算法言語 ALGOL 60 に関する報告 (1)*

Peter Naur 編

J.W. Backus, C. Katz, H. Rutishauser, J.H. Wegstein, F.L. Bauer, J. McCarthy, K. Samelson, A. van Wijngaarden, J. Green, A.J. Perlis, B. Vauquois, M. Woodger
〔William Turanski の追憶に捧ぐ〕

淵　一博** 伊藤文子**訳

序　文

背　景

1958年 Zürich の会議で準備された ALGOL[1,2] に関する予備報告が発表されて以来，これに対する関心が大いに高まった．

1958年11月 Mainz で行われた非公式の会合の結果，ヨーロッパ諸国から約40人の関係者が集って1959年2月 Copenhagen で ALGOL 具体化会議が開催された．“hardware group”が結成されて，紙テープコードのレベルに至るまで協力して作業を行うことになった．またこの会議によって Regnecentralen (Copenhagen) から，Peter Naur 編集の ALGOL Bulletin が発行されるに至り，それはその後の討議の場としての役目を果した．1959年6月 Paris における ICIP 会議の間に，公式非公式を含め数回の会合が開かれた．これらの会議によって主として ALGOL の定式化を受け持ったグループの意図に関して，いくつかの誤解のあることが明らかになったが，同時にその努力の価値が広く認められているということも明らかになった．討議の結果 ALGOL を改良して，最終報告を準備するために，1960年1月に国際会議を開くことが決定された．約50人が出席した1959年11月のパリでのヨーロッパ ALGOL 会議では，7人のヨーロッパ代表が選ばれ，1960年1月の会議に出席することになった．彼等は次の機関を代表している．

Association Française de Calcul, British Computer Society, Gesellschaft für Angewandte Mathematik und Mechanik, Nederlands Rekenmachine Genootschap. この7代表は1959年12月 Mainz で最終的な準備会合を開いた．

一方合衆国では ALGOL の変更あるいは修正案を出したい人は誰でも，その意見を発表機関である ACM Communication に送るように要請され，それらの意見は ALGOL の変更を考慮する際の基礎となった．Share および USE 組織は共に ALGOL 作業グループを作り，両組織はプログラム用言語に関する ACM 委員会に代表を送った．この ACM 委員会は1959年11月に Washington で会合し，それまでに ACM Communication に送られてきた ALGOL に関するすべての意見を審議した．また1960年1月の国際会議に出席するため7人の代表が選ばれ，この7代表は1959年12月に Boston で最終的な準備会合を開いた．

1960年1月の会議

デンマーク，イギリス，フランス，ドイツ，オランダ，スイス，合衆国からの13代表[3]は1960年の1月11日から16日まで Paris で会議を行なった．

---

* Report on the Algorithmic Language ALGOL 60 (Comm. ACM 3 No.5 (1960)) の訳の第1部である．本文に述べてあるように，ALGOL は算法を記述するための形式的言語である．我々が普通に事物を記述する，いわゆる対象言語に対して，言語の形式的，構造的な面を抽象して構成した形式的言語をメタ言語と呼ぶ．計算過程を記述した文章を与えて，計算機がそれをコードに翻訳できるためには，この言語は形式的に完全なものである必要がある．言語の形式的な構造をシンタックスという．記述が完全であるためには，そのシンタックスの記述も再び形式的である必要がある．それに対し，言語の意味内容に立ちいたった記述はセマンテックであるといわれる．この報告の記述法は，最初にシンタックスを与え，それを実例，セマンティクス，その他で補なっている．

訳についていえば，シンタックスの項では，原文を殆んどそのまま残した．説明文中では，シンタックスで使われている語についても可能な範囲で，適当な個所で対応を与えながら，訳語を使用することにした．訳語は暫定的なものであるから，今後検討されて統一的言語が制定される必要があるだろう．訳の最後に原語と訳語の対照表を添える予定である．

** 電気試験所電子部

1) 予備報告—International Algebraic Language, Comm. ACM 1, No. 12 (1958), 8

2) Report on the Algorithmic Language ALGOL by the ACM Committee on Programming Languages and the GAMM Committee on Programming, edited by A.J. Perlis and K. Samelson, Numerische Mathematik Bd. 1, S. 41～60 (1959)

3) William Turanski はアメリカ・グループの一員であったが，1960年1月会議の直前，自動車事故で死亡した．

この会議に先立ち，Peter Nauer は予備報告および準備的会合の勧告に基づいて新しい草案を書き下したが，これを会議では報告の基礎として採用した．ついで，会議はその報告の各項目についての意見の一致を得るために進められた．本報告は委員会の考えの和集合と，同意の共通部分を表わしたものである．

ALGOL 予備報告におけると同様に，Reference Language（基準言語），Publication Language（発表言語）およびいくつかの Hardware Representation（金物での表現）の三つの異なった言語のレベルを認める．

Reference Language

（1） 本委員会の使用する言語である．

（2） 定義を行なう言語である．

（3） 文字は相互理解の便宜によって定めるものであって，個々の計算機での制限，コーダの記法，あるいは純粋な数学上の記法によるものではない．

（4） 編集ルーチンの作製者にとって基本的な参考と手引になるものである．

（5） すべての hardware representation の手引となるものである．

（6） publication language を局所的に妥当な hardware representation に逐字訳するための手引となるものである．

（7） ALGOL 自身の主な印刷物には reference representation を用いるものとする．

Publication Language

（1） publication language では，印刷や筆記の慣用に応じて reference language を変更することを許す．（例えば添字，空白，指数，ギリシャ文字などについて）．

（2） 過程の記述と伝達に使われる．

（3） 用いる文字は国によって違ってよいが reference representation とは意味を違えずに一致を保証するものでなければならない．

Hardware Representations

（1） hardware representation はいずれも標準的な入力装置の文字の数に制限があるために止むを得ず reference language を圧縮したものとする．

（2） それは特定の計算機の文字の集合を使用し，その計算機用のトランスレータが取扱うことのできる言語である．

（3） それは publication language もしくは，reference language からの逐字訳のための特殊な諸規則を持っていなければならない．

reference language と publication language との間の逐字訳には，なかんづく，次の規則を推奨する．

Reference language	Publication language
添字角括弧 〔 〕	角括弧内の行を下げ，括弧を除く．
指数 ↑	その指数を上げる．
丸括弧 ( )	丸括弧 ( )，角括弧 〔 〕 大括弧のいずれでもよい．
基数 10	その 10 と次の整数を上げ，必要な積の符号を入れる．

Reference language の説明

口にするねうちのあることなら，はっきり言って，無駄口はたたかない．
Ludwig Wittgenstein

1. 言語の構造

序文に述べたように算法を記述するこの言語（algorithmic language）には三つの異なった表現方法がある．すなわち reference language, hardware representations, および publication language である．以下は，reference language について述べる事にする．すなわち，この言語で定義される一切の対象は，与えられた記号の集合によって表現される．他の二つの表現との違いは記号の選び方にあるに過ぎない．構造と内容は，すべての表現について同じでなければならない．

ALGOL の目的は計算過程を記述することにある．計算規則の記述に用いられる基本概念は，数，変数，および関数を構成要素として含む周知の算術式である．そのような式から，算術合成の規則を適用することによって，この言語の自己完結的な（self-contained）単位，すなわち，assignment statenment* と呼ばれる陽に表わされた形の式が合成される．

計算過程の流れを示すために，ある非算術的 statement ないし statement clause が追加される．これは，例えば，計算の statement の二者択一，あるいは，反復などを表わすものである．これらの statement の機能としては，ある statement が他を参照

* これらの術語は後ではっきり規定される．後で規定される語が，前の部分の説明に使用されることがある．部分を理解するには全体を知らなければならない．〔訳注〕

することが必要になるから，statement にはラベルを附すこともある．一連の statement は statement bracket を挿入することにより，結合されて複合 statement となることがある．

statement は declaration によって補われる．この declaration はそれ自身計算命令ではないが，トランスレータに，statement に現われる対象の存在とその性質，例えば，変数が値として取る数の class，数の配列の次元，あるいはさらに関数を定義する規則の集合のようなものをも知らせる．declaration は，それぞれ1個の複合 statement に附加され，それに対してだけ有効である．declaration を含む複合 statement は block と呼ばれる．

プログラムは一つの自己完結的な複合 statement である．すなわち，別の複合 statement に含まれず，また自らに含まれない他の複合 statement を使うこともない複合 statement である．

次に，この言語のシンタックス（syntax）とセマンティクス（semantics）について述べよう[4]．

1・1 シンタックス的記述のための形式化

シンタックスは，メタ言語的な式（metalinguistic formulae）の助けによって記述される[5]．

その解釈は次の例によって最もよく説明される．

<*ab*>::=(|[|<*ab*>(|<*ab*><*d*>

角括弧 < > で囲まれた文字の列は，メタ言語の変数（metalinguistic variable）を表わし，その変数の値は一連の記号（symbol）である．記号（mark）::= および |（後者は or（または）の意味をもつ）はメタ言語の連結詞*である．式の中で，変数あるいは連結詞でない記号はそれ自身（またはそれに類似する記号の類）を表示する．式の中でいくつかの記号および（あるいは）変数を並べておくことは，表示される列の並んでいることを意味する．かくして上の式は変数 <*ab*> の値を形成するための回帰的（recursive）な規則を与えるものである．すなわち，この式は<*ab*>の取りうる値が，(，または，[，または，<*ab*> の規則に合ったある値が与えられて，それに (を続けるか，変数 <*d*> のある値を続けるかによって得られる値である，ということを示す．もしも <*d*> の値が10進数字であれば*，<*ab*> の値は次のようなものである．

[(((1 (3 7 (
(1 2 3 4 5 (
(((
[8 6

学習を容易にするためメタ言語の変数（すなわち前例の ab のように < > の中に現われる一連の文字）の見分けをつけるために用いる記号（symbol）は，対応する変数の性質を大体表わすような単語を選んだ．このような形で表わされた単語が文中の他の場所で使われる場所には，それに対応するシンタックス的な定義を参照するものとする．また，式によっては数箇所で与えられるものもある．

〔定　義〕
<empty>::=
（すなわち symbol が空（から）である．）

2. Basic symbol（基本記号），Identifier，Number（数），および String（連鎖）

基本概念

reference language は次の基本記号から組立てられる．

<basic symbol>::=<letter>|<digit>|
<logical value>|<delimiter>

2・1 Letter（文字）

<letter>::=a|b|c|d|e|f|g|h|i|j|k|l
|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A
|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P
|Q|R|S|T|U|V|W|X|Y|Z

この alphabet は任意に制限したり，他の区別のつく字（すなわち，数字（digit）とか論理値（logical-value），限定詞（delimiter）と一致しない範囲で）に拡張できる．

文字は個々では意味を持たない．それは，Identifier および「連鎖[6]」を形成するために使われる．（2・4，Identifier，2・6，String 参照）

---

[4] 演算の精度が，一般には指定されていない，とか，ある過程の結果が定義されていない，という時の意味は，いつも次のように解釈するものとする．すなわち，プログラムに附随する情報が，仮定する精度，仮定する演算の種類，および計算の実行中に起りうるようなすべての事態に対してとるべき動作の道筋を指定しているということを条件にして，プログラムだけが計算過程を充分に定義するのである，と．
〔訳注〕例えば除算の場合，分母が0であるときは商は「定義されていない」．分母が0であるときにどうするかは，個々のプログラムによって異なる．それは個々のプログラムがそれぞれ規定すべきものであって，一般には「定義されない」ものである．

[5] J.W. Backus, The syntax and semantics of the proposed international algebraic language of Zürich ACM-GAMM conference. ICIP Paris, June 1959 参照

* 簡単に言えば，<a>::=B|C|D の意味は，「a は，B または C または D である．」あるいは，「変数 a のとりうる値は，B か C か D かのどれかである」ということである．〔訳注〕

* 2・2・1 参照〔訳注〕

2・2・1 Digit（数字）

<digit>::=0|1|2|3|4|5|6|7|8|9

数字は「数」および identifier および「連鎖」を形成する為に使う．

2・2・2 Logical value（論理値）

<logical value>::=**true**|**false**

論理値は固定した明白な意味をもっている．

2・2・3 Delimiter（限定詞）

<delimiter>::=<operator>|<separator>|
<bracket>|<declarator>|<specificator>
<operator>::=<arithmetic operator>|
<relational operator>|<logical operator>|
<sequential operator>
<arithmetic operator>::=＋|－|×|／|÷|↑
<relational operator>::=＜|≦|＝|≧|＞|≠
<logical operator>::=≡|⊃|∨|∧|¬
<sequential operator>::=**go to**|**if**|**then**|
else|**for**|**do**[7]
<separator>::=，|．|$_{10}$|：|；|：＝|**#**|**step**|**until**|
while|**comment**
<bracket>::=(|)|[|]|‘|’|**begin**|**end**
<declarator>::=**own**|**Boolean**|**integer**|**real**|
array|**switch**|**proceduce**
<specificator>::=**string**|**label**|**value**

限定詞は固定した意味をもっている．その大部分は明白なものであるが，そうでないものについては，後に適当な個所でその意味を与える．

空白とか改行のようなタイプ印刷上の事柄は，reference language では，何らの意味を持たない．しかし，読む事を容易にするためにそれを自由に使っても構わない．

プログラムの記号の中に文章を含ませる目的で次の“comment”便法を用いる．

一連の基本記号	これと同等，
；**comment**＜；を含まぬ任意の系列＞；	；
begin comment ＜；を含まぬ任意の系列＞；	**begin**
end＜**end** または；または **else** を含まぬ任意の系列＞	**end**

同等というのは，ここでは右側の欄に示された三つの symbol は，連鎖の外に何があっても，プログラムの動作には何の影響も与えないで同一行の左欄に示した構造の記号列でおきかえられるという意味である．

2・4 Identifier

2・4・1 シンタックス

<identifier>::=<letter>|<identifier>
<letter>|<identifier><digit>

2・4・2 例

q
Soup
V17*a*
a 34 KTMNs
MARILYN

2・4・3 セマンティックス

Identifier には内在的な意味はないが，「単純変数」，「配列」，「ラベル」，「スイッチ」，「手順」を Identify する役目を果す．これは自由に選んでも構わない．（ただし，3・2・4 Standard Function 参照）．

同じ identifier は二つの異なった量を示すのに使用してはならない．ただし二つの量がそのプログラムの declaration で定義されたところにしたがって切離された範囲にある場合は例外である．（2・7 および 5. Declaration 参照）

2・5 Number（数）

2・5・1 シンタックス

<unsigned intiger>::=<digit>|
<unsigned intiger><digit>
<integer>::=<unsigned integer>|
＋<unsigned integer>|－<unsigned integer>
<decimal fraction>::=．>unsigned integer>
<exponent part>::=$_{10}$<integer>
<decimal number>::=<unsigned integer>|
<decimal franction>|<unsigned integer>
<decimal franction>
<unsigned number>::=<decimal number>|
<exponent part>|<decimal number>
<exponent part>
<number>::=<unsigned number>|
＋<unsigned number>|
－<unsigned number>

2・5・2 例

0　　-200.084　　$-.083_{10}-02$
177　　$+07.43_{10}8$　　$-_{10}7$

---

[6] 特に注意すべきことは reference language 全体を通じて，太字が独立した基本記号（2・2・2 および 2・3 参照）を定義するのに用いられていることである．これらは，それを構成している個々の文字とは何の関係もないと解釈される．この報告の中では太字は他の目的には使われない．

[7] **do** は for statement の中で用いられる．それは予備報告（ALGOL 60 には含まれていない）の **do** のどれとも何の関係もない．

.5384　　$9.34_{10}+10$　　$_{10}-4$

$+0.7300$　　$2_{10}-4$　　$+_{10}+5$

2・5・3 セマンティックス

10進数（decimal number）はその慣習的な意味をもつ．指数部（exponent part）は10の整数巾として現わされる尺度因子である．

2・5・4 Type（型）

整数（integer）は **integer** 型であり他のすべての数は **real** 型である．

2・6 String（連鎖）

2・6・1 シンタックス

<proper string>::=‘または’を含まない基本記号の任意の系列 >|<empty>

<open string>::=<proper string>|

‘<open string>’|<open string><open string>

<string>::=‘<open string>’

2・6・2 例

‘5K„−‘[[[‘∧=／:’Tt”

‘..This#is#a#‘string”

2・6・3 セマンティックス

この言語が任意の基本記号の系列を取扱えるようにするために，連鎖の引用符‘および’を導入する．記号#は空白を意味する．それは連鎖の外では何の意味をもたない．

連鎖は「手順」の actual parameter として使われる．（3・2 Function Designator および 4・7 Procedure Statement 参照）

2・7 Quantity（量），Kind（種類）および Scope（範囲）．

次の種類の量が区別される．「単純変数」「配列」「ラベル」「スイッチ」および「手順」

量の範囲とはその量に関する identifier のための declaration が成立つ statement の集合であり，あるいは，ラベルに関しては*，そのラベルが，次に実行するもの（successor）として現われる statement をもつような，statement の集合のことである．

2・8 Value（値）および Type（型）

値とは順序づけられた一組の数（特例：単一の数），あるいは順序づけられた一組の論理値（特例：単一の論理値），あるいはラベルのことである．

シンタックスでの単位のあるものは，値をもつといわれる．それらの値は一般にはそのプログラムの実行中に変化するであろう．式の値とその構成要素は第3章で定義する．配列 identifier の値は，それに対応する「添字つき変数」の配列の，順序づけられた一組である．（3・1・4・1 参照）

各種の“型”すなわち **integer, real, Boolean** は基本的には値の性質を表示する．シンタックスでの単位に附随する型は，それらの単位の値についていう．

3. Expression（式）

この言語で算法の過程を表わすプログラムの主要構成要素は「算術式」，「論理式」，および「指定式」である．これらの式の構成要素は，ある種の限定詞を除き，論理値，数，変数，関数指定詞，および，基本的な「算術作用素」，「関係作用素」，「論理作用素」，それに「順序作用素」である．変数と関数指定詞のシンタックス的定義は，共に式を含むから，式とその構成要素の定義は必然的に回帰的となる．

<erpression>::=<arithmetic expression>|

<Boolean expression>|<designational expression>

3・1 Variable（変数）

3・1・1 シンタックス

<variable identifier>::=<identifier>

<simple variable>::=<variable identifier>

<subscript expression>

::=<arithmatic expression>

<subscript list>::=<subscript expression>|

<subscript list>, <subscript expression>

<array identifier>::=<identifier>

<subscripted variable>::=<array identifier>

〔<subscript list>〕

<variable>::=<simple variable>|

<subscripted variable>

3・1・2 例

epsilon

det$A$

a 17

$Q[7,2]$

x[sin($n\times pi/2$),$Q[3,n,4]$]

3・1・3 セマンテックス

変数は単一の値に与えられる呼称である．その値は表現の中で他の値を形成するのに使われ，また assignment statement（4・2 節）によって自由に変更できる．特定の変数の値の型はその変数自身に対する，あるいはそれに対応する配列 identifier に対する de-

* ラベルについては declaration がない．他の三つに対しては，それに対する declaration がある．〔訳注〕

claration (5・2節 Array Declaration 参照) の中で定義される.

3・1・4 Subscript (添字)

3・1・4・1 添字つき変数 (subscripted variable) は, 多次元の「配列」(5・2節 Array Declaration 参照)の構成要素となる値を指定する. 添字リスト(subscripted list) の「算術式」は, それぞれ添字つき変数の一つの添字位置を占め添字と呼ばれる. 添字の完全なリストは添字括弧〔 〕で括られる. 添字つき変数が指す配列の要素は, その添字の実際の数値で指定される (3・3 節 Arithmetic Expression 参照).

3・1・4・2

各添字位置は **integer** 型の変数の如き働きをし, その添字の値を求めることは, この仮想的な変数への附値と同じことであると解される (4・2・4 節参照).

添字つき変数の値は添字表現の値が配列の添字限界内にある場合にのみ定義されるものとする (5・2 節 Array Declaration 参照).

3・2 Function Designator (関数指定詞)

3・2・1 シンタックス

<procedure identifier>::=<identifier>
<actual parameter>::=<string>|
<expression>|<array identifier>|
<switch identifier>|<procedure identifier>
<letter string<::=<letter>|
<letter string><letter>
<parameter delimiter>::=,|)
<letter string>:(
<actual parameter list>
::=<actual parameter>|
<actual parameter list>
<parameter delimiter>
<actual parameter>
<actual parameter part>::=<empty>|
(<actual parameter list>)
<function designator>
::=<procedure identifier>
<aciual parameter part>

3・2・2 例

$\sin(a-b)$

$J(v+s\ ,n)$

R

$S(s-5)$ Temperature: (T) Pressure:(P)

Compile (':=') Stack: (Q)

3・2・3 セマンティックス

関数指定詞は単一の, 数値あるいは, 論理値を定義する. それは, procedure declaration (5・4 節 Procedure Declaration 参照) によって定義された与えられた規則の集合を, actual parameter の固定された集合に適用する事によって得られるものである. actual parameter の指定を支配する規則は, 4・7 節 procedure statement で与える. あらゆる procedure declaration が関数指定詞を定めるわけではない.

3・2・4 Standard Function (標準関数)

Identifier のいくつかは解析学の標準関数用に保存しておくべきである. これは procedure として表わされるであろう. そのリストには次のものを含めることが望ましい.

abs(E)	表現 E の値の絶対値
sign(E)	E の値の符号 ($E>0$ に対して+1, $E=0$ に対して 0, $E<0$ に対して −1)
sqrt(E)	E の値の平方根
sin (E)	E の値の sine
cos (E)	E の値の cosine
arctan(E)	E の値の arctangent の主値
ln (E)	E の値の自然対数
exp (E)	E の値の指数関数 (e^E)

これらの関数はすべて **real** 型, **integer** 型の変項のいずれにも作用するものと解される. **integer** 型の値をとる sign (E) を除いて, それらすべては **real** 型の値をとる. 特に定めた表現では, これらの関数, は, declaration を陽に出さなくても使用できる. (5 章 Declaration 参照)

3・2・5 Transfer Function

量および表現の任意の対の間で transfer function というものを定義し得るものとする. 標準関数の中に次の transfer function を含めることが望ましい.

entire (E)

これは **real** 型の表現を **integer** 型の表現に移し, それに E の値を越えない最大の整数値を附与するものである.