STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

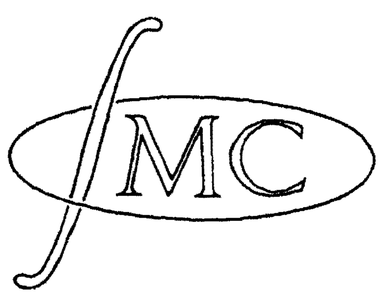
REKENAFDELING

R 890

Een systeem van geheugengebruik voor de programmeertaal ALGOL 60 en voor uitbreidingen daarvan

door

J. Nederkoorn



mei 1963

The Mathematical Centre at Amsterdam, founded the 11th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications, and is sponsored by the Netherlands Government through the Netherlands Organization for Pure Research (Z.W.O.) and the Central National Council for Applied Scientific Research in the Netherlands (T.N.O.),by the Municipality of Amsterdam and by several industries.

S u m m a r y

The first part of this report describes a memory organization suitable for dealing with ordinary arrays and own arrays with dynamic bounds in ALGOL 60 computer programs.
The second part discusses extension of ALGOL 60. A proposal is made to transform ALGOL 60 into a general language allowing the description of many classes of operations on many classes of objects, including lists.
In the third part it is shown how the memory organization devised for arrays may be used to implement this general language.
The system uses a stack or push-down-list, a chain of references intertwined with it, a counter-stack for storing all objects, the lenghts of which are unknown during compilation, and a free-space-list intertwined with the counter-stack. A garbage collector is used for disentangling counter-stack and free-space-list, thereby gaining storage space for stack or counter-stack.

1. Woord vooraf

In nova fert animus mutatas dicere formas
corpora.....

P.O. Naso

In dit verslag komt een voorgesteld systeem van geheugengebruik in electronische rekenmachines aan de orde, dat beoogt twee vraagstukken tegelijk op te lossen:

a. een doeltreffende organisatie van het werken met gewone arrays en own arrays in ALGOL 60 programma's.
b. uitwerking van een zodanig vertaal- en uitvoeringssysteem voor ALGOL 60, dat de voor allerlei uitbreidingen van de taal noodzakelijke maatregelen gemakkelijk kunnen worden genomen.

De bewering is, dat de kwaliteit van de oplossing voor a niet of nauwelijk lijdt onder de aanwezigheid van neven-doel b en omgekeerd.

Met het oog hierop bestaat dit verslag uit drie delen:

I een uiteenzetting van een mogelijke organisatie voor de arrays, geheel los van de uitbreidingen;

II een bespreking van de wenselijkheid van uitbreiding van de taal, alsmede van de vorm waarin deze uitbreiding syntactisch tot stand zou moeten komen;

III een uitwerking tot in bijzonderheden van een eenvoudig stelsel van uitbreiding van het vertaalsysteem, waardoor dit geschikt wordt voor het verrichten van zeer algemene klassen van bewerkingen op zeer algemene klassen van in rekenautomaten voorstelbare zaken.

De hier neergelegde denkbeelden zijn afkomstig uit uiteenlopende bronnen.

Naast de opgegeven literatuur hebben inspirerend gewerkt de kennismaking met LISP, de proefnemingen van Van de Laarschot en ondergetekende betreffende inlijving van een deel van LISP

in ALGOL 60 en het werk van Dr. Kruseman Aretz over het rekenen met getallen van veranderlijke lengte. Men zie de hierover te verschijnen verslagen.

Het systeem is in verschillende stadia op de Rekenafdeling van het Mathematisch Centrum besproken en het heeft daar bepaald niet onder geleden. Van Prof. van Wijngaarden is een groot deel van het over STORE meegedeelde afkomstig. De belangrijkste vereenvoudigingen en preciseringen dank ik aan dr. F.E.J. Kruseman Aretz, die ook op zich heeft genomen dit geschrift vóór de uitgave na te lezen en in onderdelen op zijn houdbaarheid te toetsen. Verscheidene andere medewerkers van de Rekenafdeling hebben wezenlijk bijgedragen door eerdere voorstellen met kracht als onvoldoende te verwerpen.

Het systeem is op dit ogenblik niet op een rekenautomaat geprogrammeerd. Misschien zal er gelegenheid zijn, dit werk ter hand te nemen bij het beschikbaar komen van de door het Mathematisch Centrum bestelde nieuwe machine, de Electrologica X8.

Amsterdam, mei 1963

J. Nederkoorn

Literatuur:

[1] Dijkstra, Dr. E.W., ALGOL-60 Translation, MR 35 en ALGOL Bulletin supplement nr. 10.

[2] Collins, George E., A method for overlapping and erasure of lists, CACM '60-12, Blz. 655.

[3] Bauer, F.L., Sequential Formula Translation, CACM'60-2, blz. 76.

[4] Van de Laarschot, P.J.J., en J. Nederkoorn, Beschrijving van de tweede ALGOL 60-vertaler van het MC, te verschijnen in de MR-serie.

[5] Backus, J.W., a.o., Revised Report on the Algorithmic Language ALGOL 60, edited by Peter Naur, Regnecentralen, Kopenhagen 1962.

[6] Wirth, N., A generalization of ALGOL, thesis University of California, 1962.

2.

2.1 Gewone arrays op de X1

In het systeem van Dijkstra en Zonneveld (MCI-ALGOL) staan de arrays in een stapel.
Dit is een aaneengesloten stuk machinegeheugen van veranderlijke lengte. In beginsel is voor de stapel het gehele geheugen, na aftrek van de ruimte voor het programma en voor het complex van arithmatische en administratieve hulproutines, beschikbaar.
De stapel bestaat uit cellen, te weten rekencellen en blokcellen in vrije afwisseling. De stapel groeit en slinkt, doordat tijdens de uitvoering van een ALGOL 60 - programma voortdurend cellen "op de stapel" gelegd worden en daar ook om allerlei redenen weer vanaf verdwijnen. In de stapel hoeft nooit te worden geschoven. Het vrijgeven van werkruimte betreft altijd de bovenste cel(len).
De rekencellen, die men o.a. boven op de stapel kan aantreffen, bevatten in hoofdzaak w a a r d e n, dienstdoend bij de uitwerking van expressies. De bovenste waarden zijn aan de beurt om aan een operatie te worden onderworpen. Na het uitvoeren van een operatie op 2 operanden is in het algemeen de interesse in de operanden verdwenen en daarentegen het éne resultaat belangrijk geworden. De stapel neemt dan dus met één rekencel af. Na voltooiing van een uitspraak zijn de bovenste rekencellen van de stapel verdwenen en is de bovenste cel van de stapel altijd een blokcel. Was de uitspraak een toekenningsuitspraak, dan is de waarde van het rechterlid in een of meer blokcellen opgeborgen om daar wat langer te worden onthouden.
Blokcellen zijn er in de stapel op elk moment evenveel als er bij de uitvoering van het ALGOL-programma onafgewerkte blokken zijn. Een blokcel bestaat in het algemeen uit 3 stukken (waarvan alleen het eerste deel nooit leeg is):
I Het koppelingsstuk. Hierin worden de betrekkingen van

het blok met andere blokken geadministreerd. Een blokcel heeft een fysisch adres in de stapel, PW genaamd.
In het koppelingsstuk worden nl. de PW's bewaard van (de blokcel van) het tekstueel omvattende blok, van het aangroepende blok - van belang als het beschouwde blok een procedure is -, het terugkeeradres, d.w.z. het adres in het vertaalde ALGOL-programma, waar de werkzaamheden na voltooiing van het beschouwde blok moeten worden voortgezet, enz. enz.

II Het stuk voor de eenvoudige variabelen. De lengte van dit stuk kan per blok verschillen, doch is voor verschillende activeringen van eenzelfde blok gelijk en kan reeds tijdens vertaling worden vastgesteld. Voor de eenvoudige variabelen, die in een blok worden gedeclareerd, stelt men nl. een hoeveelheid geheugenruimte beschikbaar, die per variabele alleen van het type (real, integer, of Boolean) afhangt.

III Het stuk voor de arrays. De lengte van dit stuk moet per activering van het blok worden vastgesteld en afgeleid uit de waarden van de ondergrenzen en bovengrenzen der verschillende dimensies op het moment van die activering. Behalve plaatsruimte voor de elementen van de arrays, de geïndiceerde variabelen, bevat dit stuk ook de z.g. storage functions van de arrays, een rij administratieve hulpgrootheden, die het aan de subroutine INDEXER mogelijk maken uit een rij indexwaarden de geheugenplaats van het bijbehorende element te vinden ten koste van een aantal vermenigvuldigingen. Deze INDEXER moet in actie komen voor iedere raadpleging van of waarde-toekenning aan een geïndiceerde variabele.

Blokverlating houdt in, dat een of meer blokcellen boven aan de stapel worden vrijgegeven. Op grond van het koppelingsstuk van de daarna bovenste blokcel worden dan een aantal toestandgrootheden van het systeem zodanig gewijzigd, dat het programma correct verder kan worden uitgevoerd. Voor own arrays met dynamische grenzen is in dit

systeem geen plaats. Wat de gewone own variabelen en de own arrays met constante grenzen betreft, kan men zich echter uitstekend behelpen door ze in het buitenste blok (beter: in de onderste blokcel) onder te brengen. Zou men een own array met dynamische grenzen in de stapel onderbrengen en zou bij hernieuwde activering van het blok, waarin dit array is gedeclareerd, de vereiste totale geheugenruimte voor dit array blijken te zijn gegroeid of geslonken, dan zou de stapel moeten worden uit- of aangeschoven en dat levert vanwege de talrijke onderlinge verwijzingen in de blokcellen welhaast onoverkomelijke moeilijkheden op.

2.2 Own arrays op de X1.

Om nu toch het hanteren van own arrays met veranderende grenzen in ALGOL-programma's mogelijk te maken is door Prof. Dijkstra in samenwerking met Van de Laarschot en schrijver dezes een systeem ontwikkeld, dat werkt met een z.g. contra-stapel. Dit systeem is sedert enige tijd op de X1 van het Mathematisch Centrum beschikbaar onder de naam MC II-ALGOL. Het vrije geheugen wordt nu van twee kanten "aangevreten". Onderaan "staat" de gewone stapel met een top die rijst en daalt op grond van de gebeurtenissen in het programma. Bovenaan "hangt", als een soort brokkelige stalactiet, de contrastapel en bezet de ruimte van af het bewegende "plafond" tot aan het fysische einde van het geheugen. Deze contrastapel bevat uitsluitend de elementen van de own arrays. De storage functions (met immers tijdens vertaling reeds vaststelbare lengte) passen uitstekend in de gewone stapel. Het ontbreken van administratieve verwijzingen in de contra-stapel maakt schuiven daarin minder bezwaarlijk. Groeien of slinken van own arrays leidt tot uit- of aanschuiven van de contrastapel en bijwerken van de boekhouding in de gewone stapel. Zuinigheidshalve wordt de contrastapel aansluitend gehouden. Gaten worden daarin dus niet getolereerd. Hoewel

het systeem aan strenge theoretische eisen voldoet, blijkt het practisch niet zonder bezwaren. Het voornaamste is, dat in verreweg de meeste ALGOL-programma's slechts een bescheiden deel van het machinegeheugen wordt gebruikt. Komen in zo'n programma own arrays voor met op een bepaald moment krimpende grenzen, dan wordt er dus onnodig en tijdrovend in heen en weer geschoven. Een ander bezwaar is dit, dat een "normaal" programma (d.i. een programma zonder own arrays) bij uitvoering met dit vertaalsysteem op het stuk van snelheid voelbaar heeft te lijden onder de omstandigheid, dat er steeds op de aanwezigheid van own arrays wordt gerekend.

Het is zonder twijfel een droevig feit, dat de meeste ALGOL-vertalers niet voorzien in het gebruik van dynamische own arrays.Niet zozeer uit practisch oogpunt, b.v. omdat een aantoonbaar belangrijke klasse van problemen daarzonder moeilijker of helemaal niet programmeerbaar is. Maar vooral, omdat hieruit blijkt, dat de vertalerschrijvers achterblijven bij de taalontwerpers. Het zou zo moeten zijn, dat minstens alle wezenstrekken van talen, op dit moment beschikbaar om automatiseerbare processen in te beschrijven, ook werkelijk in een verscheidenheid van methoden op machines van verschillend type gerealiseerd zouden zijn, opdat zowel de machinebouwers, als de groep dergenen, die inventief met programmeertalen bezig zijn, hieruit inspiratie zouden kunnen putten voor verdergaande ontwerpen. De huidige situatie werkt eerder toekomstige verarming, dan verrijking of veralgemening van het programmeerarsenaal in de hand. De bewering is, geloof ik, gewettigd, dat het hierna te schetsen systeem geschikt is om hierin verbetering te brengen.

2.3 Een nieuwe contra-stapel met vrije keten

Het voorstel is, alle arrays onder te brengen in een contra-stapel, die op de zelfde wijze als in MC II-ALGOL aan de bovenkant van het geheugen hangt en onderaan wordt begrensd door een bewegend plafond. Dit plafond is als fundamentele

toestandsgrootheid van het programma altijd op een vaste plaats te vinden. Op elk moment tijdens het uitvoeren van een ALGOL-programma bestaat de contra-stapel uit een aantal segmenten, ieder bestaande uit een even aantal, tenminste 2, aansluitende geheugenplaatsen. Deze segmenten staan zelf weer aansluitend in de contrastapel en vallen uiteen in twee groepen, die elkaar afwisselen: bezette segmenten en segmenten, behorend tot de z.g. vrije keten. Beide groepen kunnen leeg zijn. In dat geval heeft de contrastapel zijn eenvoudigste gedaante:

$$\begin{aligned} \text{plafond} &= \text{laatste plaats geheugen} \\ \lceil \text{plafond} \rceil &= 0 \end{aligned}$$

Lees voor rechte haken: de inhoud van.
De vrije keten bestaat echter gewoonlijk uit een of meer schakels van ongelijke lengte, met twee administratieve plaatsen bovenaan.

$$\begin{aligned} \lceil S_n \rceil &= S_{n+1} \\ \lceil S_n - 1 \rceil &= \text{aantal adressen in } n^{de} \text{ schakel} \end{aligned}$$

Aan de uiteinden geldt:

$$\begin{aligned} S_0 &= \text{laatste plaats geheugen} \\ S_1 &= \text{hoogste plaats van laatste (= onderste) schakel} \\ \lceil \text{plafond} \rceil &= \lceil S_{1+1} \rceil = 0, \text{ ten teken dat de keten eindigt.} \end{aligned}$$

De vrije keten kan tengevolge van deze opzet op elk moment van boven naar beneden worden afgetast. We zorgen, dat altijd geldt:

$$S_{n+1} < S_n$$

met andere woorden: de keten is gestrekt. Bij het vrijkomen van bezette segmenten (altijd minstens 2 aansluitende woorden) worden deze niet zonder meer aan het eind van de vrije keten vastgekoppeld, doch zodanig ingelast, dat de

keten gestrekt blijft. Indien het nieuwe stuk fysisch bij een of twee vrije schakels blijkt aan te sluiten, wordt het daarmee verenigd.
De ruimte boven het plafond is altijd nuttig bezet. Komt hiervan een stuk, grenzend aan het plafond, vrij, dan wordt dit laatste opgehoogd. De bedoeling is duidelijk: zo min mogelijk schakels in de vrije keten, het plafond zo hoog als de nuttig bezette ruimte toelaat.

2.4 Declaratie van een gewoon array; sleutels en portalen

Het beginsel, dat de elementen van een array opvolgend in het geheugen staan, laten we los en daarmee de bereikbaarheid van elk element via een productsom met in beginsel evenveel termen als dimensies. Dit zal wat extra ruimte kosten voor de meerdimensionale arrays, waartegenover tijdwinst komt te staan.
De eerste dimensie van een arraydeclaratie krijgt een uitzonderingspositie, de grenzen van deze dimensie worden per definitie geacht de lengte der rijen te bepalen en deze rijen worden wel aansluitend geborgen.
Voor elke rij van een matrix of tweedimensionaal array is nu een indexregister nodig, bevattend het geëxtrapoleerde nulpunt van die rij, d.i. het geheugenadres voor het (misschien denkbeeldige) element met kolomindex 0 van die rij. De vector indexregisters heeft de lengte van een kolom en is zelf weer aansluitend geborgen. Het is duidelijk, dat dit systeem van onderbrengen zich ook voor aantal dimensies groter dan 2 laat generaliseren en dat elk element snel bereikbaar zal zijn via één of meer vectoren indexregisters en wel door middel van één of meer optellingen, afgewisseld met opdrachten van het type 2B0 X 0B. Dit is een machinekodeopdracht, die de inhoud van B-register of adresmodificatieregister vervangt door "de inhoud van zijn inhoud" of m.a.w. die de inhoud van het B-register opvat als een

fysisch machine-adres en de inhoud van het daardoor aangewezen machine-woord in het B-register plaatst. Ook hier valt de taak van het opsporen van het fysisch adres van een geïndiceerde variabele toe aan de subroutine INDEXER.

Na declaratie van een array zijn ingevuld:

a. de sleutel. Sleutels zijn twee-woords- eenheden in een blokcel in de gewone stapel. Ze worden op de zelfde wijze geadresseerd als variabelen door middel van een systeem van dynamische adressering, dat hier bekend wordt ondersteld. Zie [1]. De onderscheiding tussen deel II en III van de blokcel kan komen te vervallen. De totale lengte van deze stukken samen kan nu al tijdens vertaling worden vastgesteld en dit heeft o.m. een vereenvoudiging van het koppelingsstuk tot gevolg, die hier niet nader zal worden uitgewerkt (de z.g. WW of werkruimtewijzer, nodig voor de "sprong uit de function designator" kan vervallen). Wat de inhoud van de sleutel betreft, geldt:

[sleutel] = portaal, d.i. een plaats in de contra-stapel

[sleutel +1] = oudere sleutel.

De sleutels vormen zodoende een keten, de sleutelketen, die uitsluitend van nieuw naar oud kan worden doorlopen. TSK (top sleutelketen), een fundamentele toestandsgrootheid van het programma, geeft te allen tijde toegang tot die keten.

b. het portaal. Dit is een rijtje van 2 × aantal dimensies +1 woorden in de contra-stapel.

[portaal] = geëxtrapoleerd nulpunt van elementvector of van vector indexregisters van laatste dimensie

[portaal +1] = ondergrens eerste dimensie

[portaal +2] = bovengrens " "

[portaal +3] = ondergrens tweede "

[portaal +4] = bovengrens " "

enz. enz.

De 2^e en volgende portaalwoorden dienen uitsluitend om de door het array bezette ruimte - samen met de inhoud van de indexregisters - vast te leggen. Desnoods kan daarom, als dit practischer blijkt, de bovengrens telkens worden vervangen door de lengte van een dimensie. Het laatste portaalwoord bergen we negatief, om het einde van het portaal te markeren. Straks zal blijken, dat de eerste bits van [portaal] nog een soort-specificatie moeten bevatten, die voor arrays overigens 0 kan zijn.
c. de eventuele vectoren indexregisters.

Bij declaratie van een gewoon array wordt voor portaal, vectoren van indexregisters en vectoren van elementen de vrije keten van boven naar beneden afgezocht naar geschikte vrije slierten van voldoende lengte. Deze voldoende lengte is steeds naar boven afgerond op een even aantal woorden, om te bereiken, dat, wordt er iets vrijgegeven, dit altijd in de vrije keten kan worden ingelast, die minstens twee woorden per schakel eist. Vinden we niet voor alle te bergen vectoren geschikte vrije slierten, dan moet het plafond uiteraard worden verlaagd.

2.5 Declaratie van een own array

De inhoud van het own-begrip is in het officiele ALGOL 60-rapport niet voldoende scherp gedefinieerd, in het bijzonder niet voor recursieve procedures. Eenvoudshalve beperken we ons hier tot de z.g. statische own-interpretatie, die er op neerkomt, dat er van elke gedeclareerde own grootheid in het programma te allen tijde slechts een exemplaar zal bestaan.
Deze duiding van het own-begrip zal van die in MC I-ALGOL (de vertaler van Dijkstra en Zonneveld) slechts verschillen door de toelating van dynamische grenzen voor own arrays (in MC II-ALGOL is overigens met een geheel andere uitleg geëxperimenteerd).
De vertaler kan de own grootheden nu zonder meer toerekenen

aan de buitenste blokcel en kan overigens volstaan met ergens een lijstje dimensies van own arrays af te leveren. De operatie START, het begin van elk ALGOL 60-programma, heeft aan dit lijstje genoeg om de sleutels van de own arrays vast in te vullen en hun portalen te reserveren. Deze portalen zijn leeg en zullen uitsluitend nullen bevatten, waarbij weer de lengte van het portaal gemarkeerd dient te zijn.

Bij het declareren van een own array (de declarerende opdracht staat natuurlijk in het algemeen niet in het buitenste blok) moet nu per vector, hetzij van indexregisters, hetzij van elementen, worden nagegaan of deze misschien geheel of gedeeltelijk in het doorsnee-array valt. Uiteraard zal het doorsnee-array leeg kunnen zijn en dit bij de eerste declaratie ook steeds zijn. Is een vector uit het doorsnee-array niet leeg, dan is in beginsel fysisch transport van de elementen nodig. Krimpt een vector , dan kan iets aan de vrije keten worden toegevoegd . Dijt een vector uit, dan moet omgekeerd aan de vrije keten ruimte worden gevraagd. Het lijkt dan gewenst, deze ruimte weer van boven af te gaan zoeken, waardoor misschien het plafond kan stijgen.

2.6 Blokverlating

Bij blokverlating (hetzij aan het eind van een blok, hetzij bij een go to statement naar een globale label) wordt de sleutelketting van boven naar beneden afgelopen. De arrays, waarvan we daarbij de sleutels ontmoeten, worden vrijgegeven en in de vrije keten opgenomen. Dit proces eindigt, zodra TSK komt te wijzen naar een plaats onder de nieuwe PW (zie 2.1).
Uiteraard kan de vertaler veelal reeds uitmaken of een blok al dan niet arrays bevat en of het dus nodig is bij blokverlating deze maatregelen te nemen.

Is een programma "normaal" in die zin, dat het geen echte dynamische own arrays - of andere, verderop te bespreken malversaties - bevat, dan zal de array-declarerende opdracht steeds tot de conclusie komen, dat de vrije keten leeg is en dat het te declareren array met portaal aansluitend kan worden geborgen. Aangezien de portaalgegevens op één na uitsluitend dienen voor het vrijgeven, kan nu met minder gegevens worden volstaan en het vrijgeefmechanisme een eenvoudiger weg inslaan.

2.7 De ophaaldienst

Er bestaat natuurlijk een reëel gevaar, dat de twee stapels in elkaar zullen groeien. Daarom onderzoeken we bij blokintroducties (misschien alleen bij sommige typen daarvan) en op elk tijdstip, waarop plafondverlaging nodig blijkt, of er na de corresponderende stapeluitbreidingen nog 100 plaatsen over zijn tussen de top van de gewone stapel en het plafond. Zo niet, dan moet een ophaaldienst in werking treden, die de vrije keten tot een enkele, lege schakel gaat terugbrengen en daardoor ruimte winnen tussen de stapels. Het is echter duidelijk, dat in een "normaal" programma de ophaaldienst nooit enig effect sorteren zal. Komt hij onverhoopt toch in actie, dan zal het doorgaans al te laat zijn. De ophaaldienst zelf zou bij het voltooid zijn van zijn werk kunnen eisen, dat minstens 20 of 40 plaatsen vrij zijn en anders stoppen.

Bij de organisatie van de ophaaldienst kan men de voorkeur geven aan een plaatsbesparende tijdrovende versie of omgekeerd.

De eerste eenvoudige versie neemt van de vrije keten de bovenste vrije schakel in ogenschouw en doorloopt nu de sleutelketting, alle bijbehorende portalen indexregisters en elementvectoren afzoekend naar het de vrije schakel van onderen belendende stuk. Dit wordt dan zo ver mogelijk naar boven opgeschoven, de beschouwde vrije schakel is geliquideerd en

we beginnen aan de volgende.
Ten koste van wat meer programmeerruimte kan men het aantal cycli over de sleutelketting belangrijk beperken, door elke slëert, die de zoekcyclus tegenkomt en die onder de vrije schakel blijkt te liggen, daar zo mogelijk in onder te brengen.

Overzien we het voorafgaande en vergelijken we dit met de methoden van MC I-ALGOL en MC II-ALGOL, dan blijkt:

a. het rekenen met array-elementen gaat vlugger voor aantal dimensies > 1, ten koste van wat extra plaatsruimte;
b. in programma's met uitsluitend gewone arrays gaat het declareren en vrijgeven vrij vlot;
c. het optreden van own arrays maakt declaratie en blokverlating wat omslachtiger.
d. de ophaaldienst maakt geen moeizamer indruk, dan het obligate schuiven en reorganiseren bij own array declaraties in MC II-ALGOL en komt bovendien slechts zelden in actie.

3.

3.1 <u>De wenselijkheid van uitbreiding van ALGOL 60; het begrip valuta;</u>

Op één punt is de noodzaak tot uitbreiding van de taal evident en onomstreden: ALGOL 60 kent geen in- en uitvoer. De daarvoor noodzakelijke voorzieningen zijn dan ook voor de verschillende machinetypen op uiteenlopende wijze gekozen.
Andere overwegingen nopens uitbreiding zijn:

a. men kan in ALGOL 60 niet of zeer moeilijk aanwijzingen geven omtrent de nauwkeurigheid, waarmee de elementaire operaties moeten worden uitgevoerd;

b. operaties op complexe getallen, driehoeks matrices, matrices, die in hoofdzaak uit nullen bestaan, netwerken en andere structuren eisen in ALGOL 60 onevenredig veel tijd of plaats;

c. overziet men talen, die toegespitst zijn op de behandeling van speciale objecten (LISP, COMIT enz.) dan blijken deze gemeenschappelijke trekken met elkaar of met ALGOL te vertonen, zoals statements, conditionele expressies, functies, declaraties enz. De oorzaak is natuurlijk, dat deze trekken voortkomen niet uit de beschouwde objecten, maar uit de wijze waarop het menselijk brein met objecten überhaupt omspringt. Maar wat ligt dan alleen al uit een oogpunt van arbeidsbesparing meer voor de hand, dan de notaties voor al deze gelijksoortige activiteiten te normaliseren en alles in één taal onder te brengen, met één vertaler, één stel service-programma's enz.

d. machine - afhankelijke autocodes - hoewel doorgaans minder machtig dan ALGOL 60 - zijn vaak beter geïntegreerd in het overige software pakket van een machine en hebben daardoor vaak deel aan de grotere flexibiliteit van zo'n software pakket. Het zou prettig zijn dit overige software pakket direct als uitbreidings-pakket van een ALGOL-systeem te kunnen concipieren.

Het heeft uiteraard niet ontbroken aan pogingen aan een deel van deze noden of wensen tegemoet te komen. Maar deze pogingen laboreren aan verschillende kwalen. Ofwel zij zijn niet voldoende algemeen, ofwel ze doen afbreuk aan de bereikte mate van internationale uitwisselbaarheid door invoering van onnodige eigen hebbelijkheden, ofwel ze moeten toch nog een belangrijk stuk interne logica expliciet in ALGOL 60 uitdrukken en de gebruiker met allerlei strikt genomen overbodige taken belasten. We zullen precies moeten nagaan, wat hieraan te doen valt. Daarbij bekijken we de uitbreidingsbehoeften eerst van de semantische kant, daarna zoeken we de gewonnen begrippen syntactisch onder te brengen in ALGOL 60, en tenslotte bezien we de technische aspecten van de zaak in het vertaalsysteem.

ALGOL 60 kent vier soorten waarden, reële en gehele getallen, Booleans en labels.

Het valt daarbij direct op, dat de labels stiefmoederlijk behandeld zijn: men kan geen variabelen voor labels declareren, wel specificeren. Er is dus slechts één vorm van waarde-toekenning voor labels, n.l. aan value formele variabelen (zie 4.7.3 van [5]). Voor vertaler - schrijvers - tussen haakjes - een hoogst ondankbare zaak; men is verplicht het apparaat te scheppen, dat waarde-toekenning van labels aan variabelen mogelijk maakt, doch men weet voorlopig dit apparaat nagenoeg niet te zullen gebruiken.

Nog beperkter is het gebruik van strings. Dit zijn geen waarden; er zijn wel string variabelen, doch uitsluitend bij naam geroepen formelen. Deze beperking is veel hinderlijker omdat strings echte objecten van de taal zijn, waarop men in 2.6.3 en 4.7.5.1 van [5] als het ware uitgenodigd wordt allerlei operaties te definieren.

Sommen we nu op, welke lotgevallen aan de logische of getal-waarden kunnen overkomen:

a. Ze kunnen uitdrukkelijk worden vermeld in de tekst van een programma.

b. Nieuwe waarden kunnen tot stand komen door operaties. Hun bestaan is dan gewoonlijk ephemeer, want:
c. Operaties maken in het algemeen een eind aan het bestaan der operanden. Na berekening van bijv. <E1>*<E2> zijn de waarden van de expressies E1 en E2 in beginsel vergeten.
d. Functies zijn een bijzonder geval van operaties. Ook de nieuwe waarde, die door berekening van een functie tot stand komt is ephemeer en de operanden zijn verloren.
e. Toekenning van een waarde aan een passend gedeclareerde variabele is het enige middel om de levensduur van een waarde te verlengen.
f. De toekenning wordt te niet gedaan door toekenning van een andere waarde aan de zelfde variabele of
d. door verlating van het blok waarin de variabele is gedeclareerd.

Verder valt op te merken, dat we de type-declaratie logisch niet nodig hebben. Op elk moment van de berekening is uit de historie van een waarde af te leiden wat zijn type is. In MC I-ALGOL heeft dit ook zijn sporen nagelaten, doordat zoals we weten het type van een procedure daarin volstrekt verwaarloosd wordt. Men kan zeggen dat een declaratie als

variable ,b,c

logisch voldoende zou zijn (als we bereid zijn de hier en daar nodige afrondingen expliciet te commanderen), maar dat de extra informatie die de type-declaratoren geven de vertalerschrijver in staat stelt compactheid of snelheid te winnen.

We stellen nu, dat aan nagenoeg alle uitbreidings-behoeften van ALGOL 60, die tot dusver bekend zijn, kan worden voldaan, door invoering van nieuwe soorten objecten, voortaan aan te duiden als "vreemde valuta" of kortweg "valuta". Het enige,volstrekt legitieme,uitbreidingsmechanisme, dat we daarvoor nodig hebben, is dat van de procedure met een pro-

cedure lichaam, geschreven in machine code (MCP); zie 4.7.8 en 5.4.6 van [5]. De valuta is ingedeeld in een aantal soorten, dat in beginsel onbeperkt kan toenemen. De fysische representatie van een valuta soort is volkomen vrij, doch voor iedere soort welgedefinieerd, zij het niet constant van lengte. Voor iedere valutasoort zijn een aantal operaties toelaatbaar, die nieuwe valuta, eventueel van andere soort kunnen scheppen. Ook operaties tussen verschillende valuta-soorten zijn denkbaar. Eenmaal de operaties in MCP-vorm gegeven, moet de gebruiker volledige vrijheid hebben om toekenningsuitspraken, conditionele expressies, recursieve en andere procedures van de ene soort naar de andere (of naar de zelfde), in ALGOL 60 op te schrijven.
Alle onder a - d beschreven lotgevallen van waarden dienen voor de valuta in beginsel analoog te worden gerealiseerd.

Simpele voorbeelden van valuta-MCP's:

read string	een function designator, die een string leest.
read list	een function designator, die een "list" à la Mc Carthy invoert.
cons (a,b)	de bekende LISP-operatie.
mult (a,b)	een function designator, die een product van a priori onbekende lengte aflevert,nl. zo lang als nodig om het resultaat exact voor te stellen, en die met integers of met getallen van zijn eigen soort gevoed mag worden.
invmat (a)	die een in ..iehoeksvorm gegeven symmetrische matrix inverteert.

Dit alles ligt zeer voor de hand, maar ook van ambitieuzer voornemens komt nu de uitvoering binnen ons bereik te liggen: inhomogene arrays, arrays van procedures, procedures, die uit allerlei gegevens nieuwe procedures uitrekenen en

deze uitvoeren of voor uitvoering beschikbaar stellen enz. enz.

3.2 Orthodoxe uitbreiding van ALGOL 60

Gaan we na, hoe de geschetste wenselijkheden correct in ALGOL 60 kunnen worden ingepast.
Eerst moeten er valuta-scheppende MCP's komen, vergelijkbaar met read of met het gewoon voorkomen van constanten in een expressie.
De vrijheid, die MC I-ALGOL en MC II-ALGOL met "read" namen, nl. dat het type onbeslist is, moeten we ook opeisen. Sterker: we zijn bereid van al deze MCP's de heading in de ALGOL-tekst op te nemen met de aanhef real procedure als concessie aan de syntaxis, maar we eisen het recht op, daarbij aan iets geheel anders te denken. Anders gezegd: het reële getal, dat deze MCP's beloven af te leveren, zal een codewoord zijn voor een valuta-grootheid. Wie eenmaal aanvaardt, dat we op deze wijze de in het officiele rapport gelegde band tussen de symbolen van een ALGOL-tekst en hun betekenis tot aan de grenzen van zijn elasticiteit uitrekken, zal er ook weinig bezwaar tegen maken, dat:

1. gewone en geïndiceerde reële variabelen gebruikt worden voor toekenningsuitspraken in valuta, bijv. ;a: = mult (a,b);
   Doen we daarna a:=5, dan moet de eerst toegekende valuta uit het geheugen verdwijnen.
2. conditionele expressies in - voor het oog - reële variabelen in feite valuta-expressies kunnen voorstellen.
3. Bij de toekenning aan een reële procedure-naam in feite een valuta - function - designator zijn waarde kan krijgen.

3.3 Andere uitbreidingsmogelijkheden van ALGOL 60.

In het vorenstaande werd slechts de lijn doorgetrokken, die we bij de erkend noodzakelijke uitbreiding van ALGOL 60 met in- en uitvoerprocedures volgden; daartoe werden nl. machinecode-procedures geprogrammeerd, waarmee een begeleidend commentaar aan ALGOL onbekende begrippen als: eerstvolgend getal op band, wagen terug op schrijfmachine enz. associeerde. Toch kan ik me voorstellen, dat het iemand tegen de borst blijft stuiten reële variabelen als onderdak voor codewoorden te misbruiken.

Een andere overweging is deze: in plaats van er naar te streven, dat voor een bepaalde machine een taal wordt gerealiseerd, die een echte deelverzameling is van ALGOL, kan men zich ook tot doel stellen een systeem te maken, dat ALGOL 60 tot echte deelverzameling heeft. Het eerste doel is principieel onbereikbaar omdat ALGOL geen in- en uitvoer kent en practisch steeds minder nastrevenswaard, naarmate de MCP-bibliotheken uit elkaar groeien.Bereikbaarstelling van het tweede doel doet, voor de gebruiker, die zich weet te beheersen, dan nog minder afbreuk aan de internationale uitwisselbaarheid en verstaanbaarheid dan die van het eerste.

Tenslotte is duidelijk, dat wie bereid is enkele kleine ingrepen in ALGOL 60 toe te laten (bijv. een extra type-declarator, of het toelaten van string als declarator, en als type van arrays en procedures) taken aan de vertaler kan delegeren, die anders dynamisch tot een goed eind moeten worden gebracht. Daarmee kan iets aan efficiency worden gewonnen en de ALGOL-tekst wordt een duidelijker afspiegeling van zijn inhoud.
Bij de hierna volgende uiteenzetting van de organisatie in de machine, zullen we evenwel van het orthodoxe uitbreidingssysteem van 3.2 uitgaan.

4.

4.1 Valuta-scheppende MCP's (machine code procedures)

Alle valuta-scheppende MCP's zijn als reële procedures gedeclareerd. De meeste werken analoog aan "read". Worden ze als function designator geactiveerd, dan zoeken ze in de contrastapel onderdak voor een portaal en een of meer slierten, waarin de nieuwe entiteit binair is gerepresenteerd. Het nulde portaalwoord bevat ook de soortspecificatie van de valuta. In de gewone stapel wordt een waarde afgeleverd op een plaats, die na beëindiging van MCP de bovenste rekencel zal zijn. Deze afgeleverde waarde is geen echt reëel getal maar weer een sleutel, dus een tweewoord-eenheid, die het portaal bevat en verder terwille van de gemakkelijke herkenbaarheid liefst waarneembaar verschillend dient te zijn van gewone reële getallen. Aan onze representatie van reële getallen legt dit de eis op een overtollig bit te bevatten. De sleutel wordt evenals de array-sleutels in de sleutelketen opgenomen en wel aan de top.

Wordt een valuta-scheppende MCP niet als function designator, doch als statement geactiveerd, dan heeft hij uitsluitend het nevengevolg van het opsteppen van een invoermachinisme. Er wordt dan geen rekencel met sleutel als inhoud aan de stapel toegevoegd en in de sleutelketen ingelast en natuurlijk ook in de contrastapel geen ruimte bezet. Duidelijk is, dat deze bijzondere wijze van aanroepen of in de stapel (zoals in MC I ALGOL) of alleen in het objectprogramma (zoals in MC II ALGOL) moet zijn uitgedrukt. Er is ook ruimte voor het scheppen van een valuta-grootheid door hem in het ALGOL-programma eenvoudig te noemen, zoals men een constante in een expressie neerschrijft. Dit moet dan gebeuren in de vorm van een string, als actuele parameter meegegeven aan een soortgebonden MCP, bijv.

Boolmat (‹ 010110111 ›)

waarmee men een ongedeclareerde eendimensionale Boolean

matrix zcu kunnen introduceren.
Dit laatste wijst er op, dat men het valuta-begrip als een nadere uitwerking van het in ALGOL gegeven string-begrip kan voorstellen.

4.2 Valuta-verwerkende MCP's

De rol die in de normale arithmetiek door de rekenkundige en relationele operatoren wordt gespeeld, moet geheel door dit soort MCP's worden overgenomen. Ook de rol van "print" en andere uitvoerprocedures.
Alle typen zijn dan cok toelaatbaar: de real, integer, of Boolean procedures alsmede de statement - procedure. Uit het begeleidende commentaar zal aan de programmeur-gebruiker moeten blijken, dat een deel van de real procedures als valuta-procedure bedoeld is.
Alle valuta-verwerkende MCP's proberen in beginsel hun operandwaarden te vernietigen. Dit lukt ze altijd, tenzij de waarden ooit aan variabelen waren toegekend en deze toekenning nog van kracht is. Dat is dan in het portaal bespeurbaar, zoals straks zal blijken. Vernietigen betekent hier vrijgeven uit de contrastapel en aan de vrije keten toevoegen onder ontkoppeling van de sleutel.

4.3 Algemene valuta procedures

Naast de valuta-MCP's is er in het systeem plaats voor door gebruikers vrijelijk in ALGOL 60 te declareren valuta-verwerkende (en desgewenst afleverende) procedures in alle bekende soorten. Vindt in het lichaam van zo'n procedure geen waardetoekenning aan globalen of aan de procedurenaam plaats, dan moeten na voltooiing van de procedure alle operanden en ook de tussentijds gevormde valuta weer vernietigd zijn. Dit gaat volkomen automatisch goed. Alleen moeten we nu een bijzondere eis stellen aan de door de vertaler in het objectprogramma gegenereerde subroutine STORE REAL PROCEDURE

VALUE, die verschijnt naar aanleiding van de waardetoekenning aan de procedurenaam in de ALGOL-tekst.
In MC I ALGOL ging deze subroutine na in de stapel of de procedure in kwestie als statement, dan wel als functie was aangeroepen. In het eerste geval ging de waardetoekenning niet door. Welnu, in dit geval moet weer geprobeerd worden de toe te kennen waarde in de contrastapel te vernietigen. In MC II ALGOL gaat dit anders, maar analoge maatregelen zijn ook in de daar gevolgde methode inpasbaar. Is in STORE REAL PROCEDURE VALUE deze afdeling ingebouwd, dan kunnen ook de valuta-scheppende of- afleverende MCP's met vrucht van deze operatie gebruik maken.

4.4 De toekenningsuitspraak voor valuta

We komen hier aan de eerste en enige plaats in het systeem waar van de numeriek geïnteresseerde gebruiker ten behoeve van de generalisatie een noemenswaard, zij het gering, offer in rekensnelheid zal worden gevraagd. Uit het verloop van deze beschrijving zal verder vanzelf blijken, hoe door een geringe concessie op het stuk van de syntactische orthodoxie dit onderdeel aanmerkelijk kan worden opgepoetst en het snelheidsoffer geheel kan komen te vervallen.
Beschouwen we de toekenning:

; real a ; a: = readstring;

De toe te kennen waarde is in de bovenste rekencel van de stapel vertegenwoordigd door een sleutel. De in het object-programma als vertaling van ":=" dienende subroutine STORE zal dit moeten opmerken op grond van de onderscheidbaarheid van reële getallen enerzijds en sleutels anderzijds, of, als men daaraan de voorkeur geeft, op grond van het in de sleutelketen al dan niet bereikbaar zijn.
Ook is het zeer wel mogelijk, dat de oude waarde van a een sleutel is van in het algemeen een ander valuta-portaal.
Mede met het oog op constructies als

a: = a

en a: = a: = readstring

dient STORE als volgt te werk te gaan (we nemen aan dat bij blokintroductie de plaatsen voor de reële variabelen zijn schoongeveegd, d.w.z. dat hun een voorlopige waarde is toegekend, die het onmogelijk maakt ze met sleutels te verwisselen). Een deel van deze beschrijving geldt ook voor STORE REAL PROCEDURE VALUE.
Kijk eerst naar oude waarde linkerlid. Is dit een sleutel, noteer deze dan op een vaste plaats.
Is rechterlid sleutel (d.w.z. ligt er een sleutel op de stapel, in de bovenste rekencel)? Zo ja, schrijf hem dan op de variabele gereserveerde plaats en las hem aldaar in de sleutelketting zò, dat deze gestrekt blijft. De inlassing op de oude plaats kan nu vervallen, indien het een "ephemere" sleutel betrof, d.w.z. een sleutel, die niet uit een blokcel in een rekencel was gecopieerd, doch die slechts in die rekencel bestond en aldaar in de sleutelketting was ingelast.
Verhoog daarbij in het portaal de grootheid tau, die bij het scheppen van de string op nul gesteld is en die voor alle valuta-soorten in een vaste portaalplaats staat, met één. De bedoeling is, dat tau de van kracht zijnde toekenningen turft. Als een sleutel van een linkerlid genoteerd was, laag dan in deszelfs portaal de tau af en geef, als de tau nul wordt, de gehele valuta vrij; de sleutel wordt dan uit de sleutelketen verwijderd.

Een kleine moeilijkheid doet zich voor als de variabele in het linkerlid geïndiceerd blijkt te zijn. Wie dit niet wil verbieden, zal verplicht zijn de elementvectoren bij declaratie van een real array schoon vegen.
Wordt nu een array-element gebruikt voor toekenning van valuta-waarden, dan moet het denkbeeld van de gestrekte sleu-

telketen worden losgelaten. De sleutel wordt op de plaats van de geïndiceerde variabele geschreven en de inlassing geschiedt "onder" de array-sleutel. We bereiken dan, dat de aldus gebonden valuta bij blokverlating slechts dan door tau-aflaging de kans loopt vrijgegeven te worden, als juist het blok verlaten wordt, waarin het array is gedeclareerd.

4.5. De rol van tau in lijstvormige valuta

Het is ongetwijfeld nog te vroeg om nu al te voorspellen welke valuta-soorten te eniger tijd actueel kunnen worden.
Niettemin laten zich onder de mogelijke wijzen van machine-representatie van valuta nu reeds minstens twee principieel verschillende vormen onderscheiden. Wie een vermenigvuldiging heeft uitgevoerd, zeg 5 × 7, en alleen in het resultaat geïnteresseerd is, kan volstaan met 35 te onthouden.
De vermelding, dat deze waarde in casu ontstaan is uit 5 en 7 doet dan niet veel meer terzake en de interne representatie van deze laatste getallen kan zonder bezwaar worden uitgeveegd.
Het kan - zij het minder doeltreffend - ook anders. We zouden een integer kunnen voorstellen door een lijstje adressen van priemen met hun multipliciteiten. In dat geval zouden we er zorg voor moeten dragen geen priemgetal te overschrijven zolang er nog een onthouden integer voor zijn representatie op hem rekent.
Ongeveer op deze wijze stellen wij ons de representatie voor van de uit LISP en misschien nog uit andere verwante talen over te nemen soorten valuta. Ontstaat daarbij bijv. een lijst uit twee andere, dan worden de twee bestanddelen niet gecopieerd; het is voldoende in het nieuwe portaal een administratief verband tussen de twee samenstellende delen te leggen. Bovendien moet de tau in hun portalen met één worden verhoogd. Vrijgeven van een lijst betekent dan in eerste instantie slechts vrijgeven van zijn portaal en tau-aflaging in de portalen van zijn componenten, met natuurlijk herhaling

van dit proces waar daarbij een tau nul wordt en met een speciale behandeling van primitieve componenten. De levensvatbaarheid van dit denkbeeld is door proefnemingen van Van de Laarschot en schrijver dezes met de X1-vertaler van Dijkstra en Zonneveld voldoende aangetoond.

4.6 Het toevoegen van nieuwe valuta-soorten

Wie een nieuwe valuta-soort aan het hier geschetste ALGOL-systeem wil toevoegen zal niet alleen de valuta-scheppende en verwerkende MCP's dienen te ontwerpen, doch ook faciliteiten moeten inbouwen in de ophaaldienst, in de blokverlating en in de beide bovenvermelde STORE-operaties.
Blokverlating en STORE moeten een gemeenschappelijke subroutine aanroepen: het voorwaardelijk vrijgeven. Deze zal een strooisprong over de valuta-soorten inhouden en daardoor verwezen worden naar soortgebonden vrijgeef-mechanismen. Voor de nulde soort (die in STORE niet voor zal komen), de soort van de arrays, is het vrijgeven uiteraard onvoorwaardelijk d.i. niet van tau afhankelijk. In de ophaaldienst zal een dergelijke strooisprong voor moeten komen. Op deze plaatsen moet het systeem open zijn en dienen voorzieningen voor uitbreiding te zijn getroffen. De valuta-afleverende MCP's zullen de ophaaldienst voorwaardelijk aanroepen. Enige omzichtigheid is daarbij wel op zijn plaats. Immers, deze valuta-afleverende MCP's kunnen bezig zijn aan het opbouwen van structuren van in beginsel onbeperkte complexiteit. Voor het administreren daarvan zullen ze dus zeker de gewone stapel nodig hebben. Maar ook de ophaaldienst (die gewoonlijk begint als er nog minstens 100 plaatsen boven de stapel vrij zijn) moet zoeken in structuren van onbeperkte complexiteit. Men mag hopen, dat hij daarbij eerst behoefte aan het volle aantal vrije plaatsen werkruimte zal gaan gevoelen, nadat hij gelegenheid heeft gevonden de contrastapel wat in te korten, door verplaatsing van onderste slierten naar boven. Lukt dit niet, dan moet hij maar overgaan naar de volgende sleutel.

I n h o u d