動的な制御集合をもつ文法について†

山田 攻†† 石田純一†† 野口正一†††

文脈自由文法による言語導出において，ある部分の導出が行われると制御集合の要素が変化し，残りの部分の導出を制御するような動的な制御集合をもつ文法を提案する．また，その制御対象を明確にするとともに，表現能力を高めるため，文法が複数の部分文法によって構成され，言語の導出は部分文法ごとに段階的に行われるような多段形式の文法について述べる．この文法はプログラミング言語における文脈依存，例えば宣言やブロック構造に関わる各種の制約を文脈自由文法の形で簡明に表現することを可能にし，これまで意味解析として別に扱われていた部分を構文解析の問題として統合することができる．これらの具体例を ALGOL 60 のサブセットを用いて示す．また，この文法によって生成される言語の構文解析の考え方についても述べる．

1. はじめに

制御集合 (control set) の概念は Ginzberg らによって提案され[1]，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]．

一方，コンパイラ生成系では，プログラミング言語における宣言文やブロック構造に関わる制約は属性文法[3]を用いて表し，その処理は意味解析として独立に扱われることが多い[4]．属性文法は合成属性と相続属性の双方向の属性により，高い記述能力をもち，自動生成の範囲を広げることに大きく貢献したが，意味規則の表現や解析が容易であるとは言いがたい．プログラミング言語における宣言とその利用との関係は，見方を変えれば，一種の文脈依存と考えられる．したがって，この文脈依存を言語導出時の構文上の問題として，文脈自由文法の形式で，より簡明にすることができれば，プログラミング言語の定義に有力な手段を与えるとともに，意味解析を構文解析に統合して扱うことが可能になる．

本論文では，主としてプログラミング言語の記述，あるいはその解析という立場から，制御集合の概念を用いて，ある種の文脈依存を文脈自由文法の形をとって表す文法モデルについて述べる．

具体的には，ある部分の導出が行われると，与えられた制御集合の要素が動的に変化し，残りの部分の導出を制御するような文法を提案する．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可能にするため，文法が複数の部分文法からなり，言語導出は部分文法ごとに段階的に行われるような多段形式の文法について検討する．そして，実際にこの文法モデルを用い，意味解析で扱われる各種の制約が構文表示で簡明に記述可能であり，その解析もまた，構文解析の一部として容易に実現できることを示す．

本文では，まず2章でこの文法の考え方を述べ，3章でその厳密な定義を与える．そして，4章ではプログラミング言語に対する記述を具体例を用いて説明し，5章ではその解析の考え方について述べる．

2. 動的な制御集合に基づく言語導出の概念

宣言文と実行文の関係を次のような簡単な文法で考えてみる．非終端記号は<α>の形で表している．

[例1]

<プログラム>→begin<宣言><代入文>end
<宣言> →real<名前の並び>;
<名前の並び>→<名前>
→<名前の並び>, <名前>
<名前> →A
→<名前>A
<代入文> →<変数>:=<変数>
<変数> →<名前>

この構文表示には現れないが，一般にプログラミング言語では宣言した名前の使用だけが許される．このような制限を言語導出時の制約として表すため，次の

† A Grammar with Dynamic Control Sets by OSAMU YAMADA, JUNICHI ISHIDA (Muroran Institute of Technology) and SHOICHI NOGUCHI (Research Institute of Electrical Communication, Tohoku University).

†† 室蘭工業大学

††† 東北大学電気通信研究所

形を考える．

(1) すべての非終端記号は与えられた制御集合をもち，その要素である終端記号列の導出だけを許す．

(2) 制御集合の要素は言語導出が行われる過程で動的に変化し，それ以後の導出を制御する．

これを例1の文法に適用して，名前の宣言とその利用との関係を表現すれば次のようになる．各非終端記号に与えられる制御集合のうち，<変数>の制御集合だけが空，他の制御集合は構文表示で導出が許されるすべての終端記号列を含むようにする．そして，宣言文で<名前>から終端記号列を導出するたびに，その記号列を<変数>の制御集合に加えれば，代入文では宣言された名前の導出だけが許されることになる．

いま，制御集合の要素を動的に変更するための記述を【と】でくくって表せば，例1の場合は，その3行目と4行目の<名前の並び>の生成規則を次のように書き換えることで表現できる．ただし，記号 α の制御集合は $C(\alpha)$ で，記号 β が導出した終端記号列は下線を引いて $\underline{\beta}$ で表している．また，制御集合 $C(\alpha)$ に x なる要素を追加する，取り去るという処理を，それぞれ $C(\alpha)\cup\{x\}$, $C(\alpha)-\{x\}$ で表すことにする．

<名前の並び>→<名前>
【C(<変数>)$\cup\{\underline{<名前>}\}$】
→<名前の並び>, <名前>
【C(<変数>)$\cup\{\underline{<名前>}\}$】

一般に制御集合は，その文法が導出する言語のサブセットとして，言い換えれば，初期記号に対して取られるが[1),2)]，ここではすべての非終端記号に対して定義し，その要素の動的な変更も認めている．しかし，制御集合の要素が動的に変化すると，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生じるため，導出方法に何らかの制限を設ける必要がある．その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2つの形を検討する．

(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グループごとに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．

最初の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特徴をもつ．その反面，動的な制御は自分自身の右側にしか行えない欠点をもっている．

第二の方法は，文法を複数の部分文法で構成し，言語の導出は部分文法ごとに段階的に行う方法である．1つの部分文法の導出が終了したとき，その導出文にいずれかの部分文法の初期記号を含めば，さらにその部分文法によって導出を続けていく．このときの上位の導出が下位の導出を制御する形である．

この両者を組み合わせ，上記(2)の各段階で(1)を適用すれば，さらに表現能力を高めることができる．

例1の場合は最左導出という条件だけで十分であるが，(2)の方法で表すと例2のようになる．文法を2つに分け，部分文法1による導出がそのあとの部分文法2による導出を制御している．ここで≪α≫はその部分文法では終端記号であるが，他の部分文法の初期記号であることを示す記号として用いている．

[例2]

(部分文法1：プログラム)

<プログラム>→begin<宣言>≪代入文≫end
<宣言> →real<名前の並び>;
<名前の並び>→<名前>
【C(<変数>)$\cup\{\underline{<名前>}\}$】
→<名前の並び>, <名前>
【C(<変数>)$\cup\{\underline{<名前>}\}$】
<名前> →A
→<名前>A

(部分文法2：代入文)

<代入文> →<変数>:=<変数>
<変数> →<名前>
<名前> →A
→<名前>A

この文法による実際の導出を導出木の形で示せば**図1**のようになる．制御集合 C(<変数>) の初期状態は空であるが，部分文法1で宣言文を導出したことによって，その要素に A と AA が加えられ，その後を導出する部分文法2では変数としてこの2つの名前が使える．

一般には，ある部分文法による導出が他の複数の部分文法の初期記号を含んでもよいし，自分自身を再帰的に使用してもよい．この多段形式の文法は，その導出の形態から，後で述べるように ALGOL 系言語に

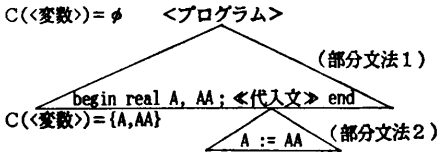


図1 例2の文法による導出例
Fig. 1 A sentence produced by the grammar in Example 2.

おけるブロック構造によく適合する.

3. 形式的な定義

3.1 基本的な定義と制御集合

まず基礎的な定義を以下に示すが，基本的には文献5)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（以後単に文法と書く）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，$V=V_N \cup V_T$ とし，$S\in V_N$ は初期記号である．$x=uAv$, $y=uwv$, $A\in V_N$, $u, v, w\in V^*$ なるとき，生成規則 $A\to w\in P$ が存在するならば，$x\Rightarrow y$ と書く．また，ある $z_1, z_2, \cdots, z_n\in V^*$ $(n\geqq 0)$ が存在して $x\Rightarrow z_1$, $z_1\Rightarrow z_2, \cdots, z_n\Rightarrow y$ 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 $x\underset{G}{\overset{*}{\Rightarrow}}y$ と書く．文法 G が生成する言語を $L(G)=\{\lambda\in V_T^* \mid S\underset{G}{\overset{*}{\Rightarrow}}\lambda\}$ で表す．（定義終）

以下，本論文では言語の生成に寄与することのない記号や規則はあらかじめ除去されているものとして述べる.

[定義2]

文法 $G=(V_N, V_T, P, S)$ において，任意の $A\in V_N$ に対し，$C(A)\subseteq D(A)=\{x\in V_T^* \mid A\overset{*}{\Rightarrow}x\}$ が与えられたとき，$L_c(G)$ を次のように定義する.

$L_c(G)=\{\lambda\in V_T^* \mid S\overset{*}{\Rightarrow}uAv\overset{*}{\Rightarrow}uwv=\lambda$ なる任意の $A\in V_N, u, v, w\in V_T^*$ に対して $w\in C(A)\}$

この $L_c(G)$ を制御集合 $C(A)$ をもつ文法によって生成される言語という．（定義終）

$D(A)$ は A から導出可能なすべての終端記号列の集合であるが，制御集合 $C(A)$ をそのサブセットにとることで A からの導出を制限している．定義2の $C(A)$ の要素は固定であるが，その動的な変化を認めた形が次の定義3である.

[定義3]

文法 $G=(V_N, V_T, P, S)$ において，各非終端記号には制御集合 $C(A)$, $A\in V_N$ が与えられたとする．この G による言語生成の過程で，ある部分導出 $B\overset{*}{\Rightarrow}x\in V_T^*$ が行われると，特定の制御集合 $C(A)$ の要素が

$C(A)\cup\{y\}$ または $C(A)-\{y\}$

なる形で変化するとき文法 G は動的な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．言語の導出が特に最左導出に限定されるとき，左から右への限定制御であると言う．（定義終）

3.2 多段文法

次に文法が複数の部分文法からなり，部分文法を1つの単位として段階的な言語導出を行う文法を定義する．最初の導出を行う特定の部分文法から導出を開始し，各部分文法の初期記号を終端記号と見なして導出を行う．そして，その導出が終了した時点で，導出文中にいずれかの部分文法の初期記号が含まれていれば，その部分文法によってさらに導出を続けていく.

この考え方を図式的に表せば**図2**のようになる．導出文中に部分文法の初期記号が現れると，それを頂点とする導出木が順次作られ，n 段の導出木を構成する．使用された部分文法に注目すれば，これが1つの木構造をなす．以後この木構造での上下関係をもって「上位の段の導出」「下位の段の導出」と表現する.

これをまとめると次のようになる.

[定義4]

n 個の文法 $G_i=(V_{Ni}, V_{Ti}, P_i, S_i)$, $(1\leqq i\leqq n)$ 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

$L(M)=\{\lambda\in(V_M-V_S)^* \mid$ 導出中に現れるすべての S_k $(0<k\leqq n)$ に対して，ある $z_1, z_2, \cdots, z_m\in V_M^*$, $z_i=uS_kv$, $z_{i+1}=uxv$, $S_k\underset{G_t}{\overset{*}{\Rightarrow}}x$ $(0<i<m)$ があって，$S_s\overset{*}{\Rightarrow}z_i\overset{*}{\Rightarrow}z_2\overset{*}{\Rightarrow}\cdots\overset{*}{\Rightarrow}z_k=\lambda\}$

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．（定義終）

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

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

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

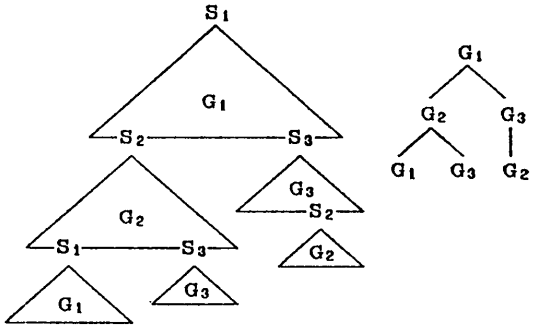


図2 多段文法の概念
Fig. 2 Conception of the multi-stage grammar.

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しただけであるから，明らかに $L(G)=L(M)$ である．ただし，多段文法では部分文法単位で導出が行われるという導出の順序に関する制約がある．

定義4では自分自身が再帰的に呼ばれても複数の段を構成するが，これを回避したいときは，その初期記号を S_i から $S_i' \notin (V_N \cup V_T)$ に変え，$S_i' \to S_i$ なる生成規則を P_i に追加することで表せる．

この多段文法で，自分の下位の段の導出に対しては制御集合の要素の変更を認めたものが定義5である．

[定義5]

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．その言語導出において，各部分文法の動的な制御記述はその段の導出開始時点の制御集合に対する変更であり，自分の下位の段の導出に対して有効であるとき，多段文法 M は動的な制御集合をもつと言う．また，各段内においても最左導出に限定された動的制御を認めるとき，各段では左から右への限定制御であると言う．　　(定義終)

多段文法において制御集合が変化していく様子を図3に示した．図中の記号の意味は次のとおりである．

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

C_i：部分文法 G_i $(1 \leqq i \leqq 3)$ に与えられた制御集合を表す．また，制御集合 C_i が変化したことを添字を追加して C_{ia}, C_{iab}, …と表している．

ただし，G_1 による導出は，すべ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，G_3 による導出では変化がないと仮定している．また，導出木（三角形）の内部の G_i と C_i はその部分文法と制御集合によって導出したことを表し，外側の C_i はそのとき変化した制御集合の状態を表している．

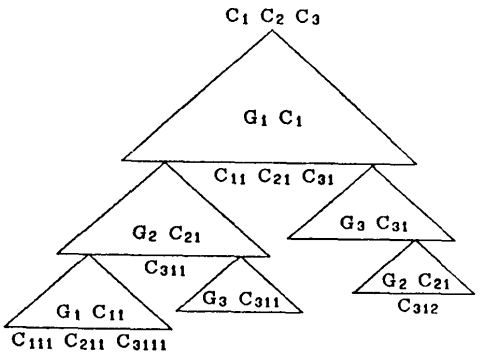


図3 多段文法での制御集合の変化例
Fig. 3 An example of changes of control sets in the multi-stage grammar.

4. プログラミング言語に対する記述

4.1 初期集合の記述

$G=(V_N, V_T, P, S)$ において，その制御集合はすべての $A \in V_N$ に対して定義されるものであり，$C(A) \subseteq D(A)$ 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ない．しかしプログラミング言語での宣言文や定義文による変化は，あらかじ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．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．

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十分と考えられる．

$C(A)=D(A)$ または $C(A)=\phi$　　(ϕ：空)

4.2 動的な制御の記述

言語導出に伴って生じる制御集合の要素の変化は，$A \overset{*}{\Rightarrow} x \in V_T^*$ なる導出が行われたとき，部分文法 G_i において，ある部分導出 $B \overset{*}{\Rightarrow} y$ を認めるか否かのいずれかである．したがって，動的な制御の記述は生成規則中に A が含まれるものに

$[x, G_i : C(<B>) \cup \{y\}]$ または

$[x, G_i : C(<B>) - \{y\}]$

なる制御規則を付加することで表現できる．しかし，これまでわれわれが行った記述の試みでは，A が導出する記号列のうちのある特定のもの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 x を省略する形でよいと思われる．

このとき，x は A から導出された結果の記号列であり，y の一部として x を含むことができるので，実際の記述では「A が導出した終端記号列」を意味する記号が必要になる．これは記号に下線を施して $\underline{A}$ と表すことにする（A が複数個含まれるときは何個目かを表す添字をつけることで区別する）．

4.3 宣言に関わる制約の記述

コンパイラ生成系などで意味論として扱われている主要部分は，変数や名札の宣言に関する各種の制約，あるいはブロック構造に伴う有効範囲の問題などである（名札を立てることを，ここでは宣言として扱う）．動的な制御集合をもつ多段文法は，部分文法単位で導出を行うことから，以下に述べる形でこれらを簡明に表現できる．

基本的には，宣言とそれを使用する部分を別な部分文法で表し，宣言部分の導出が先に行われるように構成する．それぞれの具体的な記述例は非常に簡略化した ALGOL[6),7)] のサブセットを用いて**図4**に示した．この文法は動的な制御集合をもつ3個の部分文法からなり，導出時の各段では左から右への限定制御であると仮定している．また，制御集合の初期値は下記の2つを除いては $C(\alpha)=D(\alpha)$ であるとする．

$G_3: C(<変数>)=\phi$, $G_3: C(<名札>)=\phi$

（1） 二重宣言の禁止

変数や名札を宣言すると，その記号列をそれぞれの制御集合から取り去る．左から右への限定制御により，以後は，その段で同一の名前を宣言することができなくなる(図4の文法では *1 の記述)．

(部分文法 G_1: プログラム)
<プログラム>→≪ブロック≫
(部分文法 G_2: ブロック)
<ブロック>→begin<宣言><文の並び>end
【$G_3: C(<変数>) \cup D(<変数>)$, ……*3)
$G_3: C(<名札>) \cup D(<名札>)$】 ……*3)
<宣言> →real<変数の並び>;
<変数の並び>→<変数>
→<変数の並び>, <変数>
<変数> →<名前>
【$G_2: C(<変数>) - \{<変数>\}$, ……*1)
$G_2: C(<名札>) - \{<変数>\}$, ……*1)
$G_3: C(<変数>) \cup \{<変数>\}$, ……*2)
$G_3: C(<名札>) - \{<変数>\}$】 ……*4)
<文の並び>→<文>
→<文の並び>; <文>
<文> →<名札のない文>
→<名札>: <名札のない文>
<名札のない文>→≪実行文≫
→≪ブロック≫
<名札>→<名前>
【$G_2: C(<変数>) - \{<名札>\}$, ……*1)
$G_2: C(<名札>) - \{<名札>\}$, ……*1)
$G_3: C(<名札>) \cup \{<名札>\}$, ……*2)
$G_3: C(<変数>) - \{<名札>\}$】 ……*4)
<名前>→A
→<名前>A
(部分文法 G_3: 実行文)
<実行文>→<代入文>
→<飛び越し文>
<代入文>→<変数>:=<算術式>
<算術式>→<変数>
→<算術式>+<変数>
<変数>→<名前>
<飛び越し文>→goto<名札>
<名札>→<名前>
<名前>→A
→<名前>A

図4 ALGOL 60 サブセットに対する記述例
Fig. 4 An example of a subset of ALGOL 60 with dynamic control sets.

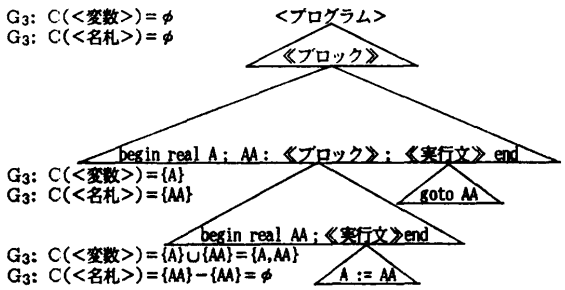


図5 図4の文法による導出例
[Fig. 5 A sentence produced by the grammar in Fig. 4.

（2） 宣言された変数や名札だけの使用許可

実行文における変数や名札の制御集合は，その初期値を空にしておき，宣言されるたびに，それぞれの制御集合に加えていく(図4の文法では *2 の記述)．

（3） 異なるブロックでの再定義

ブロックを表現する部分文法の最初の導出で，宣言を行う変数や名札の制御集合を初期状態に戻す（図4の文法では *3 の記述，次の項目と対で用いる)．

（4） ブロック構造における名前の有効範囲

宣言が行われるたびに，上位のブロックで定義された他の用途の同じ名前をその制御集合から取り去る．これにより，同じ名前が再定義されれば，それ以前の宣言は無効になり，再定義がなければ上位の宣言がそのまま有効となる(図4の文法では *4 の記述)．

このほか，図4では begin end 等を綴り記号として示したが，予約語として扱うときは，宣言文における変数や名札の制御集合から，予約語と同じ綴りの記号列を取り除く記述を行えばよい．また，複数の型を扱う場合は変数を型ごとに異なる非終端記号で表すことで表現できる．**図5**にはこの文法によって実際に導出を行った例と，そのときの制御集合の変化を示した．

5. 構文解析法

この章の内容は，一般には意味解析と言われる部分であるが，本論文では言語導出上の問題であるため，構文解析法として述べる．

前章では宣言やブロック構造に関わる各種の制約を，構文表示上で簡明に表せることを示したが，その解析もまた，構文解析の一部として容易に実現できる．

制御集合自体は，非終端記号とそれから導出される終端記号列，導出の許可／禁止のフラグからなる単純な表に相当し，制御集合の要素が動的に変化するもの

だけを対象にすればよい．要素の追加や削除の記述を伴う生成規則が適用されると，表の作成や変更が行われ，その対象となった非終端記号からの導出が行われたとき，表を参照することになる．このことは，手書きコンパイラにおける表の作成とその利用によく対応しており，言語の定義に基づく自動解析を，手書きに近い形で実現するものと言える．

ただ，制御集合の要素は動的に変化するため，構文解析の各時点での制御集合の確定方法が問題となる．2章で述べた生成規則の適用順序に関する制限方法のうち，最左導出の場合は，構文解析も左から右へ行えば，常にその時点の制御集合が得られるから特別な手法は不要である．多段文法表現に関しては，各部分文法の導出範囲の識別を必要とするので，以下にその方法を述べる．

5.1 汎用的な方法

制御集合を考慮しなければ，3.2 節で述べたように，与えられた多段文法とは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，最初はこの等価な文法 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．この導出木上では，各部分文法による導出がそれぞれ1つの部分木を構成しているから，部分木の頂点の記号によって対応する部分文法とその導出範囲は明確となる．次に部分文法単位で導出木の上位部分から順に再び調べれば，最上位の導出を行う制御集合は与えられた初期集合である．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くから，動的な制御を含めた完全な解析ができる．

5.2 境界を定める規則による方法

前節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，この問題も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」や「文」(「ブロック」とか「飛び越し文」といったもの) を1つの単位として見れば，それぞれは予約語や区切り記号によって，種別や左右の境界が特徴づけられており，その識別も比較的容易である．このような特徴は，使う人の書きやすさ，見やすさを考えれば，言語一般がもつごく自然な形と言える．本論文で述べた部分文法は，この構成要素とは1対1の対応ではないが，その集まりと考えることができる．それゆえ，このようなキーとなる記号に注目して，使用した部分文法の種類とその導出範囲を確定する一次解析[8]を行えば効率的な構文解析が行えると考えている．

図4の文法で言えば<ブロック>，<宣言>，<名札>，<飛び越し文>，<代入文>を構成要素にとると，begin end real goto := ; : がキー記号にあたり，これに注目して解析対象の文を左から右へ調べるだけで，構成要素の種別と範囲を確定できる．この結果から部分文法の導出範囲を決定することは容易であるから，このような一次解析を行うことで効率化を図れる．

6. おわりに

本論文では動的な制御集合をもち複数の部分文法からなる多段形式の文法について述べた．この文法はプログラミング言語の定義に際して，文脈依存を文脈自由文法の形で表現する手段を提供するとともに，意味解析として別に扱われてきた宣言文の内容に依存する事項を構文解析に統合して扱うことを可能にしている．とくに，基本構造の1つであるブロック構造によく適合し，これに関わる制約を簡明に表現できる．

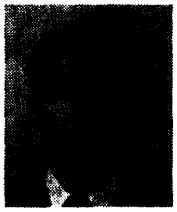
これを実際に，コンパイラ生成系などの実システムに適用するためには，記述を簡略化するための手段や，プログラミング言語一般を広く受け入れるためのレベルアップ，例えば，下位の導出を行う部分文法の制御集合自体を定義する能力をもつ形などの検討を行う必要があると考えている．このような実システム上の問題については，現在引続き検討を加えているので，改めて別の機会に報告したい．

参考文献

1) Ginzberg, S. and Spanier, E. H.: Control Sets on Grammars, *Math. System Theory*, Vol. 2, No. 2, pp. 159-177 (1968).
2) 例えば Kasai, T.: A Universal Context-Free Grammar, *Inf. Control*, Vol. 28, No. 1, pp. 30-33 (1975).
3) Knuth, D. E.: Semantics of Context-Free Languages, *Math. System Theory*, Vol. 2, No. 2, pp. 127-145 (1968).
4) 佐々政孝：コンパイラ生成系，情報処理，Vol. 23, No. 9, pp. 802-817 (1982).

5) Hopcroft, J. E. and Ullman, J. D.: *Formal Languages and Their Relation to Automata*, Addison-Wesley (野崎昭弘, 木村 泉訳：言語理論とオートマトン, サイエンス社 (1971)).
6) Naur, P. (ed.): Revised Report on the Algorithmic Language Algol 60, *Comm. ACM*, Vol. 6, No. 1, pp. 1-17 (1963).
7) 日本工業規格 JIS C 6210-1972 JIS ハンドブック情報処理, 日本規格協会 (1978).
8) 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論文集, pp. 215-216 (1984).

(昭和63年 7 月 20 日受付)
(平成 元 年 10 月 11 日採録)



山田 攻 (正会員)

昭和17年生. 昭和39年室蘭工業大学電気工学科卒業. 同年室蘭工業大学電気工学科助手. 昭和 48 年より同大学情報処理教育センター助教授, 現在に至る. 教育用システム, コンパイラ生成系などの研究に従事. 著書「電子計算機入門」(工学図書). 電子情報通信学会会員.

石田 純一 (正会員)

昭和23年生. 昭和48年室蘭工業大学修士課程 (電子工学) 修了. 昭和 49 年より室蘭工業大学情報処理教育センター助手, 現在に至る. 教育支援システム, プログラミング言語の研究に従事. 電子情報通信学会, 日本自動制御協会各会員.

野口 正一 (正会員)

昭和5年生. 昭和 29 年東北大学工学部電気工学科卒業. 昭和 35 年同大学院博士課程修了. 工学博士. 昭和 46 年東北大学電気通信研究所教授. 昭和 59 年東北大学大型計算機センター長. 主として情報システム構成論, 知識処理に関する研究に従事. 著書「情報ネットワーク理論」(岩波)「知識工学基礎論」(オーム社) など.