講　　座

ALGOL N について

(III) Semantical Notion

佐久間　紘　一*

3. はじめに

〖前前回，前回と2回にわたり，和田英一により *ALGOL N* の報告第2版の **1.** と **2.** の紹介がされた．今回は **3.** 全体を日本語訳を中心として解説する．**3.** では *elaboration* を記述するのに使用される概念，およびそれらに関する記法を定義する．〗

3.1 *Quantity*

Quantity は抽象的な対象であり，*program* の *elaboration* の経過を記述するのに導入されている．おのおのの *quantity* は，その *type*, *value*, *projection* を持つ．Q が *quantity* のとき，$\mathrm{t}(Q)$ によりその *type* を，$\mathrm{w}(Q)$ によりその *value* を，$\mathrm{h}(Q)$ によりその *projection* を表わす．

〖$\bar{E}$ を *legal program* P の *direct constituent* とする．そのときに $\bar{E}$ の *elaboration* の結果，*quantity* か *label* が得られる．*quantity* Q が得られたのなら，その *type* $\mathrm{t}(Q)$ は *type* $\mathrm{t}(\bar{E})$ と同じものになる．これの当然の帰結として $\bar{E}$ の *elaboration* の結果得られる *quantity* の *type* は，コンパイル時に決定する．〗

3.2 Value

Value は，*program* の *elaboration* の主目的である．

おのおのの *type* T は，T に特有の *value* の集合 $\mathbf{W}[T]$ を持つ．

3.2.1 **effect**-*type*

Type **effect** には，ただ一つの *value* である *done* しかない．すなわち

$$\mathbf{W}[\mathbf{effect}]=\{done\}$$

である．

〖*done* は，すべての *type* を統一的に扱うために，人為的に導入されている．すなわちすべての *type* が，*value* を持つようにするために，*done* が導入されている．

この *value* を無視すれば，**effect**-*type* を，*value* を持たない *type* ということができる．さらにこのような奇形な *type* が望ましくなければ，*type* **effect** の 〈expression〉 を，"*type* の無い 〈expression〉"，すなわち "*statement*" と解釈することもできるであろう．〗

3.2.2 **real**-*type*

Type **real** の *value* は，通常の意味での実数である．

1969, *−32768*, *3.14159*

のような通常の10進表示の他に，次のような記法を使用する：

Rで，すべての実数の集合を表わす．すなわち．

$$\mathbf{W}[\mathbf{real}]=\mathbf{R}.$$

Iで，**R**の部分集合としての，すべての整数の集合を表わす．

$0(R)$ で，実数 R を丸めることによって得られた整数を表わす．

〖〔 〕をガウスの記法，すなわち $〔R〕$ は実数 R をこえない最大の整数としたときに

$$0〔R〕=〔R+0.5〕.$$〗

$\mathrm{w}(J)$ で，〈number〉 J によって表現される実数を表わす．これは次のように厳密に定義される：

(1) J が 〈digit〉 のときには

$\mathrm{w}(0)=0$,
$\mathrm{w}(1)=1$,
$\mathrm{w}(2)=2$,
$\mathrm{w}(3)=3$,
$\mathrm{w}(4)=4$,
$\mathrm{w}(5)=5$,
$\mathrm{w}(6)=6$,
$\mathrm{w}(7)=7$,

* 京都産業大学理学部計算機科学科

$$w(8)=8,$$
$$w(9)=9,$$

とする．

（2） J が，$n\geq 2$ で

“⟨digit⟩X_1⟨digit⟩X_2……⟨digit⟩X_n”

の形のとき

$$w(J)=w(X_1)\times 10^{n-1}+w(X_2)\times 10^{n-2}+\cdots\cdots+w(X_n)$$

とする．

（3） J が，$n\geq 1$ で

“.⟨digit⟩X_1⟨digit⟩X_2……⟨digit⟩X_n”

の形のとき

$$w(J)=w(X_1)\times 10^{-1}+w(X_2)+10^{-2}+\cdots\cdots+w(X_n)\times 10^{-n}$$

とする．

（4） J_1 が

“⟨digit⟩$_{\circ\circ\circ}$”

の形で，かつ J_2 が

“.⟨digit⟩$_{\circ\circ\circ}$”

の形であるとき，J が

“J_1J_2”

という形であれば，

$$w(J)=w(J_1)+w(J_2)$$

とする．

（5） J_1 が

“⟨digit⟩$_{\circ\circ\circ}$”

の形であるとき，J が

“$_{10}+J_1$”

という形であれば

$$w(J)=10^w$$

とする．ただしここで

$$w=w(J_1)$$

とする．

（6） J_1 が

“⟨digit⟩$_{\circ\circ\circ}$”

の形であるとき，J が

“$_{10}-J_1$”

の形であれば

$$w(J)=10^{-w}$$

とする．ただしここで

$$w=w(J_1)$$

とする．

（7） J_1 が

“⟨digit⟩$_{\circ\circ\circ}$”

の形で，かつ J_2 が

“$(_{10}+|_{10}-)$⟨digit⟩$_{\circ\circ\circ}$”

の形であるとき，J が

“J_1J_2”

の形であれば

$$w(J)=w(J_1)\times w(J_2)$$

とする．

〖*Type* **real** は，通常の算術計算および配列の添字のために用意されている．〗

3.2.3 **bits**-*type*

Type **bits** の *value* は，“*bit-string*” である．*Bit-string* は

“$b_1b_2\cdots\cdots b_n$”

なる有限列である．ここで n は 0 以上の整数であり，“この *bit-string* の長さ” という．また $i=1, 2, \cdots$, n の各 b_i は “*bit*”，すなわち **0** あるいは **1** である．

次の記法を使用する：

$B[i]$ で *bit string* B の，i 番目の *bit* を表わす．すなわち B が

$$b_1b_2\cdots\cdots b_n$$

という上に説明した *bit-string* であり，i が $1\leq i\leq n$ の整数のときは

$$B[i]=b_i$$

である．

B ですべての *bit-string* の集合を表わす．すなわち

$$\mathbf{W}[\mathbf{bits}]=\mathbf{B}$$

である．

$n\geq 0$ の整数に対し $\mathbf{B}[n]$ で，長さ n のすべての *bit-string* の集合を表わす．

$l[B]$ で，*bit-string* B の長さを表わす．

BO で，長さ *0* の *bit-string* を表わす．

$w(J)$ で，⟨bits⟩J によって表現される *bit-string* を表わす．これは次のように定義される．

（1） $w(\mathbf{0})=\mathbf{0}$,

（2） $w(\mathbf{1})=\mathbf{1}$,

（3） $n\geq 2$ で，$i=1, 2, \cdots\cdots, n$ で X_i が **0** か **1** のときに，J が

“$X_1X_2\cdots\cdots X_n$”

の形であれば

$$w(J)=w(X_1)\,w(X_2)\cdots\cdots w(X_n)$$

とする．

〖*Type* **bits** は，比較式などの判定の結果を表示するために用意されている．〗

3.2.4 **string**-*type*

Type **string** の *value* は，"*string*"（すなわち"文字列"）である．*String* は

"'$c_1c_2\cdots\cdots c_n$'"

なる有限列である．ここで n は0以上の整数であり，"この *string* の長さ"という．また c_i は *character* である．

すべての *character* の集合と〈character〉X という形をしたすべての基本記号の集合との間には，1対1の対応があり，その対応を $c(X)$ で表わす．

次の記法を使用する：

$C\langle i\rangle$ で *string* C の i 番目の *character* を表わす．すなわち C が

'$c_1c_2\cdots\cdots c_n$'

という上に説明した *string* であり，i が $1\leq i\leq n$ の整数のとき

$$C\langle i\rangle = c_i$$

とする．

C ですべての *string* の集合を表わす．すなわち

$$\mathbf{W}\langle\mathbf{string}\rangle = \mathbf{C}$$

である．

$n\geq 0$ の整数に対し $\mathbf{C}\langle n\rangle$ で，長さ n のすべての *string* を表わす．

l(C) で，*string* C の長さを表わす．

CO で，長さ 0 の *string* を表わす．

␣ で，c(へ) を表わす．

w(J) で，〈string〉J により表現される *string* を表わす．すなわち次のように定義される：

(1) $\mathrm{w}(\text{`\ '}) = \mathrm{CO}$,

(2) $n\geq 1$ で，$i=1,2,\cdots\cdots,n$ で X_1 が〈character〉のとき，J が

"'$X_1X_2\cdots\cdots X_n$'"

の形であれば

$$\mathrm{w}(J) = \mathrm{c}(X_1)\mathrm{c}(X_2)\cdots\cdots\mathrm{c}(X_n)$$

とする．

［*Type* **string** は，入出力での *character* を扱うために用意されている．］

3.2.5 **reference**-*type*

Type **reference** の *value* は，*quantity* である．

［同じ *variable* に，*elaboration* の進行にともないいくつかの異なった *type* の *value* を assign することを許すために，ALGOL **68** では *union* というものを導入した．しかし，*union* を言語の中に入れると複雑になり，また *union* を使用した *program* も，*type* **reference** があれば *union* 無しでも書けると考え，ALGOL N には *union* の方法を取り入れていない．

Type **reference** は他に，**structure**-*style* の中で使用し，リストを作るのにも使用される．アドレスとして用意されていると考えればよい．］

3.2.6 **array**-*style*

T' がある *type* のとき T を

"**array** 〔 〕 T'"

の形をした *type* とする．

(1) 空集合 ϕ は，*type* T の *value* である．

(2) v と u を $v\leq u$ なる整数とし，$i=v, v+1, \cdots\cdots, u$ で Q_i が *type* T' の quantity とする．そのとき集合

$$\{\langle v, Q_v\rangle, \langle v+1, Q_{v+1}\rangle, \cdots\cdots, \langle u, Q_u\rangle\}$$

は，*type* T の *value* であり，v はこの *value* の"下限"といい，u はこの *value* の"上限"という．

次の記法を使用する：

v(W) で，W の"下限"を表わす．

u(W) で，W の"上限"を表わす．

$W\langle i\rangle$ で，W の i 番目の要素を表わす．

これらは次のように定義される：

(1) v と u は $v\leq u$ なる整数で，$Q_v, Q_{v+1}, \cdots\cdots, Q_u$ が *type* T' の *quantity* で

$$W = \{\langle v, Q_v\rangle, \langle v+1, Q_{v+1}\rangle, \cdots\cdots, \langle u, Q_u\rangle\}$$

のとき

$$\mathrm{v}(W) = v,$$
$$\mathrm{u}(W) = u,$$
$$W\langle i\rangle = Q_i \quad (v\leq i\leq u \text{ の整数 } i \text{ に対し})$$

とする．

(2) $W=\phi$ のときには

$$\mathrm{v}(W) = 1$$
$$\mathrm{u}(W) = 0$$

とする．

［(1) の場合で $i<v$ あるいは $u<i$ のとき，および (2) のときには $W\langle i\rangle$ は定義されない．］

Q が **array**-*style* の *quantity* で $W=\mathrm{w}(Q)$ であり，i が整数のとき $Q\langle i\rangle$ で $W\langle i\rangle$ を表わす．

［**array**-*style* は，同じ *type* のものを一つにまとめるために用意されている．

集合

$$\{\langle v, Q_v\rangle, \langle v+1, Q_{v+1}\rangle, \cdots\cdots, \langle u, Q_u\rangle\}$$

が，ある **array**-*style* の *type* **array** 〔 〕 T' の *value* であるとき，必ずしも各 Q_i の *projection* は一致するとは限らない．

array (**array** (*1*, *2*, *3*)
array (*4*, *5*)
array (*6*))

と書くことにより

1, *2*, *3*
4, *5*
6

のような三角行列を作ることもできる.〕

3.2.7 structure-*style*

$n \geq 0$ で, $S_1, S_2, \cdots\cdots S_n$ は互いに相異なる〈selector〉とし, $T_1, T_2, \cdots\cdots, T_n$ を *type* とし, T を

"**structure** (S_1T_1, ……, S_nT_n)"

の形の *type* とする.

$i=1, 2, \cdots\cdots, n$ で, Q_i を *type* T_i の *quantity* とする. そのとき集合

$\{\langle S_1, Q_1\rangle, \langle S_2, Q_2\rangle, \cdots\cdots, \langle S_n, Q_n\rangle\}$

は *type* T の *value* である.

〔$n=0$ の場合の { } は空集合を表わす.〕

次の記法を使用する:

$W[S]$ で, S により選ばれた W の要素を表わす. すなわち W が

$\{\langle S_1, Q_1\rangle, \langle S_2, Q_2\rangle, \cdots\cdots, \langle S_n, Q_n\rangle\}$

という上に説明した集合であり, $1 \leq i \leq n$ のある整数 i があって, S が S_i であるとき

$W[S]=Q_i$

とする.

〔S が $S_1, S_2, \cdots\cdots, S_n$ のどれとも異なるか, $n=0$ のときには, $W[S]$ は定義されない.〕

$W=\mathrm{w}(Q)$ であり Q が **structure**-*style* の *quantity* で, S が〈selector〉のとき, $Q[S]$ で $W[S]$ を表わす.

A が

"(〈letter〉|〈digit〉)$_{\infty\infty}$"

の形で〈selector〉S が

"A:"

の形のとき

$W[A:]$ および $Q[A:]$

はおのおの

$W[A]$ および $Q[A]$

と略することができる.

〔**structure**-*style* は, いくつかの異なる *type* のものを一つにまとめるために用意されている.〕

3.2.8 procedure-*style*

$n \geq 0$ で, $T_1, T_2, \cdots\cdots T_n, T'$ を *type* とするとき, T を

(**procedure** (T_1, ……, T_n) T')

の形の *type* とする.

J を空でない〈procedure donor〉とする. そのとき〈procedure notation〉

"**procedure** (T_1, ……, T_n) $T'J$"

は *type* T の *value* である.

〔1. 上の *type* T の *value* W は

W が〈mark〉を含まず, かつ

W が条件 (L 1) を除き *legal program* のすべての条件を満たすとき, *well formed* であるという.

procedure-*style* の *value* は, 次のいずれかにより *quantity* に assign される:

(1) *standard variable declaration* により;

(2) 〈procedure notation〉の *elaboration* により;

(3) 〈code〉の *elaboration* により;

(4) 〈procedure call〉の *elaboration* により.

P が *legal program* で, P に含まれているおのおのの〈code body〉は (通常の意味で)「正しい」ものとする. そのとき P の *elaboration* のときに, *quantity* に assign される **procedure**-*style* の *value* は, *well-formed* のものだけである.

2. *Value* W が

"**procedure** (T_1, ……, T_n) T
:(V_1, ……, V_n) E"

のとき, $V_1, V_2, \cdots\cdots, V_n$ を W の「*formal parameter*」と呼び, E を W の「*procedure body*」と呼ぶ.

3. Q を *quantity* とする.

Q の "*deep value*" は次のように再帰的に定義される:

(1) Q の *type* が **arry**-*style* でも **structure**-*style* でもないときには, $\mathrm{w}(Q)$ が Q の *deep-value* である.

(2) Q が **array**-*style* で, $\mathrm{w}(Q)$ が

$\{\langle v, Q_v\rangle, \langle v+1, Q_{v+1}\rangle, \cdots\cdots, \langle u, Q_u\rangle\}$

の形で, かつ Q_i の *deep value* が w_i $(i=v, v+1, \cdots\cdots, u)$ のとき, Q の *deep value* は

$\{\langle v, w_v\rangle, \langle v+1, w_{v+1}\rangle, \cdots\cdots, \langle u, w_u\rangle\}$

である.

(3) Q が **structure**-*style* で, $\mathrm{w}(Q)$ が

$\{\langle S_1, Q_1\rangle, \langle S_2, Q_2\rangle, \cdots\cdots, \langle S_n, Q_n\rangle\}$

の形で, かつ Q_i の *deep-value* が w_i $(i=1, 2, \cdots\cdots, n)$ のとき, Q の *deep value* は

$\{\langle S_1, w_1\rangle, \langle S_2, w_2\rangle, \cdots\cdots, \langle S_n, w_n\rangle\}$

である．

ときどき，*value* w(Q) を，*deep value* に対して「*surface value*」と呼ぶ．

Q_1 と Q_2 を，**array** 〔 〕 **real** の *quantity* とし，Q_1 の *value* を

$\{\langle 1, 1\rangle, \langle 2, 2\rangle, \langle 3, 3\rangle\}$

とする．このときに Q_1 の *surface value* を Q_2 に assign した後に，$Q_2[1]$ に 4 を assign すれば，$Q_1[1]$ の *value* は，やはり 4 になる．これに対し，Q_1 の *deep value* を Q_2 に assign した後に，$Q_2[1]$ に何を assign しようが，$Q_1[1]$ の *value* は，1 のままである．

4. 〈procedure call〉 の *elaboration* のときの parameter-mechanism は，*ALGOL 60* でと同じ意味での「*call-by-name*」である．しかし他の種類の parameter-mechanism も，容易に指定できる．

4.1 **procedure**-*style* の *value* が

"**procedure** ($T_1, \cdots\cdots, T_n$) T:
($V_1, \cdots\cdots, V_n$)
begin let U_{i_1} **be** V_{i_1} ;
.....
let U_{i_k} **be** V_{i_k} ; E **end**"

の形とする（ここで $1 \leq i_1 \leq i_2 \leq \cdots\cdots \leq i_k \leq n$ とする）．

この *value* を *actual parameter* $F_1, F_2, \cdots\cdots, F_n$ で呼んだら，〈expression〉

"**begin let** U_{i_1} **be** F_{i_1};
.....
let U_{i_k} **be** F_{i_k}, E **end**"

が *elaborate* される．これは

"U_{ij} で F_{ij} で表わされる *quantity* を代表させる ($j=1, 2, \cdots\cdots, k$)．そして E を *elaborate* する"

ということを意味する．

U_{ij} と F_{ij} のこのような parameter-mechanism を "*call-by-quantity*"（**1.3** 参照）と呼ぶ．この parameter-mechanism は，通常の *FORTRAN-compiler* の parameter-mechanism とほとんど同じものである．

4.2 U と F の parameter-mechanism が，*call-by-quantity* の形で指定されていて，F が

"**copy** F'"

あるいは

"**new** F'" の形のときには，**copy** F' あるいは **new** F' のどちらが使用されているかにより F' で表現される *quantity* のおのおのの *surface value* あるいは *deep value* によって新たに *generate* された *quantity* を U が代表する．

U と F' のこのような parameter-mechanism を，"*call-by-*(*surfcce* あるいは *deep*) *value*" という．]

3.3 Implementation dependent factor

言語の通常の implementation においては，必ずしもすべての *value* が実現できるのではなく，多くの *value* は，その implementation によるある種の方法により bias される．このような bias は，ある種の *projection* の自動的な作用として，形式化することができる（**3.4, 6.** 参照）．これらの特種な *projection* は，おのおのの implementation により定まる様々な factor に合うようにすることができる．この factor のある特別な場合として，*value* にいかなる bias も与えない恒等 *projection* の理想的な場合もありうる．

これらの factor の一つは，実現可能な *value* の集合であり，この集合の要素を，"*available*" *value* と呼ぶ．

[*Available value* は，その implementation によって用意されている形式（たとえば 2 進補数表示で 32 bit というような形式）で表現できる *value* である．整数を例とすれば，絶対値があまりにも大きくないものは，多くの implementation において *available* な整数である．絶対値があまりにも大きい場合には，bias を受け，0 になる implementation もあるだろうし，その付号はそのままとし，絶対値は *available value* の中で一番大きい絶対値にする implementation もあるだろう．]

次の記号は，*available value* に密接に関係する implementation dependent factor のいくつかを表わす．

R1 で，充分大きい絶対値を持つ負の数か $-\infty$ を表わす．それはすべての *available* な実数の下限として働く（**3.4.3** 参照）．

R2 で，充分大きい正の数か $+\infty$ を表わす．それはすべての *available* な実数の上限として働く（**3.4.3** 参照）．

R3 で，充分小さな正の数か 0 を表わす．それは *available* な実数の，標準的な精度として働く（**3.4.3** 参照）．

I1 で，充分大きな絶対値を持つ負の数か $-\infty$ を表わす．それはすべての *available* な整数の下限として働く（**3.4.4** 参照）．

I2 で，充分大きい正の整数か＋∞を表わす．それはすべての *available* な整数の上限として働く (**3.4.4** 参照)．

I3 で，充分大きい正の整数か＋∞を表わす．それは *available bit-string* の *length* の上限として働く (**3.4.5** 参照)．

I4 で，充分大きい正の整数か＋∞を表わす．それは *wailable string* の *length* の上限として働く (**3.4.7** 参照)．

K で *character* の集合を表わす．それは *available character* の集合として働く．

Implementation dependent factor のためのこれらの記号の他に，次の記法を使用する．

C′ で **K** の *character* からなるすべての *string* の集合を表わす．

C′〔*n*〕で，**K** の *character* よりなる *length n* のすべての *string* の集合を表わす．ここで *n* は，$n \geq 0$ の整数とする．

3.4 *Projection*

3.4.1 *Q* が *quantity* で，その *type* t(*Q*) が *T* のとき，*projection* h(*Q*) は，次の意味で *type T* の *projection* である：それは *type T* の任意の *value* に適用できる operation であり，そのおのおのの適用の結果は，同じ *type* の *value* か，(時によると) *label* になる．

W が *tpye T* の *value* であり，*H* が *type T* の *projection* であるとき

p(*H*, *W*)

は，*W* に *H* を適用することを意味する．結果が *value W′* である．あるいは，結果が *label L* であることを示すために，それぞれ記法

p(*H*, *W*)⇒*W′*

あるいは記法

p(*H*, *W*)⇒L

が使用される．*H* が *W* にいつ適用するかに無関係に，結果が *H* と *W* によって定まる場合には

H(*W*)

で，その結果を示す．

[[*Type T* の *value W* が，*type* が *T* である *quantity Q* に代入されるときには，*projection* h(*Q*) が *W* に自動的に適用され，もし p(h(*Q*), *N*)⇒*W′* なら *Q* の新しい *value* は *W′* になり，もし p(h(*Q*), *W*)⇒*L* なら代入は中断される．p(h(*Q*), *W*)⇒L とは，*label L* の宣言されているところに飛ぶことを示す．]]

3.4.2 任意の *type T* に対し，ある定まった *projection* HO〔*T*〕が *implementation* により用意されており，それは *type T* の *available value W* に対し，条件

HO〔*T*〕(*W*) = *W*

を満たす．

real, bits, string 以外の任意の *type T* に対しては，いかなる *projection* も，*program* の中に explicit に指定することはできず，HO〔*T*〕だけが使用できる．**real, bits, string** の *type* に対しては，〈real notation〉，〈bits notation〉あるいは〈string notation〉によって表現された *quantity* は，その〈modifier〉により指定された *projection* を持つことになる．

T が **real, bits** あるいは **string** の場合には，*T* のためにさらに H1〔*T*〕, H2〔*T*〕の二つの *projection* が，implementation により用意される (**3.4.3**～**3.4.8** 参照)．

3.4.3 H1〔**real**〕で，*type* **real** のためのある定まった *projection* を表わす．

W を H1〔**real**〕の値域，すなわち H1〔**real**〕により得られるすべての *value* の集合とする．そのとき **W** は，*available* な実数の集合である．さらに

(1) R1 は **W** の下限

(2) R2 は **W** の上限

(3) $0 \neq x \in \mathbf{W}$, $0 \neq y \in \mathbf{W}$, $x < y$ で $x < z < y$ なる **W** の元 *z* が存在しなければ

$$y - x < \frac{1}{2}(|x| + |y|) \times \mathrm{R3}$$

とする．

R が実数で

R1 < *R* < R2

なら，H1〔**real**〕(*R*) はある適当な丸めの操作により *R* から得られた **W** の中のある *value* である．

R が実数で

R ≤ R1 あるいは R2 ≤ *R*

のときには，H1〔**real**〕(*R*) は，定まっていない **W** のある元になる．

3.4.4 H2〔**real**〕で，*type* **real** のためのある定まった *projection* を表わす．

W を H2〔**real**〕の定義域とする．そのとき **W** は *available* な整数の集合になる．さらに

$\mathbf{W} = \{I \mid I \in \mathbf{I} \wedge \mathrm{I1} \leq I \leq \mathrm{I2}\}$

となる.

R が実数で

I1≤*R*≤I2

のとき, H2〔**real**〕(*R*) は, ある適当な丸めの操作("o" であることが望ましい) によって *W* より得られた **W** の中のある *value* になる.

R が実数で

R<I1 あるいは I2<*R*

のとき, H2〔**real**〕(*R*) は, 定まっていない **W** のある元になる.

3.4.5 H1〔**bits**〕で, *type* **bits** のためのある定まった *projection* を表わす.

W を H1〔**bits**〕の値域とする. そのとき **W** は *available bit-string* の集合である. さらに

W={*B*|*B*∈**B**∧l(*B*)≤I3}

となる.

B が *bit-string* で l(*B*)≤I3 のとき

H1〔**bits**〕(*B*)=*B*

となる.

B が

"$b_1b_2\cdots\cdots b_n$"

の形の *bit-string* とする. ここで n=l(*B*)>I3 で b_1, b_2, ……, b_n が *bit*, そして *B'* は

"$b_1b_2\cdots\cdots b_{13}$"

の *bit-string* を表わすことにする. このとき

H1〔**bits**〕(*B*)=B'

になる.

3.4.6 H2〔**bits**〕で, *type* **bits** のためのある定まった *projection* を表わす.

B を *length n* の *bit-string* とする.

n=*0* (すなわち *B*=BO) のとき

H2〔**bits**〕(*B*)=**0**

となる.

n≥*1* のとき

H2〔**bits**〕(*B*)=*B*〔*1*〕

となる.

3.4.7 H1〔**string**〕で, *type* **string** のためのある定まった *projection* を表わす.

W を H1〔**string**〕の値域とする. そのとき **W** は, *availble string* の集合である. さらに

W={**C'**〔*n*〕|n≤I4}

となる.

C を *length n* の *string* とする.

C∈**C'**〔*n*〕で *n*≤I4 のとき

H1〔**string**〕(*C*)=*C*

となる.

C∈**C'**〔*n*〕, *n*>I4 で *C* が

"$c_1c_2\cdots\cdots c_n$"

の形とする. ここで c_1, c_2, ……, c_n は *character* であり, *C'* は

"$c_1c_2\cdots\cdots c_{14}$"

の string を表わすことにする. そのとき

H1〔**string**〕(*C*)=*C'*

となる.

C∉**C'**〔n〕のとき, H1〔**string**〕(*C*) は, 定まっていない **W** のある元である.

3.4.8 H2〔**string**〕で, *type* **string** のためのある定まった *projection* を表わす.

C を *length n* の *string* とする.

n=*0* (すなわち *C*=CO) のとき

H2〔**string**〕(*C*)=␣

となる.

n≥1 で *C*∈**C'**〔n〕のとき

H2〔**string**〕(*C*)=*C*〔1〕

となる.

n≥1 で C∉**C'**〔n〕のとき, H2〔**string**〕(C') は, 定まっていない **W** のある元である.

‖*Projection* という考えは, 第1版にはなく, 第2版で導入された. 第1版では *projection* のかわりに, "*mode*" というものがあった. *mode* は *type* の小分けであり, (**I**) **概説**の中で説明されている **scale** *a*, *b* **scale** *a*, **precision** *a* (以上 **real**-*type*), **exact** *a*, **varying** *a* (以上 **bits**-*type*), **exact** *a*, **varying** *a* (以上 **string**-*type*) に対応するものと, その他に **array**-*style* の下限と上限が *mode* であった. しかし *b* **scale** *a* のようなものは, *program* でも書くことができるという議論が起き, *projection* の形になった. *Projection* とすれば, 自由なものを書くことができる. 例として, 恒等 *projection* であるが, 副作用としてその値を印刷する *projection* を, *quantity Q* の h(*Q*) としたとしよう. その場合には, *quantity Q* に assign が行なわれるごとにその値が印刷され, *program* の追跡に便利である.

このような *mode* あるいは *projection* というものは, **real**-*type* 以外の *type* では, あまり問題はない. しかし **real**-*type* では問題が多く, そのために **real**-*type* そのものを *type* から除き, **real** は *standard declaration* で処理しようという考えもある.

その場合には，**array**-*style* のための subscript にあたるものは，別の新しい *type* を用意することになる．】

3.5 *Ability*

Progoam の *elaboration* の間に，この *elaboration* に関係する *variable* は，*on* あるいは *off* の状態にある．この *on* あるいは *off* の状態を，*variable* の "*ability*" と呼ぶ．*Variable* の *ability* は，*elaboration* が進むにしたがい変わる．

Variable V の *ability* は

$$a(V)$$

により表わす．V は $a(V)$ が *on* あるいは *off* により，*on* あるいは *off* といわれる．

$a(V)$ が *on* であれば

$$q(V)$$

で示されるある *quantity* が，V に対応ずけられている．

【簡単にいうと，*variable* はその *proper interior* の中に *declaration* を含む *assemblage* の入口で *off* にされ，その *declaration* の *elaboration* の終わりで *on* にされる．

ALGOL ***60*** などでは，*off* にされるのは *program* の実行の初めと，その *variable* の *declaration* を含む *block* の出口である．*ALGOL N* では，手続き自身が **procedure**-style のデータになり，またリストなどの処理のために *off* にされる時点が *ALGOL* ***60*** などと異なるようになっている．】