意味論的メタ言語の形をした計算機言語*

渡　辺　　坦**

Abstract

A programming Language in the form of a semantic meta-language is discussed. As the language is capable to accept new types of operation and operand, can be used as a common nucleus of various problem oriented languages. Programming a problem in this language proceeds as follows:

(1) Describe the rough algorithm of it introducing new phrases.

(2) Explain the syntax and semantics of the introduced new phrase using or introducing other phrases.

(3) Repeat the process (2) until all phrases are reduced to known phrases.

The logical consistency of the language is devised using mappings from symbol to symbol or from symbol to set of symbols.

1. まえがき

従来のプログラムは，一般に，意味定義ずみの用語を用いて処理操作を表現する形態である．ここでは逆に，まだ意味の明確に定義されていない用語を用いて処理操作を書き表わし，導入した用語の使用形式と意味をさらに他の用語を用いて書き表わすことを幾重にもくり返し，ついに全用語を既知の記号に帰着させるプログラミング方式を定める．従来においても，サブプログラムの利用や新演算子の導入などによって，この行き方はある程度可能であったが，ここではそれを推し進め，プログラムをフローチャートのように，人間の思考過程に沿って，大局から細部へ書き進めることを言語設計の基本態度とする．

今後，多種多様の問題向き言語が要求される傾向にあるが，この形式の言語を採用すると，既知とする記号の組を適用分野に応じて定めることにより，それらの問題向き言語を共通の核となる言語の上に構成できる．

新しい用語の導入を可能にするために，新記号の使用形式と意味を記述する方法を定式化する．いま，あるプログラムの名前をpとしよう．本報告の方法に従うと，そのプログラムはpという記号の意味を説明する記号列であり，pの中にはさらに，pで使っている用語を説明する記号列を幾重にも含む．すなわち，プログラム自体を一つの「意味論的メタ言語」で書くことになる．このメタ言語を説明するために，ALGOL N[1] の記述に使われたメタ言語を利用することにし，それをここでは「修正バッカス記法」という．

議論の繁雑になるのを防ぐために，はじめはプログラムの見やすさ，書きやすさを度外視して話しを進めるが，あとでプログラムを解りやすい形に変えることを考える．

拡張可能な汎用言語は，ALGOL 68[2]，ALGOL N，GPL[3] などの形で発表されているが，本報告はそれらとくらべ，すでに述べた言語設計の基本方針において，また，セマンティクスを記号と記号，あるいは記号と記号類の間の写像を用いて説明する点などにおいて，性格を異にしている．プログラム中に，まだ意味の明確に定義されていない記号の使用をゆるす考えは，**4.** で言及されている．詳細文法を自由に定めうる言語という面でならば，ほかに，ICES,[5] PLAN[6] などが実用化されているが，これらはプログラム形式をコマンド型に限定している．

本文で使う記号や用語の簡単な説明を **Table 1** に示す．

* A Semantic Meta-language, by Tan Watanabe (Central Research Lab., Hitachi, Ltd.)

** (株)日立製作所中央研究所

Table 1 Notation

記 号	説 明
$[\alpha]$	空要素または α
$\alpha\circ\circ\circ$	$\alpha,\ \alpha\alpha,\ \alpha\alpha\alpha,\cdots$
$\alpha\{\beta\}\circ\circ\circ$	$\alpha,\ \alpha\beta\alpha,\ \alpha\beta\alpha\beta\alpha,\cdots$
S^*	集合 S の元を連結してできる長さ有限のストリングの全体と，空要素とからなる集合
2^{S^*}	S^* の部分集合（記号類）の全体で構成される集合族
$\lambda[\cdots\cdots]$	新しい作用子を定義する作用子定義文
$\mu[\cdots\cdots]$	新しい記号類名を定義する集合定義文
$\xi[\cdots\cdots]$	変数の新しい宣言形式を定義する宣言定義文
φ_k	既知の記号類に対してその名前を対応させる写像
φ	プログラム p で使う記号類に対してその名前を対応させる写像
ψ	変数式や宣言式の展開を定める写像
χ_k	既知作用や集合関数に対して，その引数形式を規定する写像
χ	プログラム p で使う作用子や集合関数の引数形式を規定する写像
A	標識や定数，特殊文字などから成る基本記号の集合
A_0	括弧やコンマ以外の基本記号の集合
B	基本記号とメタ記号との合併集合
C	定数の集合
D	宣言式の集合
F_k	既知作用子の集合
F	プログラム p で使う作用子の集合
G	集合関数の集合
I	標識の集合
J	変数型ストリングの集合
L	ラベルの集合
M_k	既知の記号類名の集合
M	プログラム p で使う記号類名の集合
T	作用式の集合
U	集合式の集合
V	変数の集合
W	変数式の集合

2. プログラム例

プログラム方式の大略を示すために，一つの例をあげる．同一形状の多数の粒子が運動していて，それらは互いに衝突すると，ある確率に従って結合するとしよう．結合した粒子は六角格子点上に並ぶとし，ある核のまわりに粒子が衝突して核が成長する様子を二次元の単純なモデルでシミュレートすることを問題とする．衝突して結合する確率は，衝突点の粒子の配置と衝突時刻とから他のプログラムで算出するとしよう．このプログラムを **Fig. 1** に示すが，この例では次の記号の意味を既知とし，日本語で簡単に説明するにとどめる．

λ[……]	：新しい作用子の定義
ξ[……]	：変数の新しい宣言形式の定義
do…doend	：くりかえし
cat	：記号の連結
draw particle	：格子点上に粒子の図を描く
ascend	：作用子定義の論理的終りの表示

```
 1) λ[crystal growth
 2) ξ[stacker T S(L)
 3) χ   setexpression,formalvariable,integer
 4)     integer cat[S."N"]
 5)     array T cat[S."ELEM"]
 6)   ]
 7) ξ[point P
 8) χ   formalvariable
 9)     integer cat[P."X"]
10)     integer cat[P."Y"]
11)   ]

12)     integer N,TIME,TMAX,PX,PY,I,J
13)     array bit LATTICE(-N:N,-N:N)
14)     stacker point OLDBOUNDARY(1000)
15)     stacker point NEWBOUNDARY(1000)
16)     point P
17)     read data
18)     initialize
19)     do TIME=0 to TMAX
20)     collide
21)     doend
22)     ascend

23) λ[read data
24)     get N,TMAX
25)     ascend
26)   ]
27) λ[initialize
28)     let LATTICE=0
29)     let LATTICE(0,0)=1
30)     draw particle(0,0)
31)     clear NEWBOUNDARY
32)     ascend
33)   ]
34) λ[collide
35)     let OLDBOUNDARY=NEWBOUNDARY
36)     clear NEWBOUNDARY
37)     do for P of OLDBOUNDARY
38)     collide at P
39)     give newboundary
40)     doend
41)     ascend
42)   ]
43) λ[collide at PP
44) χ   ,point
45)     integer K
46)     let I=PP.X
47)     let J=PP.Y
48)     if LATTICE(I,J)=1 then ascend
49)     let K=LATTICE(I+1,J  )*32
    ↑         +LATTICE(I+1,J+1)*16
    ↑         +LATTICE(I  ,J+1)*8
    ↑         +LATTICE(I-1,J  )*4
    ↑         +LATTICE(I-1,J-1)*2
    ↑         +LATTICE(I  ,J-1)
50)     if probability(K,TIME)<1 then ascend
```

Fig. 1 Sample program (その 1)

result	：式の値を表示
probability	：結合確率算出用プログラム
set expression	：記号類の名前の集合を表示

```
51)      let LATTICE(I,J)=1
52)      draw particle(I,J)
53)      ascend
54)    ]
55) λ[give newboundary
56)      store when empty(I,J)
57)      if LATTICE(I,J)=0 then ascend
58)      store when empty(I+1,J  )
59)      store when empty(I+1,J+1)
60)      store when empty(I  ,J+1)
61)      store when empty(I-1,J  )
62)      store when empty(I-1,J-1)
63)      store when empty(I  ,J-1)
64)      ascend
65)    ]
66) λ[store when empty(II,JJ)
67) χ    ,integer,integer
68)      if LATTICE(II,JJ)=1 then ascend
69)      step NEWBOUNDARY.N by 1
70)      let NEWBOUNDARY.ELEM(NEWBOUNDARY.N)
    ↑    =pnt(II,JJ)
71)      ascend
72)    ]
73) λ[clear S
74) χ    ,stacker
75)      let S.N=0
76)    ]
77) λ[step N by M
78) χ    ,variable,expression
79)      let N=N+M
80)      ascend
81)    ]
82) λ[do for V of S
83)    STMT
84)    doend
85) P    priority -1.0
86) χ    ,variable,stacker,statementstring
87)      integer K
88)      do K=1 to S.N
89)      let V=S.ELEM(K)
90)      STMT
91)      doend
92)      ascend
93)    ]
94) λ[pnt(II,JJ)
95) P    priority 6.0
96)      factor,integer,integer
97)      result(II,JJ)
98)      ascend
99)    ]
100)   ]
```

Fig. 1　Sample program (その 2)

以下の記号は説明も省略する.

get	let	if
+	−	*
integer	bit	array
identifier	variable	formalvariable
factor	expression	statementstring

プログラムはかぎ括弧 [　] を用いたブロック構造をとる. ブロックの見出しはカラム3から書き, 通常のステートメントはカラム4以降から始める. カラム1の λ, ξ, χ, p で各行のタイプを示す. 矢印のある行は前の行の続きである. 大文字は変数名を表わし, 小文字や特殊文字は演算子などのキィワードを表わす.

第 2)〜6) 行の

ξ[stacker T $S(L)$…]

は, 14) の宣言が二つの宣言

integer　　OLDBOUNDARY. N

array point OLDBOUNDARY. ELEM (1000)

と同じであることを定義する. カラム1に χ のついた 3) は, パラメータ T, L の属すべき記号類を規定し, 宣言されるべき変数が S であることを示している. t が記号類名を表わすとき, array t $S(L)$ 型の宣言は t $S(1), \cdots, t$ $S(L)$ という L 個の宣言と同じであることがすでに定められているとする. そのとき, 宣言 stacker point OB (1000) は,

integer OB. N
integer OB. ELEM (1). X
integer OB. ELEM (1). Y
……
integer OB. ELEM (1000). X
integer OB. ELEM (1000). Y

という 2,001 個の宣言と同等になる.

12) から 16) でこのプログラム全体で使う変数を定義し, 17) から 22) で crystal growth の大雑把な処理を記述している. 23) 以降は, read data, initialize など, このプログラムで導入した用語の意味を順次定義する. 式を処理するさいの優先順位を定める必要があれば, カラム1に p と書いてその順位を与える.

3. 既知記号, 既知写像

言語の基本的構成要素を, 修正バッカス記法により, **Fig. 2** に示す. 文字としては, 英字, 数字, およびいくつかの特殊文字をとる. 英字で始まる英数字列を標識, 数字ばかりの列を数字列, 引用符で囲んだ文字列をリテラルなどという. これらと特殊文字とを合わせて基本記号と呼び $\boldsymbol{A}$ で表わす. $\boldsymbol{A}$ から括弧とコンマを除いた部分を $\boldsymbol{A}_0$ と表わす. Fig. 2 にあげた文字や, 修正バッカス記法で使う記号とは別に, メタ記号 $\lambda, \mu, \xi, :, ;, [,], \neg$ があって, $\boldsymbol{A}$ とメタ記号とを合わせた集合を $\boldsymbol{B}$ と表わす. $\boldsymbol{A}_0, \boldsymbol{A}, \boldsymbol{B}$ の元を連結して, $\boldsymbol{A}_0^*, \boldsymbol{A}^*, \boldsymbol{B}^*$ を作る. 以後, 本報告で扱う記号はすべ

```
<character>==<letter>|<digit>|<special character>
<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
<digit>    == 0|1|2|3|4|5|6|7|8|9
<nonstructural special character>== .|=|+|-|*|/|"|<etc 1>
<structural character>          == (|)|,|
<special character>==<nonstructural special character>|
                     <structural character>
<identifier>       ==<letter>〔<letter>|<digit>〕 ooo
<numeric string>   ==<digit> ooo
<literal>          == "〔<character>〕ooo"

< nonstructural basic symbol>==
                      <nonstructural special character>|
                      <identifier>|
                      <numeric string>|
                      <literal>|<etc 2>
<basic symbol>     ==<nonstructural basic symbol>|
                     <structural character>
<meta symbol>      ==|:|;|〔|〕|¬|λ|μ|§
 A0 ==<nonstructural basic symbol>
 A  ==<basic symbol>
 B  ==<basic symbol>|
      <meta symbol>
```

Fig. 2 Basic symbols

```
<constant>          ==<numeric string>|
                      <literal>|
                      <etc 3>
<known operator>    == +|-|*|/|
                       MOVE|JUMP|JUMPTRUE|
                       MEMBER|SETATR|RESULT|
                      <etc 4>
<known set symbol>  == ID|CONSTANT|AOSTAR|ASTAR|BSTAR|
                       KOPERATIONKEY|KSETSYMBOL|KSETFUNCTION|
                      <etc 5>
<set function>      == AND|OR|NOT|CAT|REPEAT|REPRO|<etc 6>
<variable form>     ==<identifier>〔(<variable modifier>〔,〕ooo)〕
<variable modifier> ==<constant>|<variable form>

 I  ==<identifier>
 C  ==<constant>
 Fk ==<known operator>
 Mk ==<known set symbol>
 G  ==<set function>
 J  ==<variable form>
```

Fig. 3 Known symbols

て B^* のストリングであり，記号を文字までさかのぼって扱うことをしない．B^* の部分集合を記号類ともいう．

つぎに，既知記号の集合 k を定義しよう（**Fig. 3** 参照）．A^* の記号類として，標識の集合 I，定数の集合 C，既知作用子の集合 F_k，既知の記号類の名前の集合 M_k，集合関数の集合 G があるとする．これらのうち，C, F_k, M_k, G は互いに共通部分を持たない．また，B^* の記号類に B^* の元を名前として対応させる一意写像 φ_k，および，F_k と G の引数形式を規定するための，F_k, G から M_k^* への多価写像 χ_k があるとする．この φ_k, χ_k についてもっと詳しく説明しよう．

φ_k は B^* の一部分から 2^{B^*} の一部分への写像であり，$\varphi_k(a)=\alpha$ は，「a は記号類 α の名前である」ことを意味する．φ_k の定義域は M_k を含む．$\varphi_k(a)$ は空集合でもよく，そのときは，記号類 a に属するストリングの形式が未定であることを表わす．

新たな部分集合の形成に利用する集合関数 g は，$g[a_1, \cdots, a_n]$ という形で使い，それは次の意味を持つ．

$\varphi_k(\mathrm{AND}[a_1, \cdots, a_n]) = \bigcap_{i=1}^{n} \varphi_k(a_i)$

$\varphi_k(\mathrm{OR}\ [a_1, \cdots, a_n]) = \bigcup_{i=1}^{n} \varphi_k(a_i)$

$\varphi_k(\mathrm{NOT}[a\quad]) = (\varphi_k(a)$ の $\boldsymbol{B^*}$ 内補集合$)$

$\varphi_k(\mathrm{CAT}[a_1, \cdots, a_n]) =$連結集合 $\varphi_k(a_1)\cdots\varphi_k(a_n)$

$\varphi_k(\mathrm{REPEAT}[i, m, n, p])$

$=$連結集合 $\varphi_k(p_m)\varphi_k(p_{m+1})\cdots\varphi_k(p_n)$

ただし，$p_m, p_{m+1}, \cdots$ は，ストリング p に含まれる i に $m, m+1, \cdots$ を代入したストリング

$\varphi_k(\mathrm{REPRO}[i, m, n, p]) = \bigcup_{i=m}^{n} \varphi_k(p_i)$

ただし，p_i については前項と同じ，

χ_k は，$\boldsymbol{F_k}, \boldsymbol{G}$ の各元に対して $\boldsymbol{M_k^*}$ の部分集合を対応させる写像で，次の意味を持つ．

$m_0 m_1 \cdots m_n \in \chi_k(f)$

ただし，$f \in \boldsymbol{F_k}, m_i \in \boldsymbol{M_k}$

であれば，記号類 $m_1, \cdots, m_n$ の元 $x_1, \cdots, x_n$ をとると，$f(x_1, \cdots, x_n)$ は許される表現であり，それは記号類 m_0 に含まれることを意味する．とくに

$m_0 \in \chi_k(f)$

ただし，$f \in \boldsymbol{F_k}, m_0 \in \boldsymbol{M_k}$

であれば，f を引数なしで使えることを意味する．

$m_1 \cdots m_n \in \chi_k(g)$

ただし，$g \in \boldsymbol{G}, m_i \in \boldsymbol{M_k}$

であれば，記号類 $m_1, \cdots, m_n$ の元 $x_1, \cdots, x_n$ をとると，$g[x_1, \cdots, x_n]$ は許される表現であることを意味する．

$\boldsymbol{k} = \{\boldsymbol{C}, \boldsymbol{F_k}, \boldsymbol{M_k}, \boldsymbol{G},$ メタ記号$\}$ を既知記号といい，φ_k, χ_k を既知写像という．また，集合

$$\boldsymbol{J} = \boldsymbol{I} \cup \bigcup_{n<\infty} \{a(b_1, \cdots, b_n) \mid a \in \boldsymbol{I}, b_i \in \boldsymbol{J} \cup \boldsymbol{C}\}$$

の元を変数型ストリングといい，あとで変数を表わす記号として使う．

4. プログラム

4.1 プログラムの概形

プログラム形式を **Fig. 4** に示すが，そこに記載されていない label, variable, variable expression などの説明は次節に譲る．プログラムの大雑把な形は

$\lambda[f(x_1, \cdots, x_n)\urcorner\ m_0, m_1, \cdots, m_n\urcorner$

$\mu[\cdots\cdots]\ \ \mu[\cdots\cdots]\ \ \cdots\cdots$

$\xi[\cdots\cdots]\ \ \xi[\cdots\cdots]\ \ \cdots\cdots$

$u_1[v_1];\ \ u_2[v_2];\ \ \cdots\cdots$

$l_1 : f_1(t_{11}, t_{12}\cdots);$

$\cdots\cdots$

$l_k : f_k(t_{k1}, t_{k2}, \cdots);$

$\lambda[\cdots\cdots]\ \ \lambda[\cdots\cdots]\ \ \cdots\cdots\ \ ]$　　(1)

と一般に表わされる．

式(1)のプログラムは，ストリング $f(x_1, \cdots, x_n)$ が

$(l_1 : f_1(t_{11}, t_{12}, \cdots); \cdots l_k : f_k(t_{k1}, t_{k2}, \cdots);)$

と同等であることを記述するものである．m_0 は $f(x_1, \cdots, x_n)$ の属する記号類を表わし，$m_1, \cdots, m_n$ は引数 $x_1, \cdots, x_n$ の属すべき記号類を表わす．引数部$(x_1, \cdots, x_n)$ と，それに対応する $m_1, \cdots, m_n$ はなくてもよい．f をこのプログラムで定義している作用子という．

集合定義文と名づける $\mu[\cdots\cdots]$ は

$\mu[m(x_1, \cdots, x_n)\urcorner u_1, \cdots, u_n\urcorner S]$　　(2)

の形であり，$m(x_1, \cdots, x_n)$ という名前の記号類を定義する．$u_1, \cdots, u_n$ は引数 $x_1, \cdots, x_n$ の属すべき記号類を示す．引数はなくてもよい．S は集合関数を用いて他の記号類を複合するストリングであり，記号類 $m(x_1, \cdots, x_n)$ とはどんな記号類であるかを表現する．

宣言定義文と名づける $\xi[\cdots\cdots]$ は

$\xi[m(x_1, \cdots, x_n)[x]\urcorner u_1, \cdots, u_n\urcorner$

$d_1;\cdots\cdots d_l;\quad]$　　(3)

の形であり，一連の変数定義文 $d_1;\cdots\cdots d_l;$ を簡単に $m(x_1, \cdots, x_n)[x];$ と表わすことを定義する．

$u[v];$ 型ストリングである変数定義文は，v で表わされる変数の属する記号類の名前が u であることを宣言する．

作用子定義文と名づける $\lambda[\cdots\cdots]$ は，新たな作用子を定義するストリングで，(1)式と同じ構造を持つ．プログラム自体も一つの作用子定義文である．

$l : f(t_1, t_2, \cdots);$ 型のストリングを作用文といい，l をそのラベル，$f(x_1, \cdots, x_n)$ を作用式という．f は既知作用子であるか，このプログラムに含まれている作用子定義文で定義されている作用子であるか，あるいは，このプログラムを含んでいるプログラム内の作用子定義文で定義されている作用子のいずれかである．f の実引数 t_i は，定数であるか，あるいは，$f_i(t_{i1}, t_{i2}, \cdots)$ という形の作用式や，変数，ラベルのうちのいずれかである．

4.2 プログラムの詳細

前節の説明では，まだプログラムの意味が明確でなく，また，プログラム形式も一部分説明し残してある．本節では，プログラム p に対して各種の集合や写像を

```
< program>                  ==<operator definition>
<operator definition>       ==
     λ[<oprt key>[(<oprt formal parm>{,}ooo)]          ┐
       <oprt def type>[,<oprt formal parm type>]ooo    ┐
       [<set definition>]ooo
       [<declaration definition>]ooo
       [<variable definition>]ooo
       [<operation statement>ooo
       [<operator definition>]ooo]
<oprt key>                  ==(<identifier>
                              <nonstructural special character>)ooo
<oprt formal parm>          ==<identifier>
<oprt def type>             ==<set expression>
<oprt formal parm type>     ==<set expression>

<set definition>            ==
       μ[<set key>[(<set formal parm>{,}ooo)]     ┐
                  [<set formal parm type>{,}ooo]  ┐
         <set statement>ooo ]
<set key>                   ==<identifier>
<set formal parm>           ==<identifier>
<set statement>             ==<set expression>;

<declaration definition>    ==
       ξ[<set key>[(<set formal parm>{,}ooo)][<formal variable>]  ┐
                  <set formal parm type>{,}ooo
         <declaration statement>
<formal variable>           ==<identifier>
<declaration statement>     ==<set expression>[<variable expression>];
<set formal parm type>      ==<identifier>
<set expression>            ==<known set symbol>|
                              <set key>[(<set exp actual parm>{,}ooo)]|
                              <set function>[<set function actual parm>{,}ooo]
<set exp actual parm>       ==<constant>|
                              <set formal parm>|<oprt formal parm>
<set function actual parm>==<constant>|<set formal parm>|
                              <formal variable>

<variable definition>       ==ξ[<set expression>[<variable expression>]]

<operation statement>       ==[<lable>:]<operation expression>;
<operation expression>      ==<operator>[(<operation exp actual parm>{,}ooo)]
<operator>                  ==<known operator>|
                              <oprt key>
<operation exp actual parm>==<constant>|<variable>|<label>|
                              <operation expression>

<variable expression>       ==
<variable>                  ==   } あとで定める
<label>                     ==
```

Fig. 4 Syntax

定義し，それらを使ってメタ記号の持つ意味，およびプログラムの形式と意味の明確化をはかる．

プログラム p に対し，作用子の集合 $\boldsymbol{F}$，記号類名の集合 $\boldsymbol{M}$，作用式の集合 $\boldsymbol{T}$，集合式の集合 $\boldsymbol{U}$，変数式の集合 $\boldsymbol{W}$，宣言式の集合 $\boldsymbol{D}$，ラベルの集合 $\boldsymbol{L}$，変数の集合 $\boldsymbol{V}$，および，記号類に名前をつける写像 φ，作用子や集合関数の引数形式を規定する写像 χ，変数式や宣言式の「展開」を定める写像 ψ を，Fig. 4 と，下記の(1)から(20)までの規則によって定義する．

(1) $\boldsymbol{F} \supset \boldsymbol{F}_k$

(2) $\boldsymbol{M} \supset \boldsymbol{M}_k$

(3) $a \in \boldsymbol{B}^*$ に対して $\varphi_k(a)$ が定義されていれば，
$\varphi(a) \supset \varphi_k(a)$

とする．

(4) $g \in \boldsymbol{G}$ ならば $\chi(g)=\chi_k(g)$，
$f \in \boldsymbol{F}_k$ ならば $\chi(f)=\chi_k(f)$

とする．

(5) $T \supset C$　$T \supset V$　$T \supset L$

(6) $U \supset M$

(7) $g \in G$, $m_1 \cdots m_n \in \chi(g)$ のとき，$t_i \in \varphi(m_i)$ なる t_i に対して，

$g[t_1, \cdots, t_n] \in U$

と定義する．(M の元に対しては，あとで述べるように，2^{B^*} への写像 φ が定義されている．したがって，集合関数に対する φ_k の定義と合わせると，すべての集合式 $u \in U$ に対して 2^{B^*} への写像 φ が定義されていて，集合式は B^* の記号類を書き表わすことになる．)

(8) $W = J \cup \{u \in U \mid \varphi(u) \in J\}$

(9) $w \in W$ かつ $w \in U$ ならば $\psi(w) = \varphi(w)$

$w \in J$ かつ $w \in U$ ならば $\psi(w) = \{w\}$

とする．変数式 $w \in W$ に対して $\psi(w)$ を w の展開という．

(10) $m \in I, x_i \in I, u_i \in U, s \in U$ として，ストリング

$\mu[m(x_1, \cdots, x_n)\urcorner u_1, \cdots, u_n\urcorner s]$

がプログラム p に集合定義文として含まれるならば，

$m \in M$,　$u_1 \cdots u_n \in \chi(m)$,　$x_i \in \varphi(u_i)$

であるとし，また，$t_i \in \varphi(u_i)$ なる t_i をとると

$m(t_1, \cdots, t_n) \in M$,

$\varphi(m(t_1, \cdots, t_n)) = \varphi(s')$,

$\varphi(m) \supset \bigcup_{t_i \in \varphi(u_i)} \varphi(m(t_1, \cdots, t_n))$

であると定義する．ここに，s' は s に含まれる $x_1, \cdots, x_n$ に $t_1, \cdots, t_n$ を代入したストリングを表わす．とくに，$n=0$，すなわち引数なしの場合，上記は

$\mu[m\urcorner\urcorner s]$

ただし，$m \in I, s \in U$

が p に集合定義文として含まれるならば，

$m \in M$,　$\varphi(m) \supset \varphi(s)$

と定義する，となる．このような場合，以後簡略化して，「引数なしの場合も同様」といって表わす．

(11) $D' = \{u[w] \mid u \in U, w \in W\}$

$D = D' \cup \{u \in U \mid \varphi(u) \subset D'\}$

(12) 宣言式 $d \in D$ が，

$u[w]$ 型ストリングのときは $\psi'(d) \supset \{d\}$,

集合式のときは $\psi'(d) = \varphi(d)$

とする．

(13) $m \in I, x_i \in I, x \in I, u_i \in U, d_j \in D$ として，

$\xi[m(x_1, \cdots, x_n)[x]\urcorner u_1, \cdots u_n\urcorner d_1; \cdots d_l;]$

がプログラム p に宣言定義文として含まれるならば，

$m \in M$,　$u_1 \cdots u_n \in \chi(m)$,　$x_i \in \varphi(u_i)$

であるとし，また，$t_i \in \varphi(u_i)$ なる t_i をとると，

$m(t_1, \cdots, t_n) \in M$,

$\psi'(m(t_1, \cdots, t_n)[x]) \supset \bigcup_{j=1}^{l} \psi'(d_j')$,

$\varphi(m) \supset \bigcup_{t_i \in \varphi(u_i)} \varphi(m(t_1, \cdots, t_n))$

であると定義する．ここに，d_j' は d_j に含まれる $x_1, \cdots, x_n$ に $t_1, \cdots, t_n$ を代入したストリングを表わす．$\varphi(m(t_1, \cdots, t_n))$ については(16)で説明する．引数なしの場合も同様に定義する．

(14) 写像 ψ' は規則(12)，(13)のみで定義される．(規則(13)の場合，(12)と合わせると，

$\psi'(m(t_1, \cdots, t_n)[x])$

$= \{m(t_1, \cdots, t_n)[x]\} \cup \bigcup_{j=1}^{l} \psi'(d_j')$

となる．一般には，d_j もまた他の宣言定義文で定義されているので，ψ' の定義は recursive である．)

(15) 宣言式 $d \in D$ に対して写像 ψ を

$\psi(d) = \bigcup_{u[w] \in \psi'(d)} \{u[a] \mid a \in \psi(w)\}$

と定め，$\psi(d)$ を d の展開という．

(16) $u \in U, w \in W$ として，ストリング $u[w]$; が変数定義文としてプログラム p に含まれているならば

$V \supset \psi(w)$

$V \supset \{a \mid m[a] \in \psi(u[w])\}$

$\varphi(u) \supset \psi(w)$

と定義する．また

$m[a] \in \psi(u[w])$ なら $a \in \varphi(u)$

と定義する．

変数の定義を例を使って検討してみるために，プログラム p に次の宣言定義文が含まれているとしよう．

```
ξ[ARRAY(U, L)[A]¬
  SETEXPRESSION,
    INTEGEREXPRESSION¬
  U[REPRO[I, 1, L,
    APPENDMOD[A, I]]];]
ξ[POINT[P]¬¬
  INTEGER[APPENDMOD[P, CHARX]];
  INTEGER[APPENDMOD[P, CHARY]];]
```

ただし，APPENDMOD は集合関数，CHARX, CHARY は記号類名で，それらに対して写像 φ は，$a, b, b_1, \cdots, b_n$ を A_0^* のストリングとするとき，

$\varphi_k(\text{APPENDMOD}[a, m]) = \{a(m)\}$

$\varphi_k(\text{APPENDMOD}[b(b_1, \cdots, b_n), m])$

$=\{b(b_1, \cdots, b_n, m)\}$

φ_k(CHARX)$=\{X\}$

φ_k(CHARY)$=\{Y\}$

と定められているとする．このとき，ストリング

ARRAY(POINT, 5)[OR[A, B]];

が変数定義文として p に含まれていれば，写像 ψ', ψ, φ, および変数集合 $\boldsymbol{V}$ に関して，次の関係が成り立つ．

ψ'(POINT[P])
$=\{$POINT[P],
INTEGER[APPENDMOD
[P, CHARX]],
INTEGER[APPENDMOD
[P, CHARY]]$\}$

ψ'(ARRAY(POINT, 5)[A])
$=\{$ARRAY(POINT, 5)[A],
POINT[REPRO
[I, 1, 5, APPENDMOD[A, I]]]$\}$

ψ(ARRAY(POINT, 5)[OR[A, B]])
$=\{$ARRAY(POINT, 5)[A],
ARRAY(POINT, 5)[B],
POINT[A](1)], ⋯, POINT[A(5)],
POINT[B(1)], ⋯, POINT[B(5)],
INTEGER[A(1, X)], ⋯
⋯, INTEGER[A(5, X)],
INTEGER[A(1, Y)], ⋯
⋯, INTEGER[A(5, Y)],
INTEGER[B(1, X)], ⋯
⋯, INTEGER[B(5, X)],
INTEGER[B(1, Y)], ⋯
⋯, INTEGER[B(5, Y)]$\}$

$\boldsymbol{V}\supset\{$A, B, A(1), ⋯, A(5), B(1), ⋯, B(5),
A(1, X), ⋯, A(5, X), A(1, Y), ⋯, A(5, Y),
B(1, X), ⋯, B(5, X), B(1, Y), ⋯, B(5, Y)$\}$

φ(ARRAY)$\supset\{$A, B$\}$

φ(ARRAY(POINT, 5))$\supset\{$A, B$\}$

φ(INTEGER)
$\supset\{$A(1, X), ⋯, A(5, X), A(1, Y), ⋯, A(5, Y),
B(1, X), ⋯, B(5, X), B(1, Y), ⋯, B(5, Y)$\}$

(17) $l\in\boldsymbol{J}$, $f\in\boldsymbol{F}$, $t_i\in\boldsymbol{T}$ として，ストリング

l: $f(t_1, \cdots, t_n)$;

が作用文としてプログラム p に含まれているとき，

$l\in\boldsymbol{L}$

と定義し，l をこの作用文のラベルという．

(18) $f\in\boldsymbol{F}$, $u_0u_1\cdots u_n\in\chi(f)$ であれば，$t_i\in\varphi(u_i)$ なる $t_i\in\boldsymbol{T}$ に対して

$f(t_1, \cdots, t_n)\in\boldsymbol{T}\cap\varphi(u_0)$

であるとする．引数なしの場合も同様である．すなわち，作用子 f につけた引数が，写像 χ で規定された形式に合致した作用式であるとき，その引数つき作用子は作用式であるという．

(19) プログラム p に作用子定義文として含まれる(1)式型ストリングにおいて，

$f\in\boldsymbol{A}_0{}^*$, $x_i\in\boldsymbol{I}$, $m_i\in\boldsymbol{U}$, $u_i[v_i]\in\boldsymbol{D}$,
$l_j\in\boldsymbol{J}$, $f_j(t_{j1}, t_{j2}, \cdots)\in\boldsymbol{T}$

となっていれば

$f\in\boldsymbol{F}$, $m_0m_1\cdots m_n\in\chi(f)$, $x_i\in\boldsymbol{V}\cap\varphi(m_i)$

であると定義する．規則(18)と合わせると，このとき，$t_i\in\varphi(m_i)$ なる $t_i\in\boldsymbol{T}$ をとると

$f(t_1, \cdots, t_n)\in\boldsymbol{T}\cap\varphi(u_0)$

となる．引数のない場合も同様である．また，(1)式において，ラベル $l_1, \cdots, l_k$ は省略されていてもよい．

(20) プログラム（すなわち作用子定義文）p が，作用子定義文 q を含んでいるとする．p の作用子，記号類名，ラベル，変数の集合をそれぞれ $\boldsymbol{F}_p$, $\boldsymbol{M}_p$, $\boldsymbol{L}_p$, $\boldsymbol{V}_p$ と表わし，q のそれらを $\boldsymbol{F}_q$, $\boldsymbol{M}_q$, $\boldsymbol{L}_q$, $\boldsymbol{V}_q$ と表わした場合，これらの間に次の関係が成り立つとする．

$\boldsymbol{F}_p\subset\boldsymbol{F}_q$ $\quad\boldsymbol{M}_p\subset\boldsymbol{M}_q$
$\boldsymbol{L}_p\subset\boldsymbol{L}_q$ $\quad\boldsymbol{V}_p\subset\boldsymbol{V}_q$

すなわち，外側のプログラムで定義されている記号は，内側のプログラムで使用可能であるとする．

上記の規則(1)から(20)により，プログラムの形式，新記号の導入しかた，メタ記号の持つ意味などが明確になった．

4.3 プログラムの意味づけ

前節までででプログラムに持たせる意味もほぼ明らかになったが，まだ，新しく定義する記号とそれを定義している記号列との，意味上の対応が厳密になされていない．この対応を明らかにする一つの方法として，導入した新記号をそれを定義している記号列で置き換えることを考える．

作用式の集合 $\boldsymbol{T}$ を拡張して，作用文の列を括弧でくくった形のストリング

$(l_1$: $f_1(t_{11}, \cdots)$; l_2: $f_2(t_{21}, \cdots)$;
$\cdots l_k$: $f_k(t_{k1}, \cdots)$;$)$

も含めた集合を $\boldsymbol{T}'$ と表わす．そして，任意の作用式に対し，それに含まれている新しく導入された記号を，つぎつぎとそれを定義している記号列でおきかえ，$\boldsymbol{T}'$

のストリングへと帰着させる写像 π を考える．$t'\in T'$ に含まれる作用子が，recursive に定義されたものを除いて，すべて既知作用子であるとき，t' の持つ意味はもう既知であるとしよう．π は，作用式を意味の既知な T' の元に射影する写像である．

以下に，上記のことを定式化する．

プログラム p に対し，ストリング集合 T' および写像 φ' を，次の規則により定義する．

(1) $T'\supset T$

(2) $u\in U$ のとき $\varphi'(u)\supset\varphi(u)$

(3) $f\in F$, $m_0m_1\cdots m_n\in\chi(f)$, $t_i\in\varphi'(m_i)\cap T'$ であるとき，次の関係が成り立つ．

$$f(t_1,\cdots,t_n)\in T'\cap\varphi'(m_0)$$

(4) 作用子 f が(1)式の形の作用子定義文で定義されているとすれば，$t_i\in\varphi'(m_i)$ なる $t_i\in T'$ をとれば

$$(l_1'\colon f_1(t_{11}',\cdots);\ \cdots l_k'\colon f_k(t_{k1}',\cdots);)\in T'\cap\varphi'(m_0)$$

が成り立つ．ただし，l_i', t_{ij}' は，l_i, t_{ij} に含まれる $x_1,\cdots,x_n$ に $t_1,\cdots,t_n$ を代入したストリングである．

つぎに，写像 π を以下の規則によって定義する．

(1) $c\in C$ ならば $\pi(c)=c$

(2) $v\in J$ ならば $\pi(v)=v$

(3) T' の定義規則(4)の場合，次式が成り立つ．

$$\pi(f(t_1,\cdots,t_n))=\pi((l_1'\colon f_1(t_{11}',\cdots);\cdots l_k'\colon f_k(t_{k1}',\cdots);))$$

(4) $(l_1\colon t_1;\cdots l_k\colon t_k;)\in T'$ ならば

$$\pi((l_1\colon t_1;\cdots l_k\colon t_k;))=(l_1\colon\pi(t_1);\cdots l_k\colon\pi(t_k);)$$

(5) $f\in F_k$, $f(t_1,\cdots,t_n)\in T'$ ならば

$$\pi(f(t_1,\cdots,t_n))=f(\pi(t_1),\cdots,\pi(t_n))$$

(6) 作用子が recursive に定義されているとき，上記(1)から(5)の規則だけでは，写像 π によるおきかえには無限のくり返しが起こるので，最初のくり返しが発生したところでおきかえ操作を打ち切る．

5. プログラム表記上の便法

これまで，作用式の形を $f(t_1,\cdots,t_n)$，集合式の形を $g[t_1,\cdots,t_n]$，変数の形を $a(b_1,\cdots,b_n)$ としてきたが，このような括弧つきの形だと，引数にまた式を使うことを幾重にもくり返した場合，非常に見づらくなる．プログラムの見やすさは実用上非常に大切な問題なので，メタ記号や括弧，記号の配列順序などを工夫して，プログラムを変形する必要がある．しかし，その場合，変形結果はまた4章での形に一意的に復元可能でなければならない．

変形規則の一例として，2章の例で用いたプログラム表記上の便法を説明する．

(1) 作用式を $f(t_1,\cdots,t_n)$ の形に限定しないようにするため，作用子を小文字または（括弧やコンマ以外の）特殊文字で表わし，変数を大文字で表わして両者を区別し，引数を区切るコンマや括弧を省略してよいとする．

(2) 作用子を必ずしも作用式の先頭に置く必要がないようにするため，作用子に構文解析上の優先順位をつけ，作用式を operator precedence 型にする．

(3) let $A=B$ のように，作用子を主となる見出し記号といくつかの補助記号との組み合わせとして表わしてよいとする．

(4) 変数の形を $a(b_1,\cdots,b_n)$ に限定せず，b_i が（括弧やコンマ以外の）特殊文字のときは，括弧やコンマをはずした形でもよいとし，その形を変数定義文の中で，記号を連結する関数 cat を用いて表現する．

(5) プログラムを行に分割し，文の区切りをメタ記号；の代わりに改行で示す．

(6) 作用子定義文(1)式，集合定義文(2)式，宣言定義文(3)式において，見出し部分 $f(x_1,\cdots,x_n)$, $m(x_1,\cdots,x_n)$, $m(x_1,\cdots,x_n)[x]$ の区切りを記号 ┐ で示すかわりに，見出し部分の行をカラム3以前から書き，他の行をカラム4から書いて両者を区別する．また，引数の属すべき記号類名を並べた行の終りを記号 ┐ で示す代わりに，その行のカラム1に記号 χ をつける．

(7) (1)式の形の作用子定義文が，一つの作用文をなす作用式の見出しとなる作用子を定義するときは，記号類名の指定 m_0 を省略してよいとする．

(8) 宣言定義文(3)にて，宣言される変数 x を書く位置を見出し行の最後尾としなくてよいとし，その位置を引数の記号類名を指定する行に書く formal variable という記号の位置で示す．

(9) 作用子定義の論理的な終りを示すとき，閉じ括弧] の直前に飛ばせる代わりに，記号 ascend で示す．

6. 検　　討

この方式を実用化するには，プログラム表記上の便法とプログラムの内部表現とをはっきり定め，コンパイラを作成しなければならない．

内部表現は，処理操作のアルゴリズムを処理対象（定

数や変数）の型にあまり依存しないで記述できる形式が望ましい．そのためには，すべての処理対象を同一形式の「指針」を使って間接的に表現することが考えられる．そうすれば，プログラム自体も処理対象と同一の内部表現形式をとるようにできよう．

この方式のプログラミングシステムにおいて，頻繁に使う記号を定義するストリングを問題の種類に応じて組にして保存し，プログラミングのさい，それらの記号を既知記号と同等に扱えるようにすれば，さまざまの問題向き言語を構成できる．その場合，これらの問題向き言語はすべて，4章のメタ言語，および少数の既知記号を共通の核として持つ．この核のコンパイラを作り，それに対して，定義ずみとして保存してある記号を既知記号と同等に扱う機能を付け加えれば，この方式に従うすべての問題向き言語に対するコンパイラを構成できたことになる．

プログラムや処理対象をすべて間接的な内部表現形式にすると処理時間が大幅に増す．その対策としては，記号の属性や値のうち，言語処理の段階ですでに判明しているものに対して，その特性に合った能率よい内部表現に変える部分評価過程をコンパイラに組み入れればよい．

前章までで述べた言語体系には，改善を要する点がいくつかある．とくに，作用式の引数として作用式を使うが，プログラムの表現形式を使用に便利なように変形した場合，引数を区切るコンマと作用式中に含まれるコンマとが混同されるおそれがある．したがって，混同のおそれのあるコンマを含む作用式を引数とするときには，その作用式を括弧でくくるなどの条件を課する必要がある．

また，同一記号を異なる意味で使うことをゆるしていないが，プログラムは個々の作用子定義文，集合義文，宣言定義文をそれぞれ一つのブロックとみると，かぎ括弧によるブロック構造を持つので，記号の定義域をそれを定義しているブロック内に限れば，記号の重複をゆるすことができる．さらに集合定義文，宣言定義文などにおいて，既知作用子を PL/I の compile time statement に相当する形で使うことをゆるせば，定義能力が大幅に強化される．

7. むすび

記号の形式と意味を定義することの定式化をはかり，それをもとに，従来の汎用計算機言語とは行き方を異にする自由度に富んだ，汎用かつ問題向きとなる計算機言語を構成した．この言語のコンパイラを作成する段階では，論理の面でも実用の面でも，まだ問題点が出てくると思われるが，逐次改善していきたい．

おわりに，貴重な助言をいただいた上智大学村田晴夫氏，日本原子力研究所浅井清氏，日立製作所中央研究所中田育男氏に深く謝意を表する．

参考文献

1) 岩村，島内，高須，他5名：算法言語の設計—記述—処理の研究，京都大学数理解析研究所講究録—66 (1969)
2) A. van Wijngarden 編：Draft Report on the Algorithmic Language, ALGOL 68, Mathematisch Centrum 発行 (1968)
3) J. Garwick：GPL, A Truly General Purpose Language：Comm. ACM, Vol. 11, No. 9, pp. 634～638 (1968)
4) 浅井 清：オンラインプログラミングのための方法，第9回プログラミングシンポジウム報告集 (1968)
5) D. Ross：ICES System Design, MIT (1967)
6) IBM社：Problem Language Analyzer (PLAN) Program Description Manual, Form H 20-0594 (1969)

（昭和45年2月27日受付）