講　　座

ALGOL N について

(VI) Program の動的作用

筧　　捷　彦*

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），
Operations（島内），
Elaboration（島内）
の順に紹介してきた．このあとを受けて，今回と次回とで，**ALGOL N** の報告第2版の **5.** Dynamic behavior of programs, **6.** Standard declarations を，その日本語訳を中心として紹介する．〗

5. *Program* の動的作用

〖この章では，**4.** ***Operations*** で述べた *operation* を用いて，*legal program* の *elaboration* を定義する．

特定の *program* が *elaborate* される以前に，*cosmos* と呼ばれるものが *create* されているものとする．*osmosc*は，**if then else**, :=, +, −, ×, ／ といった通常の programming language では固有の operation と考えられている *formula* などの集まりのことである．こうした operation としてあらかじめ用意しておくことが望ましいものについては，*standard declaration* として **6.** で述べる．〗

5.1 *Normal Program*

legal program P が，つぎの条件を満たすとき，「*normal*」であるという．

（1） *P* は〈mark〉を含まない．

（2） 前以って宣言された *variable* のどれに対しても，*P* の *proper declaration* は無い．

（3） 前以って宣言されてはいない，どの *variable* に対しても，*P* の *proper declaration* が多くともひとつしかない．

（4） どの *label* に対しても，*P* の *proper declaration* が多くともひとつしかない．

5.2 *Creation*

〖extensible languageで，任意の *frame* を定義して用いることができる，といっても *program* を書こうとする場合に，**real**-*type* に対する＋や−まで定義しなければならないとすると不自由である．そこで，一応通常の programming language に用意されている operation は，あらかじめ準備されているものとしたい．

ALGOL 68 では，こうした標準の operation についても，その報告の中で，program の一部とする形で定義を与えている．このために，たとえば任意の **array**-*style* に関する operation が用意されたとすれば，その無限種類の *type* 毎に *body* を *elaborate* して作り出さねばならず，不要な混乱を招いている．

ALGOL N では，あらかじめ準備された operation の集合を *cosmos* と呼び，これを公理的に定義する．各々の *frame* に対する *declaration* は便宜上，*program* と同種の形を用いて記述するが，これらを *elaborate* した結果，*cosmos* が作られるのではなく，*cosmos* の構造についての単なる一記述にすぎない，という立場をとる．〗

5.2.1 特定の *program* の取扱いに先立って，最初に，前以って宣言された要素（*variable*, *skeleton*, *mark* および *mark* の順序対）とその属性の集合である「*cosmos*」が「*create*」されなければならない．

{ *cosmos* の構造は，以下に述べる条件が守られている限りこの言語内では指定されず，*standard declaration* の集合として記述される．

standard declaration の集合として推薦するものは，第 6, 7 章に掲げてある．}

cosmos Ω は，*variable* とその属性（*type*, *projec*-

* 東京大学工学部計数工学科

tion および *value*) と *quantity* の順序組の集合 Γ, *skeleton* とその属性 (*type* と *body*) の順序対の集合 Σ, *mark* とその属性 (*facing*) の順序対の集合 Φ, 2つの *mark* の順序対とその属性 (*priority*) の順序対の集合 Π, そして *label* の集合 $\mathbf{L}_0$ から成る.

{ $\Omega=\langle\Gamma, \Sigma, \Phi, \Pi, \mathbf{L}_0\rangle$ }

5.2.2 前以って宣言された *variable*

Γ は, *variable* とその *type, projection, value* および *quantity* の順序組の集合である.

Γ の要素を構成しているすべての *variable* の集合を $\mathbf{V}_0$ とする. V を $\mathbf{V}_0$ の要素とする. $t(V)$, $h(V)$, $w(V)$ と $q(V)$ は, それぞれ V の *type, projection, value* と *quantity* とする.

{ $\Gamma=\{\langle V, t(V), h(V), w(V), q(V)\rangle \mid V\in\mathbf{V}_0\}$ }

このとき, $t(V)\in\mathbf{T}$ であり, $h(V)$ と $w(V)$ とは *type* $t(V)$ の *projection* と *value* である. とくに, $t(V)$ が **procedure**-*style* なら $w(V)$ は *normal program* である.

この場合, V は *type* $t(V)$ (*projection* $h(V)$, *value* $w(V)$) として, 前以って宣言されているという.

{ 一般に, *type* $t(V)$ のどの *value* W に対しても $(h(V))(W)=w(V)$.

さらに, 一般に $t(V)$ は **procedure**-*style* である. }

$q(V)$ は *quantity* である. $q(V)$ を Q とすると,

$t(Q)=t(V)$,

$h(Q)=h(V)$,

$w(Q)=w(V)$ そして

$a(V)=on$

が, 任意の *variable* $V\in\mathbf{V}_0$ につき成立する.

$\mathbf{Q}_0$ で, $V\in\mathbf{V}_0$ なる *quantity* $q(V)$ すべての集合をあらわす.

{ $\mathbf{Q}_0=\{q(V)\mid V\in\mathbf{V}_0\}$. }

$\mathbf{V}_0$ は互いに素な2つの部分集合 $\mathbf{V}_e$ と $\mathbf{V}_h$ とから成る. $\mathbf{V}_h$ の要素を「*hidden variable*」という.

{ $\mathbf{V}_0=\mathbf{V}_e\cup\mathbf{V}_h$, $\mathbf{V}_e\cap\mathbf{V}_h=\phi$. }

5.2.3 前以って宣言された *skeleton*

Σ は *skeleton* とその *type* と *body* の順序組の集合である.

Σ の要素を構成するすべての *skeleton* の集合を $\mathbf{Z}_0$ とする. $\mathbf{Z}_0$ の要素を Z とする. $t(Z)$, $b(Z)$ を順に Z の *type* と *body* とする.

{ $\Sigma=\{\langle Z, t(Z), b(Z)\rangle \mid Z\in\mathbf{Z}_0\}$ }

Z は *compatible* である. Z を $\langle G, O\rangle$ とする. ただし, G は *frame*, O は *type-list* とする. さらに O を $(T_1, \ldots, T_n)$, $n\geq 0$ とする.

$t(Z)\in\mathbf{T}$, $b(Z)$ は *hidden variable* であり, $t(b(Z))$ は **procedure** $(T_1, \ldots, T_n)t(Z))$ である.

この場合, Z は *type* $t(Z)$ (*body* $b(Z)$) として「前以って宣言さ」れているという.

5.2.4 前以って宣言された *facing*

Φ は, *mark* とその *facing* との順序対の集合である.

Φ の要素を構成するすべての *mark* の集合を $\mathbf{M}_0$ とする. $\mathbf{M}_0$ の要素を M とする. $f(M)$ を M の *facing* とする.

{ $\Phi=\{\langle M, f(M)\rangle \mid M\in\mathbf{M}_0\}$ }

M は *mark* であり, $f(M)$ は, *double-faced, left-faced, right-faced* または *non-faced* である.

この場合, M の *facing* が $f(M)$ として「前以って宣言さ」れているという.

5.2.5 前以って宣言された *priority*

Π は, 2つの *mark* の順序対とその *priority* との順序対の集合である.

Π の要素を構成するすべての *mark* の順序対の集合を $\mathbf{N}_0$ とする. $\langle M, N\rangle$ を $\mathbf{N}_0$ の要素とする. $p(\langle M, N\rangle)$ を $\langle M, N\rangle$ の *priority* とする. ただし, M, N は *mark* である.

{ $\Pi=\{\langle M, N\rangle, p(\langle M, N\rangle)\mid\langle M, N\rangle\in\mathbf{N}_0\}$ }

$p(\langle M, N\rangle)$ は *natural* か *reverse* である.

この場合, $\langle M, N\rangle$ の *priority* は, $p(\langle M, N\rangle)$ として, 前以って宣言されているという.

さらに, 任意の *mark* M と N について,

$M\in\mathbf{M}_0$ または $N\in\mathbf{M}_0$

のとき, そしてそのときに限り

$\langle M, N\rangle\in\mathbf{N}_0$

である.

5.2.6 $\mathbf{L}_0$ はただひとつの *label* から成る. その *label* を L0 とする.

{ L0 は標準の割出し点を意味し, *program* の *elaboration* の結果 L0 が得られた後の行動がどうなるかは, **ALGOL N** の範囲外のことがらである. }

5.3 *Program* の *hidden variable* 独立性

P を (*straight language* における, *semi legal, legal* または *normal*) *program* とする. P は, 以下の条件を満たすとき, 「*hidden variable* 独立な」 *program* であるという.

$\bar{V}$ を, ⟨identifier⟩ の形をした P の *direct con-*

stituent とする.

$\bar{V}$ に対する *declaration* であるような, P の *proper declaration* が無いなら,

$V \notin \mathbf{V}_k$.

5.4 *Normalization* と *program* の *elaboration*

P を *hidden variable* 独立な *legal program* とする. $\mathbf{V}_1$ で, P の *proper declaration* によって宣言されたすべての *variable* の集合を表わす. $\mathbf{L}_1$ で, P の *proper labelling* により *label* されたすべての *label* の集合をあらわす.

core
let $\mathbf{V} \leftarrow \mathbf{V}_0 \cup \mathbf{V}_1$;
(a(V) $\leftarrow$ *off*;) for $V \in \mathbf{V} - \mathbf{V}_0$
let $\mathbf{L} \leftarrow \mathbf{L}_0 \cup \mathbf{L}_1$;
let $\mathbf{Q} \leftarrow \mathbf{Q}_0$;
r(P)$\Rightarrow P_1$;
n(P_1)$\Rightarrow P_2$;
e(P_2)
end of core

結果が *quantity* Q であるときは, かくして P の *elabolation* が終了する. 結果が *label* L であるときの P の *elabolation* は未定義である.

5.5 *Extended language* における *program*

extended language における 〈expression〉 を「*extended language* における *program*」という. *extended language* におけるどの *program* P に対しても, *straight language* における *program* P' があって, **1.3.1**～**1.3.22** の規則のいくつかを順に適用することにより, P' から P が得られる. この場合 P' を,「P の *straight form*」であるという.

extended language における *program* の *normalization* と *elaboration* とは, その *straight form* に対するものによって定義する.

1.3.19 が適用されたときには, *extended language* における *program* の *straight form* は一意的には決らない. しかしながら, *parameter-mechanism* により, これらのいくつかの *straight form* の *elabolation* の効果は, ほとんど等価である.

6. *Standand declarations*

この章では, **ALGOL N** を使用する場合に, 後立てとなる *standard declaration* として推奨する組 **SD** を記述する. **SD** は, **ALGOL N** の一般的な使い方に対応する *cosmos* の構造を規定している.

SD の要素は, 〈declaration〉 であり, その左方に "＊" がつくこともある. そして (**SD** *i*. *j*. *k*) の形で引用番号がつけられている.

SD の各〈declaration〉は, *normal* な形で書かれていなければならない. すなわち, 〈variable declaration〉の場合には "**be**" の右辺は *normal* でなければならないし, 〈formula declaration〉の場合には "**represent**" の右辺は 〈identifier〉 でなければならない.

以上の制約事項にかかわらず, 続み易くするためにここでの記述には多くの *mark* と *formula* とが用いられている. さらに, *extended language* の形さえ用いられているが, もとの *normal* な形へ再構成することは容易である.

特に, 〈formula declaration〉

"**let** 〈frame〉 G **represent** 〈expression〉 F" は, F が 〈identifier〉 でない場合には,

＊"**let** V **be** F" と

"**let** G **represent** V"

の 2 つの 〈declaration〉 におきかえられる. ただし V は, 新たに *generate* された *variable* である.

SD の要素が, 〈variable declaration〉

"**let** 〈identifier〉 V **be** 〈expression〉 F"

である場合を考える.

F を *elaborate* して得られる *quantity* を Q とする.

このとき, この 〈declaration〉 は,

$\langle V,\ \mathrm{t}(Q),\ \mathrm{h}(Q),\ \mathrm{w}(Q),\ Q \rangle \in \boldsymbol{\Gamma}$

＊ つきなら $V \in \mathbf{V}_k$

＊ なしなら $V \in \mathbf{V}_e$

{ 多くの場合, F は 〈procedure notation〉 で空でない 〈procedure donor〉 をもつ. したがって, t(Q), h(Q), w(Q) は, *elaboration* 無しで直接 F から得られる. }

SD の要素が, 〈formula declaration〉

"**let** 〈frame〉 G **represent** 〈identifier〉 V" ただし, t(V) が

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

の形をしている場合を考える.

このとき, この 〈declaration〉 は

$\langle \langle G, (T_1, \ldots, T_n) \rangle,\ T,\ V \rangle \in \boldsymbol{\Sigma}$ を意味する.

SD の要素が, 〈mark declaration〉

"let ⟨mark⟩ *M* operate
⟨left priority⟩ *Z* ⟨right priority⟩ *Z'*"
である場合を考える.

この ⟨declaration⟩ は,

(1) *M* の *facing* (f(*M*))

(2) 任意の ⟨mark⟩ *N* について, これが ⟨*M*, *N*⟩ または ⟨*N*, *M*⟩ に対する *reverse declaration* であるか否か

を, **2.4.1** と同様に意味する.

そして, このとき

⟨*M*, f(*M*)⟩∈**Φ**

M と *N* を ⟨mark⟩ とする. ⟨*M*, *N*⟩ に対して *reverse declaration* が **SD** 中にあれば,

⟨⟨*M*, *N*⟩, *reverse*⟩∈**Π**.

⟨*M*, *N*⟩ に対して *reverse declaration* が **SD** 中になく, *M* または *N* の *facing* が **SD** 中で *declare* されているなら,

⟨⟨*M*, *N*⟩, *natural*⟩∈**Π**.

6.1 Basic formulas

(SD 1.0.1)
let dummy, nil operate before all left after right

(SD 1.0.2)
let type, copy, new, enproc, enref operate before all left after right

(SD 1.0.3)
let deref operate before all left

(SD 1.0.4)
let match, as operate after all right

(SD 1.0.5)
let else operate before then left after right

(SD 1.0.6)
let if operate before all left

(SD 1.0.7)
let then operate after right

(SD 1.0.8)
let ←, := **operate before then, else,** , **left after right**

(SD 1.0.9)
let ≡, ≠, =, ≠ **operate before then, else** , , ←, := **left after** $^{-1}$, $^{-}$ **right**

(SD 1.1.1)
let dummy represent procedure( ): *dummy*

(SD 1.1.2)
∗**let** *dummy* **be effect**

{ **(SD 1.1.1)** は,

"**let dummy represent procedure**()**effect:**()**begin** *dummy* **end**"

を, *extended language* で書いたものである.

effect と **dummy** との相違は, *quantity* を新たに *generate* するか, しないか, にある. }

(SD 1.2.1)
let nil represent procedure()**reference:** *nil*

(SD 1.2.2)
∗**let** *nil* **be begin let** *a* **be reference;**
a ← **enref** *a*;
a **end**

(SD 1.3.1) *T*∈**T** とする.
let type()**represent**
procedure(*T* **name** *a*)(*T*): (*T*)

(SD 1.4.1) *T*∈**T** とする.
let copy()**represent**
procedure(*T* **quantity** *a*)(*T*):
code (*p*1: *a*)(*T*):
core let *Q* ← parameter;
g(*Q*) ⇒ *Q'*;
t(*Q'*) ← *T*;
h(*Q'*)← h(*Q*[*p*1:]);
w(*Q'*)← w(*Q*[*p*1:]);
⇒*Q'*
end of core

(SD 1.5.1) *T*∈**T**, *T*∉**T** [**array**]∪**T** [**structure**] とする.
let new() **represent**
procedure(*T* **name** *a*)(*T*): (**copy** *a*)

(SD 1.5.2) *T*∈**T** [**array**] とする.
let new()**represent**
procedure(*T* **name** *a*)(*T*):
begin let *u* **be lower bound** *a*;
let *v* **be upper bound** *a*;
let *i* **be** *u*−*1*;
array [*u*: *v*]
(**begin** *i*:=*i*+*1*; **new** *a*[*i*] **end**)
end

(SD 1.5.3) *T* を "**structure** (*S*$_1$*T*$_1$, *S*$_2$*T*$_2$, ..., *S*$_n$ *T*$_n$)" とする. *S*$_1$, *S*$_2$, ..., *S*$_n$ は ⟨selecter⟩, *T*$_1$, *T*$_2$, ..., *T*$_n$∈**T** である.

let new() **reperesent**
procedure(T **name** a)(T):
structure (S_1 **new** a[S_1],
S_2 **new** a[S_2],
. . .
S_n **new** a[S_n])

{ **type, copy, new** はおのおの *parameter* と同じ *type* の *quantity* を *generate* する．その違いは，その *type* の標準の *quantity* であるか，*parameter* と *surface value* の同じ *quantity*であるか，あるいは *parameter* と *deep value* の同じ *quantity* であるか，にある． }

(SD 1.6.1) $T\in$**T**, T' を "**procedure**()(T)" とする．

let enproc() **represent**
procedure(T **name** a)(T'):
procedure()(T): ( ) a

(SD 1.7.1) $T\in$**T** とする．

let enref() **represent**
procedure(T **quantity** a)**reference**:
code(*p1*: a)**reference**:
core let Q ← parameter;
g(**Q**) ⇒ Q';
t(Q') ← **reference**;
h(Q') ← H0 [**reference**];
w(Q') ← {Q[*p1*:]};
⇒Q'
end of core

(SD 1.8.1) $T\in$**T** とする．

let deref()**match**() **represent**
procedure(**reference** a, T **name** b)**Boolean**:
code(*p1*: a)**Boolean**:
core let Q ← parameter;
let w(Q[*p1*:])≡{Q'};
if t(Q')≡T
then → next, else → $L1$;
let W ← **1**;
→ $L2$;
$L1$: let W ← **0**;
$L2$: g(**Q**) ⇒ Q';
t(Q') ← **bits**;
h(Q') ← H2 [**bits**];
w(Q') ← W;
⇒ Q'
end of core

{ *parameter a* が refer している *quantity* が，*parameter b* の *type* と同じ *type* を有するとき **true**，そうでなければ **false**. }

(SD 1.8.2) $T\in$**T** とする．

let deref()**as**() **represent**
procedure(**reference** a, T **name** b)(T):
if deref a **match** b
then code(*p1*: a)(T):
core let Q ← parameter;
let w(Q[*p1*:])≡{Q'};
⇒ Q'
end of core
else type T

[enref(), **deref**()**as**() は，**reference-***type* における基本的な operation である．他の *type*，たとえば，**array-***style* を取り上げて考えてみると，それぞれ〈array donor〉，〈array element〉に匹敵するものである．

「*program* の *type* に関しての静的な検査が完全にできること」を目標とすれば，**reference-***type* はそれが refer している *type* に応じて別の *type* としなければならない．この方針をとると，list 処理をしようという場合に，たとえば linear list に対応する *type* T は，T=**structure**(*flag*: **string**, *chain*: **reference** T) というふうに回帰的にしか定義できないことが生ずる．

ALGOL 68 では，こうした回帰的に定義されるものをも *type* として認めている．このため，*type* の識別は，いわば context free language の同値問題にあたる操作を要することになり，その厳密な定義を与えることにおいても **ALGOL 68** の報告は完全でない．

ALGOL N では，**reference-***type* を唯一のものとして，*type* の回帰的定義を取り除いた．同時に，**reference** をはがす operation については，その結果得られるであろう *type* を別に明記しない限り，*type* の静的検査は不可能となった．

これは他の *type* に対する operation とは大きく異なる．そこで **reference-***type* に関する operation は，すべて *standard declaration* においてのみ，定義することにした. **]**

(SD 1.9.1) $T\in$**T** とする．

let ()**else**() **represent**
procedure(T **name** a, T **name** b)

```
      structure (then:enproc T, else:enproc T)
    : structure (then:enproc a, else:enproc b)
```

(SD 1.10.1) $T\in$**T** とする.

```
  let if()then() represent
      procedure(bits a, (T else T) b)(T):
      code(p1:a, p2:b)(enproc T):
        core let Q ← parameter;
             let B ← w(Q[p1:]);
             let n ← l(B);
             if n≠0 then → next, else → L1;
             (if B[i]=1 then → L2,
                        else → next;)
                        for i=1, 2, ..., n
          L1: ⇒ Q[p2:] [else:]
          L2: ⇒ Q[p2:] [then:]
        end of core
    ( )
```

(SD 1.10.2)

```
  let if () then () represent
      procedure (bits a, effect name b):
        (if a then b else dummy)
```

(SD 1.11.1) $T\in\boldsymbol{T}$ とする.

```
  let ()←() represent
      procedure(T quantity a, T quantity b):
      begin code(p1:a, p2:b)(T):
        core let Q ← parameter;
             let Qa ← Q[p1:];
             let Qb ← Q[p2:];
             let H ← h(Qa);
             p(H, W) if W' → next, L ⇒ L;
             w(Qa) ← W';
             ⇒ Qa
        end of core;
        dummy
        end
```

(SD 1.12.1) $T\in$**T**, $T\notin$**T[array]** とする.

```
  let ( ):=( ) represent
      procedure(T quantity a, T quantity b):
        (a ← new b)
```

(SD 1.12.2) $T\in$**T[array]** とする.

```
  let ( ):=( ) represent
      procedure(T quantity a, T quantity b):
      begin let ua be lower bound a;
            let ub be lower bound b;
            let va be upper bound a;
            let vb be upper bound b;
            let u, v, i be Integer;
            u:=if ua≥ub then ua else ub;
            v:=if va≥vb then va else vb;
            for i:=u step 1 until v do
              a[i]:=b[i]
      end
```

{ ()←() と ():=() とのちがいは, assign されるものが, *surface value* であるか, *deep value* であるか, にある. }

(SD 1.13.1) $T\in$**T** とする.

```
  let ( )≡( ) represent
      procedure ((T) a, (T) b) Boolean:
      code(p1:a, p2:b) Boolean:
        core let Q ← parameter;
             if w(Q[p1:])=w(Q[p2:])
              then → next, else → L1;
             let W ← 1;
             → L2;
        L1: let W ← 0;
        L2: g(Q) ⇒ Q';
             t(Q') ← bits;
             h(Q') ← H2 [bits];
             w(Q') ← W;
             ⇒ Q'
        end of core
```

(SD 1.13.3) $T\in\boldsymbol{T}$ とする.

```
  let ( )≠( ) represent
      procedure((T) a, (T) b) Boolean:
        (¬ a≡b)
```

(SD 1.14.1) $T\in$**T**, $T\notin$**T[array]**, $T\in$**T[structure]** とする.

```
  let ( )=( ) represent
      procedure ((T)a, (T)b) Boolean: (a≡b)
```

(SD 1.14.2) $T\in$**T[array]** とする.

```
  let ( )=( ) represent
      procedure ((T)a, (T)b) Boolean:
      begin let t be true;
            let ua be lower bound a;
            let va be upper bound a;
            if (ua=lower bound b) ∧
              (va=upper bound b)
            then begin let i be Integer;
```

```
        for i:=ua step 1 until va do
            t:=t ∧ (a[i]=b[i])
        end
      else t:=false;
      t
  end
```

(SD 1.14.3) T を "**structure** (S_1T_1, S_2T_2, ..., S_nT_n)" とする. S_1, S_2, ..., S_n は ⟨selector⟩, T_1, T_2, ..., $T_n \in \mathbf{T}$ である.

```
let ( )=( ) represent
    procedure((T) a, (T) b)Boolean:
    begin let t be true;
        t:=t ∧ (a[S1]=b[S1]);
        t:=t ∧ (a[S2]=b[S2]);
        . . .
        t:=t ∧ (a[Sn]=b[Sn]);
        t
    end
```

(SD 1.14.4) $T \in \boldsymbol{T}$ とする.

```
let ( )≠( ) represent
    procedure((T) a, (T) b)Boolean:
        (¬ a=b)
```

{( )≡( ), ( )≠( ) と ( )≠( ) とは, それぞれ *surface value* が等しいか否か, *deep value* が等しいか否かに応じて, **true** または **false** となる. }

(昭和47年5月24日 受付)