4. ALGOL の動向

井上謙蔵（東大物性研）

1. AX と AY

昨年の3月に，森口先生の代理としてIFIPのWG2.1に出席したので，その報告をしなければならないわけであるけれども，その後WG 2.1の4−th meetingが9月14日から19日にかけてBaden(Austria)で開かれ，そのreport をみることができたので，両者を含め，またACMやALGOL Bulletinなどの文献にあらわれた関係論文等を含めてALGOLの動向の如きものを報告したい．

3月のTutzingにおける3−rd meetingでは

Subset ALGOLの決定

IFIP−I／Oの決定

Future ALGOLに関する討論

がテーマであつた．Subset ALGOLとIFIP−I／Oについては，C. ACMのVol. 7，No. 10(1964)[1) 2)] の626頁と628頁に結論が載せられているから，それをみていただきたい．また情報処理の昨年最終号[3)]に解説記事があるので，それも参照していただきたい．唯一つお断りしておかなければならないのはfunction designatorのside effectを禁止する条項の文章に対して日本からは代案をもつていつたところ，それはdefinitionに対するexplanationの欄にのせることになつたのであるが，今度C. ACMに出たものをみると，どうしたわけかそれが落ちている．この理由はいずれたしかめたいと思つている．

Tutzingの会議の主な関心事はfuture ALGOLの討議であつた．future ALGOLというのは，一つにはALGOL 60の拡張案であり，もう一つはもつと根本的に新しい何かである．5日間にわたる討議をへて出された結論は

AX：ALGOL 60の改善

AY：meta languageによる厳密な定義をもつnew language

を，これからの研究の課題とするということであつた．はじめからAXとAYに類する二つの言語を考えようとする空気があつた．それらはA6X，A6Yと呼ばれていたが，討論しているうちにAX，AYということになつた．まだ出発点もきまらないのであるから到着の日附けをきめるのは早すぎるというわけかも知れない．

元来，WG2.1にはALGOL 60のextensionの必要を考えていたグループと，言語の

記述のために必要なaxiomaticな要素のみを残して，他の文法的規定はすべて捨てる形でgeneralizationを行なおうとするグループがあつた．3-rd meetingではこの二つの考え方をめぐつて討論が行なわれ，結局上記の如き結論となつたわけであるが，これは必ずしもgeneralistがYでextensionalistがXを作るというわけではない．あとでのべるように結論においては，ALGOL 60の改良案としてのXに関して，かなり両者の一致点があるのである．4-th meetingではXがそなえるべき特徴について討議が行なわれ，最終的にWijngaadenが，彼のmeta-languageでかいたXの草案を，次の5-th meetingに用意してくることになつた．

2. Generalization

Generalizationのチヤンピオンは Wijngaaden[4] である．彼の考えは，プログラム用言語の文法は，その言語で書かれたプログラムに対する計算機の動作を規定するものであるが，このような規定のもつともaxiomaticなものを選びだして，それだけで計算機に要求されるすべての動作を記述しようとすることである．

一つ定義をつけ加えることは，その定義と矛盾する動作を否定することであるから，可能性をきりちぢめることにほかならない．従つて，言語の表現形式の上に，計算機に開かれてくる様々な可能性を残しておこうとするならば，この様な定義を行なうべきではない．文法などは定義すべきでない，つまり何もきめないのが一番よいということになる．しかし何もきめないと何もできないということになるから，言語のはたらきのうち公理的なものだけをきめようというのである．例えばvalueというのが，この公理の一つであつて，今あるsource programをよんでいるうちにopen string $x+y$ に対し

value $\{x+y\}$

なる動作を計算機がおこすとする．計算機の中にはこのsource programのすでに読まれ，それ以上は行きつけないところまで解釈されたtruth の集合 v が入つているとする．value $\{x+y\}$ に対する計算機の動作は，まずこの v を逆順に吟味することである．そして

value {<sum 1>+<term 1>}
= value {value<sum 1>+value<term 1>}

なるtruthを発見したとき，このtruthがvalue $\{x+y\}$ に適用できるかどうかをしらべるのが，計算機の次の動作である．すでに読まれたプログラムの部分にはALGOL 60の定義を，このgeneral language 構成要素から順々に組立てる部分があるとする．そうすると機械は，くり返し v を調べることによつて，

x in <letter>
<letter> in <identifier>

<identifier> in <simple variable>
<variable> in <primary>
<primary> in <factor>
<factor> in <term>
<term> in <sum>

を順々に発見し，はじめにみつけたtruthがvalue {x+y}に適用可能であることを知る．次に機械はvalue xを求める動作をおこす．やはりvを吟味して

$x=3$

なるtruthをみつけたとすると

value x = value 3

となつて，再びvを調べる．ところが今度は

value <name 1> = <name 1>

以外，上式に対応するものは発見できなかつた．そこで

value $x=3$

である．value yに対しても

value $y=4$

となつたとする．そこで

value {x+y} = value {3+4}

となる．再び3+4のvalueをvの中に求めて

$3+4=7$

なるtruthを発見したとする．そこで

value {x+y} = value {7}

であるが，再びvをしらべ

value {name 1} = name 1

以外に発見できなかつたとすると，最終的にx+yの値は

value {x+y} = 7

になる．もしvの中にtruthとして

value {<variable>:=<expression 1>}
= {<variable> = value <expression 1>}

があれば，これでassignment statementの評価が行なわれ，例えばz=7というようなtruthが新らたにvに加わることになる．このようなやり方で，valueとかinのごとき適当な公理的動作が与えられたならば，任意の言語を組立てつつ，そのプログラムを消化できそうに見える．尤もWijngaadenも何を基礎として採用したらよいか苦しんでいるらしく，Tutzingでの予報にもかかわらず，4－th meetingでは前進した議論が行なわれ

なかつたようである．

Wijngaadenの考えを使つて，ALGOL 60に類似の文法をもつnew languageがWirth[5]によつて記述されそのコムパイラが作られている．この言語の特徴はWijngaadenと共通の考え方であるが宣言というものをなくしてしまおうということである．そもそもidentifierはその使い方によつて性格がわかるのであるから，宣言はredundantな情報である．ただblock構造を保存するためにだけ，label以外の局所的に用いられるidentifierを

new i, k

という形で宣言する．arrayの宣言もないのであるから，subscripted variableがあらわれる毎に，あるarrayの要素がつけ加わつていくわけである．procedureの宣言もないわけであるが，その代りに，これもWijngaadenの考えであるが，stringのevaluationはopen stringとするという規則が定められている．即ち，通常のassignment statement

$z := u + v$

は，$u+v$のevaluationがなされて，その値でzの値がおきかえられるのであるがassignment statement

$v :=$ " $x+y$ "

は，open string $x+y$によつてvの値をおきかえる．従つて$z := u+v$の前に$v :=$ " $x+y$ "があれば$u+v$のevaluationはすでに評価されたvの値，即ち$x+y$を再評価することによつてその値をうる．このようなやり方で単純にして且elegantな文法をまとめあげている．

Wirthの言語では

$a := 6, 12, 23, 14$

というようなassignmentを行なうことができる．これはaが4個の要素をもつベクトルであるということである．この考を更に一般化して単純変数も，要素が一つのlistとみなして，すべての変数をlistによつておきかえるのが，Garwick[6]の考え方である．この考え方では，任意のoperationはlistに対するoperationになる．例えば

$a := b$, $a + b$

等はlist element全部のassignmentや和である．

以上でgeneralizationの方向でのnew languageの特徴づけを，おわることにするが，大変面白いことには，のちに説明するextensionの立場からのnew languageの特徴と，そうかけはなれたものでないことである．文法の単純にしてelegantな記述ができるならば，generalizationの要素にかなうのであるから，これは当然なことかも知れない．

話がYからXの方にそれてきたので，ここで簡単にYのしめくくりをやつておいて，Xの方に話を進める．

WG 2.1の4－th meetingと同じ時期に，WiennaでIFIP主催のFormal Language Descriptive Languageという名称のWorking Conferenceがあつた．その会議の後におけるWG 2.1の評価には，AYを定めるために会議があまり役に立たなかつたという意見がかなり存在し，WG 2.1のテーマは時期尚早であるという意見もかなりあつた．機械が処理しうる形にmeta languageを定めるということは，それによつて記述される言語に不明確さを残さない点で大きな意味があるであろうが，機械の発展を考えると，axiomatic な形にそれを定着させることが，現段階で果して可能なのであろうかという気もする．Wijngaadenのやり方に対して，problem oriented language というものは機械語の繁雑さからのがれるのが一つの目的であるけれども，これでは同じ繁雑さに戻つてしまうという批判もある．

3. Extension

Extensionalistの重要な代表とみられるロシアのErshov[7]たちの考え方を次に説明する．

ロシアではALGOL 60が固まる以前に，PPと称する自動プログラム用言語があつた．しかし，ALGOL 60の発表以後，これを支持して，Input Languageという名で，ALGOL 60の拡張案を出し，かつそのコムパイラALPHA systemが作製された．ALGOL 60から拡張されている主な点を，次に述べよう．

まずarrayからはじめる．宣言

$$\underline{\text{array}}\ A[1:2, 1:3, 1:4] - \underline{\text{array}}\ 4\times 10$$

はarray Aの各要素が更に4×10のarrayであることを定義する．このsub－arrayに名前をつけたいときは

$$A[i, j, k] = \| q[i, j, k, l, m] \|$$

と宣言する．逆に$A[, 2, 3]$は2，3番目の添字に2と3をもつAの要素からなるarrayをreferすることになる．

$$|1.5, 2.6, 3.9|$$

は一つのvectorをあらわし

$$\left| |1, 2, 3|, |1, 4, 9|, |1, 8, 27| \right|$$

は

$$\left| \begin{matrix} 1, & 2, & 3, \\ 1, & 4, & 9 \\ 1, & 8, & 27 \end{matrix} \right|$$

なる一つのmatrixをあらわす(formation)．一方

$$d[1]:=\cdots:=d[4]$$

は $d[1]:=d[2]:=d[3]:=d[4]$ の意味であり

$$x[1],\cdots,x[n]$$

は $x[1], x[2], x[3], \cdots, x[n]$ の意味である等々，なので

$$|\,|a[1,1],\cdots,a[1,n]|,\cdots,|a[m,1],\cdots,a[m,n]|\,|$$

で

$$\begin{vmatrix} a_{11}, a_{12}, \cdots, a_{1n} \\ a_{21}, a_{22}, \cdots, a_{2n} \\ \vdots \\ a_{m1}, a_{m2}, \cdots, a_{mn} \end{vmatrix}$$

をあらわすことができる．これらに関係して，×は，スカラー積に用いられ :=，+，−，／，÷等はelement毎の演算，•はelement毎の掛算に用いられる．例えば

assignment statement

$$z:=|t[i[1]],\cdots,t[i[n]]|\bullet|t[j[1]],\cdots,t[j[n]]|$$

でzは $t_{i_1}t_{j_1}, t_{i_{1+1}}t_{j_{1+1}}, \cdots$ がassignされるvectorで

$$u:=|t[i[1]],\cdots,t[i[n]]|\times|t[j[1]],\cdots,t[j[n]]|$$

におけるuの値は

$$t_{i_1}t_{j_1}+t_{i_{1+1}}t_{j_{1+1}}+\cdots$$

である．

formationは，一階高いarrayを作るのに使用されるが，compositionの

$$|[2]\ A,B,C|$$

はarray，A, B, Cが二番目の添字の方向に並べられたarray

$$|A\,\vdots\,B\,\vdots\,C|$$

をあらわす．

Input Languageでは

```
begin complex  u , x ;
    u = a+i b ; a=0 ; b=1 ;
    x:=3+( y+4×z )×u+sin( u×z )
end
```

なる形で宣言complexを用いることができる．$u=a+\underline{i}\,b$，$a=0$，$b=1$はinitial valueに関する宣言である．statement の中で直接$\underline{i}$を使えないことに注意されたい．

以上のほかにiterationのためにsuperscript を使用することができるとか，函数宣言を使うことができるとか，その他の便利な手段が与えられている．

次にNaurのAXに対する重要な要求[8]を説明する．Naurはextensionalist ではないが，特定の機械に対して働く共通言語ということに深い関心をもつている．彼はALGOL 60が共通性を一面的にのみ考えすぎて，実際のoperationをあいまいにしている点を批判する．そこでNaurはAXの中に，機械の特徴を指定できるような手段を与えるべきことを提案する．例えば

epsilon (u)

は

(u+epsilon (u)=u)V(u−epsilon (u)=u)

がfalseになるような最小の正数値を与えるfunctionとする．このときプログラム

```
begin real tol; tol:=10↑(−n);
if epsilon (2.7)<tol then
    begin real e, t; integer p;
    e:=2; p:=2;
    for t:=1/p, t/p while t>tol do
        begin e:=e+t; p:=p+1 end
    end
else
```

は $e=1+1/1!+1/2!+1/3!+\cdots$ を許容誤差10^{-n}がepsilon (2.7) より大なるとき，single precisionで計算するが，許容誤差がepsilon (2.7) (machine dependentな量!!) より小さければmultiple precisionの計算を行なわなければならない．それがelse以下に続くのであるが，そのためには，その計算機が扱いうる整数(single precision)の最大数を与えるprocedure

integer procedure max integer

の如きものがなければならない．このようなenvironmentを考慮に入れる方法が，更にいくつか提案されている．

またNaurの提案はstring, label, switch, あるいはnon−rectangular array, procedure等々多方面に及んでいる．

ALGOL Bulletinには，AXについての様々な提案が提出されている．その中でAB 16に出されたDuncan and Wijngaadenの

Cleaning up ALGOL 60

はAXへの多くの寄与を含むものと思われるが，AB16を手にすることができなかつたので，内容について一言も述べられないのは残念である．

4. 4−th meetingにおけるAXの特徴づけ

さきに述べたようにWG 2.1の4−th meetingでは，AXについての特徴づけが行なわれ，そのsyntactical なまとめがWijngaadenにゆだねられた．この特徴づけに関する討論の結論を箇条書きにまとめてみる．

1. complex, complex procedure, complex array,
2. Naurの述べている如き，environment enquiryの手段，
3. higher precision の演算を与える方法，
4. label, identifier, address variable
5. irreducible data element は Boolean element, character, integer, real である．
6. string variableを定め，stringのcatenation, insertion, deletion 等を行う手段，
7. Cartesian

上にあげた1から7の項目の中で，7については今のところ何を意味するのかさっぱりわからない．また4の label variableというのはその valueがlabelである variable であつて

$$lv := L1 ; \cdots\cdots ; \underline{go\ to}\ lv$$

等として使用できるが，identifier variable, address variable というのは label variableと同じものか，ちがうものか不明である．5のirreducible data というのはdataとして基本的なものは，これだけという意味であろう．

Wijngaadenのまとめは，彼の好みによつて他人の主張を適当に取捨選択するであろうし，また新しいものをつけ加えるであろう．5−th meetingは彼のreport を出発点として再びAXの精密化のdiscussionに進むはずである．

以上で大さつぱに，WG 2.1を中心とするfuture languageについての考え方と，その内容を説明した．AYはまだ芽の状態にもなつていないようである．AXの明文化の作業を通じて，Wijngaadenのlanguageの能力がためされるのかも知れない．AXはある程度の方向づけがされたようであるがその内容については，全く流動的な状態と考えられる．従つてAXについての寄与を行いうるのは，正に今の時期が最適であろう．ABへの投稿は無条件で歓迎されることと思う．最後にABの編輯者の住所を記しておく．

F.G. Duncan, Lubeckstraat 71,
THE HAGUE,
the Netherlands.

文　献

1) Report on IFIP SUBSET ALGOL 60 (IFIP); C. ACM, Vol. 7(1964) No. 10, 626.

2) Report on Input—Output Procedures for ALGOL 60; C. ACM, Vol. 7 (1964), No. 10, 628.

3) 情報処理, 5巻(1964), 6号

4) A. van Wijngaaden: Generalized ALGOL; Annual Review in Automatic Programming, Vol. 3(1963), Pergamon Press, 17.

5) N. Wirth: A Generalization of ALGOL; C. ACM, Vol. 6(1963), No. 9, 547.

6) J. V. Garwick: Remark on Further Generalization of ALGOL; C. ACM, Vol. 7(1964), No. 7, 422.

7) A. P. Yershov, G. I. Kozhukhin and U. M. Voloshin: Input Language for Automatic Programming System, 1963, Academic Press.

8) P. Naur: Proposals for a new language; AB 18, 23.

本 PDF ファイルは 1965 年発行の「第 6 回プログラミング―シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html