プログラミング言語 14－1
（１９８７．１２．１８）

動的な制御集合をもつ文法について

山田 攻　　石田純一　　寄松史士　　野口正一
（室 蘭 工 業 大 学）　　（東北大学 通研）

あらまし： プログラミング言語では一般に何らかの宣言機能あるいは定義機能をもっており、これらの文の内容に依存する制約、例えば変数の宣言の有無といったものは一種の文脈依存である。しかし、無限個の導出が可能な中から選択されたある部分導出の結果が他の部分の導出を制約するものであるから、従来の句構造文法でこれを表現することはできない。これに対応する１っの方法として、ある部分の導出が行われると制御集合が動的に変化して、残りの部分の導出を制御するような形の文法を提案する。制御集合が動的に変化すると、一般には生成規則の適用順序によって生成されるものが異なるというあいまいさをもつ。これを避けるとともに、言語機能のモジュール化をねらいとして、文法が複数の部分文法によって構成され、言語の生成は部分文法毎に段階的に行われるような多段文法の考え方を示す。これにより、これまで意味論として属性文法などを用いて別に表わされていたかなりの部分が構文の問題として統一的に扱えるようになる。また、この動的な制御集合をもつ多段文法を用いたＡＬＧＯＬ系言語の記述や構文解析についても触れる。

ON THE GRAMMAR WITH DYNAMIC CONTROL SETS

Osamu YAMADA*,Junichi ISHIDA*,Fumio YORIMATU* and Shoichi NOGUCHI**

* Muroran Institute of Technology　27-1,Mizumoto-cho,Muroran-shi,Hokkaido,Japan
**Research Institute of Electrical Communication,Tohoku University
2,katahira-cho,Sendai-shi,Miyagi-ken,Japan

This paper presents a grammar with control sets such as elements are changed dynamically. The grammar consist of sub-grammars and generates the language in stages every time each sub-grammar is used. The derivation is restricted by the control sets which are varied in the upper stages. It can be represented syntactically most restrictions based on declarations in the program. A method of syntactic analysis for the language generated by this grammar is also discussed.

1、はじめに

制御集合（control set）の概念はGinzberg[1]らによって提案され、生成規則の形を制限するのではなく、生成規則の適用を制限する方向の研究の重要性が指摘された。その後、制御集合をもつ文法によって生成される言語の性質、とくにあるクラスの言語で導出を制御して他のクラスの言語導出を行うことの可能性についての研究が行われてきた[2]。

本論文では、主として実際のプログラミング言語を記述するという観点から、この制御集合の概念を用いて従来の句構造文法[3]の形では表現できず、意味論として別に扱われてきたかなりの範囲をとり込むことができるような新しい形の文法を提案する。

一般にプログラミング言語では、その能力の大小はあっても、何らかの宣言機能や定義機能を持っている。しかし、これにかかわる制約、例えば変数名の宣言の有無といったものは、ほとんどの場合、事前に与える有限個の生成規則だけで表す文脈依存文法で表現することは不可能である。なぜならば、この文脈依存は宣言や定義部分の導出を行った結果として生ずるものであり、無限個の部分導出が可能な中から選択されたある記号列が他の部分導出に与える制約であるからである。

このことから、本論ではある部分の導出が行われた結果として、制御集合が動的に変化し、残りの部分の導出を制御するような文法を提案する。この動的な制御を記述する部分とその制御対象となる部分の関係を明確にする方法はいくつか考えられるが、ここでは主として文法が複数の部分文法からなり、言語の生成は部分文法毎に段階的におこなわれるような多段表現の文法について述べる。

2、諸定義

まず基礎的な定義を以下に示すが、基本的には文献4）の定義に準拠している。

［定義１］

$G=(V_N,V_T,P,S)$を文脈自由文法（以下単に文法と書く）とする。ここにV_N, V_T, P はそれぞれ、非終端記号、終端記号、生成規則の有限集合であり、$S \in V_N$は初期記号である。$V=V_N \cup V_T$とし、とくに断わりのないものについては、V_N, V_T, V^*の要素を次の記号で代表させる。

$V_N=A,B,\cdots$　$V_T=a,b,\cdots$

$V^*=u,v,\cdots z$

生成規則を$A\rightarrow\omega$で表し、Aを生成規則の左辺、ωを右辺と呼ぶ。$x=uAv$、$y=u\omega v$ なるとき、$A\rightarrow\omega \in P$が存在するならば、$x \Rightarrow y$と書く。またあるω_1、ω_2、・・ω_n（$k \geqq 0$）が存在して$x \Rightarrow \omega_1$、$\omega_1 \Rightarrow \omega_2$、・・・$\omega_n \Rightarrow y$なるとき、$x \overset{*}{\Rightarrow} y$と書いて導出と呼ぶ。導出がとくに文法Gの生成規則によることを明示するときは$x \overset{*}{\underset{G}{\Rightarrow}} y$と書く。

［定義２］

文法$G=(V_N,V_T,P,S)$が生成する言語を$L(G)$で表す。

$$L(G)=\{x \in V_T^* \mid S \overset{*}{\Rightarrow} x\}$$

$\lambda \in L(G)$を言語$L(G)$の文という。また一般にある記号の集合$V_A \subseteq V$が与えられ、$A \overset{*}{\Rightarrow} x \in V_A^*$かつ任意の$y \in V_A^*$（$x \neq y$）に対して$x \overset{*}{\Rightarrow} y$なる導出は存在しないとき、$y$を$V_A^*$で終端したAからの導出結果という。言語$L(G)$の文は$V_T^*$で終端したSの導出結果である。

［定義３］

文法$G=(V_N,V_T,P,S)$において、$D(A)=\{\omega \in V^* \mid A \Rightarrow \omega\}$とし、任意の$C(A) \subseteq D(A)$が与えられたとき$L_C(G)$を次のように定義する。

$L_C(G)=\{\lambda \in V_T^* \mid S \overset{*}{\Rightarrow} uAv \overset{*}{\Rightarrow} u\omega v \overset{*}{\Rightarrow} \lambda$ なる任意の$A \in V_L$、$u,v,\omega \in V^*$に対して$\omega \in C(A)\}$

この$L_C(G)$を制御集合Cをもつ文法によって生成される言語という。

3、動的な制御集合をもつ多段文法の定義

［定義４］

文法$G=(V_N,V_T,P,S)$において、任意の$A \in V_N$について、その制御集合$C(A)$と記号の集合$V_A \in V$が与えられたとき、Gによる言語生成の過程で、V_A^*で終端した導出$A \overset{*}{\Rightarrow} x$が行われると、特定の制御集合$C(B)$が

$C(B)-\{y\}$ または $C(B) \cup \{y\}$, $y \in D(B)$

なる形で変化するとき文法Gは動的制御集合をもつという。yはxの一部または全部をその記号列中に含んでよい。

動的な制御はある部分導出が行われた結果として、すでに導出を終了した確定部分が残りの部分の導出を

制御することを意味する。しかし動的な制御集合をもつ文法は生成規則の適用順序によって導出可能な文が異なるというあいまいさを持つ。これを避けるためには、導出の順序に関して何らかの制限を設ける必要があり、その制限方法によっていくつかのサブクラスが作られるが、ここでは次の３つの形を提案する。

（１）言語の生成を最左導出に限定する

（２）生成規則を分割して段階的な導出を行い、段の上下関係に基づく制限を設ける

（３）上記（２）の各段階に（１）を適用する

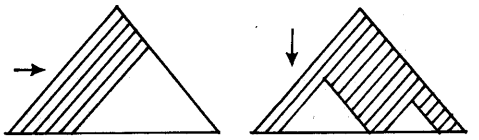
最左導出に限定する方法は、その形が単純かつ明確であり、構文解析もまた左から右へ行えば、制御集合が動的に変化することによる複雑さを回避できる。しかしながら動的な制御は自分自身の右側にしか行えず、対象とする言語の仕様に大きな制約を与えることになる。

第二の方法は生成規則が制御を記述する部分と制御対象の部分に分けられることを前提にしている。プログラミング言語で言えば「宣言部」と「実行部」といった形で生成規則がいくつかのグループを形成していることを意味する。

この２つの方法の概念を、導出木を用いて図式的に表せば図１のようになる。図１の導出木で斜線部分が導出を終了した確定部分を表し、この結果によって導出が制限される部分を斜線なしで表している。（１）が左から右への制御に対し、（２）は上から下への制御に相当する。

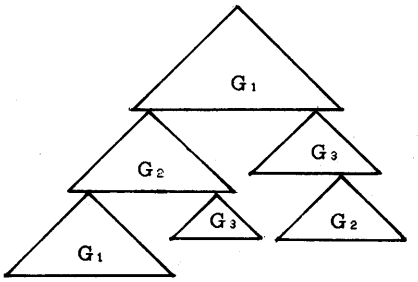
三番目の方法は段階的な導出で上から下への制御を行い、更に各段内では左から右への制御を認めるものであるが、（１）、（２）の組み合せであって、本質的な相違はない。

この（２）の概念を明確にするため、文法が複数の部分文法からなり、その言語導出が部分文法毎に段階的に行われるような多段形式の文法を定義する。この考え方は図２の（イ）のように、部分文法を１つの単位として順番に導出が行われ、全体としては（ロ）のように各部分文法が導出した範囲が１つの部分木をなすものである。以下、本論文ではこの第二の方法を中心に述べる。

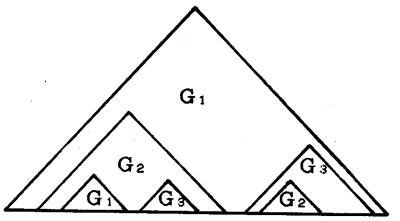


（イ）　　（ロ）

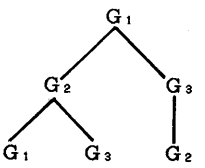
図１　動的制御の方向



（イ）



（ロ）



（ハ）

図２　多段文法の概念

［定義５］

n個の文法$G_i=(V_{Ni},V_{Ti},P_i,S_i)$、$i=1,2,\cdots,n$ があって、

$H=\{G_1,G_2,\cdots,G_n\}$ 、$S_i\neq S_j$ $(i\neq j)$

とするとき、$M=(H,G_S),G_S\in H$ が生成する言語 $L(M)$を次のように定義する。

$V_S=\{S_1,S_2,\cdots,S_n\}$, $V_M=V_{T1}\cup V_{T2}\cup\cdots\cup V_{Tn}$とするとき

$L(M)=\{\lambda\in(V_M-V_S)^* \mid$ あるω$_1$,ω$_2\cdots$,ω$_m\in V_M^*$, ω$_i=uS_kv$, ω$_{i+1}=uxv$, $0<i<m$ があって $S_S\Rightarrow\omega_1\Rightarrow\omega_2\Rightarrow\cdots\Rightarrow\omega_k=\lambda$ かつ $S_k\underset{G_k}{\Rightarrow}x$ $\}$

このMを多段文法、各G_iをMの部分文法、G_SをMの初期文法と言う。

上記の各部分文法G_i($i=1,2,\cdots,n$)においてその初期記号S_iを除いては非終端記号V_{Ni}が互に重複しないように記号をつけかえ

$V_N=\cup V_{Ni}$, $V_T=\cup V_{Ti}$, $P=\cup P_i$、

$G=(V_N,V_T,P,S)$

とおけば多段文法Mは文法Gの生成規則の集合を単純に分割だけであるから、明らかに$L(G)=L(M)$である。ただし、多段文法では部分文法を１つの単位として導出が行われているという導出の順序に関する制約がある。導出された個々の文について言えば、導出に使用された部分文法がその適用順序により１つの木構造をなす。図２の（イ）あるいは（ロ）の場合には（ハ）のような部分文法の木構造ができる。以後この木構造での上下関係をもって「上位の導出」「下位の導出」と表現する。

［定義６］

多段文法Mを構成する部分文法はそれぞれ与えられた制御集合をもつとする。この多段文法Mによる言語の生成において、部分文法の動的な制御記述は自分の下位の導出に対して有効となり、自分自身に対しては同一の記号列の再導出の禁止（制御集合からの削除）のみが許されるとき多段文法Mは動的な制御集合をもつという。

多段文法において制御集合が動的に変化していく様子を図３に示した。ただし、使用している記号の意味は次の通りである。

$M=(\{G_1,G_2,G_3,\},G_1)$

C_i：部分文法G_i($i=1,2,3$）に与えられた制御集合

C_i', C_i'', C_i'''：制御集合が変化したことを$'$をつけて示す。

（ ただし、G_iによる導出は、全ての部分文法の制御集合を変化させ、G_2による導出はG_3の制御集合だけを変化させると仮定している。また図中の１つの三角形はその中に書かれている部分文法G_iと制御集合C_iとによって導出したことをを表す。）

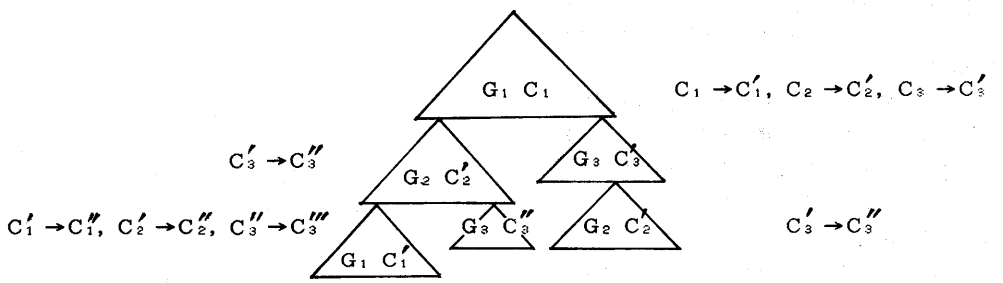


図３ 多段文法での制御集合の変化

4、制御集合の記述

（1）初期集合の記述

G＝（V_N,V_T,P,S）において、その制御集合C(A)はすべてのA∈V_Nに対して定義されるものであり、C(A)⊆D(A)を満足する任意の集合であるから、一般にはその記述は容易ではない。しかしプログラミング言語では、宣言文あるいは定義文によって動的に変化するのは、予め構文表示された最大の範囲から特定のものが禁止されるか、あるいは逆に最大の範囲は定まっているが最初は許されるものがなく、特定のものが許可されていくかのいずれかである。したがって、プログラミング言語の場合の初期集合としては次のいずれかの表現で充分と考えられる。

C(A)＝D(A)　または　C(A)＝∅　(∅：空)

例えば2重宣言の禁止は前者の場合であり、宣言された変数の使用許可は後者の場合に該当する。

（2）動的な記述

動的な制御の記述は一般にV_A⊆Vが与えられ、A≒x∈V_A*なる導出が行われたとき、ある制御集合C(B)にB≒yなる導出を認めるか否かのいずれかである。したがって、許可か禁止かを表す記号をεとすれば、動的な記述は一般に、文法Gの生成規則でその左辺がAであるものに［x，ε，B≒y］なる情報を付加することで表現できる。

例）G_i：A→u　［x，ε，B≒y］

このとき、xはAから導出された結果の記号列であり、それをyの一部として含むことができるので、「V_A*で終端したAからの導出結果」を意味する記号が必要となる。いま、これを記号に下線を施すことであらわせばV_A*に属するAからの導出結果を$\underline{A}$と表せる。

多段文法の場合はこれに制御対象の部分文法の指定が入り次のようになる。

G_i：A→u　［x，G_j，ε，B≒y］

この動的な記述をALGOL60およびPASCALのサブセットに対して行った試みでは、Aが導出する記号列のうちのある特定の場合についてだけ動的な制御を必要とする場合は生じていないので、多くの場合は［ε，B≒y］という形でよいと思われる。

（3）記述例

「宣言された変数のみ使用可」を記述例として次のページの図4に示した。図5はこの文法による1つの導出例を表している。例はPASCAL[5]からとったものであるが、構文表示は非常に簡略化してある。

5、ALGOL系言語への適用

これまで述べた動的な制御集合をもつ多段形式の文法表現を用い、コンパイラ生成システムの作成を目的にALGOL60[6]およびPASCALを記述する試みを行った。まだ記述仕様は仮のもので、手書きによる確認段階ではあるが、この結果からは次のように言える。

ALGOL60については、いわゆるコンパイラーにおける「文法上の誤り」あるいは「翻訳時の誤り」と言われるものは、ほとんど表現可能になる。具体的には次のような事項である。

イ、予約語
　初期の文法の最初の導出で宣言文における＜変数名＞から予約語と同じ綴りの導出を禁止する

ロ、変数の多重定義の禁止
　自分自身と同一の記号列の再導出を禁止する

ハ、宣言の有無
　実行文関係を1つの部分文法とし、その中の＜変数名＞の制御集合の初期状態は空にしておき、宣言が行われる度にその要素として追加する

ニ、型の対応
　上記ハを型毎に記述する

ホ、名札と飛び先
　＜名札＞は＜実行文＞から分離し、上位の部分文法で導出する

ヘ、ブロック構造に付随する制約
　ブロックを1つの部分文法にすれば上位（外側）のブロックでの名札や変数名の有効範囲が下位（内側）を包括する形となる

一方、記述が不可能な、あるいは困難な問題も存在している。とくにPASCALでは基本的なところに大きな問題を抱えているが、それらの問題を列挙すれば次のようになる

イ、利用者が定義した型に属する変数の区別
　予め与えられていない型を定義した場合、該当する制御集合がなくそれに属する要素を他のものと区別できない。

ロ、数値の比較を必要とする記述
　定数の大きさの制限あるいはPASCALにおける＜名札＞の一致（整数表現の先行する0の扱い）等は表現できない（大きさに制限がある

$M=(\{G_1, G_2\}, G_1)$

$G_1=(V_{N1}, V_{T1}, P_1, <プログラム>)$、$G_2=(V_{N2}, V_{T2}, P_2, <実行部>)$

P_1：<プログラム>→ program；<宣言部><実行部>.

……

<型宣言>→ var <実数宣言>：real；

→ var <整数宣言>：integer；

<実数宣言>→ <実変数>

→ <実数宣言>，<実変数>

<整数宣言>→ <整変数>

→ <整数宣言>，<整変数>

<実変数>→ <名> [許可、<実変数>≒<名>]

<整変数>→ <名> [許可、<整変数>≒<名>]

<名> → <英字>

→ <名><英字>

……

P_2： <実行部>→ <文の並び>

……

<変数> → <実変数>

→ <整変数>

<実変数>→ <名>

<整変数>→ <名>

<名> → <英字>

→ <名><英字>

……

制御集合（初期状態）：G_2のC(<実変数>)=∅、C(<整変数>)=∅、それ以外は

任意のA∈(V_{N1}∪V_{N2}) に対して C(A)=D(A)

図４ 動的な制御集合をもつ多段文法の例

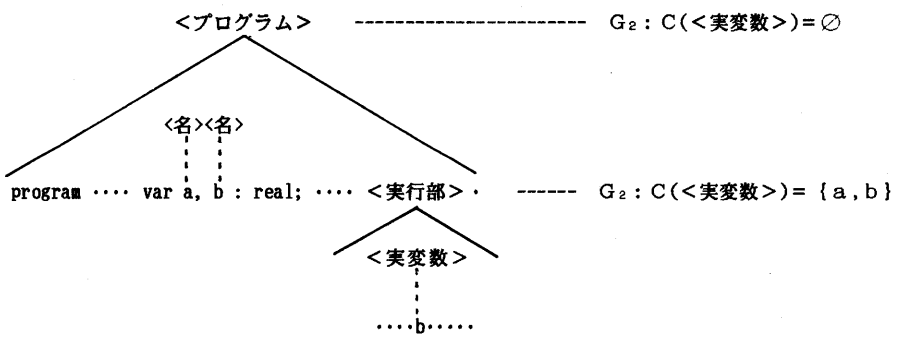


図５ 動的な制御集合をもつ多段文法による導出例

ので理論上は表現できるが現実的でない）

ハ、例外的な取扱事項の表現

例えば、ＡＬＧＯＬ６０でｉｆ文だけはその文の中へｇｏｔｏ文で飛び込めるといった事項

このうちイに関しては上位の部分文法が下位の部分文法の制御集合自体を定義する機能、言い換えれば下位の部分文法を変形する能力を必要としており、現在の文法モデルをレベル１とすればレベル２、レベル３という形で上位のクラスを設定する必要がある。しかし、現在のレベルでもコンパイラーにおける構文解析とコード生成の間で行われている意味解析の基本部分は構文上の問題として扱える。

このほか、実際の記述を行うためにはこのような文法自体の問題とは別に、制御集合の動的な変更を記述する超言語の仕様の確定とその検証を行う必要がある。

６、構文解析法

制御集合の初期状態は４節（１）で述べたC(A)=D(A)またはC(A)=∅のいづれかであるとすれば、制御集合自体は、その動的な部分も含め非終端記号Aとそれから導出される記号列の表である。しかし、これが動的に変化するため、構文解析では解析の各時点での制御集合の内容をどのようにして確定するかが問題となる。

３節で述べた生成規則の適用順序に関する制限方法のうち、最左導出による場合は構文解析も左から右へ行えば、常に導出時と同一の制御集合が得られるから特別な手法は必要としない。多段文法表現による場合は各部分文法の導出範囲の識別を必要とするので、以下にその解析方法を述べる。

（１）汎用的な方法

制御集合を考慮しなければ、多段文法自体は通常の文脈自由文法を単純に分割して得られるものであるから、３節で示したように、生成される言語も導出木も等価であるような１個の文脈自由文法Ｇを容易に作ることができる。したがって解析を２段階に分け、最初は文法Ｇに基づき、従来から提案されている各種の解析方法を用いて導出木を作る。この導出木を見れば、部分導出木の頂点の記号から、対応する部分文法とその導出範囲は明確となる。次にこの導出木の上位部分から順に再び調べれば、最上位の導出を行ったときの制御集合は与えられた初期集合であり、以後は上位部分の解析を終了した時点でその直後の導出を行う部分文法の制御集合は確定しているから、動的な制御を含めた完全な解析ができる。

（２）境界を定める規則による方法

前項の方法は動的な制御集合をもつ任意の多段文法に適用できる広さをもつが、導出木全体を保存し、あとで再び調べ直す必要がある。このため効率や記憶容量の面に難がある。しかし、もし導出木を作ることなく各部分文法が導出する範囲を識別する簡単な手法があれば、これらの問題を取り除くことができる。

一般にプログラミング言語では、その構成要素である「部」（宣言部とか実行部といったもの）や「文」（型宣言文とかｉｆ文といったもの）を１っの単位として見れば、それぞれは予約語や区切り記号等によって、左右の境界やその種別（部分木の頂点の記号）が特徴づけられている。各部分文法にこのようなキーとなる記号が存在するならば、解析対象の文を見るだけで図６のように、適用された部分文法の種類とその導出範囲を示す外形のみの導出木が得られる。したがって、この解析のあと上位の部分から順に導出木を作っていけば、制御集合が動的であっても、導出木作成の時点で完全な解析を行うことができる。

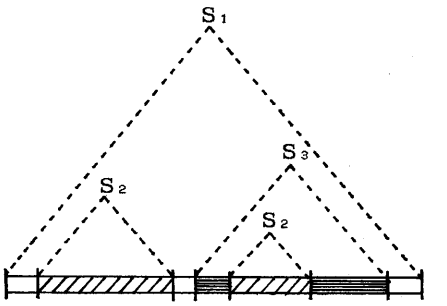


図６境界規則による１次解析

この方法は動的な制御集合をもたない通常の文脈自由文法においても、その生成規則を適当な大きさに分割して部分文法を作り、その中にキー記号を含むようにできるならば一般的な構文解析法[7]として有効なものである。この方法は、２－パス構文解析の１っのモデルとして考えることもでき、１－パス目に各部分文法が導出する範囲を識別することによって、

イ．部分毎の最も適切な構文解析法の選択

ロ．翻訳の平行処理

ハ．エラー処理の適切化

等を可能にする特徴を持っている。また、このような部分文法化は言語機能をモジュール化して表すことになるので、機能の追加、修正を容易に行なうことができる。したがって、コンパイラ生成システムのように、不特定の言語を対象にしたり、言語仕様の変更に即応する能力が要求される場合に適していると言える。

7、おわりに

コンパイラ生成システムを作成する際、宣言文や定義文の内容に依存する事項は属性文法[8]などを用い、意味解析として独立に扱われることが多い[9]。本論ではこれを動的な制御集合および部分文法による多段形式の表現により、構文の問題として統一的に扱う文法モデルについて述べた。我々は現在、教育利用を目的にコンパイラ生成システムを作成する試みを開始したが、ここで述べた文法モデルと境界記号に基づく２－パス構文解析法により、ある程度実用に耐える表現の容易さと実行効率をもつシステムができることを期待している。この構文解析法については、プログラミング言語を多段文法で実際に記述した結果と合せ、別に報告したいと考えている。

文献

1）Ginzberg,S.& Spanier,E.H. "Control sets on grammars" Math.system Theory,2, p159(1968)

2）例えば Kasai.T "A universal context-free Grammar " Inf.& Control,28,p30 (1975)

3）Chomsky,N. "Three models for the description of languages" IRE Trans. Information Theory IT2,p113 (1956)

4）Hopcroft.J.E.& Ullman.J.D, "Formal languages and their relation to automata" Addision-Wesley

5）ISO国際規格 ISO7185-1983 （石畑 清、筧俊彦 ,安村通晃 訳「ＰＡＳＣＡＬの標準化 － ＩＳＯの規格全訳とその解説」共立出版(1984)

6）日本工業規格 JIS C 6210-1972 電子計算機プログラム用言語ＡＬＧＯＬ（水準７０００）

7）山田攻、石田純一、野口正一 「キー記号の概念を導入した構文解析法について」 電気４学会北海道支部大会 p216 (1984)

8）Knuth,D,E "Semantics of context-free languages" Math.system Theory 2,p130

9）例えば 佐々政孝 「コンパイラ生成系」 情報処理 Vol.23,No9, p802