

Recursive 言語に関する一考察*

藤 田 輝 昭**

1. はじめに

われわれの日常使用するプログラム言語は, Backus normal form (BNF) であらわすことのできる constituent structure 文法と, その記述をおぎなう非形式的な規則とによって構成される. この非形式的な部分は, たとえば ALGOL 文法では “semantical rules” として与えられているが, その中には identifier や label に関する制限のように, 明らかにシンタックス規則と考えられるものも含まれている.

他人の書いたプログラムを読む場合, われわれはこれらのシンタックス規則にしたがって構造分析を行ない, 有限時間でそれが構造的に正しいプログラムであるかどうかを判定しているわけで, このことは実在のプログラム言語が recursive であることを示唆している. Putnam 1961 は, 自然言語について同じ論法を適用し, 現実の文法によって生成される言語は recursive だろう, といっている.

Context-free (CF) 言語はいうまでもなく recursive であるが, 現実のプログラム言語は最初に見たように, ある CF 言語の部分集合であり, 非形式的な規則をそのままの形で見る限り, それが recursive であるかどうかはわからない.

本稿では ALGOL 60 を例にとって, Naur 1963 のうち BNF で書かれていないシンタックス規則の部分 WFP (well-formed program) という述語の非形式的な定義であると考え, この述語が recursive であることを確認する. したがって ALGOL 60 文法によって生成される言語は recursive であることがわかる.

一方 acceptor の側から考えると, finite-state 言語が有限オートマトンによって, また CF 言語がプッシュダウン・オートマトンによって特徴づけられるのと同様に, recursive 言語の族は, 定義により, チューリング機械によって accept される言語の族にはかな

らない.

Gilbert 1966 の “analytic grammar” も, acceptor としてのチューリング機械の一種と考えられる. 彼のアプローチでは phrase-structure (PS) 文法によって生成されたストリングに, 適当な recursive 関数 (“selection function”) によるふるいわけ (“scan”) を加えることによって well-formed string を判定する. すると, analytic grammar によって解析される言語の族は recursive 言語の族と一致する (ibid. Theorem 1, 2). 特に PS 文法は analytic grammar と equivalent である (ibid. Theorem 3).

本稿の結果は ALGOL 60 に対して analytic grammar を作成し得ることを示しているが, PS 文法と scan を構成する具体的な方法はまだわかっていない.

【記号および用語】 本稿の記号および用語のうち文法に関する部分は Ginsburg 1966 にしたがう. また Kleene 1952 (p. 71) のように記号の名前として同じ記号を用いることは, 本稿の場合混乱を起こすおそれがある. そこで Quine 1961 (p. 23) に準じて, 記号の「名前」を指すのに single quotation mark を用いる.

2. CF 言語の算術化

ALGOL 60 文法の述語化は次の 2 段階によって完成される.

- (1) BNF で書かれた部分 (CF 文法) の述語化
- (2) 述語 WFP の定義

当然その前提として ALGOL 60 の算術化が必要である. 算術化と CF 文法の述語化とは, ALGOL 60 に限らずすべての CF 文法に共通の手順を踏んで進行するので, ここでは CF 文法一般に関するメタ理論の形で展開する.

以下 CF 文法 $G=(V, \Sigma, P, \sigma)$ を固定して考える. よく知られているように G は条件:

- (*) G は reduced grammar であり, $X \rightarrow X$ ($X \in V - \Sigma$) の形のプロダクションを持たない.

を満たしていると仮定して差支えない.

まず V の元に適当な順序づけをおこない, 3 から始

* A note on recursive languages, by T. Fujita (Mitsui Mutual Life Insurance Co.)

** 三井生命保険相互会社

まる奇数を次々に対応させなければならない。後で定義する述語の recursiveness を保証するため、この順序づけには若干の注意を要する。

ここで採用するのは Ingerman 1966 (p. 55 f) の canonical ordering である。すなわちまず、プロダクション $X \rightarrow \varphi$ があって $Y \in V$ が φ の中に現われれば $Y \leq X$ とする。したがってどの $X \in \Sigma$, $Y \in V - \Sigma$ なら $Y \leq X$ となることはなく、 $X \in \Sigma$, $Y \in V - \Sigma$ なら $Y \leq X$ となることはない。この二つの関係が不変であるように、上の順序づけの中から loop を消して行く。プロダクションは有限個であり、 G は reduced と仮定されているので、Ingerman のアルゴリズムは必ず成功する。

このアルゴリズムでは順序づけと同時に suffix が与えられ、 σ の suffix は 0 である。そこで V の元を suffix の逆順に並べる。同じ suffix のものはどう並べてもよい。こうしてできた列に 3, 5, 7, ... と割り当てて行けばよい。

この写像によって V の元 X に対応する奇数を $p(X)$ とする。任意のストリング $\varphi = x_1 x_2 \dots x_n \in V^*$ の Gödel 数 $\bar{\varphi}$ は

$$\bar{\varphi} = 2^{p(x_1)} 3^{p(x_2)} \dots p_n^{p(x_n)} \quad (p_i \text{ は } i \text{ 番目の素数})$$

である。特に null string には $20=1$ が対応する。また単一の character からなるストリングがその character 自体と区別されることを注意しておく。

なお、 $'p(x)'$ というのは informal な記号 (メタメタ記号) であり、述語の形式的な定義の中ではそれぞれ対応する奇数が使われなければならない。

こうして V^* を自然数全体の集合 N の中に埋めこむことができたので、次の課題は与えられた CF 文法をこれらの Gödel 数に関する理論と考え、各 non-terminal を表現する述語を定義することである。このメタ理論において使用する論理記号を

$$\neg \vee \wedge \supset \equiv \forall \exists =$$

とする。また“ただひとつ存在する”という意味に‘ $\exists!$ ’を使用する。‘ $\forall$ ’と‘ $\exists$ ’を除いては ALGOL 60 の logical operator と同じであるが、ALGOL 60 の basic symbol はすでに N の元と考えられているので混同のおそれはない。

定義のための等号として、関数については、‘ $=$ ’、述語については、‘ $\equiv$ ’を用いるが、これも混乱を生ずることは考えられない。

最初に定義する基本的な述語と関数は Gödel 1931 の 1~9 と同じものである (#6 だけを Gödel 1934

にならって変更してある)。

$$\#1 \quad x|y \equiv \exists z(z \leq x \wedge x = yz)$$

x は y で割り切れる。

$$\#2 \quad \text{Prim}(x) \equiv \neg \exists z(z \leq x \wedge z \neq 1 \wedge z \neq x \wedge x|z) \wedge x > 1$$

x は素数。

$$\#3 \quad 0 P_r x = 0$$

$$(n+1)P_r x = \mu y \{y \leq x \wedge \text{Prim}(y) \wedge x|y \wedge y > nP_r x\}$$

$nP_r x$ は x の n 番目の素因数。

$$\#4 \quad 0! = 1$$

$$(n+1)! = (n+1) \cdot n!$$

$$\#5 \quad P_r(0) = 0$$

$$P_r(n+1) = \mu y \{y \leq (P_r(n)! + 1) \wedge \text{Prim}(y) \wedge y > P_r(n)\}$$

$P_r(n)$ は n 番目の素数。

$$\#6 \quad nGl x = \mu y \{y \leq x \wedge x|P_r(n)^y \wedge \neg x|P_r(n)^{y+1}\}$$

$nGl x$ は、 x に対応するストリングの n 番目の character の Gödel 数。

$$\#7 \quad l(x) = \mu y \{y \leq x \wedge y P_r x > 0 \wedge (y+1)P_r x = 0\}$$

x に対応するストリングの長さが $l(x)$ である。

特に $l(1) = 0$ 。

$$\#8 \quad x*y = \mu z \{z \leq [P_r(l(x) + l(y))]^{x+y}$$

$$\wedge \forall n[n \leq l(x) \supset nGl z = nGl x]$$

$$\wedge \forall n[0 < n \leq l(y) \supset (n + l(x))Gl z = nGl y]\}$$

$x*y$ は x と y の concatenation。

$$\#9 \quad R(x) = \alpha^x$$

x が character の Gödel 数である場合、この character だけからなるストリングの Gödel 数が $R(x)$ 。

3. CF 文法の述語化

命題 1. 条件 (*) を充たす CF 文法 $G = (V, \Sigma, P, \sigma)$ が与えられたとき、 G の nonterminal を

$$V - \Sigma = \{X_1, X_2, \dots, X_k\} \quad (X_1 = \sigma, X_i \text{ は互に相異なる})$$

とすると、1-place recursive predicate

$$E_1, E_2, \dots, E_k$$

を作って、任意の $\varphi \in \Sigma^*$ に対し

$$X_i \overset{*}{\Rightarrow} \equiv E_i(\bar{\varphi}) \quad (1 \leq i \leq k)$$

であるようにできる。

(証明) まず P を並べかえて

$$X_1 \rightarrow \varphi_{11}$$

$$\begin{aligned} X_1 &\rightarrow \varphi_{12} \\ &\vdots \\ X_i &\rightarrow \varphi_{ij} \\ &\vdots \\ X_k &\rightarrow \varphi_{k, N(k)} \end{aligned}$$

とする。\$N(i)\$ は \$X_i\$ を左辺に持つプロダクションの数である。各 \$\varphi_{ij}\$ に対応して述語記号 \$\Phi_{ij}\$ を定め、\$\Phi_{ij}(x)\$ を次のように定義する。

$$\text{まず } \varphi_{ij} = Y_1 Y_2 \cdots Y_\nu \ (\nu \geq 0, Y_r \in V)$$

とする。\$Y_1, \dots, Y_\nu\$ の中にちょうど \$s\$ 個の nonterminal \$Z_1, \dots, Z_s\$ (\$0 \leq s \leq \nu, Z_r \in V - \Sigma\$) を含むとき、terminal だけでできている substring \$\kappa_0, \dots, \kappa_s\$ (null string もあり得る) を作って

$$\varphi_{ij} = \kappa_0 Z_1 \kappa_1 \cdots Z_s \kappa_s$$

とする。

(1) もし \$s=0\$ すなわち \$\varphi_{ij} = \kappa_0 \in \Sigma^*\$ なら、

$$\Phi_{ij}(x) \equiv x = R'(\kappa_0)$$

ここで一般に

$$\kappa = T_1 T_2 \cdots T_\mu, T_r \in \Sigma$$

とするとき、

$$\mu > 0 \text{ なら } R'(\kappa) = R(\bar{T}_1) * R(\bar{T}_2) * \cdots * R(\bar{T}_\mu)$$

$$\mu = 0 \text{ なら } R'(\kappa) = 1$$

と定義する。

(2) \$s > 0\$ のとき、\$Z_1, \dots, Z_s\$ に対応する述語を \$\Pi_1, \dots, \Pi_s\$ として、

$$\begin{aligned} \Phi_{ij}(x) &\equiv \exists y_1 \cdots y_s \{ y_1, \dots, y_s \leq x \\ &\quad \wedge x = R'(\kappa_0) * y_1 * R'(\kappa_1) * \cdots * y_s * R'(\kappa_s) \\ &\quad \wedge \Pi_1(y_1) \wedge \cdots \wedge \Pi_s(y_s) \} \end{aligned}$$

こうして定義された \$\Phi_{ij}\$ によって

$$\begin{aligned} E_i(x) &\equiv \Phi_{i1}(x) \vee \Phi_{i2}(x) \vee \cdots \vee \Phi_{i, N(i)}(x) \\ (1 \leq i \leq k) \end{aligned}$$

とする。構成の仕方から明らかに \$\varphi \in \{\varphi_{ij}\}\$ については

$$X_i \rightarrow \varphi \equiv \varphi = \varphi_{i1} \vee \cdots \vee \varphi = \varphi_{i, N(i)} \equiv E_i(\bar{\varphi})$$

だから、任意の \$\varphi \in V^*\$ に対し

$$X_i \xrightarrow{*} \varphi \equiv E_i(\bar{\varphi})$$

であることはすぐにわかる。

\$E_i\$ が recursive であることを示すには \$\Phi_{ij}\$ が recursive であることを見ればよい。

\$\Phi_{ij}\$ の構成のうち、(1) の場合には \$\Phi_{ij}\$ は primitive recursive である。(2) の場合右辺の \$y_1, \dots, y_s\$ はいずれも \$\leq x\$ であり、\$\Phi_{ij}\$ 自身が右辺にあらわれるときは、たとえば \$\Pi_\nu = \Phi_{ij}\$ であったとすると、条件 (*) によって \$y_\nu < x\$ である。したがって、\$V\$ の元に対する Gödel 数の与え方を考えると、\$\{\Phi_{ij}\}\$ は

同時的 course-of-value recursion を許すことになる。いいかえれば \$\Phi_{ij}\$ は recursive に定義される。(証終)

4. ALGOL 60 の recursiveness

ALGOL 60 の CF 文法が条件 (*) をみたしていることはいうまでもないので、各 nonterminal (“metalinguistic variable”) に対し、この命題の意味でそれを“表現”する述語を作ることができる。ただし comment に関する約束は次のように BNF によって与えられているものとする。

(a) 4.1.1 (Naur 1963) への追加:

(1) 冗長になるので書かないが、‘;’を除く basic symbol の sequence として <c-string> (comment string) を、またそれからさらに ‘end’ と ‘else’ を除いたものとして <e-string> を定義する。

(2)

$$\begin{aligned} \text{<semicolon>} &::= ; | \text{comment <c-string>} ; \\ \text{<begin phrase>} &::= \text{begin} | \text{begin comment} \\ &\quad \text{<c-string>} ; \\ \text{<end phrase>} &::= \text{end} | \text{end <e-string>} \end{aligned}$$

(b) 4.1.1 の修正

$$\begin{aligned} \text{<compound tail>} &::= \text{<statement>} \\ \text{<end phrase>} &| \text{<statement>} \\ \text{<semicolon>} &\text{<compound tail>} \\ \text{<block head>} &::= \text{<begin phrase>} \\ \text{<declaration>} &| \text{<block head>} \\ \text{<semicolon>} &\text{<declaration>} \\ \text{<unlabelled block>} &::= \text{<block head>} \\ \text{<semicolon>} &\text{<compound tail>} \end{aligned}$$

次の目標は WFP を定義することである。ただしここでは Naur 1963 に述べられたあらゆる context-sensitive なシンタックス規則を網羅するのではなく、簡単のため基本的な次の規則に限定して考える:

(1) 同一ブロック内で同じ label が 2 度現れてはならない。

(2) go to statement の行先は同じブロックの内部か、またはその外側のブロックの内部に限り、かつ label として定義されていなければならない。

(3) Label 以外の identifier はそれを含むブロックか、またはその外側のブロックにおいて declare されなければならない。同一ブロック内では異なる declaration は許されない。

まず CF 文法の部分で次の述語が定義されているも

のとする:

述語	対応する nonterminal
Lval (x)	<logical value>
Delm (x)	<delimiter>
Ident (x)	<identifier>
Gotost (x)	<go to statement>
Stmnt (x)	<statement>
Ctail (x)	<compound tail>
Prog (x)	<program>
Dcl (x)	<declaration>

ここで <declaration> のシンタックスにおいて,
下記の nonterminal が導入される部分はすべて < ...
D > の形に変更され, 新たに

< ... D > ::= < ... >

の形の規則が追加されているものとする.

述語	nonterminal
Svar D (x)	<simple variable>
Array D (x)	<array identifier>
Sw D (x)	<switch identifier>
Proc D (x)	<procedure identifier>

この変更は, 上記の述語を declaration の context
の中でのみ真であるようにするためである.

また, basic symbol X によって作られる single-
character string の Gödel 数は $R(p(x))$ である
が, これを 'Q(X)' と略記する (これもメタメタ記
号である).

以下の定義では, primitive recursive であることを
を強調するため, redundant な不等式を挿入してあ
る.

#10 $Vdlm(x) \equiv Lval(R(x)) \vee Delm(R(x))$

x は logical value または delimiter

#11 $x \text{ Head } y \equiv x \leq y \wedge \exists z(z \leq y \wedge y = x * z)$

#12 $x \text{ Tail } y \equiv x \leq y \wedge \exists z(z \leq y \wedge y = z * x)$

#13 $x \text{ Occ } y \equiv x > 1 \wedge \exists z(z \leq y \wedge z \text{ Head } y \wedge x \text{ Tail } z)$

'Occ' は "occurs in" をあらわす.

#14 $\text{Labelist}(x) \equiv \exists z[z \leq x \wedge \text{Ident}(z) \wedge$

$\times x = z * Q(:)]$

$\vee \exists y, z[y, z \leq x \wedge \text{Ident}(z) \wedge$

$x = y * z * Q(:) \wedge \text{Labelist}(y)]$

x は label list すなわち:

<label> : <label> : ... :

#15 $x \text{ Stmntin } y \equiv \text{Prog}(y) \wedge x \text{ Occ } y \wedge \text{Stmnt}(x)$

$\wedge \exists z[z \leq y \wedge z \text{ Tail } y \wedge \text{Ctail}(z)$

$\wedge x \text{ Occ } z \wedge \forall u, v[u, v \leq z \wedge$
 $z = u * x * v \supset \text{Labelist}(u)]\}$

x は program y の内部の statement. ここで "内
部" というとき, y に含まれる (y 以外の) block
または compound の内部を含まない.

#16 $x \text{ Labelin } y \equiv \text{Prog}(y) \wedge x \text{ Occ } y \wedge \text{Ident}(x)$

$\wedge \exists u, v[u, v \leq y \wedge u \text{ Stmntin } y$

$\wedge v \text{ Stmntin } y \wedge u = x * Q(:) * v]$

x は program y の内部の label.

#17 $x \text{ Identin } y \equiv \text{Prog}(y) \wedge \text{Ident}(x)$

$\wedge \exists z[z \leq y \wedge z \text{ Stmntin } y \wedge x \text{ Occ } z$

$\wedge \exists m, n, u, v[m, n \leq l(z) \wedge u, v \leq z$

$\wedge z = u * x * v \wedge l(n) = m \wedge l(u * x)$

$= n \wedge Vdlm(m \text{ Gl } y)$

$\wedge Vdlm((n+1) \text{ Gl } (y))]\}$

x は program y の内部の identifier

#18 $x \text{ Gotolabel } y \equiv x \text{ Identin } y$

$\wedge \exists z[z \leq y \wedge z \text{ Stmntin } y$

$\wedge \text{Gotost}(z) \wedge \exists u, v, n\{u, v$

$\leq z \wedge n \leq l(z) \wedge z = u * x * u \wedge n$

$= l(u * x) \wedge [Q(\text{go to}) \text{ Tail } u$

$\vee Q(\text{then}) \text{ Tail } u \vee Q(\text{else})$

$\text{Tail } u] \wedge \neg(n+1) \text{ Gl } z = p(\{\})\}$

x は program y の内部にある go to statement
の中の go to label.

#19 $x \text{ Dclby } y \equiv x \text{ Occ } y \wedge \text{Dcl } y$

$\wedge \{\text{Svar } D(x) \vee \text{Array } D(x)$

$\vee \text{Sw } D(x) \vee \text{Proc } D(x)\}$

x は declaration y によって declare される.

#20 $\text{WFP}(x) \equiv \text{Prog}(x)$

$\wedge \forall y(y \leq x \wedge y \text{ Occ } x \wedge \text{Prog}(y) \supset$

$\forall u\{u \leq y \wedge u \text{ Labelin } y \supset \exists v[v \leq y$

$\wedge \text{Ctail}(v) \wedge v \text{ Tail } y \wedge u \text{ Head } v \wedge \forall t$

$(t \leq y \wedge \text{Ctail}(t) \wedge t \text{ Tail } y \wedge u \text{ Head } t$

$\supset t = v)\}\}$

$\wedge \forall u\{u \leq y \wedge u \text{ Gotolabel } y \supset$

$\exists v[v \leq x \wedge v \text{ Occ } x \wedge \text{Prog}(v)$

$\wedge y \text{ Occ } v \wedge u \text{ Labelin } v]\}$

$\wedge \forall u\{u \leq y \wedge u \text{ Identin } y \wedge \neg u \text{ Labelin } y \supset$

$\exists z[z \leq x \wedge y \text{ Occ } z \wedge z \text{ Occ } x$

$\wedge \text{Prog}(z) \wedge \exists !v(v \leq z \wedge v \text{ Occ } z$

$\wedge \text{Dcl } v \wedge u \text{ Dclby } v)\}\}$

述語 WFP は CF 文法に対して定義された各述語
に関して primitive recursive であり, したがって

命題 2: #20 で定義される述語 WFP は recursive である。

Well-formed program の概念が論理的体系における“証明可能”の概念などと異なるのは、“証明の列”のような、与えられたストリングの構造だけによって決まらない対象を要求しない点にある。したがって、シンタックス規則がどんなに context-sensitive であっても、それがストリングの構造だけに関するものであって、ストリングを生成する仕方に関する記述を含まなければ、上と同様にして WFP の概念を effective に（つまり recursive に）定義できるに違いない。そのようにして得られる文法は decidable であり、現存するプログラム言語はいずれもこの条件を満足していると考えられる。

5. Recursive 言語とその文法

述語 WFP の定める言語 L は、CF 文法によって生成される言語 $\bar{L}$ の部分集合である。すなわち、 L は $\bar{L}$ から L に属さない (well-formed でない) ストリングを排除することによって得られる。

一般に $L_1 \subset L_2$ で、 L_1 を定義する述語 ϕ が L_2 を定義する述語 ψ から recursive に定義される場合、“ ϕ は L_2 から recursive に定義される”ということにする。

CF 言語は recursive だから、CF 言語 $\bar{L}$ から、recursive に定義される言語 L は recursive である。この場合 $\bar{L}$ を L の“CF extension”と呼ぼう。

逆に任意の $L \subset \Sigma^*$ が recursive であるとき、 L を部分集合とする CF 言語 $\bar{L}: L \subset \bar{L} \subset \Sigma^*$ が少なくともひとつ存在する。事実 Σ^* はそのひとつである。そのような $\bar{L}$ のひとつをとると、 $\bar{L}$ も L も recursive だから $\bar{L}-L$ も recursive である。したがって L を定義する述語 ϕ は L から recursive に定義される。いいかえれば $\bar{L}$ は L の CF extension である。なぜなら、 $\bar{L}$ を定義する述語を ψ 、 $\bar{L}-L$ を定義する述語を Π とすると

$$\phi(x) \equiv \psi(x) \wedge \neg \Pi(x)$$

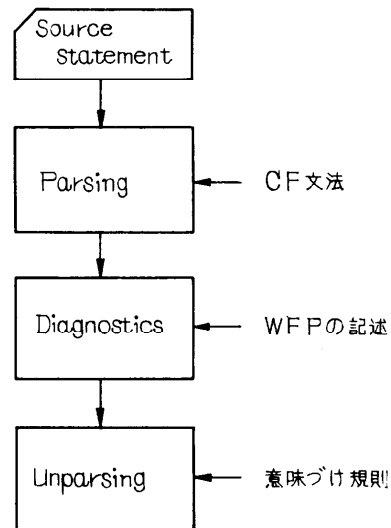
Π の定義関数は recursive、したがって μ -recursive だから、initial function だけから μ -operator を含む recursion によって定義されており、故に上式の右辺には ϕ を含まない。こうして、

命題 3: ストリングの集合 L が recursive であることは、 L が CF extension を持つことと同等である。

6. むすび

従来の syntax-directed compiler のひとつの弱点は、CF 文法の形に記述できない文脈上の制約については、あらかじめ確実に守られているものと仮定せざるを得ない点にあった。

これまでと同じ形の CF 文法と意味づけ規則 (semantic rules) に、さらに WFP を記述するインプットを加えてやると、context-sensitive な diagnostic routine を持った compiler ないし analyzer ができる。



一階述語論理は決定不能なので一般的な決定アルゴリズムは存在しないのだが、今の場合には現れる述語がすべて recursive、特に quantification がすべて finite だという利点があるので、任意に与えた x に対して WFP(x) の真偽を決定できる。また WFP の決定に必要な substring だけを parsing の段階で残しておけば、能率のよい diagnostics が可能になる。

ただし本稿で使った Gödel numbering は現実の implementation では実用にならないので、concatenation function “*” を適当な方法で導入しなければならない。

参考文献

- 1) Gilbert, P.: On the Syntax of Algorithmic Languages. J. ACM, 13, no. 1, 90-107 (1966)
- 2) Ginsburg, S. The Mathematical Theory of Context Free Languages, McGraw-Hill (1966)

- 3) Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatsh. f. Math. u. Physik, **38**, 173-198 (1931)
- 4) Gödel, K.: On Undecidable Propositions of Formal Mathematical Systems, in M. Davis 1965: The Undecidable, Raven Press (1934)
- 5) Ingerman, P.Z.: A Syntax-Oriented Translator, Academic (1966)
- 6) Kleene, S.C.: Introduction to Metamathematics, North-Holland および Van Nostrand (1952)
- 7) Naur, P.: Revised Report on the Algorithmic Language ALGOL 60. Comm. ACM, **6**, no. 1, 1-17 (1963)
- 8) Putnam, H.: Some Remarks in the Theory of Grammar, Proc. of Symposia in Applied Math., AMS, **12**, 52-42 (1961)
- 9) Quine, W.V.O.: Mathematical Logic, Revised Ed., Harvard Univ. Press (1951)

(昭和 41 年 12 月 1 日 受付)