

Ik wil niet nalaten hier een ontwikkeling te vermelden die in Amerika aan de gang is, nl. de taal LOGLAN (*Log*ical *Lan*guage).[17]) Deze taal behoort thuis in de klasse van de kunsttalen zoals Esperanto, dus primair bedoeld voor menselijk gebruik. De makers ervan (BROWN c.s.) beogen echter iets anders. Men kan met denken slechts zover komen als de werktuigen om dit denken mogelijk te maken toelaten (Hypothese van WHORF). Ons technisch kunnen is gebaseerd op een systeem van denken in symbolen. Onze wiskundige waarheden kunnen zeer concreet gesymboliseerd worden. De programmeertalen sluiten zich zich hier in zoverre bij aan, dat ze ook een zeer strenge structuur en een onontkoombare logica hebben. BROWN tracht de taal LOGLAN zodanig logisch in te richten dat alle voorwaarden om zich gemakkelijk scherp uit te drukken aanwezig zijn, o.a. door het volledig opnemen van de symbolische logica als structuurelement in de syntaxis. Voorts hebben alle tekens, wiskundetekens en interpuncties een uitspreekbaar en schrijfbaar woord als equivalent en zijn de woorden op een eenduidige manier te separeren ook als geen spaties geschreven worden. Men heeft dit bereikt door als predikaten (de zelfstandige en bijvoeglijke naamwoorden, bijwoorden en werkwoorden) steeds woorden van vijf letters van de structuur medeklinker – klinker – medeklinker – medeklinker – klinker of medeklinker – medeklinker – klinker – medeklinker – klinker te gebruiken. De „kleine woordjes" zoals voegwoorden, interpuncties, voornaamwoorden zijn van de structuur klinker of klinker – klinker of medeklinker – klinker.

Of van zo'n kunsttaal inderdaad enige vooruitgang verwacht mag worden en of de hypothese van WHORF daarmee gecontroleerd kan worden is een open vraag maar wel is de gelijkenis tussen ALGOL en LOGLAN treffend. Het is verwonderlijk dat de makers van LOGLAN dit niet opgemerkt schijnen te hebben. Men hoeft slechts ALGOL om te draaien en er een N achter te zetten.

In het kielzog van de automatische programmeertalen komen nog tal van andere problemen die ik hier niet onvermeld wil laten. Moderne machines zijn te duur om ongebruikt gelaten te worden. Ze moeten middelen hebben om automatisch op een ander program over te gaan zodra een program stagneert, hetzij

door een programmeerfout, hetzij door wachttijd bij invoer en uitvoer. Eenzelfde situatie doet zich ook voor als men een program kortstondig wil uittesten en daartoe een werkend program wil onderbreken, dat zijn loop weer mag hervatten zodra de test klaar is. Benevens zeer summiere technische voorzieningen in de machine (z.g. interruptiefaciliteiten), vereist deze techniek van werken een uitgebreid stel organisatieprograms waarmee de machine kan bijhouden welk program aan de beurt is, waar het staat, hoeveel ruimte en welke organen het in beslag neemt enz. Deze zogenaamde operationele systemen gaan hand in hand met de automatische programmeertalen en vormen daar wat de compilatoren betreft een onverbrekelijk geheel mee. Denken we alleen al eens aan het testen van programs in ALGOL. De gebruiker heeft niet de minste notie meer hoe het machineprogram er uit is komen te zien na de vertaling en waar het in het geheugen staat. Maar hij wil wel graag te zien krijgen waar het program fout gegaan is en wel in de taal waarin de gebruiker het program opgesteld heeft en niet in de inwendige machinetaal. Men noemt dit terugvertalen wel decompileren.[18]) Dit decompileren is soms nog moeilijker dan het compileren. Niet altijd is het mogelijk het program te decompileren in dezelfde termen omdat bij de compilatie essentieel informatie verloren is gegaan over de oorspronkelijke namen die inwendig vervangen kunnen zijn door getallen. Ook is de decompilatie niet eenduidig. Men kan uit een program dat $b \times c + a$ doet niet zien of er $a + b \times c$ gestaan heeft.

De meeste programmeertalen zijn tot nu toe descriptief van karakter. Zij helpen nog niet essentieel mee om de problemen op te lossen. Zij zijn nog slechts een werktuig om een uit te voeren proces mede te delen. Het vinden van de weg en de keuze van de methode is meestal de taak van de mens. De overgang naar talen die meer doen is niet sprongsgewijs maar is meer een kwestie van gradueel verschil. Zoals ook de wiskundetaal zich ontwikkeld heeft van kortschrift als uitdrukkingsmiddel van gevonden waarheden tot een transformatiemechanisme om nieuwe waarheden te vinden. Zo tekent zich ook al vaag de weg af die de programmeertalen zullen gaan. Reeds nu kan een programmeertaal dienen om een andere taal te vertalen, te transformeren. Een naam T staat niet meer voor een object maar voor een procedure die transformatieregels uitdrukt. Er zijn al programs gemaakt die

automatisch stellingen kunnen bewijzen.[19]) Er zijn programs en programmeertalen ontwikkeld en in ontwikkeling om algebraïsche manipulatie door de machine te laten verrichten. Ook wil ik wijzen op de in ontwikkeling zijnde informatie-algebra, welke calculus moet dienen om voor sommige klassen van problemen de oplossingsmethode te zoeken.[20])

Na dit vertoog over automatisch programmeren, assembleer-, compileer- en interpreteerprograms, talen, vertalers en operationele systemen zal het U duidelijk zijn geworden dat een rekenautomaat waarlijk niet zijn grote mogelijkheden aan het apparaat zelf ontleent maar dat de „software" hier een grote, zelfs overheersende rol in speelt. Een machine, kant en klaar opgesteld en bedrijfsklaar maar zonder programbibliotheek is een onbruikbaar instrument. Eerst moet men de beschikking hebben over de „software package", het samenstel van serviceprograms, bibliotheekprograms en bijbehorend operationeel systeem voor men effectief problemen kan aanpakken. Het maken van deze software is evenzeer constructie te noemen als het maken van de machine zelf en behoort daarmee ook typisch thuis op het gebied van de ingenieurswetenschappen. Dit vak is in deze vorm nog vrijwel onbekend aan onze nederlandse universiteiten en hogescholen. Het is in de praktijk al zeer goed merkbaar dat er aan technici die deze soort problemen kunnen aanpakken een grote behoefte bestaat. Deze behoefte zal in de komende tijd van snelle opkomst van rekenautomaten, in het bijzonder voor de administratieve toepassingen, alleen groter worden.

Tenslotte wil ik de hoop uitspreken, dat we geen geautomatiseerde ingenieurs zullen krijgen. Bij alle takken van wetenschap en techniek kan vaardigheid alleen verkregen worden door zelf met de problemen te worstelen en te trachten ze door hard werken tot een oplossing te brengen. Daarbij moet men bereid zijn steeds weer opnieuw automatismen aan te leren. Het is juist een van de essentiële eigenschappen van de mens dat hij de flexibiliteit bezit zich steeds weer nieuwe routines eigen te maken. Door dat deel van het werk aan machines over te laten, waar men zelf vaardigheid in heeft, kan men zich vrij maken om op hoger plan zijn werk voort te zetten. Ondanks de rekenautomaten zullen de kinderen op de lagere school dus nog steeds rekenen moeten blijven leren. Er zit in het zelf aanleren van een routine ook een spelelement en een voortdurende bevestiging dat men op

dit onderwerp met techniek de problemen weer de baas kan. De taal zelf is in alles onze basistechniek en het hanteren van woorden, die een gegeneraliseerde begripsinhoud hebben en niet meer dan een symbool zijn, stelt ons in staat om algemene waarheden te vinden en uit te drukken. De taal wordt ook slechts als automatisme aangeleerd door noeste vlijt en dagelijkse oefening. Door dit niet te vergeten kunnen wij mens blijven en de automatisering voor vooruitgang aanwenden inplaats van onze gaven te verleren door een geautomatiseerde mens te worden.

Zeer gewaardeerde Toehoorders,

In de eerste plaats wil ik hier graag mijn dank uitspreken aan het Delfts Hogeschoolfonds voor de benoeming tot bijzonder hoogleraar aan deze Hogeschool.

Mijne Heren Curatoren van deze bijzondere leerstoel,

U zeg ik dank voor het vertrouwen in mij gesteld door mij te willen voordragen voor deze leerstoel. Het getuigt van een vooruitziende blik een leerstoel voor een zo nieuw vak als dit in te stellen. Ik hoop het door U in mij gestelde vertrouwen waard te zijn en ik zal mijn beste krachten aan de mij opgedragen taak geven.

Mijne Heren Hoogleraren,

In mijn gevoel is het pas korte tijd geleden dat ik velen van U als leermeester had. Ik hoop dat ook nu mijn verhouding tot U die van collega is geworden ik nog steeds veel van U mag leren.

Mijne Heren Hoogleraren van de onderafdeling der Wiskunde,

De vele persoonlijke contacten die ik sinds lange tijd al met U gehad heb geeft mij volkomen het gevoel in een vertrouwde omgeving te komen. De wijze waarop U mij onmiddellijk hebt opgenomen in Uw midden heeft dit nog bevestigd.

Hooggeleerde van Heel, Hooggeleerde de Bruijn,

De unieke gelegenheid die mij gedurende mijn studie enerzijds in de afdeling natuurkunde, in de vorm van materiële mogelijk-

heden, anderzijds als speurwerkassistent bij het Delfts Hogeschoolfonds door U geboden werd is van beslissende betekenis geweest voor mijn ontwikkeling.

Hooggeleerde Kosten,

Met dankbaarheid denk ik terug aan de jaren dat ik onder Uw leiding op de mathematische afdeling van het Dr Neherlaboratorium der PTT in de gelegenheid ben geweest mij op het gebied van de rekenmachines verder te bekwamen.

Hooggeleerde van Wijngaarden,

Onze contacten dateren al uit het grijze verleden van het rekenmachinetijdperk in Nederland. Hoewel gij formeel nooit mijn leermeester zijt geweest moet ik toch zeggen dat ik door onze vele gesprekken en vooral door de inspiratie en de prikkel van de ontwikkeling bij het Mathematisch Centrum zeer veel geleerd heb. Dat gij als mijn promotor hebt willen optreden vervult mij met grote dankbaarheid. Ik hoop dat ook in de toekomst onze contacten even insprierend en veelvuldig mogen zijn als in het verleden.

Hooggeachte Kock,

Eigenlijk heb ik van U de eerste scholing gehad in het logisch denken. Ik herinner mij nog ieder woord van Uw lessen over het oplossen van problemen zijnde „het toepassen van wetten op het gegeven in verband met het gevraagde". De strengheid en klaarheid waarmee U het bouwwerk der elementaire natuurkunde hebt opgetrokken blijft ook nu nog voor mij model staan.

Mijnheer de Directeur-Generaal der PTT,

U ben ik dank verschuldigd voor de toestemming om een deel van mijn tijd aan het onderwijs te geven. Ik acht het een zeer gelukkige omstandigheid nog steeds bij het bedrijf werkzaam te zijn, daar de bread and butter problems uit de praktijk toch de uiteindelijke bron van inspiratie voor een nieuwe aanpak van problemen is. Ik hoop dat omgekeerd ook het bedrijf kan profiteren van de ontwikkeling in de Technische Hogeschool.

Collega's en Medewerkers van het Dr. Neherlaboratorium der PTT,

De sfeer van goede samenwerking en onderlinge discussies is een bijzonder prettige. Het soort werk dat met rekenmachines gedaan wordt is geen solitair werk maar is bij uitstek teamwerk. Vele ideeën zijn in teamverband uitgebroed en de kennis die ik hier zal uitdragen is voor een niet gering deel door Uw medewerking uitgekristalliseerd.

Dames en Heren Studenten,

Al is mijn vak maar een keuzevak, toch hoop ik dat enige onder U zich er toe aangetrokken zullen voelen. Er is ongetwijfeld een grote gelijkenis tussen de schakeltechniek die zich bezighoudt met het ontwerpen van hardware en de wiskunde en logica van programmeersystemen aan de zijde van de software. De schakeltechniek mag zich aan de Technische Hogeschool op een grote belangstelling verheugen. Ik heb dus goede hoop dat er voldoende puzzelaars en pluizers onder U zullen zijn die er behagen in scheppen om zich in het vak van programmering te gaan verdiepen. Want ik kan U verzekeren dat het element van puzzle en spel bij de rekenmachinetechniek een grote rol speelt. U kent waarschijnlijk wel de voldoening die het geeft voor een moeilijk wiskunde vraagstuk een „elegante" oplossing te vinden. Zulke situaties zijn er legio in het rekenmachinevak en ieder programmeur zal U beamen dat het maken van programs zeker geen automatisme maar eerder een kunst is. Ik hoop er veel toe bij te dragen U deelgenoot te maken in deze typische ingenieurswetenschap die techniek is en kunst tegelijk.

Ik heb gezegd.

Literatuur

1. Charles Babbage. Passages from the life of a philosopher. Longman, Green, Longman, Roberts, & Green, London, 1864.
2. A. M. Turing. On computable numbers. Proc. Lond. Soc. **42** (1936) 230.
3. J. Juley. The ballistic computer. Bell Lab. Rec. **25** (1947) 5.
4. 25 Jahre Entwicklung programmgesteuerter Rechenanlagen vom mechanischen Schaltglied über Relais, Röhre, zum Transistor. Jubiläumsfestschrift der Firma Zuse KG, Bad Hersfeld, 1961.
5. A. Goldstine, H. H. Goldstine, The electronic numerical intergrator ENIAC. Math. Tabl. Aids Comput. **2** (1946) 97.
6. H. Rutishauser. Über automatische Rechenplananfertigung bei programmgesteuerten Rechenmaschinen. Z. angew. Math. Mech. **32** (1951) 255.
7. C. Böhm. Du déchiffrage de formules logico-mathématiques par la machine même dans la conception du programme. Diss. Zürich, 1952.
8. A. J. Perlis, K. Samelson. Report on the algorithmic language ALGOL. Num. Math. **1** (1959) 41.
9. P. Naur (Ed.) et al. Report on the algorithmic language ALGOL 60. Num. Math. **2** (1959) 106.
10. COBOL, Report to the conference on data systems languages. Dept. of Defense, Washington D.C., 1961.
11. Programmer's Reference Manual FORTRAN. Int. Business Mach. Corp, New York, 1956.
12. J. McCarthy. Recursive functions of symbolic expressions and their computation by machine. Part 1. Comm. Ass. Comp. Mach. **3** (1960) 184.
13. P. M. Woodward, D. P. Jenkins. Atoms and lists. Comp. J. **4** (1961) 47.
14. A. Newell (Ed.). Information Processing Language V. Prentice Hall, Englewood Cliffs N. J. ,1961.
15. V. H. Yngve. The COMIT system for mechanical translation. In: Information Processing, Oldenbourg, München, 1960, p. 183.
16. F. L. Bauer, K. Samelson. Maschinelle Verarbeitung von Programmsprachen. In: Digitale Informationswandler. Vieweg, Braunschweig, 1962.
17. J. C. Brown. LOGLAN. Sci. Amer. **202** (1960) no. 6, 53.
18. M. H. Halstead, Machine independent computer programming. Spartan, Washington D.C., 1962.
19. H. Gelernter. Realisation of a geometry theorem proving machine. In: Information Processing, Oldenbourg, München, 1960, p. 273.
20. R. F. Clippinger. Information algebra. Comp. J. **5** (1962) 180.