論　文

能率のよいトップダウン型構文解析プログラムの自動作成について*

吉　村　一　馬**

Abstract

A Syntax Analysis Program for LL(k) grammar by means of the top-down method is easy to understand.

It can easily be linked to Semantic Programs and be modified when the syntactic structure has to be changed. On the contrary, it could only deal with simple languages; it was hardly employed in order to construct compilers for practical languages, e. g. FORTRAN.

In the present report, an algorithm will be offered for an automatic generater of Syntatix Analysis Program for an extended LL(k) grammar. An optimization algorithm for a generated program will also be offered. In the program, no pattern-matchings are required for nonterminal symbols. Finally, the algorithm will be extended so that keeping the cleanness of the top-down method, it may deal with the Syntax Analysis of Fortran. A simple example of the extended algorithm is also given.

1. はじめに

コンパイラの構文解析部分を自動的に作成しようとする試みは，十数年前から行われている．Knuth の LR(k) 文法，Lewis, Stearns の LL(k)[1] 文法を対象とする構文解析法などが発見されてから，その実用化のきざしが見えてきた．LL(k) 方法では，構文解析のプログラムがトップダウン式に作成出来る．従って作成された構文解析プログラムは判りよく，変更や意味処理の文の挿入が容易なので，いくつかの試みがなされてきた[2)3)]．しかし FORTRAN での〈変数〉=〈数式〉，〈変数〉=〈論理式〉のように実際の言語に対し処理出来ない個所のある欠点があった．また，Backes[2)] の方法は判りよいが，非終端記号の照合を行うなど，照合に冗長な点があった．

この論文では，まず拡張された LL(k) 文法に対し，その構文解析プログラムを自動的に作成するアルゴリズムを与え，次に作成された構文解析プログラムの最適化の方法をのべる．これによって，先読みの記号を除くと，一番新しく読み込まれた記号を照合していくだけで構文解析が出来，照合が成功したとき必ず次の記号の読み込みをともなうプログラムが得られる．次にトップダウン方式の見やすさを残し，FORTRAN などに使える様にアルゴリズムを拡張する．

2. 構文解析アルゴリズムの表現

この章では，構文解析のアルゴリズムを表現する言語を説明し，それを用いて，構文解析プログラムをどのように作成するかを述べる．

2.1 構文解析記述言語

構文解析記述言語として，拡張された Floyd Production の表記法を次の様に定義する．以下の節ではこの言語を用いて構文解析アルゴリズムが表現される．

〈ラベル〉‖〈パターン〉〔〈先読み記号列〉〕
(〈S−処理部〉)(〈F−処理部〉)

〈ラベル〉はこの文につけられた名前である．照合させたい記号列をパターン欄に，先読み記号列を〔 〕の中にそれぞれ書く．〈パターン〉，〈先読み記号列〉に書かれた語の列が，入力文の対応する部分とそれぞれ一致するとき，〈S−処理部〉で定められる処理を行ない，一致しないとき〈F−処理部〉で定められる処理を行なう．パターン欄が σ のときは S−処理部だけ

* On the generating method of an efficient top-down Syntax analysis program by Kazuma YOSHIMURA (Central Research Laboratory HITACHI, Ltd.).

** (株)日立製作所中央研究所

行う．〈S−処理部〉も〈F−処理部〉も同じ形のものが書け，ラベル，数，#，＊，ERROR, RETURN などを ‘,’ で区切って並べると，その書かれた順にそれに対応する処理を行う．ラベルはそのラベルのついた文の呼び出しを，# は次の文の呼び出しをそれぞれ表わし，RETURN は呼び出しに対する戻りを表わす．ただし，処理部の最後に書かれたラベル，# は，それぞれ，そのラベル，次の文へのとびこしを表わす．＊は1記号の読み込みを，ERROR はエラー処理プログラムの呼び出しをそれぞれ表わす．数はそれに対応する意味処理を行わせるために使用し，生成規則による還元，表の操作，コードの生成などが必要に応じて書ける．処理部の最後に＊と数を書いてはいけない．

2.2 Backus 表記法で書かれた文法の構文解析記述語言への変換法

2.2.1 諸 定 義

文法 $G=(\mathbf{V}, \Sigma, P, S)$ を文脈自由文法[4] とする．ここで

$\mathbf{V}$: 有限個の記号の集合，

Σ: $\mathbf{V}$ の部分集合，Σ の元を終端記号といい，$\mathbf{V}-\Sigma$ の元を非終端記号という．

P: 文法の生成規則で，その要素を $\xi\rightarrow W$ の形に表わす．ここで $\xi\in\mathbf{V}-\Sigma$, $W\in\mathbf{V}^*$.

S: 文法 G の開始記号．

$W\in\mathbf{V}^*$から導出される終端記号列の先頭 k 個をとり出した記号列を表わす $h^k(W)$ を次の様に定義する．

定義 1 $h^k(W)=\{\alpha\,|\,|\alpha|=k,\ \alpha\in(\Sigma\cup\vdash),\ S\overbrace{\vdash\cdots\vdash}^{k}\Rightarrow^{*}\beta W\gamma\Rightarrow^{*}\beta\alpha\delta\}$

ここで $|\alpha|$ は記号列 α の字数を表わす．

定義 2 文法 G が次の条件をみたすとき，ALL(k) 文法と呼ぶ，これは LL(k) 文法[1]の拡張になっている．

(1) 任意の $A\in\mathbf{V}-\Sigma$ に対し，

$A\Rightarrow^{*}BV\Rightarrow^{*}AV',\ V, V'\in\mathbf{V}^*, B\neq A, B\in\mathbf{V}-\Sigma$

となる B は存在しない．

(2) P の元を→の左辺の非終端記号によって類別し，その類を次の様に表わす．

$$\left.\begin{array}{l}A\rightarrow u_1\\A\rightarrow u_2\\\vdots\\A\rightarrow u_{m_A}\end{array}\right\}\ (2.1)\qquad\left.\begin{array}{l}A\rightarrow Au_1'\\A\rightarrow Au_2'\\\vdots\\A\rightarrow Au_{n_A}'\end{array}\right\}\ (2.2)$$

ここで，$u_1, u_2, \cdots, u_{m_A}$ の先頭の記号は A でないとする．このとき，

$$D^k(A)\equiv\bigcup_{i\neq j}(h^k(u_i)\cap h^k(u_j))\qquad(2.3)$$

ここで，$i, j=1, 2, \cdots, m_A$

$$E^k(A)\equiv\bigcup_{i\neq j}(h^k(u_i')\cap h^k(u_j'))\qquad(2.4)$$

ここで，$i, j=1, 2, \cdots, n_A$

$N^k(A)\equiv\{\alpha\,|\,S\overset{*}{\Rightarrow}\gamma A\delta$ かつ $\gamma A\delta$ の A は (2.2) 式の生成規則でできたものはない．$\alpha\in h^k(\delta)\}$ としたとき，すべての非終端記号 A について

$$D^k(A)=\phi,\ E^k(A)=\phi,\ \bigcup_{i=1}^{n_A}h^k(u_i')\cap N^k(A)=\phi$$

となる．ここで ϕ は空集合を表わす．

$D^k(A)=\phi$ は，これから構文解析の処理をしようとする部分が A に還元されるのは判っているが，$A\rightarrow u_1, A\rightarrow u_2, \cdots, A\rightarrow u_{m_A}$ の内のどれによって行われるかが決定出来るための十分条件であり，$E^k(A)=\phi$ は A に還元されたあとで，$A\rightarrow Au_1', \cdots, A\rightarrow Au_{n_A}'$ のどれで還元されるか決定出来るための十分条件である．また $h^k(u_i')\cap N^k(A)=\phi$ は，左回帰の演算の処理が続けられるかを決定するための十分条件を与えている．

定義 3 $W_i\in 2^{\mathbf{V}^*}$, $S_i\in 2^{\Sigma^*}$ $(i=1, 2, \cdots, n)$ が次の条件をみたすとき，S_i は W_i を分離するための特徴記号列集合と言う．

任意の $\omega_i\in W_i$ に対し，$h^{|\omega|}(\omega_i)\ni\omega$ かつ $j\neq i$ なるすべての j, $\omega_j\in W_j$ に対し $h^{|\omega|}(\omega_j)\ni\omega$ となる $\omega\in S_i$ が存在し，ω の長さ 0 以上の先頭部分記号列 ω' をとると $j\neq i$ で $h^{|\omega'|}(\omega_j)\ni\omega'$ となる $\omega_j\in W_j$ が存在する．

2.2.2 変換のアルゴリズム

ここでは ALL(k) 文法から拡張された Floyd Production 言語への変換アルゴリズムをのべる．

$W_i\in 2^{\mathbf{V}^*}$ を分離するための特徴記号列の集合 S_i $(i=1, \cdots, n)$ を次の方法で計算する．

1) $S_j=\{a\,|\,a\in h^1(\omega_j),\ \omega_j\in W_j\}$, $k=1$ とおく．

2) $\{S_j\}$ の2つ以上の要素に属する α があり，$|\alpha|\leqq k$ なるとき，α を要素にもつものを $S_{j1}, \cdots, S_{jm}$ とする．S_{jl} $(l=1, \cdots, m)$ の各々から α を除き，すべての $\omega_{jl}\in W_{jl}$ に対し $h^{k+1}(\omega_{jl})$ の要素で先頭が α となる記号列を S_{jl} に加える．2)を行える限り行う．

3) 2つ以上の S_j に含まれるものがあれば $k=k+1$ として 2)を行う．

特徴記号列が存在すれば上の処理は有限で終わる．

(2.1) 式のどの生成規則を適用するかを決定するため，$u_1, \cdots, u_{m_A}$ を分離するための特徴記号列の集合

$C^k(A, u_j)$ $j=1, \cdots, m_A$ 及び $u_1', \cdots, u_{nA}'$, $N^k(A)$ を分離するための特徴記号列の集合 $L^k(A, u_j')$, $j=1, \cdots, n_A$, $M^k(A)$ を計算する.

$$C^k(A)=\bigcup_{i=1}^{m_A} C^k(A, u_i)$$

$$L^k(A)=\bigcup_{i=1}^{n_A} L^k(A, u_i')$$

とおく.

(1) 記号列が A に還元されることが判っているときどの生成規則で A に還元するかを決定するプログラムを作成する手続 $P_1(A)$ についてのべる.

$C^k(A)$ の記号列の先頭の記号で異なるものを $a_1, a_2, \cdots, a_l$ とし, a_i を先頭にもつ記号列を $a_iV_{i1}, a_iV_{i2}, \cdots, a_iV_{it_i}$ とする.

このとき,

$$\left.\begin{array}{lllll} A\| & a_1[V_{11}] & (g_{11}) & (\#) & \\ & \vdots & & & \\ & a_i[V_{ij}] & (g_{ij}) & (\#) & \\ & \vdots & & & \\ & a_l[V_{lt_l}] & (g_{lt_l}) & \text{(ERROR)} & \end{array}\right\} \quad (2.5)$$

$i=1, 2, \cdots, l$　$j=1, 2, \cdots, t_i$

を作成する*.

ここで g_{ij} はラベルで, $a_iV_{ij} \in C^k(A, u_i)$ となるとき, Al を表わしている. Al は次に述べる手続き $P_2(A)$ で作られる. ALL(k) 文法では, $a_iV_{ij} \in C^k(A, u_i)$ となる u_i は唯一である.

(2) $A \to u_i$ の還元が行われる筈と判っているとき, u_i の各要素について, 実際にそうなっていることを確かめるためのプログラムを作成する手続き $P_2(A)$ について述べる.

$u_i=X_1X_2\cdots X_m$, $X_j \in \mathbf{V}$

とすると,

$$\left.\begin{array}{ll} Ai\| & s_1 \\ & \vdots \\ & s_m \end{array}\right\} \quad (2.6)$$

ここで s_j は $1 \leq j < m$ に対し, X_j が終端記号か非終端記号かに従って,

X_j $(*, \#)$ (ERROR)

或いは,

σ $(X_j, \#)$

をそれぞれ表わす. s_m は X_m が終端記号か非終端記号かに従って,

X_m $(*, k, \$)$ (ERROR),

σ $(X_m, k, \$)$

をそれぞれ表わす.

ここで k は $A \to u_i$ が k 番目の生成規則であることを示し, それに対応する出力をする意味を持っている. (2.2)式において, $n_A=0$ のときは, 上の $ を RETURN で置きかえる. $n_A>0$ のときは, $ をラベル $A\#$ で置きかえる.

(3) 次に (2.2) 式で左回帰の還元を更に行なうか, 還元を終えるのを決定するプログラムを作る手続き $P_3(A)$ を述べる.

$L^k(A) \cup M^k(A)$ の元の先頭の記号で異なるものを $b_1, b_2, \cdots, b_S$ とし, b_j を先頭にもつ記号列を $b_jW_{j1}, b_jW_{j2}, \cdots, b_jW_j\alpha_j$ とする. このとき

$$\left.\begin{array}{lllll} A\#\| & b_1[W_{11}] & (g_{11}) & (\#) & \\ & \vdots & & & \\ & b_i[W_{iij}] & (g_{iij}) & (\#) & \\ & \vdots & & & \\ & b_S[W_{S\alpha_S}] & (g_{S\alpha_S}) & \text{(ERROR)} & \end{array}\right\} \quad (2.7)$$

を作る. ここで $i=1, 2, \cdots, S$

$j=1, 2, \cdots, \alpha_i$

F−処理部は最後のものを除いて, (#) となっている. g_{iij} は $b_iW_{iij} \in L^k(A, u_i')$ のときラベル $A\#l$ を表わし, $b_iW_{iij} \in M^k(A)$ のとき RETURN を表わす. 仮定から b_iW_{iij} は $L^k(A, u_i')$ のどれかの1つか, または $M^k(A)$ にしか属しない.

(4) 最後に $A \to Au_j'$ の還元が行なわれる筈と判っているとき, u_j' の各要素について, 実際そうなっていることを確めるためのプログラムを作成する手続き $P_4(A, u_j')$ について述べる. これは手続き $P_2(A)$ に類似している. $u_j'=Y_1Y_2\cdots Y_n$ とすると,

$$\left.\begin{array}{ll} A\#j\| & s_1 \\ & \vdots \\ & s_n \end{array}\right\} \quad (2.8)$$

を作成する. ここで s_i は $1 \leq i < n$ のとき

Y_i が終端記号か非終端記号かに従い,

Y_i $(*, \#)$ (ERROR)

或いは

σ $(Y_i, \#)$

を表わす. s_n は

Y_n が終端記号か非終端記号かに従い,

Y_n $(*, q, A\#)$ (ERROR)

或いは

---

* このようにすべての非終端記号に対して, それをラベルとするプログラムが作られる. これは入力記号列の先頭部分をその非終端記号に還元するプログラムである. 以下の手続きで作られる Ai, $A\#$, $A\#j$ などは解析の便宜上作られるラベルである. しかし, これらのものも, サブプログラムとして呼び出すことが出来る.

σ $(Y_n, q, A\#)$

を表わす.

ここで q は $A \rightarrow Au_j'$ が q 番目の生成規則であることを示す.

定理 1 手続き $P_i(A)$ $(i=1,2,3,4)$ をすべての非終端記号 A について行って生成されたものに,

START‖ σ $(S, \#)$

$\vdash$ (EXIT) (ERROR)

を付けて得られるプログラムは START を入口とし, EXIT を出口とする文法 G の構文解析プログラムになっている.

2.3 最適化の方法

前節の手続きで得られたプログラムの最適化の方法をのべる.

2.3.1 プロダクションの統合

式 (2.5), (2.7)において, 照合欄の記号が同じで, 先読み記号列が異なるものがいくつかあるとする. その先読み記号列の情報は分岐のためだけに使い, 誤りの検出は読み込んだあとで行わせることにより, 文の統合を行う. このために誤りのある文を正しい文として認識する恐れがないだけでなく, 入力文での誤りの発見場所がずれるとしても k 個までである (先読みで k 個先の誤りが発見されることもあり得るから). 手続きは次の様に表わされる.

(2.5) 式の照合部に書かれている記号が a_i となっている文を S-処理部によって類別したとき, 類の個数が最大となる文を全部消去する. 次に照合欄が a_i である文の最後の文の次に, 消去した文の先読み記号列だけ消去したものを挿入する. この処理で, 先読み記号列は異なるが, S-処理部は同じものが一つに統合される. (2.5) 式の最後の文の F-処理部の ERROR は最初 # におきかえておき, この統合の手続きを照合欄の先頭が a_i のものについて, $i=1,\cdots,l$ まで行う. そのあとで, 最後の行の F-処理部を ERROR に変える.

(2.7) 式では, 更に照合欄の記号もこめて分岐先で照合されるので, 分岐先の決定にだけ照合欄の情報を使えばよい. 従って, $g_{11},\cdots,g_{s\alpha_s}$ の内, 最も重複の回数の多いラベルを g とする. g を S-処理部にもつ文を全部消去する. 最後の文の F-処理部を (♯) にし, その次に σ (g) をつけ加える.

補題 1 この統合の処理を前節で作った構文解析プログラムにほどこしても, プログラムはもとのプログラムに等価である.

このことは先読みによる誤りの検出をやめても, 読み込みの所で改めて調べられるから明らかである.

この統合は, ある一つの文に分岐するものが数多く, 他は各1個あてに分岐する様になっているとき, 効果が大きい. 照合の回数を大きく減らし, しかも行数を減らすので, 文法を表現するメモリの数も減らせる.

2.3.2 照合の処理のあとまわし

照合の結果, その S-処理部でまた同じ照合を行なう場合が起るので, 照合の操作を後へ遅らすことで文の単純化をはかるものである.

$$\left.\begin{array}{lll} W_1 & (S_1) & (\#) \\ W_2 & (S_2) & (\#) \\ \vdots & & \\ W_{n-1} & (S_{n-1}) & (\#) \\ W_n & (S_n) & (\mathrm{ERROR}) \end{array}\right\} \qquad (2.9)$$

のように, F-処理部に (♯) がつながり, 最後に ERROR で終るものをプロダクション・リストと呼ぶことにする. 式 (2.5), (2.7) はその例になっている. これは読み込んだ記号が (先読み記号列もつないで) $W_1,\cdots,W_n$ のどれであるかを調べようというものである. このリストの最後の行は σ (S_n) の形でもよいことにしておく. (2.9)式のリストにおいて, i 番目の文の S-処理部を $(S_{i1}, S_{i2},\cdots)$ とする. ここで S_{i1} はラベルとする. S_{i1} のラベルのついたプロダクション・リストの中に照合部が W_i となるものがあり, (2.9) 式のプロダクション・リストの最後が,

1) σ 文でないと, i 番目の文をプロダクション・リストの最後の次に, 照合部を σ に入れかえて移し, 今まで最後の行であった文の F-処理部を (♯) でおきかえる.

2) σ 文のときは, その処理部 (S_n) が (S_i) と一致するときは, i 番目の文を消す. (S_n) と (S_i) が一致しないときは, S_{i1} のラベルのついたプロダクション・リストの中で S_i と同じ照合部をもつ文の S-処理部を (Q) とするとき (S_i) を $(Q, S_{i2},\cdots)$ と書きかえる.

1)の処理を $S_{11}, S_{21},\cdots, S_{n1}$ の内重複度の一番大きいラベルを S-処理部の先頭にもつ文から始める.

補題 2 プログラムにこの照合のあとまわしの処理をほどこしても等価なプログラムになる.

この手続きにより, 照合の処理が呼ばれるプログラムの方にまとめられる場合が多いので, 必要な記憶容量を縮めることが出来る. また照合の回数も減少す

る.

2.3.3 照合の処理のまえだおし

照合の結果判った情報を出来るだけ使用し，照合回数を出来るだけ減らそうというものである.

(1) プロダクション文

W_i $(S_{i1}, \cdots, S_{im})$ $(Q_{i1}, \cdots)$

において，S_{i1} と言うラベルのついたプログラムで W_i と同じ照合部をもつプロダクション文の S-処理部が $S_{j1}, \cdots, S_{jk}$ とすると，上のプロダクション文を

W_i $(S_{j1}, \cdots, S_{jk}, S_{i2}, \cdots, S_{im})$ $(Q_{i1}, \cdots)$

に書きかえる．この処理をこれ以上出来なくなるまで行なう.

(2) プロダクションの処理部に…, $S_{k1}, \cdots, S_{kl}, \cdots$ なるラベルの列があり，S_{kj} の各々が

$S_{kj}\|$　W_j　(S_j)　(RETURN)

$j=1,2,\cdots,l$ となっているとき

ラベル $S_{k1}\cdots S_{kl}$ をもつ次のプロダクション・リストをプログラムに追加し，処理部でラベルの列 $S_{k1}, \cdots, S_{kl}$ の現れている個所をすべてラベル $S_{k1}\cdots S_{kl}$ でおきかえる.

$S_{k1}\cdots S_{kl}\|$　W_1　$(S_1, S_{k2}, \cdots, S_{kl})$　(≠)

W_2　$(S_2, S_{k3}, \cdots, S_{kl})$　(≠)

⋮

w_l　(S_l)　(RETURN)

補題 3 プログラムに照合の処理の前だおしをしても等価なプログラムになる.

この手続きで，ラベルのついたプログラムの呼び出しの回数及び照合回数が大幅に減るので，構文解析に要する時間が短縮される．しかし，記憶容量の必要量は増える.

2.3.4 σ文の消去

照合欄が σ だけの文を σ 文と言う．この文の特徴は F-処理部のないことである．この σ 文を呼び出している個所（この σ 文にラベルがついていないときはすぐ上の文の処理部で # と書いてある所，ラベルのついているときは，すぐ上の # のついている処理部と，そのラベルを呼び出している所）について次のように変更する．σ 文を

σ　$(S_1, S_2, \cdots, S_n)$

とし，これを呼び出している処理部が $(Q_1, \cdots, Q_k, \#)$ とすると，その処理部を $(Q_1, Q_2, \cdots, Q_k, S_1, \cdots, S_n)$ でおきかえる．また呼び出している処理部が $(T_1, T_2, \cdots, T_r, S, T_{r+1}, \cdots)$ で，σ 文のラベルが S のとき，この処理部を $(T_1, T_2, \cdots, T_r, S_1, S_2, \cdots S_n, T_{r+1}, \cdots)$ でおきかえる．この σ 文を呼び出すすべての文について，この処理を行ったあと，この σ 文を消去する．ただし，ラベル $S_1, S_2, \cdots, S_n$ の中に # があると，# ごとに新しいラベル NS を作って σ 文の次につけ，$S_1, \cdots, S_n$ の中の # を NS でおきかえておく.

補題 4 プログラムに σ 文の消去を行っても等価なプログラムになる.

σ 文を呼び出している所での「呼び出されるプログラムの名前の列」と，置きかえによって出来る「呼び出されるプログラムの名前の列」が等しいことから明かである.

この処理により，照合の回数は減少しないし，プログラムの呼び出しの回数も減少しないが，とびこしの命令が1個減らせること，行が減って整理されること，及び S, F 処理部を一つにまとまった処理としてとらえることで能率化が考えやすいなどの長所がある.

2.3.5 その他の簡約化の方法

(1) START から到達出来ないプロダクション文を消去する.

(2) プロダクションに

$S_k\|$　ω_k　(Q, RETURN)　(Q')

または

$S_k\|$　ω_k　(Q, RETURN)　(#)

で次の文にラベル Q' がついているものがあるとする．このとき，処理部にラベルの列 …, $S_l, S_k, S_m, \cdots$ が書かれている文があると，この処理部を $\cdots S_l, S_{km} \cdots$ に書き変え，ラベル $S_{km}\ldots$ をもつ次のプログラムを加える.

$S_{km}\ldots\|$　ω_k　$(Q, S_m\cdots)$　$(Q', S_m\cdots)$

補題1～4から次のことが言える.

定理 2 定理1でのべられた処理によって得られた文法Gの構文解析プログラムに 2.3.1～2.3.5 の最適化の処理をして得られるプログラムは文法Gの構文解析プログラムになっている.

3. 例　題

Earley の与えた単純数式を例にとって，上記のアルゴリズムを示す．まず構文を **Table 1** (次頁参照) に与える.

2.2.2 でのべた方法によって作成される解析プログラムを **Table 2** (次頁参照) に与える．最適化の処理の 2.3.1 をほどこして S-処理部が Return となる文の統合を行ったあと，2.3.3 の1)の照合の前だおし，

Table 1 A Syntax of Simple Arithmetic Expression

S→*i*←E	1	T→T∗F	6
E→T	2	F→P	7
E→±T	3	F→F↑P	8
E→E±T	4	P→*i*	9
T→F	5	P→(E)	10

Table 2 A Syntax Analysis Program of the Simple Arithmetic Expression

START	‖	σ	(S,♯)	
		⊢	(Exit)	(ERROR)
S	‖	*i*	(S1)	(ERROR)
S1	‖	*i*	(∗,♯)	(ERROR)
		←	(∗,♯)	(ERROR)
		σ	(E,1,RETURN)	
E	‖	*i*	(E1)	(♯)
		±	(E2)	(♯)
		(	(E1)	(ERROR)
E1	‖	σ	(T,2,E♯)	
E2	‖	±	(∗,♯)	(ERROR)
		σ	(T,3,E♯)	
E♯	‖	±	(E♯1)	(♯)
		)	(RETURN)	(♯)
		⊢	(RETURN)	(ERROR)
E♯1	‖	±	(∗,♯)	(ERROR)
		σ	(T,4,E♯)	
T	‖	*i*	(T1)	(♯)
		(	(T1)	(ERROR)
T1	‖	σ	(F,5,T♯)	
T♯	‖	∗	(T♯1)	(♯)
		)	(RETURN)	(♯)
		±	(RETURN)	(♯)
		⊢	(RETURN)	(ERROR)
T♯1	‖	∗	(∗,♯)	(ERROR)
		σ	(F,6,T♯)	
F	‖	*i*	(F1)	(♯)
		(	(F1)	(ERROR)
F1	‖	σ	(P,7,F♯)	
F♯	‖	↑	(F♯1)	(♯)
		±	(RETURN)	(♯)
		∗	(RETURN)	(♯)
		)	(RETURN)	(♯)
		⊢	(RETURN)	(ERROR)
F♯1	‖	↑	(∗,♯)	(ERROR)
		σ	(P,8,F♯)	
P	‖	*i*	(P1)	(♯)
		(	(P2)	(ERROR)
P1	‖	*i*	(∗,9,RETURN)	(ERROR)
P2	‖	(	(∗,♯)	
		σ	(E,♯)	
		)	(∗,10,RETURN)	(ERROR)

2.3.4の σ 文の消去，および 2.3.5 1）を行ったものが **Table 3**（次頁参照）である．Table 3 のものに 2.3.2，2.3.4 を行ったものを **Table 4**（次頁参照）に，2.3.3 を(1)，(2)の順に行ったのち，共通の文を統合したものを **Table 5**（次頁参照）にあげる．Table 4，5 のものは記号の照合の無駄がない．サブプログラムの呼び出しは一般に回帰的に行われるので能率が悪いが，Table 5 のものはサブプログラムの呼び出しの回数がすくないので，処理能率がよい．

処理部の数字は句に還元される時に行う意味処理プログラムの番号と考えることが出来るので，Table 5 では意味処理をともなわない 2，5，7，10 を処理部から除いてある．コンパイラ作成のときは更に，加減，乗除，論理式での関係演算子，入出力文などの様に形が同じものは，語彙解析で同じ記号に変換し構文解析を単純にするとか，構文解析の情報を使って語彙解析を簡単にするなどの方法をとる．

4. アルゴリズムの拡張

2章であげたアリゴリズムに従って FORTRAN の

Table 3 A Simplified Program (1)

START	‖	σ	(S, ♯)	
		⊢	(Exit)	(ERROR)
S	‖	i	(∗, S2)	(ERROR)
S2	‖	←	(∗, E, 1, RETURN)	(ERROR)
E	‖	i	(T, 2, E♯)	(♯)
		±	(∗, T, 3, E♯)	(♯)
		(	(T, 2, E♯)	(ERROR)
E♯	‖	±	(∗, T, 4, E♯)	(RETURN)
T	‖	i	(F, 5, T♯)	(♯)
		(	(F, 5, T♯)	(ERROR)
T♯	‖	∗	(∗, F, 6, T♯)	(RETURN)
F	‖	i	(P, 7, F♯)	(♯)
		(	(P, 7, F♯)	(ERROR)
F♯	‖	↑	(∗, P, 8, F♯)	(RETURN)
P	‖	i	(∗, 9, RETURN)	(♯)
		(	(∗, E, P4)	(ERROR)
P4	‖	)	(∗, 10, RETURN)	(ERROR)

Table 4 A Simplified Program (2)

START	‖	σ	(S, ♯)	
		⊢	(Exit)	(ERROR)
S	‖	i	(∗, ♯)	(ERROR)
		←	(∗, E, 1, RETURN)	(ERROR)
E	‖	±	(∗, P, 7, F♯, 5, T♯, 3, E♯)	(P, 7, F♯, 5, T♯, 2, E♯)
E♯	‖	±	(∗, P, 7, F♯, 5, T♯, 4, E♯)	(RETURN)
T♯	‖	∗	(∗, P, 7, F♯, 6, T♯)	(RETURN)
F♯	‖	↑	(∗, P, 8, F♯)	(RETURN)
P	‖	i	(∗, 9, RETURN)	(♯)
		(	(∗, E, P4)	(∗, E, P4)
P4	‖	)	(∗, 10, RETURN)	(ERROR)

Table 5 A Simplified Program (3)

START	‖	σ	(S, ♯)	
		⊢	(Exit)	(ERROR)
S	‖	i	(∗, ♯)	(ERROR)
		←	(∗, E, 1, RETURN)	(ERROR)
E	‖	i	(∗, 9, F♯T♯E♯)	(♯)
		±	(∗, T, 3, E♯)	(♯)
		(	(∗, E, P4, F♯T♯E♯)	(ERROR)
T	‖	i	(∗, 9, F♯T♯)	(♯)
		(	(∗, E, P4, F♯T♯)	(ERROR)
F	‖	i	(∗, 9, F♯)	(♯)
		(	(∗, E, P4, F♯)	(ERROR)
P	‖	i	(∗, 9, RETURN)	(♯)
		(	(∗, E, P4)	(ERROR)
P4	‖	)	(∗, RETURN)	(ERROR)
F♯T♯E♯	‖	↑	(∗, P, 8, F♯T♯E♯)	(♯)
T♯E♯	‖	∗	(∗, F, 6, T♯E♯)	(♯)
E♯	‖	±	(∗, T, 4, E♯)	(RETURN)
F♯T♯	‖	↑	(∗, P, 8, F♯T♯)	(♯)
T♯	‖	∗	(∗, F, 6, T♯)	(RETURN)
F♯	‖	↑	(∗, P, 8, F♯)	(RETURN)

構文解析プログラムを作ろうとすると，〈論理1次式〉から導出された（〈論理式〉）と〈数式〉≡〈数式〉* のように，いくら先読みをしてもそのいずれかを決定出来ない個所がある．これは〈論理1次式〉のようにある非終端記号 A に対し，(2.3)式で定義した $D^k(A)$ が空にならない個所である．この章ではこのような場合を解決するアルゴリズムを与える．

* EQ を≡で表わしてある．

4.1 従属グラフの作成

1) $A\to u_1$, $A\to u_2$, …, $A\to u_l$ を(2.1)式の内で $h^k(u_i)\cap h^k(u_j)$ が空にならないことのあるものの全体とする．(A, u_i) $i=1,2,\cdots,l$ をグラフの最初の節とし，これに入口節の印をつける．

2) 各節 (A, u) において，u の先頭の字 ξ が非終端記号であるとき，$\xi\to\alpha_1$, $\xi\to\alpha_2$, …, $\xi\to\alpha_t$ を左辺が ξ である生成規則の内，左回帰なものを除く全体とす

Table 6 A Syntax of a Logical Expression

S→*j*←BE	11	BP→*j*	16
BE→BT	12	BP→¬BP	17
BE→BE∨BT	13	BP→E≡E	18
BT→BP	14	BP→(BE)	19
BT→BT∧BP	15		

Table 7 A Syntax Analysis Program of the Logical Expression (1)

S	‖	*i*	(∗, S1)	(#)
		j	(∗, S2)	(ERROR)
S1	‖	←	(∗, E, 1, RETURN)	(ERROR)
S2	‖	←	(∗, BE, 11, RETURN)	(ERROR)
BE	‖	*j*	(BT, 12, BE#)	(#)
		i	(BT, 12, BE#)	(#)
		¬	(BT, 12, BE#)	(#)
		(	(BT, 12, BE#)	(ERROR)
BE#	‖	∨	(∗, BT, 13, BE#)	(RETRUN)
BT	‖	*j*	(BP, 14, BT#)	(#)
		i	(BP, 14, BT#)	(#)
		(	(BP, 14, BT#)	(#)
		¬	(BP, 14, BT#)	(ERROR)
BT#	‖	∧	(∗, BP, 15, BT#)	(RETURN)
BP	‖	*j*	(∗, 16, RETURN)	(#)
		i	(E, BP1)	(#)
		(	(〈分離出来ない〉)	(#)
		¬	(∗, BP, 17, RETURN)	(ERROR)
BP1	‖	≡	(∗, E, 18, RETURN)	(ERROR)

注) 1) Eの処理プログラムは Table 5 のものを用いる.
2) 〈分離出来ない〉の所が2章のアルゴリズムで処理出来ない.

Table 8 An example by extended algorithm.

L@	‖	¬	(∗, BP, 17, Path-1)	(#)
		j	(∗, 16, Path-1)	(#)
		±	(N@3)	(#)
		i	(∗, 9, Path-2)	(ERROR)
N@1	‖	≡	(∗, E, 18, CRT, PC-1)	(ERROR)
N@2	‖	)	(POP(( ), ∗, CRT, PC-2)	(ERROR)
N@3	=	σ	(∗, T, 3, E#, PC-3)	
N@4	‖	σ	(2, E#, PC-4)	
N@5	‖	)	(POP(( ), ∗, 19. 10, PC-5)	(ERROR)
PC-1	‖	σ	(Path-1)	
PC-2	‖	σ	(Path-1)	
PC-3	‖	≡	(Path-3)	
		)	(Path-4)	(ERROR)
PC-4	‖	≡	(Path-3)	
		)	(Path-4)	(ERROR)
PC-5	‖	σ	(Path-2)	
Path-1	‖	σ	(14, BT#, 12, BE#, N@2)	
Path-2	‖	σ	(7, F#, 5, T#, N@4)	
Path-3	‖	σ	(N@1)	
Path-4	‖	σ	(N@5)	

ると，節 (A, u) から節 $(\xi, \alpha_1), \cdots, (\xi, \alpha_t)$ への有向枝をつけ加える．

3) 2)の手続きが出来なくなるまで行なう．

節 (η, ω) で，ω の先頭の記号が終端記号 a であるとき，この節に a が現われるということにする．2つ以上の節に現われる終端記号の全体を $I=\{x_1, \cdots, x_m\}$ とする．上記の手続きで得られた有向グラフの各節 (D, γ) に対して，次の処理を行なう．

4) $\gamma=x_i\xi\beta$ において，$x_i \in I$，β は記号列となっているとき，節 (D, γ) から節 $(\xi, \alpha_1), \cdots, (\xi, \alpha_t)$ への有向枝をつけ加える．ここで $\xi \to \alpha_1, \cdots, \xi \to \alpha_t$ を左辺が ξ となる生成規則の内左回帰のものを除いたものを全部とする．

5) 2) 及び 4) の処理がこれ以上出来なくなるまで行なう．

Table 1 の数式の構文に **Table 6** の論理式の構文を追加した文法では，BP からの導出でいくら先読みしても '(' がつづくと分離出来ないので，(BP, E≡E)

Table 9 A Syntax Analysis Program of the Logical Expression (2)

S	‖	*i*	(∗, S1)	(♯)
		j	(∗, S2)	(ERROR)
S1	‖	←	(∗, E, 1, RETURN)	(ERROR)
S2	‖	⟝	(∗, BE, 11, RETURN)	(ERROR)
BE	‖	*j*	(∗, 16, BTE)	(♯)
		i	(∗, 9, F♯T♯E♯, BP1, BTE)	(♯)
		¬	(∗, BP, 17, BTE)	(♯)
		(	(Stack&Read, L@, BTE)	(ERROR)
BE♯	‖	∨	(∗, BT, 13, BE♯)	(RETURN)
BT	‖	*j*	(∗, 16, BT♯)	(♯)
		i	(∗, 9, F♯T♯E♯, BP1, BT♯)	(♯)
		¬	(∗, BP, 17, BT♯)	(♯)
		(	(Stack&Read, L@, BT♯)	(ERROR)
BT♯	‖	∧	(∗, BP, 15, BT♯)	(RETURN)
BP	‖	*j*	(∗, 16, RETURN)	(♯)
		i	(∗, 9, F♯T♯E♯, BP1)	(♯)
		(	(Stack&Read, L@)	(ERROR)
		¬	(∗, BP, 17, RETURN)	(ERROR)
BP1	‖	≡	(∗, E, 18, RETURN)	(ERROR)
BTE	‖	∧	(∗, BP, 15, BTE)	(BE♯)
L@	‖	¬	(∗, BP, 17, BTE, N@2)	(♯)
		j	(∗, 16, BTE, N@2)	(♯)
		±	(∗, T, 3, E♯, PC-3)	(♯)
		i	(∗, 9, F♯T♯E♯, PC-3)	(ERROR)
N@1	‖	≡	(∗, E, 18, CRT, BTE, N@2)	(ERROR)
N@2	‖	)	(POP ((), ∗, CRT, BTE, N@2)	(ERROR)
N@5	‖	)	(POP ((), ∗, F♯T♯E♯, PC-3)	(ERROR)
PC-3	‖	≡	(N@1)	(♯)
		)	(N@5)	(ERROR)

注) Table 5 の5行目以下と一緒に用いる.

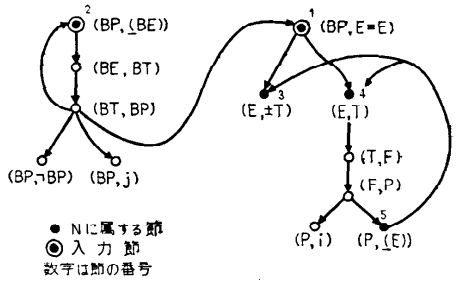

Fig. 1 An Example of the dependency Graph

と (BP,(BE)) が入口節となる. **Fig. 1** にその従属グラフをあげる.

4.2 解析手続きの作成

2章でのべた方法では, 2つ以上の生成規則のどれを適用したらよいか決定出来ない個所に対し, 上にのべた従属グラフから, どの生成規則を適用するかを決めるプログラムの作成法を述べる. このプログラムでは新しく一つのスタックを使用する.

I の元の現われる節, 入口節, 入る枝が2本以上ある節の全体を N で表わし, これに番号をつける.

1) I の元以外の終端記号をもつ節 $(\xi, a\eta\mu)$, (ここで $\eta\in\mathbf{V}-\Sigma, \mu\in\mathbf{V}^*$) から, はじめて N に属する節までさかのぼる. その節を (γ, u) とする. 節 (γ, u) から節 $(\xi, a\eta\mu)$ までの節の列を $(\gamma, u), (\eta_1, \eta_2 u_1), (\eta_2, \eta_3 u_2), \cdots, (\eta_m, a u_m)$ とし, 対応する生成規則の番号を $p_0, p_1, \cdots, p_m$ とする. ここで $\eta_m=\xi$, $u_m=\eta\mu$ とする. この時, 対

a　$(*, \eta, \bar{\mu}, p_m, \eta_m\sharp, \text{path-}l)$

を作る. これは a からはじまる記号列が u の左から最初の非終端記号に還元されることを確かめるプログラムである. ここで $\bar{\mu}$ は記号列 μ を認識するプログラムを表わす, $\eta_m\sharp$ は η_m が左回帰な生成規則をもつときに書かれる. path-l はラベル名で, 節の列 (γ, u), $(\eta_1, \eta_2 u_1), \cdots, (\eta_{m-1}, \eta_m u_{m-1})$ に対応するプログラム

path-l ‖ σ　$(\bar{u}_{m-1}, p_{m-1}, \eta_{m-1}\sharp, \cdots, \bar{u}_1, p_1, \eta_1\sharp, N@l')$

を表わす. これは η_m まで還元が終ったあと η_1 まで還元がおこなえることを確かめるプログラムである. ここで $\eta_i\sharp$ は η_i が左回帰の生成規則をもつとき書き, $N@l'$ はラベル名で, l' は節 (γ, u) の番号を表わす.

ただし, $m=1$ のときは対 a $(*, \eta, \bar{\mu}, p_1, \eta_1\sharp, N@l')$ を, $m=0$ のときは対 a $(N@l')$ を作る. すべての節について, 節に現われる I の元以外の終端記号の集合を $a_1, \cdots, a_k$ とし, 各 a_i について上にのべた対 a_i (v_i) を作り, 次のプログラムを作る.

$L@\|$　a_1 (v_1) $(\sharp)$
a_2 (v_2) $(\sharp)$
⋮
a_k (v_k) (ERROR)

2) N の各節 (ξ, u) に対し，u が $x\eta\mu$, $a\eta\mu$, $\eta\mu$ (ここで $x\in I$, $\eta\in \mathbf{V}-\Sigma$, $\mu\in \mathbf{V}^*$) に従い，それぞれ次のプログラムを作る．これは u の部分を確かめるものである．l は節 (ξ, u) の番号である．

$N@l\|\sigma$　(pop(x), $\bar{\mu}$, p, $\xi\sharp$, CRT, PC-l),
$N@l\|\sigma$　($*$, η, $\bar{\mu}$, p, $\xi\sharp$, CRT, PC-l),
$N@l\|\sigma$　($\bar{\mu}$, p, $\xi\sharp$, CRT, PC-l)

ここで pop(x) はスタックの一番上が x なら一記号ポップアップし，そうでないと誤りとするプログラムを表わす．CRT (Check & Return) は節 (ξ, u) が入口節のときのみ作り，スタックの一番上に $L@$ を呼び出した時にスタックにつめた「$L@$を呼びだした印」が入っているとそれをポップアップして呼んだプログラムへの戻りを，そうでないときは次に書かれている処理をするプログラム名である．PC-l はラベル名である．p は生成規則 $\xi\to u$ の番号を表わし，$\xi\sharp$ は ξ が左回帰の生成規則をもつときだけ作る．

3) N の元の節 (γ, u) から1つ以上の節をさかのぼって，はじめて N の節に達するまでの節の列を逆にならべて，$(\delta^1, v^1), (\delta_1^1, \delta_2^1 v_1^1), \cdots, (\delta_{m-1}^1, \delta^1{}_m v_{m-1}^1)$, $(\delta_m{}^1, u)$ $j=1, \cdots, q$, $\delta_m{}^1=\gamma$ とする．

$\delta_{mj-1}{}^1\to\delta_m{}^1 v_{m-1}{}^1$ が生成規則で，$S\overset{*}{\Rightarrow}\alpha_j\delta_{mj-1}{}^1\beta_j$ とするとき，$h^1(v_{mj-1}{}^1\beta_j)$ が $j=1, \cdots, q$ に対し，どの2つも共通部分がないとする．(γ, u) への入力枝が一つのとき，PC-$l\|\sigma$ (path-k) を作り，入力枚が2つ以上のときは，$h^1(u_{mj-1}{}^1\beta_j)$ の要素を $b_{j1}, \cdots, b_{it_j}$ とすると

PC-$l\|$　b_{11} (path-k_1)　($\sharp$)
⋮
b_{1t_1} (path-k_1)　($\sharp$)
b_{21} (path-k_2)　($\sharp$)
⋮
b_{nt_n} (path-k_n)　(ERROR)

を作る．ラベル PC-l (path-check) のついたプログラムはどの道をたどって N の節に至るかを決め，還元をつづけるものである．Path--k はラベル名で，節の列 $(\delta, v), \cdots, (\delta_{m-1}, \delta_m v_{m-1})$ に対応するプログラム

path-$k\|\sigma$　(v_{m-1}, p_{m-1}, $\delta_m\sharp$, $\cdots$, v_1, p_1, $\delta\sharp$, $N@l'$)

を表わす．ここで l' は節 (δ, v) の番号，$\delta_i\sharp$ は δ_i が左回帰の生成規則をもつときのみ作られる．

4) 2章で作られたプログラムで出来なかった所の処理部に「スタックにこの部分を呼んだと言う印をスタックし，入力記号が I の元である限りそれを読み込んでスタックにつめる処理を行うプログラム」を表わす Stack&Read· とラベル名 $L@$ を '·' で区切って書く．

Table 1 の数式の構文に Table 6 の論理式の構文を追加した文法に対し，Fig. 1 のグラフを基に，上でのべた方法を用いて作られた処理プログラムを**Table 8** (前頁参照) にあげる．それを単純化したものを2章のアルゴリズムによって作られたプログラムに加えたものを**Table 9** (前頁参照) にあげる．

構文の所だけで処理しにくい関数と添字変数の区別，論理変数と算術変数の区別は意味情報を利用することにすれば，本節でのべた方法で FORTRAN の構文解析プログラムを作成することが出来る．

5. おわりに

拡張された LL(k) 文法に対し，能率のよい構文解析プログラム作成手法を提案した．この手法は i) 読みこまれた語の照合にはスタックが不用で，ii) 非終端記号が出てこないこと，iii) 先読みの語を除いて長さ1のものを照合すればよい，iv) 手続きが判りよい，などの特徴をもっている．更に，FORTRAN などに適用出来るよう，アルゴリズムを拡張したので，LL(k) の手法の見やすさを保存し，処理出来る言語のせまいことの欠点をなくし得たと考える．

参考文献

1) Lewis. P. M. and Stearns, R. E.: Syntax-directed transductions. J. ACM 15. 465～488 (1968).
2) Backes, S.: Top-down Syntax analysis and Floyd Evans Production language. IFIP 1971
3) 吉村，西野：能率のよい構文解析システムの自動作成について，昭 47 情報処理学会大会予稿
4) Hopcroft, J. E., Ullman J. D.: Formal Languages and Their Relation to Automata, Addison-Wesley, (1969)

(昭和 49 年 5 月 17 日受付)
(昭和 49 年 11 月 6 日再受付)