Vol. 4 No. 4　　情　報　処　理　　July 1963

Phrase structure grammar の構造分析について*

長　尾　　真**

まえがき

ALGOL 60 はいわゆる Backus notation によりその規則が表現されているのであるが[1~3]，これはほぼ phrase structure grammar と考えることができる[4]．phrase structure grammar は，文章の生成(sentence generation)としての面から，主として N. Chomsky により研究されているが[5,6]，逆に一つの string が与えられたときに，それが与えられた言語生成法則から生じたものか，またそうであればその string はいかなる構造をもっているかということを調べる必要がある．与えられた言語生成法則の数が有限であり，与えられた string が有限の長さである場合には，かならず上記の問題に対する解をうることができるのであるが，それを最も能率よく実行するための工夫がなされねばならない．

stack structure[7]は数式の処理を能率よく行なうために考えられたものであるが，これが phrase structure language における文章の構造分析においていかなる位置を占めるかを考察し，その適用範囲を知ることは大切である．この論文では phrase structure grammar にいくつかの制限を加えた場合の問題を主として取扱い，その場合の文章の構造分析の方法について述べた．

1. Phrase structure grammar の一意性†

terminal symbol の集合 V_T が互に分離した n 個の部分集合 V_{Ti} に分割されているものとする．すなわち

$$V_T=V_{T1}\cup V_{T2}\cup \cdots\cdots \cup V_{Tn}$$
$$V_{Ti}\cap V_{Tj}=\phi \quad (i\neq j)$$

部分集合 V_{Ti} に α_i なる symbol を附属させ，この α_i を V_{Ti} を代表する semi-terminal symbol と名づける．α_i の集合を V_α とする．

$$V_\alpha=\{\alpha_1, \alpha_2, \cdots\cdots, \alpha_n\}$$

α_i の現われる位置には V_{Ti} の任意の一つの要素をおきかえうることを意味する．

次に characteristic symbol の集合 D_c があり，互に分離した K 個の部分集合 D_i に分割されているものとする．すなわち

$$D_c=D_1\cup D_2\cup \cdots\cdots \cup D_k$$
$$D_i\cap D_j=\phi \quad (i\neq j)$$

部分集合 D_i に δ_i なる symbol を付属させ，δ_i の集合を D で表わす．

$$D=\{\delta_1, \delta_2, \cdots\cdots, \delta_k\}$$

さらに non-terminal symbol の集合を S，その要素を β_i で表わせば，

$$S=\{\beta_1, \beta_2, \cdots\cdots, \beta_l\}$$

そこで集合 M を

$$M=V_\alpha\cup D\cup S$$

とし M の要素を一般に ρ_i で表わすものとする．

M の要素をいくつか1列にならべたものを syntactic unit と呼び φ_{ij} で表わす．

$$\varphi_{ij}=\rho_{\nu_1}\rho_{\nu_2}\cdots\cdots\rho_{\nu_m},\ \rho_{\nu_i}\in M \qquad (1)$$

ただし少なくとも一つの ν_i について $\rho_{\nu_i}\in V_\alpha\cup S$ であるものとする．また $\rho_{\nu_i}\in D$, $\rho_{\nu_{i+1}}\in D$ なるときは $\rho_{\nu_i}\rho_{\nu_{i+1}}$ を一つの $\rho_\nu\in D$ としてよいので，今後は ρ_{ν_i}, $\rho_{\nu_{i+1}}$ ともに D の要素であるという場合は考えない．

いま二つの string χ, ω があり，χ が ω でおきかえうるとき

$$\chi\rightarrow\omega$$

とかきあらわす．context-free phrase structure grammar における言語生成法則とは，χ が一つの symbol からなる場合であって，これを β とすると

$$\beta\rightarrow\omega$$

とかける．さらに β におきかえうる string が ω だけでなく，多数あるときは

$$\beta\rightarrow\omega_1,\ \beta\rightarrow\omega_2,\ \cdots\cdots$$

と書けるが，これを

$$\beta=\{\omega_1, \omega_2, \cdots\cdots\}$$

と表わす．

そこで，一般の context-free phrase structure grammar は式(1)で定義された syntactic unit

---

* Syntactic analysis for phrase structure grammar, by Makoto Nagao (Faculty of Engineering, Kyoto University)

** 京都大学工学部

† この節は主として N. Chomsky の文献(6)による．

を用いて次のように表わせる*.

$$\left.\begin{aligned}\beta_1&=\{\varphi_{11}, \varphi_{12}, \cdots\cdots, \varphi_{1m_1}\}\\ \beta_2&=\{\varphi_{21}, \varphi_{22}, \cdots\cdots, \varphi_{2m_2}\}\\ &\cdots\cdots\cdots\\ \beta_l&=\{\varphi_{l_1}, \varphi_{l_2}, \cdots\cdots, \varphi_{lm_l}\}\end{aligned}\right\}\qquad(2)$$

式（2）が phrase structure grammar であるためには，すべての β_i について（2）をみたす $V_T \cup D$ の要素のみからなる string が少なくとも一つ存在しなければならない．すなわち $V_T \cup D$ の要素からなる (terminal) string が生成できるように（2）を構成しなければならない．

このような string が存在しないような構造はここでは考えない．

phrase structure grammar（2）をみたす $V_a \cup D$ 上の string は次のようにして生成することができる．

$$\left.\begin{aligned}\eta_0&=(\beta_{01}, \beta_{02}, \cdots\cdots, \beta_{0l})=(0, 0, \cdots\cdots, 0)\\ \eta_1&=(\beta_{11}, \beta_{12}, \cdots\cdots, \beta_{1l})=(\psi_1(\eta_0), \psi_2(\eta_0), \cdots\cdots, \psi_l(\eta_0))\\ &\cdots\cdots\end{aligned}\right\}\qquad(3)$$

一般に第 i 段階では

$$\eta_i=(\beta_{i1}, \beta_{i2}, \cdots\cdots, \beta_{il})=(\psi_1(\eta_{i-1}), \psi_2(\eta_{i-1}), \cdots\cdots, \psi_l(\eta_{i-1}))$$

ここに $\psi_k(\eta_{i-1})$ は（2）の

$$\beta_k=\{\varphi_{k1}, \varphi_{k2}, \cdots\cdots, \varphi_{km_k}\}$$

の右辺の各 syntactic unit 中において，$\beta_1, \beta_2, \cdots\cdots, \beta_l$ が現われるところへ，それぞれ $\beta_{i-1,1}, \beta_{i-1,2}, \cdots\cdots, \beta_{i-1,l}$ を代入して得られる β_k に属する string の集合を表わす．

このようにして各 β_k について $V_a \cup D$ 上の string が生成されるが，ある β_k に属する生成された string の集合中に同じ string が二つ以上現われれば，その phrase structure grammar が β_k について一意的でない**．すべての β_k について文法が一意的であるとき，文法が一意的であるという[6,8]．

例 1.　$V_T=V_{T1}$，その名前を α とする．また D の要素も一つとする．

$V_a=\{\alpha\}$, $D=\{\delta\}$.

生成法則：$\beta_1=\{\alpha\}$, $\beta_2=\{\beta_1, \beta_2\delta\beta_1\}$.

$$\eta_0\begin{cases}\beta_{01}=\{0\}\\ \beta_{02}=\{0\}\end{cases},\quad \eta_1\begin{cases}\beta_{11}=\{\alpha\}\\ \beta_{12}=\{0\}\end{cases},\quad \eta_2\begin{cases}\beta_{21}=\{\alpha\}\\ \beta_{22}=\{\alpha\}\end{cases},$$

$$\eta_3\begin{cases}\beta_{31}=\{\alpha\}\\ \beta_{32}=\{\alpha, \alpha\delta\alpha\}\end{cases},\ \cdots\cdots$$

一般に n 段階では

$$\eta_n\begin{cases}\beta_{n1}=\{\alpha\}\\ \beta_{n2}=\{\alpha, \alpha\delta\alpha, \alpha\delta\alpha\delta\alpha, \cdots\cdots, (\alpha\delta)^{n-2}\alpha\}\end{cases}$$

となり，同じ string が二つ以上現われないから grammar は一意的である．

例 2.　$V_a=\{\alpha\}$, $D=\{\delta\}$.

生成法則：$\beta_1=\{\alpha, \beta_1\alpha\}$, $\beta_2=\{\beta_1, \delta\beta_2\beta_1\}$.

$$\eta_0\begin{cases}\beta_{01}=\{0\}\\ \beta_{02}=\{0\}\end{cases},\quad \eta_1\begin{cases}\beta_{11}=\{\alpha\}\\ \beta_{12}=\{0\}\end{cases},\quad \eta_2\begin{cases}\beta_{21}=\{\alpha, \alpha\alpha\}\\ \beta_{22}=\{\alpha\}\end{cases},$$

$$\eta_3\begin{cases}\beta_{31}=\{\alpha, \alpha^2, \alpha^3\}\\ \beta_{32}=\{\alpha, \alpha^2, \delta\alpha^2, \delta\alpha^3\}\end{cases}$$

$$\eta_4\begin{cases}\beta_{41}=\{\alpha, \alpha^2, \alpha^3, \alpha^4\}\\ \beta_{42}=\{\alpha, \alpha^2, \alpha^3, \delta\alpha^2, 2\,\delta\alpha^3, 2\,\delta\alpha^4, \delta\alpha^5, \delta^2\alpha^3, 2\,\delta^2\alpha^4, 2\,\delta^2\alpha^5, \delta\alpha^6\}\end{cases}$$

$\cdots\cdots$

となり文法は一意的でない．たとえば $\delta^2\alpha^4=\delta\delta\alpha\alpha\alpha\alpha$ は次のように3通りに分解することができる．

$$\begin{aligned}\delta\delta\alpha\alpha\alpha\alpha&=\delta(\delta((\alpha)\alpha)\alpha)\alpha=\delta(\delta(\alpha)\alpha)((\alpha)\alpha)\\ &=\delta(\delta\alpha((\alpha)\alpha))\alpha.\end{aligned}$$

2. Characteristic symbol をもつ non-recursive grammar

以下の議論においては文法の一意性が成立しているものとして話を進める．1で示した文法（1），（2）は一般的すぎるので次のような二つの制限をもうける．

（i）（1）において $m \geq 2$ のとき

$\varphi_{ij}=|\delta|$, $\delta \epsilon D$.

ここに $|\delta|$ は δ が syntactic unit φ_{ij} を構成する要素であることを示すものとする．

（ii）同一の $\delta \epsilon D$ が文法法則（2）中，2個所以上に現われることはない．つまり一つの δ は，ある一つの syntactic unit φ に固有のものであるとする．

このような制限をもつ文法を文法 G と名づける．

制限 (i), (ii) のもとに文法 G が recursive な構造をもっているかどうかを次のようにして調べることができる．

式（1），（2）を用いて順次 $\eta_0, \eta_1, \eta_2, \cdots\cdots$ を作って行くとき生成される，ある β_i に属する $V_a \cup D$ 上の一つの string 中に $|\delta|=\varphi_{ij}$ であるような δ が2個所以上に現われる場合がある．これはその生成過程から考えて容易にわかるように文法 G が recursive な構造になっていることを示している．

---

* 一般に行なわれている phrase structure grammar の定義は，ここに述べるのとは少しちがう．文献（5），（6）等参照．

** これには $\beta_i \to \beta_j$ のような rule が含まれていないという条件がいる[6]．

次に集合 $S \cup V_a$ に順序関係を入れることを試みる．順序としては，β_i に属する syntactic unit φ_{ij} において $|\rho| = \varphi_{ij}$, $\rho \notin D$ なる ρ があるとき $\beta_i (= \rho_k) < \rho$ とする．一般に文法 G によって関係づけられた集合 $S \cup V_a$ はこの意味で順序集合とはなっていない．しかし G は言語生成法則であるから次のことがいえる．

文法 G から適当な syntactic unit φ_{ij} をのぞいて集合 $S \cup V_a$ を半順序集合にすることができる．このとき，半順序という性質が失なわれない範囲で，のぞかれる φ_{ij} を最も少なくしたような phrase structure grammar を作りこれを G' とする．このような操作はかならず行ないうる．つまり recursion の原因となる syntactic unit φ_{ij} を除いても，β_i の右辺の項がすべてなくなるということはない．もしそうであれば文法 G は $V_a \cup D$ 上の string を生成しえなくなる．このようにして文法 G' によって規定される順序関係により集合 $V_a \cup D$ について Hasse の図式がかけることになる．

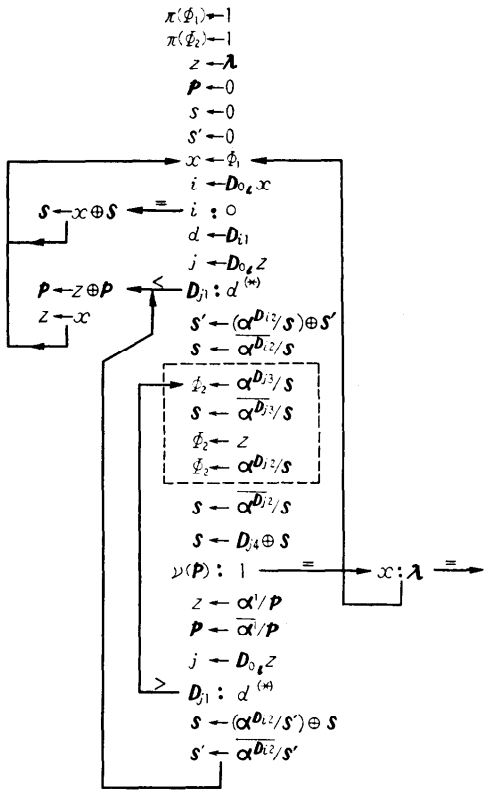
集合 D の各要素は syntactic unit に固有のものであるから $\varphi_{ij} (j = 1, 2, \cdots\cdots, m_j)$ に属する D の要素すべての集合を D_{β_i} とすると，部分集合族 $\{D_{\beta_1}, D_{\beta_2}, \cdots\cdots, D_{\beta_l}\}$ にも順序関係を考えることができて，それは集合 $S \cup V_a$ の順序関係における S の要素の順序関係に等しいとする．

このようにして D に順序関係を定めると，これは文章の構造分析において次のような意味をもつ．

（i） $\delta_i > \delta_j$ は，δ_i に関する syntactic unit を，δ_j に関する syntactic unit よりも先に処理すべきことを示す．

（ii） δ_i と δ_j が比較不可能なとき，一つの string 中の δ_i を含む適当な部分と δ_j を含むそれとは各々独立な関係であることを示す．よって比較不可能な δ_i と δ_j が一つの string 中で（D の要素のみに着目した場合）となりあっているとき，その各々は優先順位のことを考慮せず独立に処理できる．

D の要素はある syntactic unit φ に固有なものであり，φ の長さが 2 以上の場合はかならず一つ以上の



第 1 図*

Φ_1: input file で，与えられた string がはいっている．string の最後には terminal partition λ がついているものとする．
Φ_2: output file
P: D の要素のための stack
S: V_a の要素のための stack
S': S に対する補助 stack
D: 文法 G' に関する情報のはいっている matrix
D_0: $\lambda \cup D$ の列 vector
D_1: D における順序関係を示す記号の列 vector．λ は極小要素とする．
D_2: $\Delta_H \rho_i \Delta_T = \varphi_{ij}$ とするとき $\nu(\Delta_H)$ の列 vector．
D_3: 〃 〃 $\nu(\Delta_T)$ の 〃
D_4: $\rho_i \in D_{\beta_j}$ であるような β_j の列 vector．
（*） D_{j1} と d とが比較不可能の場合には分岐せずまっすぐ下へゆく．

* 図の表記法は主として K. E. Iverson の方法[9]によっている．この表記法はプログラムを実行する計算機等の特殊性によらない一般的な表記法である．主な記号の表す意味は次のとおりである．
$\pi(\Phi) \leftarrow 1$: file Φ を rewind する．
$\Phi \leftarrow a$: a を file Φ の現在点から forward record する．
$a \leftarrow \Phi$: file Φ の現在点から forward read し，a に入れる．
$x_\iota x$: vector x の要素 x_ι が $x_\iota = x$ であるような最少の ι を示す．このような ι のない場合を ° で表す．
$a^j(n)$: prefix vector of weight j. 長さ n でその初めの j 個が 1，その他は 0 である logical vector. vector の次元が明らかなときは (n) は省略する．
a^j/x: compression of x. vector a^j の 1 の存在する位置に対応する vector x の要素をその順にとり出して作った vector.
$a \oplus b$: a と b とをこの順に catenate する．

Dの要素を含んでいることから，この場合の構造分析はもっぱらDの要素の間の順序関係のみに注目して行なうことができ，その他の要素について考慮する必要はない．そこで与えられた $V_a \cup D$ 上の string が，半順序をもつ与えられた文法 G' から生成されているとするとき，その構造分析は**第1図**に示すようにすればよい．ただし syntactic unit φ が

$$\varphi_i = |\delta_{i_1}|\delta_{i_2}|$$

のようにDの要素を二つ以上含んでいる場合にはこの図はすこしかえねばならない．δ_{i_1} と δ_{i_2} とは同じ D_{β_i} に属しているので同じ優先順位である．よってこの syntactic unit 内での処理の順番は適当にきめるか，別に付帯条件があればそれにしたがうようすればよい．

図において点線でかこんだ部分は一つの syntactic unit に関する処理をするところで，目的により種々の操作を行なえばよい．ここでは単にその syntactic unit を output file に出したにとどまるが，たとえば ALGOL のような場合には，対応する object program を作ることになるし，言語翻訳の場合にはその syntactic unit に対する訳および語順を決定すること等にあたる．stack p に記憶される D の要素は top から bottom に向って

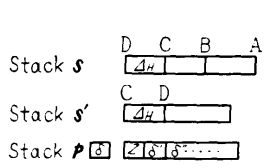
$$z > \delta_{p_1} > \delta_{p_2} > \cdots\cdots$$

の順序関係になっているので，z の次に現われる $\delta \epsilon D$ と z との順序が $z \geq \delta$ となっている場合，または z と δ とが比較不可能である場合には，z は局所的にその順位が極大である．よって z に関する演算が実行できる．

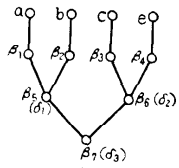
この場合に用いるべき working memory は，一般にはいわゆる last-in-first-out の stack ではうまくゆかない．**第2図**に示すように，characteristic symbol z に関係する V_a の要素が stack S 中 BC 間にあり，次の characteristic symbol δ が現われたときには，z に直接関係のない V_a の要素が CD として stack S に記憶されている場合がある．よってこれは別のところ S' へ移動させて BC 間の symbol が top になるようにする必要がある．

例 3．文法 G: $\beta_1=\{a\}$, $\beta_2=\{b\}$, $\beta_3=\{c\}$, $\beta_4=\{e\}$, $\beta_5=\{\delta_1\beta_1\beta_2\}$, $\beta_6=\{\beta_3\delta_2\beta_4\}$, $\beta_7=\{\delta_3\beta_5\beta_6\}$

この場合は $G'=G$ となり，Hasse の図式は**第3図**のようになっている．この文法より生成される β_7 に属する string $\delta_3\delta_1 abc\delta_2 e$ を，δ_1, δ_2, δ_3 のみをたよりにして構造をしらべる場合，δ_1 の次に δ_2



第2図



第3図

が現われたときには，δ_1 に関係のない要素 c が stack S の top にはいっていて，δ_1 に関する演算をすぐ実行することができない．

これは syntactic unit φ において non-terminal symbol β が連続して二つ以上ならぶときに生ずる可能性がある．これに対して φ 中で S の要素が連続せず，かならず characteristic symbol が間にはいっている場合には recursive definition をも含めて，上記のようなことは起らず，last-in-first-out の stack を用いることができる．

3. Characteristic symbol をもつ recursive grammar

文法 G' は recursive structure をもっていないから，有限の数の string しか生成しえないという点で不都合である．そこで recursive definition を許した文法Gを考える．Gにおいては，一般に集合$S \cup V_a$ は順序集合とはなりえない．しかし文法を G から G' に制限したときになりたつ順序関係を，Gにおいてもある程度保存していることは当然である．

3.1 Explicit recursion を許す場合

explicit recursion とは $\beta_i \to \varphi_{ij}$ であるときに

$$\varphi_{ij} = |\beta_i|$$

であることをいうものとする．この場合には recursion は β_i だけについて局所的であって，文法の他の部分に影響を与えないので G' について定義した順序関係は保存されている．つまり characteristic symbol の集合 D は，全体として半順序関係をもつ．しかし explicit recursion の存在する non-terminal symbol β_i に属する D_{β_i} の要素に関しては，さらにくわしい性質を調べなければならい．

簡単のために，recursion の存在する β_i に対応した D_{β_i} がただ一つの要素 δ からなるとして

$$\varphi_{ij} = |\delta|,\ \delta \epsilon D_{\beta_i}$$

とするとき，'δ, δ' なるものを定義する．

'δ: explicit recursion により β_i に関する $V_a \cup D$ 上の string 中に，δ が二つとなりあってなら

んだとき（その間に V_a の要素がはいってもよい）左側にある δ を示す．

δ': $'\delta$ とは逆に，二つならんだ右側の δ を示す．

$'\delta$ と δ' との間の優先順位は，φ_{ij} の作り方によってことなり，次のようになる．

（i） $\varphi_{ij}=|\delta|\beta_i|$

つまり characteristic symbol δ が β_i の左にある場合である．上式の右辺の β_i に $\beta_i\rightarrow\varphi_{ij}$ の規則にしたがって φ_{ij} を代入すれば

$$\varphi_{ij}=|\delta|(|\delta|\beta_i|)|=|'\delta|\delta'|\beta_i|$$

となり，これより $'\delta<\delta'$ が結論される．

（ii） $\varphi_{ij}=|\beta_i|\delta|$

この場合は（i）と逆になり $'\delta>\delta'$.

(iii) $\varphi_{ij}=|\delta|\beta_i|\beta_i|$, $\varphi_{ij}=|\beta_i|\delta|\beta_i|$

などの場合はどの β_i を先に計算すべきかは明らかでないから，優先順位をきめることはできない．

さらに β_i に属する syntactic unit が二つあって

(iv) $\varphi_{ij}=|\delta_{i_1}|\beta_i|$, $\varphi_{ik}=|\delta_{i_2}|\beta_i|$

である場合は $'\delta_{i_1}<\delta_{i_2}'$, $'\delta_{i_2}<\delta_{i_1}'$.

$\varphi_{ij}=|\delta_{i_1}|\beta_i|$, $\varphi_{ik}=|\beta_i|\delta_{i_2}|$ などの場合は (iii) と同様に，優先順位をきめることができない．

結局，explicit recursion が (i), (ii), (iv) の形の場合には，δ に関して上記のような左右の順序関係を導入することによって，syntactic unit を構成するすべての要素を調べることなく，D の要素のみに着目して与えられた string の構造分析が行なえる．

たとえば ALGOL 60 における規則

<simple arithmetic expression>::=<term>|
<simple arithmetic expression><adding operator><term>

においては上記（ii）より

$'$<adding operator> > <adding operator>$'$

となる．これよりたとえば a+b+c+d は (((a+b)+c)+d) と解釈すべきことが規則から直接要求されることである．もし上記の規則のかわりに

<simple arithmetic expression>::=<term>|
<term><adding operator><simple arithmetic expression>

となっていれば

$'$<adding operator> <<adding operator>$'$

となり a+b+c+d は (a+(b+(c+d))) と解されねばならない．

ALGOL 60 では left to right principle がとられているが，もし syntax に忠実な計算順序をとるものとすれば，この原理をとらずとも規則そのものが，その順序を自ずから規定しているとみることができる．これに対して，例えば $\varphi=\delta_1\beta_1\beta_2$ などの場合にはこの syntactic definition からだけでは β_1 と β_2 のどちらを先に求めておくべきかが明らかでない．syntax では規定できない順序に関する ambiguity はこういうところに存在する．よって，こういう場合には，semantic に順序に関する規定をもうけねばならない．

3.2 Implicit recursion を許す場合

これは文法 G 中に

$$\beta_i\rightarrow\varphi_{ij},\quad \varphi_{ij}=|\beta_k|$$
$$\beta_k\rightarrow\varphi_{kl},\quad \varphi_{kl}=|\beta_i|$$

のような規則の含まれている場合であって D の要素の優先順位はそれら互のならび方により種々かわる．この場合には **3.1** でのべた characteristic symbol に関する左右の順序関係を D のすべての要素間でしらべねばならない．そこで文法 G から生ずる string について次のような性質をしらべる．

（a） β_i に属する $V_a\cup D$ 上の string はいかなる symbol で構成されるか．

（b） 生成される string において，$V_a\cup D$ の一つの要素 ρ の直接右隣り（左隣り）に来ることのできる要素は何か．

（c） 生成される string において D の一つの要素 δ の右隣り（左隣り）に来ることのできる D の要素は何か（それら二つの D の要素の間に V_a の要素が存在していてもよいとする）．それら隣りあう二つの D の要素間の優先順位はどうなるか．

（a） 文法 G の規則（2）について次のような Boolean 行列 A を作る．行列 A は M の要素すべて $(\rho_1, \rho_2, \cdots\cdots, \rho_n)$ を行および列とする正方行列である．(ρ_i, ρ_j) に対応する行列の要素を A_{ij} とすると

$A_{ii}=1$

$A_{ij}=1$: $\rho_i\epsilon S$ で，ρ_i に属する syntactic unit φ が $\varphi=|\rho_j|$ であるようなすべての (i, j) に対して．

$A_{ij}=0$: その他の場合

一般に Boolean 行列 A, B の積 $A\wedge B$, 和 $A\vee B$ を次のように定める．$C=A\wedge B$, $D=A\vee B$ とすると

$$C_{ij}=\vee_k(A_{ik}\wedge B_{kj}),$$
$$D_{ij}=A_{ij}\vee B_{ij}.$$

そこで上記の行列 A について

$A^2=A\wedge A$

$A^3=A^2\wedge A$

……

$A^n=A^{n-1}\wedge A$

を作ってゆく．こうして $A^k=A^{k+1}$ となったときの行列 A^k を A^∞ と表わすと[10]，A^∞ の β_i に対応する行において1の存在する列の要素はすべて β_i に属する string 中に現われうることを示している．これは1でのべた文法 G からの文章生成の過程(3)と，行列 A の積を作って行く過程との対応関係をみれば容易にわかる．

(b) 次のような M のすべての要素を行および列とする Boolean 行列 B を作る．

$B_{ii}=1$

$B_{ij}=1$: $\rho_i\epsilon S$ で ρ_i に属する syntactic unit φ の最も左の要素が ρ_j である ($\varphi=\rho_j|$) ようなすべての (i,j) について．

$B_{ij}=0$: その他の場合．

行列 B と同じようにして次のような行列 C を作る．

$C_{ii}=1$

$C_{ij}=1$: $\rho_i\epsilon S$ で ρ_i に属する syntactic unit φ の最も右の要素が ρ_j である ($\varphi=|\rho_j$) ようなすべての (i,j) について．

$C_{ij}=0$: その他の場合．

さらに文法 G の syntactic unit (2) において

$\varphi=|\rho_i\rho_j|$

であるとき行 ρ_i，列 ρ_j の要素 $D_{ij}=1$，その他の (i,j) について $D_{ij}=0$ なる行列 D を作る．

そこで

$DB=D\wedge B$

$DB^2=DB\wedge B$

……

$DB^k=DB^{k-1}\wedge B$

なる Boolean 積の行列を作ってゆく．$DB^k=DB^{k+1}$ となる最終状態がえられたときこれを DB^∞ と表わす．この行列はその作り方からわかるように，その (i,j) 要素が1である場合，文法 G のもとに ρ_j は ρ_i の右側に ρ_i より優先順位が上で直接つながりうることを示す．

同様にして

D^tC^∞

を作ると，この行列の (i,j) 要素が1である場合，ρ_j が ρ_i より優先順位が上で ρ_i の左側に直接つながりうることを示す．

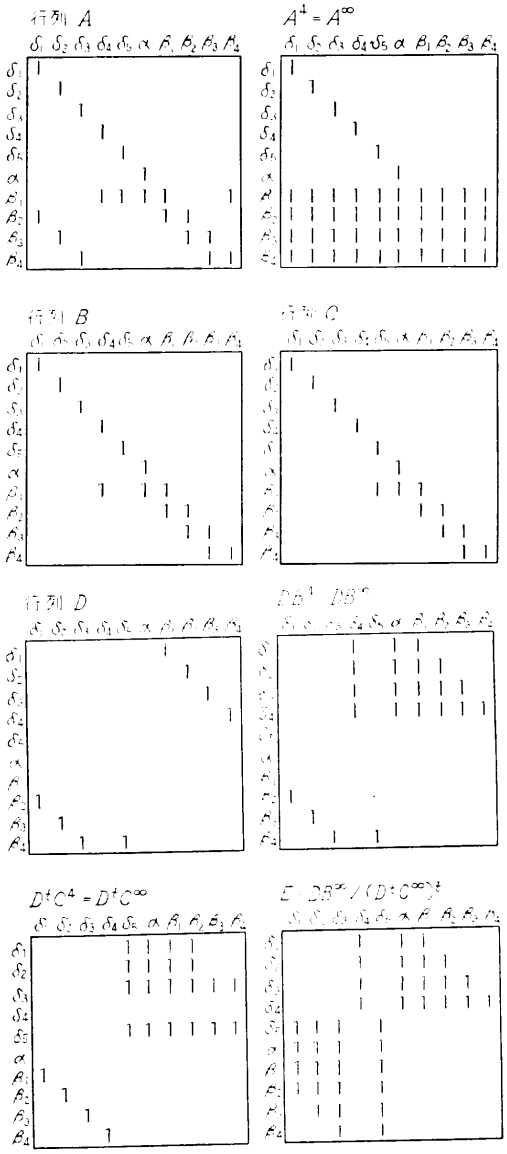
よって DB^∞ と $(D^tC^\infty)^t$ との論理和の行列

$E=DB^\infty\vee(D^tC^\infty)^t$

を作れば，その (i,j) 要素が1であるとき，ρ_i の右側に ρ_j が直接つながりうることを示している．

これらの行列を利用して与えられた string の接続関係をしらべれば，これが文法 G から生成されたものであるかどうかを test する一つの手段となりうる．

例 4. $\delta_1=\{\uparrow\}$, $\delta_2=\{\times,/\}$, $\delta_3=\{+,-\}$, $\delta_4=\{(\}$, $\delta_5=\{)\}$, $\alpha=\{n,v\}$.



第 4 図

$\beta_1=\{\alpha, \delta_4\beta_4\delta_5\}$
$\beta_2=\{\beta_1, \beta_2\delta_1\beta_1\}$
$\beta_3=\{\beta_2, \beta_3\delta_2\beta_2\}$
$\beta_4=\{\beta_3, \beta_4\delta_3\beta_3\}$

とするとき上記の諸行列は**第4図**のようになる.

たとえば行列 DB^∞ から, $\delta_1, \delta_2, \delta_3, \delta_4$ 等の直接右側へ δ_4 が来ることができて, しかも δ_4 の方が優先順位が上であることがわかるが, これは ↑(, ×(, +(などの記号列がありえて, しかも (の方を先に処理すべきことを意味している.

(c) 次のような手順で行列 F を作る.

[1]: $\rho_i\epsilon V_a \cup D$ のとき.

$F_{\rho_i\rho_i}=1,\ F_{\rho_i\rho_j}=0\ (i\neq j)$

[2]: $\rho_i\epsilon S$ のとき. ρ_i に属する一つの syntactic unit を $\varphi_{ij}=\rho_{\nu_1}\rho_{\nu_2}\cdots\cdots\rho_{\nu_n}$ とする.

$n=1$ のとき. $F_{\rho_i\rho_{\nu_1}}=1$ ([6] へ行く)

$n\neq 1$ のとき. ρ_{ν_k} を φ_{ij} の最も左にある D の要素とする. $l=1$ として [5] へ行く.

[3]: $\rho_{\nu_l}\epsilon S$ のとき. ρ_{ν_l} に属する(D の要素を含まない) V_a 上の string が生成できれば $F_{\rho_i\rho_{\nu_l}}=1$, ([4]へ行く), 生成できなければ $F_{\rho_i\rho_{\nu_l}}=1$ とし l に関する繰返しは終了する. ([6] へ行く)

$\rho_{\nu_l}\epsilon V_a$ のとき $F_{\rho_i\rho_{\nu_l}}=1$ ([4] へ行く)

[4]: $l+1\rightarrow l$ として $l<k$ であれば [3] へもどる.

[5]: $l=k$ であれば $F_{\rho_i\rho_{\nu_l}}=1$ として l に関する繰り返しは終了する. ([6] へ行く)

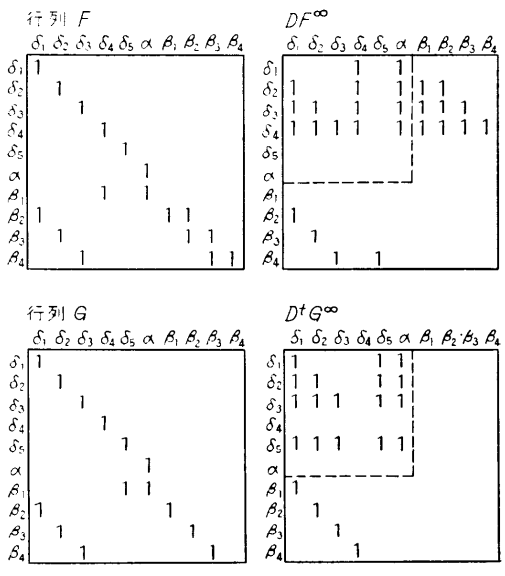
[6]: すべての syntactic unit について同様のことを行なう. ([2] へ行く)

この行列 F を用いて前と同様に DF^∞ を作る. しかるとき

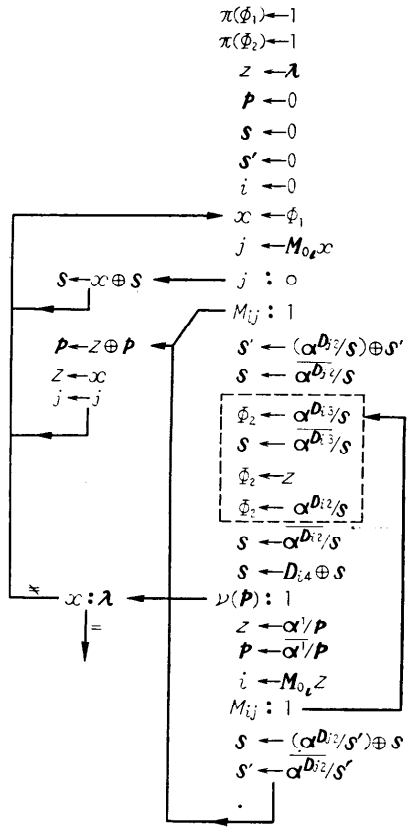
$F_{\rho_i\rho_j}=1,\ \rho_i,\ \rho_j\epsilon D$

は, 行列 F の性質と DF^∞ の作り方とからわかるように ρ_j が ρ_i の右側にくることのできる characteristic symbol で, その順番のとき $\rho_i<\rho_j$ (つまり '$\rho_i<\rho_j$') なる優先順位をもつ.

F を作った手順において, 最初に $l=n$ とし, $l-1\rightarrow l$ としながら F と同様な性質をもつ行列 G を作る. そして DG^∞ を作ると, この行列は D の要素間



第 5 図



第 6 図

M: 行列 DF^∞ 中 D に関する行, 列をとりだして作った小行列. それに行 λ をつけ加える. λ の順位は最低とする.

その他の記号は第1図のと同じ.

で右から左を見たときの優先順位を表わす.

例 5. 例4の文法から F, DF^{∞}, G, DG^{∞} を作ると**第5図**のようになる.

これらの行列, DF^{∞} または DG^{∞} を用いることにより, 文法Gから生成された $V_a \cup D$ 上の string をDの要素のみに着目して構造分析することが可能となる. そのフローチャートを**第6図**に示す. この場合も, 一つの syntactic unit 中に $\varphi=|\delta_1|\delta_2|$ のように characteristic symbol が二つ以上現われる場合には, semantic にその順序をきめる必要があり, それらの関係を DF^{∞} につけ加えねばならない.

あとがき

この論文では, 主として各 syntactic unit に固有の characteristic symbol が存在するという特殊な場合の context-free phrase structure grammar の構造分析についてのべたが, **3.2** で導入した A^{∞}, DB^{∞}, D^tC^{∞}, E 等の行列のもつ性質は characteristic symbol によらないから, phrase structure grammar 一般について成立するものである. よって, これらの行列は与えられた string の non-well-formedness を test するための一つの手段を与えるものである.

syntax と semantics との関係については種々の議論があって, たとえば semantics として syntax とは全く独立な解釈をとる場合も考えられる. ALGOL では a+b−c は syntax としては (a+b)−c を意味するのであるが, 数値計算上右の方から順次左の方へ計算を進めた方が誤差等のことを考えてよいという場合に, semantic として right to left principle をとるよう規定することもありうる.

しかし syntax と semantics とはなるべく密接な関係にある方が自然である. semantics にはしばしば不明確さを伴なうに対し, syntax は一応明確な規定であることから, 今までは semantics で定義されていたというものが, syntax として表現できるということになれば, それは一つの進歩であるとみることができるであろう. このように syntax の領域と semantics の領域との関係といった点にも今後の研究の方向があると思う.

参考文献

1) Report on the Algorithmic Language ALGOL 60, Comm. ACM, Vol. 3, No. 5, May 1960, pp. 299～314.
2) Revised Report on the Algorithmic Language ALGOL 60, Comm. ACM, Vol. 6, No. 1, Jan. 1963, pp. 1～17.
3) Backus, J.W.: ACM-GAMM Conference, ICIP, June 1959.
4) Brown, P.J.: Note on the Proof of the non-existence of a Phrase Structure Grammar for ALGOL 60, Comm. ACM, Vol. 6, No. 3, March 1963, p. 105.
5) Chomsky, N.: On certain formal properties of grammars, Information and Control, Vol. 2, No. 2 (1959), pp. 137～167.
6) Chomsky, N. and Schützenberger, M.P.: The Algebraic Theory of Context-free Languages, Computer Programming and Formal Systems, North Holland Publ. Co., Amsterdam, 1963, pp. 118～161.
7) Samelson, K. and Bauer, F.L.: Sequential formula translation, Comm. ACM. Vol. 3, No. 2, 1960, pp. 76～83.
8) Gorn, S.: Detection of Generative Ambiguities in Context-Free Mechanical Languages, J. ACM. Vol. 10. pp. 196～208.
9) Iverson, K.E.: A Programming Language, John Wiley & Sons. 1962.
10) Warshall, S.: A Theorem on Boolean Matrices. J. ACM, Vol. 9, 1962, pp. 11～12.

(昭和38年9月1日受付)