講　　座

ALGOL N について

(IV) Operations

島　内　剛　一*

はじめに

〖前回までに，**ALGOL N** について，
プログラムの外観（岩村）
構　文（和田）
Program の静的構造（和田）
Semantical Notion（佐久間）
の順で紹介してきた．このあとを受けて，今回と次回とで，**ALGOL N** の報告第2版の **4.** Operations を，その日本語訳を中心として解説する．〗

4. Operations

〖**2.** Program の静的構造で述べたように，**ALGOL N** の資格をそなえた program は，*legal program* と呼ばれる．

この *legal program* が，どのようにして *elaborate* されるかを記述することによって，program に意味が与えられる．

elaboration は，特別な条件を満足するような *legal program*（これを *normal program* という）に対してだけ与え，一般の *legal program* は，*normalization* という operation によって *normal program* に直されてから，*elaborate* されることにする．*normalization* や *elaboration* の途中で，新しい *quantity*, *variable*, *label* などが必要になることがある．このような新しいものを生成する operation を *generation* と呼ぶ．

この章では，これら *elaboration*, *normalization*, *generation* や，補助的な operation として，*substitution*, *refinement* と呼ばれるものなどを記述する．〗

4.1 *Core Language*

〖従来，プログラミング言語の記述では，構文については，BN 記法などの厳密な記述が使われているものもあったが，プログラムの働きに関しては，日常語をそのまま使い，あいまいさや矛盾のはいり込む余地を残したものが多かった．

ウィーン手法 (Wiener method) のような厳密な記述法もないことはなかったが，記述できるというだけで，実用性にはとぼしいものと思われる．

われわれは，**ALGOL N** の初版を公表するに当り，*core language* と名付けた言語を開発し，それによってプログラムの働きを記述した．このやり方は第2版でも踏襲されている．

物事を厳密に記述する方法のうちで，歴史も永く，洗練されているものに，数学の記法がある．数学では，約束として説明なしに使ういくつかの記号法のほかに，簡単な日常言語と，それらによって定義された新しい記号法などが使われる．ここでは，日常言語の使い方も，先人によって，あいまいさや，矛盾のないように習慣付けられ，われわれは無意識のうちに，それに従っている．そこで，*core language* では，その基礎を，この数学的な記法に置く．この場合，日常言語としては，英語でも日本語でも，適当なものを選べばよいわけだが，**ALGOL N** の文法書のほかの部分との折合いから，*core language* では，それを英語とした．

この日本語訳では，その部分をも日本語に訳すという考えもあろうが，できるだけ忠実に第2版の紹介をするという意味で，*core language* は，そっくりそのまま使うことにした．

core language で記述される種々の operation のうち，*substitution* だけは，普通の数学と同じように「静的」に記述することも容易だが，そのほかのものについては，なんらかの「動作」を伴い，「動的」な記述をしたほうが，理解が容易である．この動的な部分については，**ALGOL 60**（といっても，**FORTRAN** とか **PL/I**, **COBOL** などといっても同じだが）と同〗

* 立教大学理学部

じ考え方を取り入れることにする.

これに二, 三の独特な記号法を追加して, *core language* はできあがる.

core language 自身の記述は, わざと, 形式的にはしなかった. これは, つぎの二つの理由による.

(1) 形式化することによって言語の記述能力に限界が生ずるのをさけること. **ALGOL N** だけを記述するのならば, 必要十分な記述能力を設定することも可能であろうが, それによって, もっと別の言語を記述する際に能力の不足が生ずることが起こりうる. これは, はじめに, いかに強力な記述能力を定めておいたにしても, さけられない. 普通の数学を記述する言語が, 形式化されない理由の一つもこれであると思われる.

(2) *core language* を形式的に記述するには, その記述用の言語が必要になる. その言語を記述するにはどうするのかを考えると, また同じ問題にぶつかる. このようにして, どこかで必ず, 形式的記述を放棄しなくてはならなくなる. そこで, われわれは, 形式的な記述を **ALGOL N** に限った. ‖

「*core language*」は, *elaboration* (e), *generation* (g), *refinement* (r) などの operation を記述するために導入される. この言語は, 普通に数学で使われている概念, 簡単な英語の文章, **ALGOL 60** と似た構造と, つぎにあげるものを含めていくつかの記号に関する規約とから成っている:

core ... end of core

core language で書かれた本文 (text) は, "**core**" と "**end of core**" とで括られる.

$\rightarrow K$

"go to K". ここで, K は *core language* における label である.

→next

"go to next statement".

$f(A)$

"apply the operator f to the operand A".

$f(A)\Rightarrow A'$

"apply the operator f to the operand A, and let A' be the result".

$\Rightarrow A'$

"the operation is now completed with the result A'".

let $A\leftarrow A'$

"let A be what A' has been".

この規約は, A という表記法が, ここで導入されたときのものである.

$A\leftarrow A'$

"let A be what A' has been" または, "assign A' to A".

この規約は, A という表記法が, すでに導入されているときのものである.

let $A\equiv A'$

"let A be *of the form* A' ‖A' の形‖".

if $A\equiv A'\ldots$

"if A is *of the form* $A'\ldots$".

$f(A)$ if $Q\sigma$, $L\tau$

" $f(A)\Rightarrow A'$;
if $A'\in$**Q** then→next, else→K;
let $Q\leftarrow A'$; σ;
→K';
K: if $A'\in$**L** then→next, (else);
let $L\leftarrow A'$;
τ;
K':

$\sigma(i)$ for $i=m, m+1, \ldots, n$

$m\leq n$ のときは,
"$\sigma(m)\sigma(m+1)\ldots\sigma(n)$",
$m>n$ のときは, 空.

ここで, $\sigma(i)$ は, i が整数のとき, *core language* における expression や statement の列とする.

4.2 code body

‖ **ALGOL 60** における, 代入文, 条件文, 繰返し文や, 加減乗除, ベキ, 論理演算などは, **ALGOL N** においては, 言語の中核とは考えず, その動作がコードで記述されている手続き (または, formula) と考える. こうして, コードの果たす役割りが, 重要になってくる.

コードの働きは, パラメタの受渡しと, それを使っての動作の二つに分けられ, 後者は, 〈code body〉と呼ばれる形で表現される.

〈code body〉の働きは, 個々のものによって違っているので, 一括して述べることはできないが, パラメタの受渡しは, 一様な方法で行なわれる. ‖

〈code body〉 X は,

"**code** 〈structure donor〉 J X"

の形の 〈code〉 が *elaborate* されたときに, 動作を起こす. このとき, まず 〈structure notation〉

"**structure** J"

が *elaborate* され，その *elaboration* の結果として *quantity* Q が得られたならば，parameter Q を持って X が *elaborate* される．parameter Q を持っての X の *elaboration* は，

"$X(Q)$"

と表わされる．

X が *core language* によって書かれているときには，このような Q は，「parameter」として引用される．

4.3 *Generation*

elaboration の間に，三つの集合 **Q**, **V**, **L** が扱われる．これらは，それぞれ，*elaboration* に関係している，または関係したことのある，すべての *quantity*, *variable*, *label* の集合である．新しい *quantity*, *variable*, *label* が必要となったときには，**Q**, **V**, **L** を引数として，*generation* という operation g が働く．これは，つぎのようにして定義される．

4.3.1 g(**Q**) は，

core

let Q←an arbitrary abstract element ($\notin$**Q**);
Q←**Q**$\cup\{Q\}$;
$\Rightarrow Q$

end of core.

4.3.2 g(**V**) は，

core

let V←an arbitrary ⟨identifier⟩
($\notin$**V**$\cup$**L**);
V←**V**$\cup\{V\}$;
$\Rightarrow V$

end of core.

4.3.3 g(**L**) は，

core

let L←an arbitrary ⟨identifier⟩
($\notin$**V**$\cup$**L**);
L←**L**$\cup\{L\}$;
$\Rightarrow L$

end of core.

4.4 *Normalization*

legal program は，*program* の洗練と，⟨mark⟩ の除去との過程によって，*normal program* に変換される．この過程は，つぎの三つの operation x, y, z によって組み立てられる．

4.4.1 formula declaration の *simplification*

operation x は，*legal program* における ⟨formula declaration⟩ の *simplification* のために使われる．これは，⟨expression⟩ と ⟨declaration⟩ とに対して，つぎに定義されるように働く．

4.4.1.1 E が ⟨block⟩

"**begin** ⟨declaration⟩ D_1;
.....
⟨declaration⟩ D_m;
⟨identifier⟩ L_1^1: ...: ⟨identifier⟩ $L_{i_1}^1$:
⟨expression⟩ E_1;
.....
⟨identifier⟩ L_1^n: ...: ⟨identifier⟩ $L_{i_n}^n$:
⟨expression⟩ E_n **end**"

の形で，$m\geq 0$, $n\geq 1$, $i_1\geq 0$, $i_2\geq 0$, ..., $i_n\geq 0$ のとき，x(E) は，

core

x(D_1)$\Rightarrow D_1'$;
.....
x(D_m)$\Rightarrow D_m'$;
x(E_1)$\Rightarrow E_1'$;
.....
x(E_n)$\Rightarrow E_n'$;
let E'←
"**begin** D_1'; ...; D_m';
L_1^1: ...: $L_{i_1}^1$: E_1';
.....
L_1^n: ...: $L_{i_n}^n$: E_n' **end**"; $\Rightarrow E'$

end of core.

4.4.1.2 E が，⟨block⟩ 以外の ⟨expression⟩ で，

"$A_0E_1A_1E_2A_2\ldots A_{n-1}E_nA_n$"

の形とする．ここで，$n\geq 0$ で，$\bar{E}_1$, $\bar{E}_2$, ..., $\bar{E}_n$ は，$\bar{E}$ のすべての *immediate constituent* で，A_0, A_1, ..., A_n は ⟨figure⟩ である．このとき，x(E) は，

core

x(E_1)$\Rightarrow E_1'$;
.....
x(E_n)$\Rightarrow E_n'$;
let E'←
"$A_0E_1'A_1E_2'A_2\ldots A_{n-1}E_n'A_n$";
$\Rightarrow E'$

end of core.

4.4.1.3 D が ⟨variable declaration⟩

"**let** ⟨identifier⟩ V **be** ⟨expression⟩ F"

のとき，x(D) は，

core

x(F)⇒F';
let D'←
"**let** V **be** F'";
⇒D'

end of core.

4.4.1.4 D が〈formula declaration〉
"**let** 〈frame〉 G **represent** 〈expression〉 F"
のとき，x(D) は，

core

x(F)⇒F';
g(**V**)⇒V;
let D'←
"**let** V **be** F';
let G **represent** V";
⇒D'

end of core.

4.4.1.5 D が〈mark declaration〉のとき，x(D) は，

core

⇒D

end of core.

4.4.2 formula の *elimination*

operation y は，*legal program* P における〈formula〉の *elimination* のために使われる．これは，P の *direct constituent* に対して，つぎに定義されるように働く．$\bar{E}$ を P の *direct constituent* としよう．

4.4.2.1 E が〈formula〉で，$\bar{F}_1, \bar{F}_2, \ldots, \bar{F}_n$ が $\bar{E}$ の *immediate constituent* で，ここに書かれた順序に現われているものとしよう．

もし，P の *proper declaration* $\bar{D}$ で，$\bar{E}$ に対する宣言となっているものがあれば，このような $\bar{D}$ は一意的に定まる．この場合，$\bar{D}$ を

"**let** 〈frame〉 G **represent** 〈expression〉 F"

の形としよう．

もし，P の *proper declaration* D で，$\bar{E}$ に対する宣言となっているものがないならば，$\bar{E}$ の *skeleton* は前もって宣言されている．この場合，F を，$\bar{E}$ の *skeleton* が **5** (→**5.2.3**) で記述されるようにして宣言されているときの *body* としよう．

y($\bar{E}$) は，

core

y($\bar{F}_1$)→F_1';
……
y($\bar{F}_n$)⇒F_n';
let E'←
"(F)(F_1', …, F_n')";
⇒E'

end of core.

4.4.2.2 E が〈formula〉以外の〈expression〉で，
"$A_0E_1A_1E_2A_2\ldots A_{n-1}E_nA_n$"
の形とする．ここで，$n\geq 0$ で，$\bar{E}_1, \bar{E}_2, \ldots, \bar{E}_n$ は $\bar{E}$ のすべての *immediate constituent* で，$A_0, A_1, \ldots, A_n$ は〈figure〉である．このとき，y($\bar{E}$) は，

core

y($\bar{E}_1$)⇒E_1';
……
y($\bar{E}_n$)⇒E_n';
let E'←
"$A_0E_1'A_1E_2'A_2\ldots A_{n-1}E_n'A_n$";
⇒E'

end of core.

4.4.3 mark declaration と formula declaration の *elimination*

operation z は，*legal program* P における〈mark declaration〉と〈formula declaration〉の *elimination* のために使われる．これは，P の *direct constituent* に対して，つぎに定義されるように働く．

$\bar{E}$ を P の *direct constituent* としよう．

4.4.3.1 E が〈block〉
"**begin** 〈declaration〉 D_1;
……
〈declaration〉 D_m;
〈identifier〉 L_1^1: …: 〈identifier〉 $L_{i_1}^1$:
〈expression〉 E_1;
……
〈identifier〉 L_1^n: …: 〈identifier〉 $L_{i_n}^n$:
〈expression〉 E_n **end**"
で，$m\geq 0$, $n\geq 1$, $i_1\geq 0$, $i_2\geq 0, \ldots, i_n\geq 0$ のとき，z($\bar{E}$) は，

core

(if $D_j\equiv$〈variable declaration〉 then→next,
else→K_j;
let $D_j\equiv$
"**let** 〈identifier〉 V **be** 〈expression〉 F_j";
z($\bar{F}_j$)⇒F_j';
let A_j←
"**let** V **be** F_j'; "; →K_j';
K_j: let A_j←" ";

{label K_j のついている statement は，A_j に空な図形を assign することを意味する．}

K_j': ;) for $j=1, 2, \ldots, m$

z(E_1)⇒E_1';

.....

z(E_n)⇒E_n';

let E'←

"**begin** $A_1A_2\ldots A_m$

L_1^1: ...: $L_{i_1}^1$: E_1';

.....

L_1^n: ...: $L_{i_n}^n$: E_n' **end**";

⇒E'

end of core.

4.4.3.2 E が〈block〉以外の〈expression〉で，

"$A_0E_1A_1E_2A_2\ldots A_{n-1}E_nA_n$"

の形とする．ここで，$n\geq 0$ で，$\bar{E}_1, \bar{E}_2, \ldots, \bar{E}_n$ は，$\bar{E}$ のすべての *immediate constituent* で，$A_0, A_1, \ldots, A_n$ は〈figure〉とする．このとき，z($\bar{E}$) は，

core

z($\bar{E}_1$)⇒E_1';

.....

z($\bar{E}_n$)⇒E_n';

let E'←

"$A_0E_1'A_1E_2'A_2\ldots A_{n-1}E_n'A_n$";

⇒E'

end of core.

4.4.4

P を *legal program* とする．

P の *normalization* は，

n(P)

と表わされ，

core

x(P)⇒P_1; y(P_1)⇒P_2; z(P_2)⇒P_3;

⇒P_3

end of core.

と定義される．

{P_1 は，*legal program*，P_2 は，〈formula〉の形の *direct constituent* を持たない *legal program*，P_3 は，*normal program* である．さらに，P がかげの変数に対して独立ならば，P_1, P_2, P_3 もまた，かげの変数に対して独立である．(→**5**)}

4.5 *Substitution*

E を〈expression〉，〈declaration〉または〈procedure donor〉とし，$U_1, U_2, \ldots, U_k$ ($k\geq 0$) を，たがいに異なる〈identifier〉，$F_1, F_2, \ldots, F_k$ を〈expression〉としよう．このとき，

s(E; $F_1/U_1, \ldots, F_k/U_k$)

は，E の中の $U_1, U_2, \ldots, U_k$ に，同時に $F_1, F_2, \ldots, F_k$ を「*substitution* 【代入】」して得られる像となっている図形を作ることを意味する．これは，つぎのようにして回帰的に定義される．ここで，

s(E; $F_1/U_1, \ldots, F_k/U_k$)

は，しばしば，

s(E)

と略される．

4.5.1 E を〈identifier〉V としよう．

(1) もし，V が，ある $j(1\leq j\leq k)$ について，U_j であるならば，

s(E; $F_1/U_1, \ldots, F_k/U_k$)

は，

core

⇒F_j

end of core.

(2) もし，V が，どの $j(1\leq j\leq k)$ についても，U_j でないならば，

s(E)

は，

core

⇒V

end of core.

4.5.2 E を，〈go to statement〉で，

"**go to** L"

の形としよう．

このとき．

s(E)

は．

core

s(L)⇒L';

let E'←

"**go to** L'";

⇒E'

end of core.

4.5.3 E を，〈block〉で，

"**begin** 〈declaration〉 D_1;

.....

〈declaration〉 D_m;

〈identifier〉 L_1^1: ...: 〈identifier〉 $L_{i_1}^1$:

〈expression〉 E_1;

.....
⟨identifier⟩ L_1^n: ... : ⟨identifier⟩ $L_{i_n}^n$:
⟨expression⟩ E_n **end**"

の形とし，$m \geq 0$, $n \geq 1$, $i_1 \geq 0$, $i_2 \geq 0$, ..., $i_n \geq 0$ とする．このとき，s(E) は，

core

(s(D_i)⇒D_i';) for $i=1, 2, \ldots, m$
(s(E_j)⇒E_j';) for $j=1, 2, \ldots, n$
((s(L_i^j)⇒$L_i^{j\prime}$;) for $i=1, 2, \ldots, i_j$)
for $j=1, 2, \ldots, n$
let E'←
"**begin** D_1'; ... ; D_m';
$L_1^{1\prime}$: ... : $L_{i_1}^{1\prime}$: E_1';
.....
$L_1^{n\prime}$: ... : $L_{i_n}^{n\prime}$: E_n' **end**";
⇒E'

end of core.

4.5.4 E を，⟨procedure notation⟩ で，

"**procedure** (⟨expression⟩ T_1, ...,
⟨expression⟩ T_n)
⟨primary⟩ T ⟨procedure donor⟩ J"

の形とし，$n \geq 0$ としよう．このとき，s(E) は，

core

(s(T_i)⇒T_i') for $i=1, 2, \ldots, n$
s(T)⇒T';
s(J)⇒J';
E'←
"**procedure** (T_1', ..., T_n')$T'J'$";
⇒E'

end of core.

4.5.5 E を，⟨identifier⟩, ⟨go to statement⟩, ⟨block⟩, ⟨procedure donor⟩ 以外の ⟨expression⟩ とし，

"$A_0E_1A_1E_2A_2 \ldots A_{n-1}E_nA_n$"

の形としよう．ここで，$n \geq 0$ であり，$\bar{E}_1$, $\bar{E}_2$, ..., $\bar{E}_n$ は $\bar{E}$ のすべての *immediate constituent* で，A_0, A_1, ..., A_n は ⟨figure⟩ である．このとき，s(E) は

core

(s(E_i)⇒E_i';) for $i=1, 2, \ldots, n$
let E'←
"$A_0E_1'A_1E_2A_2' \ldots A_{n-1}E_n'A_n$";
⇒E'

end of core.

4.5.6 E を，⟨variable declaration⟩ で，

"**let** ⟨identifier⟩ V **be** ⟨expression⟩ F"

の形としよう．このとき，s(E) は，

core

s(V)⇒V';
s(F)⇒F';
let D'←
"**let** V' **be** F'";
⇒D′

end of core.

4.5.7 E を，⟨formula declaration⟩ で，

"**let** ⟨frame⟩ G **represent** ⟨expression⟩ F"

の形としよう．このとき，s(E) は，

core

s(F)⇒F';
let D'←
"**let** G **represent** F'";
⇒D'

end of core.

4.5.8 E を ⟨mark declaration⟩ Dとするとき，$^{s}(E)$ は，

core

⇒D

end of core.

4.5.9 E を ⟨procedure donor⟩ J としよう．

(1) もし，J が空ならば，s(E) は，

core

⇒J

end of core.

(2) もし，J が

": (⟨identifier⟩ V_1, ..., ⟨identifier⟩ V_n)
⟨primary⟩ F"

の形ならば，s(E) は，

core

(s(V_i)⇒V_i';) for $i=1, 2, \ldots, n$
s(F)⇒F';
let J'←
": (V_1', ..., V_n')F'";
⇒J'

end of core.

4.6 *Refinement*

E が ⟨expression⟩ のとき，その *refinement* r(E) の結果は，大ざっぱにいえば，E または E に含まれる *assemblage* に対して *local* であるようなすべての *variable* や *label* に対して，**g(V)** または **g(L)**

の結果の *substitution* によって，E から得られる ⟨expression⟩ のことである．厳密には，中間的な operation i(E) とともに，E の構成に関する回帰法によって定義される．

4.6.1 ⟨expression⟩ E を，

"$A_0E_1A_1E_2A_2\ldots A_{n-1}E_nA_n$"

の形としよう．ここで，$n\geq 0$ であり，$\bar{E}_1$, $\bar{E}_2$, ..., $\bar{E}_n$ は $\bar{E}$ のすべての *immediate constituent* で，A_0, A_1, ..., A_n は ⟨figure⟩ である．このとき，i(E) は，

core

(r(E_i)⇒E_i' ;) for i=1, 2, ..., n

let E'←

"$A_0E_1'A_1E_2'A_2\ldots A_{n-1}E_n'A_n$";

⇒E'

end of core.

4.6.2 E を⟨block⟩ とし，$V_1, V_2, \ldots, V_n$ を E の *proper declaration* によって宣言されているすべての *variable* とし，$L_1, L_2, \ldots, L_m$ を E の *proper labelling* によって *label* されたすべての *label* としよう．ここで，$n, m\geq 0$ である．このとき，r(E) は，

core

i(E)⇒E';

(g(**V**)⇒V_i' ;) for i=1,2,...,n

(g(**L**)⇒L_i' ;) for j=1,2,...,m

s(E' ; $V_1'/V_1, \ldots, V_n'/V_n$,

$L_1'/L_1, \ldots, L_m'/L_m$)⇒$F$;

⇒F

end of core.

4.6.3 E を，⟨procedure notation⟩ で，

"**procedure** (⟨expression⟩ $T_1, \ldots,$ ⟨expression⟩ T_n) ⟨primary⟩ T ⟨procedure donor⟩ J"

の形としよう．ここで，$n\geq 0$ である．

(1) もし，J が空ならば，r(E) は，

core

i(E)⇒F;

⇒F

end of core.

(2) もし，J が

" : (⟨identifier⟩ $V_1, \ldots,$ ⟨identifier⟩ V_n) ⟨primary⟩ F"

の形ならば，r(E) は，

core

r(F)⇒F';

(g(**V**)⇒V_i' ;) for i=1,2,...,n

s(F' ; $V_1'/V_1, \ldots, V_n'/V_n$)⇒$F''$;

let E'←

"**procedure** ($T_1, \ldots, T_n$)T:

($V_1', \ldots, V_n'$)F''";

⇒E'

end of core.

4.6.4 E を，*assemblage* 以外の ⟨expression⟩ としよう．このとき，r(E) は，

core

i(E)⇒E';

⇒E'

end of core.

(昭和47年1月4日受付)