M. Fokkinga, 12/11/74.

Notities naar aanleiding van
Notitie over behandeling van het parametermechanisme in sygl. ALGOL 60 (CB68)

Hieronder zal ik enkele opmerkingen, bezwaren en ideeen uiteenzetten die me bij het lezen van CB68 te binnen schoten bij betreffen de doelstellingen van CB68, kritiek van algemene aard en kritiek en opmerkingen van technische aard. Een toelichting volgt daarna.)

Doelstellingen van CB68.

Deze zijn me niet geheel duidelijk. Poogt dat stuk een inzicht te geven in een redelijk goed parametermechanisme dat - bijna helemaal - verwezelijkt is in ALGOL 60, of beoogt het inzicht te geven in het parametermechanisme zoals dat in ALGOL 60 gedefinieerd is, of stelt het zich ten doel informatie te verschaffen over de bewegredenen waarmee sommige facetten van ALGOL 60's par.mech. wel en andere niet behandeld zijn (opdat de wetenschappelijke leiders de stof beter kunnen presenteren?) Of is het een schets van 's schrijvers bezwaren tegen ALGOL 60's par.mech. die te herinnering kunnen worden aangenomen?

Kritiek van algemene aard op CB68

1. Onduidelijk (pag 0, r. -22 t/m -18) wordt nogtevreel de indruk gewekt dat name en valueparameters gelijksoortig zijn (zeker omdat ze beide iets met de buitenwereld te maken hebben) en te weinig nadruk wordt gelegd op het belangrijke verschil, nl. dat value "parameters" declaraties van nieuwe locale variabelen zijn en nameparameters benamingen van bij aanroep te verwachten grootheden.
2. De wens voor het bestaan van constanten is een zaak die geheel los staat van procedures. De verwarring waarover in het begin van CB68 wordt gesproken (pag 0, r. 1 e.v.), zou zeker bij studenten ontstaan die CB68 lezen. Want daarin

licht een verwezenlijking nagestreefd te worden van het concept van constanten d.m.v. het parametermechanisme! Foei!

3. Het onderscheid tussen in-, uit- en in/uitgangsparameters is mijns inziens een verheerd onderscheid.
4. De voorgestelde oplossingen dragen mijns inziens teveel een ad hoc karakter.

## Kritiek en opmerkingen van technische aard

5(a) De voorgestelde taaluitbreidingen (val, var) heur ik af.
(b) Ik zie een nieuw voorstel,
(c) waarmee en voor arrays geen moeilijkheden zijn
(d) en dat gezonde bewijsregels heeft.
6. Over expression by name heb ik nog een tweetal opmerkingen.

## Toelichting

ad1 Gelijksoortigheid versus verschil van name en value parameters.
Ik ontken niet dat het verschil niet is duidelijk gemaakt in CB 60 of CB62 (= ALGOL 68 style). Ik vind slechts dat m.i. teveel gelijksoortigheid wordt gesuggereerd. Het enige wat beide vormen van parameters met elkaar gemeen hebben is dat ze "iets met de buitenwereld hebben uit te staan". Daarmee is alles gezegd. Want name parameters zijn grootheden van de buitenwereld waaraan door de procedure geslenteld kan worden en value "parameters" zijn interne grootheden gedeclareerd in de kop van de procedure, met als enig verschil met de overige interne grootheden dat ze automatisch geïnitialiseerd worden op de bij aanroep meegegeven waarden (uit de buitenwereld). De formele value "parameters" zijn derhalve geen parameters: hun

resultaat in de vorm van een betrekking tussen verscheidene grootheden te berekenen") en niet dat de naamparameters als uitgangsgrootheden dienen ter uitbreiding van functieprocedures tot functieproc's met meervoudige waarden.
De naamparameters zijn dus in/uitgangsgrootheden en derhalve past proc wisel(x,y) wel in dat beeld.'
De behoefte aan louter uitgangsgrootheden is in CB 68 niet geschetst. Bovendien is de wel daartoe voorgestelde verwezenlijking foutief. Louter uitgangsgrootheden zullen grootheden moeten zijn die binnen de procedure gevormd worden en niet al bij aanroep bestaan. Net als bij arithmetische operatoren zullen zij impliciet opgeleverd en verwerkt moeten worden of net als bij functieprocedures zullen zij ~~als~~ via de procedurenaam ~~bosch~~ bereikbaar moeten zijn: f.value1, f.value2, ... Vergelijk met de bereikbaarheid van interne grootheden van records in Pascal, of classes in Simula.
Ter verduidelijking, alle voorbeelden van "louter uitgangsvariabelen" in CB68 en CB62 betreffen grootheden waarvan de waarde bij aanroep nog niet bepaald is en daarna wel. Dus ten aanzien van die grootheden fungeert de procedure als een meervoudige toekenning, en bij toekenningen maken we normaliter geen onderscheid tussen al wel en nog niet geïnitialiseerde variabelen!

ad 4 Ad hoc oplossingen

Ik acht het een kenmerk van goede talen dat er met een minimaal aantal concepten een maximaal aantal uitdrukkingsmogelijkheden zijn. Dit impliceert het afwijzen van nieuwe concepten in die gevallen dat met de al bestaande volstaan kan worden en het toestaan van het gebruik ervan in willekeurige context ipv bijv. in procedures alléén, en i.h.a. algemeen het nastreven van

orthogonaliteit.
De concepten die achter val steken (zoals op pag 1 van CB60) zouden niet tot procedures beperkt moeten blijven, m.a.w. niet in het kader van procedures behandeld moeten worden. De ingevoerde datatype expr en de lambda notatie is overbodig omdat met procedures hetzelfde bereikt kan worden en zelfs veel algemener.

Opmerkingen en kritiek van technische aard
Van nu af bedien ik me van Manna's notatie [.] om het effect van tekstgedeelten te beschrijven door met accenten de waarden van grootheden vóór uitvoering aan te geven.
Bijv. true {wissel(x,y)} $x=y' \wedge y=x'$.
Dit werkt veel prettiger dan het invoeren van betekenisloze $x_0$ en $y_0$ : $x=x_0 \wedge y=y_0$ {wissel(x,y)} $x=y_0 \wedge y=x_0$.

ad 5 a) Afkeuring van voorgestelde taaluitbreiding
i) Gewenste versie van proc wissel met 2 name en 2 value parameters heeft wel degelijk zin. Geef wissel maar eens de naam multiple assignation, dan staat er -bijna-:
proc multipleassignation (real var x,y ; real val $x_0$, $y_0$);
begin $x := x_0$ ; $y := y_0$ end,
met het effect
true {multiple ass.(x,y ; $x_0$, $y_0$)} $x=x_0' \wedge y=y_0'$
i.h.b. true {multiple ass.(a,b ; b,a)} $a=b' \wedge b=a'$
en ook true {multiple ass.(a,b ; a+b,2)} $a=a'+b' \wedge b=2$
ii) Ik heb groot bezwaar tegen de vorm
proc wissel2 (real val var x,y),
want wat gebeurt er bij de body
begin $x := x+y$ ; $y := x+2$ end ? Moet in de rechterleden steeds de initiele value-waarden worden genomen of alléén bij de eerste voorkomens van x en y? In beide gevallen

maar er wel in voorkomende grootheid moet ofwel als een local zijn gedeclareerd ofwel als een global zijn bekend.

De onderscheiding tussen constanten en variabelen kan, indien deze bestond in ALGOL 60 door bijv. const, ook in de global en local specificaties doorgevoerd worden:
proc vb2 (global real x, int const i ; local int const k, real const x0),
maar het gebruik van een local int const n en een global int const n is natuurlijk wel gelijkwaardig.

5c) Met het global/local concept voorzie ik geen moeilijkheden mbt array's. Global array's worden mogelijkerwijs in waarde veranderd, local array's zijn nieuw gedeclareerde met automatische initialisatie. De keus tussen beide wordt overgelaten aan de ontwerper van de algoritme, die zich moet baseren op efficientieoverwegingen of de logische probleemanalyse. De behoefte aan constante array's staat los van deze scheiding. "teller uitgangs array" C (CB 68, pag 3) is een global die hennelijk door de procedure matmul een meervoudige waardetoekenning ondergaat.

5d) Ik ben het eens met de opmerking (CB 68/pag 4, 1e alinea) dat er over het al of niet by reference meegeven nogal wat te zeggen valt.
Dit hangt nauw samen met de wenselijkheid van het nettoeffect van procedureaanroepen en de ingewikkeldheid van bewijsregels !!
De "problemen" zijn de volgende. Beschouw eens
proc vb3 (global real x, int i);
begin x := x+1 ; i := i+1 ; x := x-1 end.
Duidelijk is dat
true { vb3 (x, i) } $x = x' \wedge i = i'+1$,

maar dan is het ook wenselijk dat
true { vb3 ( A[i], i) } A[i] = A[i]' ∧ i = i'+1 ,
en dit laatste is niet waar indien "global" in ALGOL 60 als "by name" wordt vertaald.

Beschouw eens
proc vb4 ( global real x ; local int i) ;
begin x := x+1 ; i := i+1 ; x := x-1 end .
Duidelijk is dat
true { vb4 ( x ; i) } x = x' ∧ i = i'
(NB: i = i' want local int i is een andere dan bij aanroep),
maar dan is het ook wenselijk dat
true { vb4 ( A[i] ; i)} A[i] = A[i]' ∧ i = i',
en bij onzorgvuldige semantiek definitie is dit niet waar.

De gewenste nettoeffecten worden gerealiseerd (a) indien door global specificaties de identiteit van de grootheden wordt meegegeven, of anders gezegd, de overdracht "by reference" is, en (b) door geschikte hernaming van de locale, i.e. gebonden, grootheid i in i®, zó dat er geen conflicten ontstaan bij substitutie. Deze laatste regel is algemeen bekend bij substitutie processen en ook voor ALGOL's by value mechanisme gedefinieerd! Dit neem ik ook aan voor de local variabelen.
De by reference overdracht wordt voor de global en - in ALGOL 60 - by name parameters gegarandeerd indien we de volgende beperking opleggen:
(***): van de global gespecificeerde parameters mag in bij aanroep in de actuele parameters géén identifier meer dan eens voorkomen.
Immers, volgens (***) kan er alleen aan afzonderlijk als global gespecificeerde grootheden gesleuteld worden, en