ALGOL 60 Language について*

長尾　真**

はしがき

よく知られているように，ALGOL 60 Language は主として科学技術計算のために作られたAutomatic Programming System の言語であって，世界的な共通語を意図して作られたものである．ALGOL 60 Language を広く普及させることは現在重要なことであると考え，以下にこの言語の特徴，その syntax 上の問題点をなるべく多くの例題を用いて述べた．

1. ALGOL Language のおいたち

Automatic Programming System はかなり古くから研究されていたようであるが，それらはおのおの System を異にし，それら相互間の互換性についてはほとんど考慮がはらわれていなかった．しかし共通に使える言語についての要求は次第にたかまり，U.S.A. の ACM (Association for Computing Mechinary) と Germany の GAMM (Gesellschaft fur Angewandte Mathematik und Mechanik) が共同で検討した結果 Preliminary Report—International Algebraic Language を出した．これは 1958 年 5 月から，6 月にかけてである．その目的とするところはつぎの点であると述べられている．

1) The new language should be as close as possible to standard mathematical notation and be readable with little further explanation
2) It should be possible to use it for the description of computing processes in publications.
3) The new language should be mechanically translatable into machine programs.

その後 1959 年 6 月にパリで開かれた UNESCO Conference on Data Processing においてこの問題が正式にとり上げられ，さらに何回かヨーロッパおよびアメリカで会合が開かれ，またその間種々の提案がなされた．しかし ALGOL 60 にあらわれた syntactical な言語の構成法がどのようにして形成されてきたかは明らかでない．とにかく 1960 年 1 月にパリで開かれた会合において採用された ALGOL 60 Language は，これまでの Automatic programming に用いられてきた言語の体系とは本質的に性格をことにし，厳密であると同時にかなり複雑難解な体系として生れ出たわけである．

その後はこの ALGOL 60 をくわしく検討するとともに，これに対する compiler の研究，作成が各国ではじめられた．しかし ALGOL 60 の体系が検討されて行くにつれて，その syntax にいくつかの不備な点が発見され，またその言語構成の非常な自由性のためにこれらを完全に取りあつかいうる compiler を作ることの困難性が認識されてきている．今後は syntax の不備をのぞくとともに compiler が作りやすいように ALGOL 60 になんらかの制限をもうけたり，またその他別の syntactical な要素をつけ加えること (complex computation. input output など) などが議論されるであろう．しかし ALGOL 60 の基本的な構成法が変更をうけるということは考えられないし，現在すでに数多くの program が ALGOL 60 Language で表現されて発表されており，今後共種々の研究分野でこの言語を用いた研究発表がなされて行く傾向にあるものと思われる．

2. ALGOL 60 の特徴

ALGOL 60 Language はその report の最初にも書かれているように，必要最少限度のことを recursive に定義された syntax によって行ない，それ以上に不必要なことは書かない主旨であるので，その解釈は困難でうっかりすると見落してしまう点がある．以下においては ALGOL 60 syntax 中で特に注意すべき点やその special case などについて述べる．

2.1 Left to right principle

---

* On ALGOL 60 Language, by Makoto Nagao (University of Kyoto)
この解説は 10 月 27 日に当学会の ALGOL 研究会で行われた著者の講演のうちで，ALGOL 60 に関する解説の部分だけを掲載したものである。当日の討論内容については，研究会の議長が寄稿された（寄書）を参照されたい。

** 京都大学工学部

普通たとえば数式を compile するときは，arithmetic units と memory との間の数値の転送を最少にするように一番 depth の深い所から順次外側へむかって計算をして行くことが行なわれているが，ALGOL 60 では〔3.3.5〕* に定義されているように計算は left to right に行なうよう定められている．これはつぎのような例を見ればその意義が明らかになるものと思われる．

〔例1〕

real procedure $A(Z)$; **real** Z;
begin $A:=Z^3$; $Z:=Z+1$ **end**;
real procedure $B(Z)$; **real** Z; $B:=Z^5+5$;

である場合に $P: A(x)+B(x)$ と $Q: B(x)+A(x)$ とではその結果がちがう．それは

$$P:=x^3+(x+1)^5+5$$

であるに反し

$$Q:=x^5+5+x^3$$

となるからである．

そこで left to right principle を守るということになるが，left to right という言葉の解釈そのものに問題があるのであって，これは後にのべる（**3.1** 参照）．

2.2 Scope

ALGOL 60 では block とは declaration を含んだ一つの statement であり〔4.1.1〕，すべての identifier, variable などはどこかの block head で定義する必要がある（ただし label と procedure declaration の formal parameter などはのぞく）〔5〕．

そしてその block head で定義された identifier はその block の外ではなんらの意義をもたないものとされる〔5〕．

そこでたとえばつぎのような例：

〔例2〕

```
begin real A,B,C;
A:=B×C+2;
begin integer A;
A:=2+B;          Q:=A↑2+A↑0.5
end;
S:=A
end
```

においては内側の block を実行している間，外側で定義された variable A を保存しておき，内側の block から normal exit または jump instruction により外側の block に出たときにもとの値を復元する必要を生じる．これは programming technique 上少々やっかいな問題である．特に recursive procedure において重要である（**2.4** 参照）．

own のついた variable, array などでは一度計算された値はそれらの定義された block を出てからも cancel されずにのこり，再び同じ block にはいって来たときにも元の値を保持している作用をもつ〔5〕．これは program 上ではその定義された block を出ても **own** と declare された quantity を cancel しなければよい．

なお label は本質的に local なものである〔4.1.3〕．その意味はつぎの例によりわかる〔4.3.4〕．

〔例3〕

```
begin real A, B, C;·····
P: A:=B+2×C;·····
begin real A, D;·····
  Q: A:=2×B+C;
     D:=2+B+A;·····
P: C:=2×A−D;·····
     go to P;·····
     go to R;·····
  end;·····
R: go to P;·····
end
```

2.3 Procedure statement と Procedure declaration

procedure statement と procedure declaration とは普通行なわれている progromming における subroutine call と subroutine との関係に相当するものと考えることができる．しかし ALGOL 60 における parameter の linkage の方法には二つあって，procedure heading にある value part は parameter の linkage を規定するためのものである〔4.7.3〕．value part にかかれた formal parameter を parameter called by value といい，そうでない formal parameter を parameter called by name と呼ぶ．

parameter called by value: これは procedure body にはいる直前にその formal parameter に対応する actual parameter の値を計算し，その値を formal parameter に assign することである〔4.

* 以下〔3.3.5〕のような表現は ALGOL 60 Report の 3.3.5 項を参照することを示す

7.3.1〕. したがって procedure body を実行しているときには actual parameter に対して演算を行なうことはないし, procedure body から exit したときにも actual parameter の状態に変化はなく, procedure body にはいる前の値を保っている.

parameter called by name: これはその formal parameter が body 中に現われるたび, その位置に対して actual parameter を置きかえて, 演算は actual parameter に対して行なうのであって, その計算はすべて current value で行なうところが parameter called by value とは根本的にちがう点である〔4.7.3.2〕. したがって procedure body を出たときに actual parameter の状態に変更をうけているということが存在しうる.

以上のようにある paramcter を value で呼ぶか name で呼ぶかは結果にとって差異を生ずるものであるから注意がいるのである. このことを例によって示す.

〔例4〕

```
procedure Sum (Z, Y, X, A, B, C);
  begin Z:=0;
  for X:=A step B until C do Z:=Z+Y
end
```

これを用いて $U=\sum_{n=1}^{100}\frac{1}{n}$ を計算するためには,

```
Sum (U, 1/n, n, 1, 1, 100)
```

なる形の procedure statement を作ればよい. ここで $1/n$, n は Y, X に substitute (by name) することが必要であり, また U は結果がはいる storage であるから当然 name で呼ばねばならない.

一方 A, B, C, については, name で呼ぶ必要はなく, これは value part に **value** A, B, C; をつけ加える方がよいと思われる. なお記述を完全にするためには **integer** A, B, C; をつけ加えた方がよい.

一方つぎのような場合:

〔例5〕

```
real procedure R3 (X, A, B, C, D);
  value X; real X; integer A, B, C, D;
  R3:=(A×X↑3+X↑2+B×X+1)/
      (C×X↑3+X↑2+D×X+1);
```

が

```
U:=K×R3(arctan(S/cos(k×t)-(S-1)
    ×sin(k×t)), a, b, c, d)
```

で用いられたとき, もし X が parameter called by value でなかったとしたら非常にめんどうな計算をせねばならないが, これが value で呼ばれているためにつぎのような簡単な計算ですむことになる.

```
begin real X, R3;
X:=arctan(S/cos(k×t)-(s-1)sin(k×t));
R3:=(a×X↑3+X↑2+b×X+1)/(c×X↑
    3+X↑2+d×X+1);
U:=K×R3
end
```

もちろん procedure body 中の計算においても left to right principle が守られるのであってつぎのような例では特に注意が必要である.

〔例6〕

```
begin real B, D; array A〔1:10, 2:20〕;
procedure P (a, b, c, d); real a, b, c, d;
  a:=b:=c×d+a+c:
real procedure C (dd); real dd; begin
dd:=dd+5;  C:=dd-3 end C;
PROGRAM:
D:=5; B:=4; A〔10, 7〕:=-20;
P(A〔D, B+3〕, B, C(D)-4, D)
end
```

この program の計算は P なる procedure declaration を計算することになるのであるが, それにはまず name replacement を行ない

```
A〔D, B+3〕:=B:=(C(D)-4)×(D)+(A〔D,
  B+3〕)+(C(D)-4).
```

left to right principle からまず第一には

```
A〔5,7〕:=B:=
```

つぎに右辺最初の $C(D)-4$ は $D=5$ であるから,

```
C(D)-4=3, (このとき D=10)
```

つぎの (D) は

```
(D)=10
```

つぎの項は

```
A〔D, B+3〕=A〔10, 7〕=-20
```

となる. そして最後の項は

```
C(D)-4=8 (このとき D=15)
```

となり結局

```
A〔5, 7〕:=B:=18 (with the side effect D=15)
```

という結果を生ずる. これを最初から

```
A〔5, 7〕:=B:=(C(5)-4)×5+(A〔5, 7〕)+(C
  (5)-4)
```

とした場合とでは全然ちがった結果になることは明らかである.

なお procedure body に現われる identifier, variableなどがすべてその procedure に対して non-local であり，local なものが存在しないときは procedure declaration の parameter part が vacant でありうる．

〔例7〕

procedure *P*; **A**:=*B*×*C*;

(*A*, *B*, *C* はこの procedure に non-local な quantity)

Procedure statement の parameter に string がゆるされているが，これは対応する procedure declaration の body が non-ALGOL code でかかれた場合のために作られたものであって，ALGOL Language のみを body にもつ場合は関係がない．〔4.7.5.1〕．

2.4 Recursive procedure

ALGOL Language は recursive に定義されているので複雑さをましているのであるが，特に procedure が recursive に定義される場合は注意を要する．

〔例8〕

real procedure *factorial* (*n*); **real** *n*;
factorial:=**if** *n*=0 **then** 1 **else** *n*×*factorial* (*n*−1);

〔例9〕

real procedure *SIGMA* (*i*, *h*, *k*, *t*); **value** *h*; **integer** *i*, *h*, *k*; **real** *t*;
begin if *k*<*h* **then** *SIGMA*:=0 **else**
begin *i*:=*h*; *SIGMA*:=*t*+*SIGMA* (*i*, *h*+1, *k*, *t*)
end
end

こういった場合その procedure に local な quantity は新たに自身の procedure に入る前に保存する必要が生ずる．ただし **own** と declare されたものはその必要がない．この process はその procedure がくりかえし，depth が増加するたびに行なわねばならない．一方ある depth の procedure から normal exit または go to statement により外側の procedure に出るときは保存されていた外側の procedure の parameter を復活させる必要がある．よって上記の例でも計算の順序を変えて

SIGMA (*i*, *h*+1, *k*, *t*)+*t*

とすれば結果は全く異なってくる．

3. ALGOL 60 Syntax における問題点

ALGOL 60 が発表されて以来，その syntax 上の不備あるいはその syntax を実行する上での不明確な箇所がいくつか指摘されて来た．それらの代表的なものを挙げるとともに，それらに対してのべられた意見を記する．またその他具体的な提案等についていくつかのべる．これらは主として ALGOL-Bulletin によったものでありくわしくは文献（2）を参照していただきたい．

3.1 Order of evaluation に関するもの

3.1.1 Order of evaluation in arithmetic expression

演算の順序は left to right とされているが，これはかならずしも明確でない．そこでつぎのように定義することが提案されている．それはつぎの二つをはっきり区別することである．

a) The evaluation of primaries, *i. e.* substitution of numbers for variables, function designators, and expressions within parentheses.

b) The execution of operations

そうすると〔3.3.5〕はつぎのようにおきかえるべきである．

3.3.5 Precedence of evaluation and operation

3.3.5.1 Within one expression the primaries must be evaluated in order from left to right.

3.3.5.2 In accordance with the syntax of section 3.3.1 the operations on the resulting values will take place in the following order.

first: ↑
second: × / ÷
third: + −

3.1.2 *A* **step** *B* **until** *C*

〔4.6.4.2〕の *A* **step** *B* **until** *C* の説明中

LL: **if** (*V*−*C*)×sign (*B*)>0 **then go to** Element exhausted は

LL: **if** sign (*B*)×(*V*−*C*)>0 **then go to** Element exhausted とした方がよい．なぜならば *A* **step** *B* **until** *C* は普通 *A*, *B*, *C* の順序に計算を行なうから，*B* が *C* より先に現われるようすべきだとの意見である．

3.1.3 Parameter part

〔4.7.3.1〕の actual parameter (called by value) の計算は，それらの parameter が procedure

declaration の formal parameter part に現われる順に計算すべきだという意見と，value part に書かれている順に計算すべきだとの意見がある．

3.2 Boolean expression に関するもの

3.2.1 Evaluation of Boolean expression

$B1 \vee B2$（または $B1 \wedge B2$）のような Boolean expression では $B1$ が **true**（または **false**）であれば $B2$ は計算しなくてもよいか，あるいはたとえ $B1$ が **true**（**false**）であっても $B2$ も計算しなければならないかという問題がある．これは特に $B2$ を実行した時に他の variable の値が変化するといった場合に問題となる．

3.2.2 Expressions in relations

つぎのような例：

〔例 10〕

Boolean procedure *question* (A, B, C, D);
question:=**if** A **then** B **else** $C = D$

では B の type がわからない場合にはつぎのような二様の意味にとれるから好ましくない．すなわち，B が Boolean であれば

if A **then** B **else** $(C=D)$.

B が Boolean でなければ

(**if** A **then** B **else** C)$=D$.

3.3 Conditional statement に関するもの

3.3.1 Definition of conditional statement

〔4.5.3.2〕 中の sentence: "If none of the Boolean expressions of……that of a dummy statement."は好ましくない．なぜならば conditional part の Boolean はすべて計算しているから，その計算により引きおこされる変化は認めねばならないと考えられる．

3.3.2 For statement

for statement の定義は〔4.6.1〕によると

<for statement>::=<for clause><statement>|
<label>:<for statement>

である．そこで for statement の do のつぎに直接 conditional statement がくる場合としてつぎのような例が考えられる．

〔例11〕

if B **then for** q: =……**do if** $B2$ **then** $S1$ **else** $S2$

この statement は

if B **then** (**for** q: =……**do if** $B2$ **then** $S1$) **else** $S2$.

if B **then for** q:=……**do** (**if** $B2$ **then** $S1$ **else** $S2$)

の二とおりに解釈され，結果はちがうものとなる．そこでこれをさけるために **do** のつぎの <statement> は <unconditional statement> にすべきだとの意見がある．

一方 **then, else** の組合せに対してつぎのような解釈をとればこの問題は解決するという意見もある．つまり

The <delimiter **then** that corresponds to a given **else** is determined so that between the two matching symbols the numbers of **then**'s and **else**'s are equal apart from **then**'s and **else**'s situated between **begin**'s and **end**'s.

とするものである．これはまた，statement の解釈に不明確さをもつ場合には

"The largest statement found in this position" という立場にたてばよいとの意見がある．これは arithmetic expression の項〔3.3.3〕で

"The value of the arithmetic expression is then the value of the first arithmetic expression following this Boolean (the largest arithmetic expression found in this position is understood)."

と同じ考え方である．そうすると上記の〔例 11〕は一義的に解されるし 3.2.2. の〔例10〕についても

question:=**if** A **then** B **else** $(C=D)$

と解釈され B は Boolean Variable でなければならないということになる．

3.4 The definition of a program

ALGOL 60 では program とは，その最初に書かれているように，"A self-contained compound statement" と規定している〔1〕．compound statement と block の定義は〔4.1.1〕に与えられているとおりである．そこで blockを a program とするためには，それを **begin, end** でかこんで compound statement にしなければならない．つまり，

begin begin D; D;……S; S;……**end end**

となり **begin, end** が二重になることは意味がない．そこで，a program の定義〔1〕に現われる四つの compound という語を削るべきである．

3.5 A function designator changing the value of a formal variable.

〔例 12〕

real procedure *Sneaky* (Z); **value** Z; **real**

Z; begin *Sneaky*:=Z+(Z−2)↑2; W:=
Z+1 end

では当然のことながら

$P := Sneaky(k) \times W$

と　$Q := W \times Sneaky(k)$

とでは結果が異なる．しかもその原因となる variable W が formal parameter の位置に現われていないのは不都合である．そこで［5.4.4］に

"Any other identifier which occurs as a left part variable in an assignment statement within the procedure body must either be local or a formal parameter"

という項をつけ加えるべきだとの意見がある．

3.6 Integers as labels

integer を label に用いることはつぎのような例が存在しうるところから好ましくないという意見がある．

［例 13］

procedure *pop* (*Q*); procedure *Q*; begin
……*Q*(3); end;
procedure *pip* (*A*); label *A*; begin……go
to *A*; ……end;
procedure *pap* (*B*); real *B*; begin……
***q*:=*B*;……end;**
***pop* (*pap*);**
***pop* (*pip*);**

procedure *pop* 中の数字3は *pop* (*pap*) では数字として，*pop* (*pip*) では label として用いられることになり好ましくない．

3.7 Complex variable への拡張

現在の ALGOL 60 では complex variables の計算に不便であるので，**complex** なる delimiter をもうけて complex variable も簡単にあつかえるようにすべきだとの提案がある．（くわしくは ALGOL-Bulletin Supplement No. 12, "A Proposed Extension to ALGOL 60" by R.W. Hockney, 22 June, 1961）

3.8 Input and output

input, output の形をいかにすべきかについてはつぎのような方法がある．input, output に関する命令は procedure statement の形にし，それは ALGOL 60 による programのどこへでも挿入できるものにする．それに対応する procedure declaration は standard function と同様に compiler 自身がもっており，programmer がそれを書く必要がないものとする（文献（8）参照）．

3.9 Storage allocation

実際に program を書く場合，core, drum, magnetic tape など種々の storage をうまく使いこなしたい場合が生じる．こういった memory の allocation そのものを compiler が自動的に行なうことは将来の compiler 研究の一つの方向であるかもしれないが（Automatic allocation of a program），少なくとも現在では programmer が指定する必要があろう．そのために ALGOL System と矛盾しない形でなんらかの命令を作る必要があるものと思われる．（文献（8）参照）

参考文献

上記各例題，種々の見解等についていちいち文献名，著者名等明らかにしなかったが，その多くの部分については下記文献におうところが大であった．ここに感謝の意を表する次第である．

1) Report on the algorithmic language ALGOL 60: P. Naur et. al., Comm. of ACM, May 1960 または Numerische Mathematik 2,106-136. (1960)
2) ALGOL-Bulletin No. 1 (16, March, 1959) ～No. 12 (22 June 1961): Regnecentralen Copenhagen
3) A Course of ALGOL 60 Programming: P. Naur, Regnecentralen, Copenhagen, 1961.
4) An Implementation of ALGOL 60 Procedures; J. Jensen, P. Naur, Nordisk Tidskrift for Informations Behandling, Bind 1 Hefte Nr. 1. 1961
5) Thunks—A way of Compiling Procedure Statements with Some Comments on Procedure Declarations: P. Z. Ingerman, Comm. of ACM, Jan. 1961
6) Compiling Techniques for Boolean Expressions and Conditional Statements in ALGOL 60: H.D. Huskey, W.H. Wattenburg, Comm. of ACM, Jan. 1961
7) Comments on the Implementation of Recursive Procedures and Blocks in ALGOL 60: E.T. Irons, W. Feurzeig, Comm. of ACM. Jan. 1961
8) A Manual of the DASK ALGOL Language: Regnecentralen, Copenhagen. Nov. 1960

は Denmark の DASK Computer に対して作成された ALGOL 60 programming system で ALGOL 60に対する制限は非常に少ない．特に input, output

の形式および data allocation に関して興味ある system を作っておりぜひ一読をすすめる.

なお上記文献中（3）（4）（8）は情報処理学会に申し込めば入手できるものと思われる.

現在 ALGOL language に関する情報交換のための centre としてつぎの2箇所がある.

Editor, Communications ACM
Carnegie Inst. of Tech.,
Pittsburgh, 13, Pa., U.S.A.
Editor of the ALGOL-Bulletin
Regnecentralen
Gl. Carlsbergvej 2,
Copenhagen, Valby
Denmark

前者は Communications of ACM という monthly の journal を発行しており，後者は ALGOL-Bulletin を不定期に発行している. ALGOL-Bulletin の方はときどき supplement を発行しており（8）の文献もその一つである. 現在ドイツを中心として

Handbook for Automatic Computation

が企画されている（全体で数巻）. この Handbook はあらゆる種類の数学計算の test ずみの algorithm を集成するものであって，この algorithm は ALGOL language で書かれることになっている. volume 1a は ALGOL language の使用法の解説，volume 1b はその translator の解説にあてられる予定である.

また一般的な automatic programming に関する本につぎのものがある.

Annual Review in AUTOMATIC PROGRAMMING:

Edited by Richard Goodman M.A., B. Sc. Pergamon Press. 現在 Volume 1（1960）と Volume と 2（1961）が出ている.

"ALGOL 60 Language について" に対する remark*

高 橋 秀 俊**

1) 〔例 1〕について

この例で $A(x)+B(x)$ と $B(x)+A(x)$ とが違う結果を与えるのは，$A(x)$ という function designator が $Z:=Z+1$ という"副作用"をもつ procedure を呼ぶからである. このように，同じ式が計算の順序によって違う結果を生ずるのは artithmetic expression というものに対する我々の常識に反するものであって，不都合というべきである.

この不都合の原因を除くには，function designator を定義する procedure が副作用を生むことを禁止するのが妥当であろう. たとえば

"function designator を定義する procedure declaration の中には，その function designator 以外の non-local variable に値を assign するような statement があらわれてはならない". という条項を加えれば一応問題は解決されると思われる. この制限は多少窮屈かもしれないが，読んでわかりやすいという ALGOL の特徴を活かすためにはやむを得ないのではあるまいか.

なお，これに類する case の一つとして，function designator が input に関する procedure である場合がある. たとえば，READ が一つの input procedure で，input device から読み出された数値をあらわすとすると，これが2度以上あらわれる statement，たとえば

$A:=\mathrm{READ}+B\times\mathrm{READ};$

は，一方の READ が第1に読み出された数値，他方は第2に読み出された数値をあらわし，どちらが先に演算されるかで異ってしまう. これを有効に防ぐには，input procedure を function designator の形にすることを禁止するか，または，そのような function designator が一つの expression に2回以上使われることを禁止するかしなければならない.

2) 〔例 3〕についての注釈

ここで6行目に P という label が存在しないなら

* Remarks to "On ALGOL 60 Language." by Hidetosi Takahasi (University of Tokyo)
** 東京大学理学部