アルゴリズム 66－7
（1999. 1. 27）

LC構文解析法を拡張した構文解析法について

椎名 広光†, 増山 繁‡

† 岡山理科大学 〒700-0005 岡山県岡山市理大町 1-1
‡ 豊橋技術科学大学 〒441-8580 愛知県豊橋市天伯町雲雀ヶ丘 1-1

あらまし

計算可能な一般の句構造言語を構文解析する場合，文法の構造上文法記号間の交差がいくつも生じるため，入力列から作られる構文解析木は複雑になる．更に，構文解析の処理中においては，構文解析木の候補者や複数の構文解析木の生成も考慮しながら計算しなければならない．そのため，構文解析の処理中に予測した構文解析木が誤っている場合は，訂正のためにバックトラック処理が必要になり，解析時間が増大する．そこで本研究では，コンパイラなどで用いられている構文解析の中でも上昇型と下降型を組み合わせた混合型に分類されるLC構文解析法の先読み文字列の定義と生成規則の出力順を変更することで拡張を行なう．それによって，一般の句構造言語を構文解析する処理よりもバックトラック処理の回数を減少させ，また，LC文法で生成される言語よりも広い言語に対して構文解析が可能とことを示す．

キーワード LC 構文解析, LC 文法, 句構造文法, 文脈自由文法

The proposal of the extended LC parser

Hiromitsu Shiina†, Sigeru Masuyama‡

† Okayama University of Science
1-1, Ridai-cho, Okyama city, Okayama, 700-0005, Japan
‡ Toyohashi University of Technorogy
1-1, tempaku-cho, Toyohashi city, Aichi, 441-8580, Japan

abstract

When we parse recursive languages, the parse tree becomes complicated in structure. Furthermore, we must take into consideration the candidate of the parse tree and the generation of multiple parse trees in parsing processing. If the parser generates some wrong parse trees, then its backtracking process for correction is required. Therefore, in this paper, we propose an extension the LC parser as a mixed type of top-down and bottom-up parsing. We show that by using this parser the number of backtrack can be reduced and languages which is in the class larger than that of context-free languages can be proposed by changing the definition of lookahead strings.

keywords LC parser, LC grammar, phrase structure grammar, context-free grammar

1 はじめに

計算可能な一般の句構造言語を構文解析する場合，文法の構造上，入力列から作られる構文解析木には，文法記号の親が複数あって枝に交差を持つことになり，複雑になる．

更に，構文解析の処理中においては，構文解析木の候補者や複数の構文解析木生成も考慮しながら計算しなければならない．そのため，予測した構文解析木が誤っている場合は，訂正のためにバックトラック処理が必要になり，解析時間が増大する．

一方，自然言語処理などにおいては，文脈自由言語だけではなく，それより広いクラスに属する言語を処理する必要がある．しかしながら，自然言語の文法は，部分的に文脈自由文法や句構造文法の生成規則が含まれてはいるものの，ほとんどの生成規則が文脈自由文法のもので，構文解析時に最悪のケースばかりに対応していると処理時間が遅くなってしまう．

そこで本研究では，コンパイラなどで利用されている構文解析の中でも上昇型と下降型を組み合わせた混合型に分類される LC構文解析法 [8] を拡張する．それによって，予測構文解析木を作成しないで処理する方法と LC文法 [8] で生成される言語よりも広いクラスの言語を生成する文法を提案する．

なお，文脈自由文法 $G=(N,T,P,S)$，N: 非終端記号の集合，T: 終端記号の集合，P: 生成規則の集合，S: 開始記号 と入力文字列 $a_1a_2\ldots a_n\$$，$a_1,a_2,\ldots,a_n \in T$ を与える．また構文解析実行の便宜上，入力文字列の最後に終りを示す特殊記号\$ $(\notin N\cup T)$ を用意する．

2 準備

本章では，準備として，LC文法の定義および LC構文解析法について，簡単に述べる．なお，本論文における LC文法の定義は，文献 [8] に準拠する．

2.1 左隅出現でない非終端記号

最左導出の過程で得られる文法記号列である左文形式 $wA\delta$，$w\in T^*$，$A\in N$，$\delta\in(T\cup N)^*$，の最左位置の A が，生成規則の左隅の記号として導入されたものでない時，左隅でない非終端記号 A という．左隅出現でない左文形式 $wA\delta$の非終端記号 A を $S \overset{*}{\underset{\mathrm{lc}}{\Longrightarrow}} wA\delta$ と表す．

2.2 LC文法 [8]

次の条件を満たす文脈自由文法 G を $LC(k)$ 文法という．左隅出現でない任意の非終端記号 A，$S \overset{*}{\underset{\mathrm{lc}}{\Longrightarrow}} wA\delta$ と長さ kの先読み u に対して，いつも以下のような生成規則 $B\to\alpha$ が高々 1 個存在し，かつ $A \overset{*}{\Longrightarrow} B\gamma$で次のいずれかが成り立つ．

1. (i) $\alpha=C\beta$なら，u は $FIRST_k(\beta\gamma\delta)$ の要素である．
   (ii) さらに，$C=A$なら，uは $FIRST_k(\delta)$ の要素でない．
2. αが非終端記号で始まらないなら，u は $FIRST_k(\alpha\gamma\delta)$ の要素である．

ここで，$w \overset{*}{\Longrightarrow} w_1w_2\ldots w_n, w_1, w_2,\ldots, w_n\in T$ とするとき，$FIRST_k(w)$ は次のように定義される．

$$FIRST_k(w)=\begin{cases} w_1w_2\ldots w_k, & \text{if } n\geq k,\\ w_1w_2\ldots w_n, & \text{if } n<k.\end{cases}$$

2.3 LC構文解析法 [8]

LC構文解析は，上昇型と下降型を組み合わせた混合型に分類される構文解析法である．その解析の実行においては，木の第1部分木を下から上へ，第2部分木から最右部分木までそれぞれ上から下への走査を再帰的に行ないながら構文解析木を決定する．以下に，LC構文解析の動作を示す．

1. Tが葉のみの場合，
   - 訪問（根 (T)）
2. Tが葉のみでない場合，
   - LC走査 (第1部分木 (I));
   - 訪問 (根 (T));
   - 出力 (根 (T));
   - LC走査 (第2部分木 (I));

     ...
   - LC走査 (最右部分木 (I));

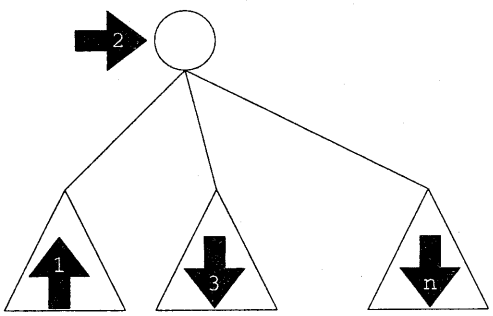

図 1: LC走査

例えば，$LC(1)$ 文法の例 $G = (N, T, P, S)$, $N = \{S, A, B\}$, $T = \{a, b, c\}$, $P = \{S \to AC,\ C \to CB,\ C \to B,\ A \to a,\ B \to b,\ B \to c\ \}$ に対して入力文字列を $I = abbb$ とする．この時，LC構文解析は，$(1)A \to a$, $(2)S \to AC$, $(3)B \to b$, $(4)C \to B$, $(5)B \to b$, $(6)C \to CB$, $(7)C \to CB$, $(8)B \to b$ の順で生成規則を当てはめて構文解析木を作成する (図2)．特に，(2) の $S \to AC$の後に (3) の $B \to b$ を当てはめるのが LC構文解析の特徴である．

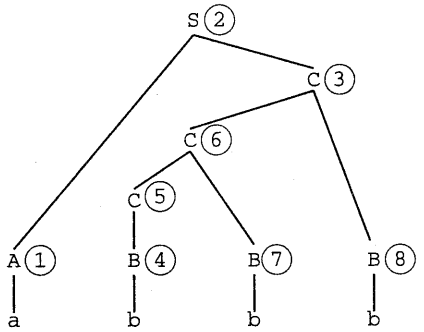

図 2: LC構文解析の例

3 LC構文解析の拡張 (ELC構文解析)

文脈自由言語以下のクラスの言語の場合，生成規則の矢印の左側が複数の文字列である条件$\alpha \to \beta \in P, |\alpha| > 1, |\beta| > 0$を満たす生成規則の適用を含まないため，構文解析木の兄弟は同じ入力文字を子供とすることはない．そのため，それぞれの文法記号から生成される部分文字列は文法の構造から決定できる．

一方，文脈自由言語より大きいクラスの言語に対する構文解析木には矢印の左側が複数文字列である生成規則の適用がされ，解析木中に図3に示すような交差をもつ（図3，$AB \to CDE$）．そのため，文脈自由言語より大きいクラスの言

語では，構文解析木の兄弟が同じ入力文字の先祖となる場合が存在する．

例えば，図3のCから生成される部分文字列とDから生成される部分文字列とは重複する部分があり，CとDは入力文字aの先祖である．

よって，Cを決定した時点でDを決定するためにDの先頭に引き戻す処理方法やEを決定した後DとCを決定する処理方法[10]をとらなければならない．

これらのどちらの処理方法もバックトラックの回数が多い．そこでバックトラックを避けるため，本研究では先読み文字列の定義と部分文字列の有効範囲を定義することによって，構文解析の左隅を決定するだけで生成規則が決定できる拡張LC構文解析(以下ELC構文解析)および$ELC(k)$文法を提案する．

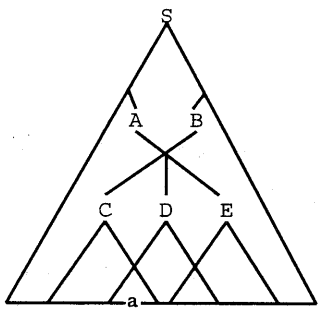

図3: 構文解析木の交差

3.1 文法定義の準備

先に，$ELC(k)$文法を定義するのに必要な用語を**第3.1.1節**と**第3.1.2節**で定義する．

3.1.1 最左導出

文脈自由文法における最左導出は，文形式中の最も左側に位置する非終端記号を書き換える導出であったが，文脈自由言語より広いクラスの言語を生成する文法では，非終端記号の代わりに書き換えられる文法記号列に変更する．そして，この最左導出の過程で得られる文法記号列である左文形式$wA\delta$の左隅出現でない左文形式$wA\delta$の非終端記号Aを$S \overset{*}{\underset{lc}{\Rightarrow}} wA\delta$と表す．

3.1.2 生成される文字列の範囲

非終端記号Aから生成される文字列を定義する．文脈自由文法では生成規則の矢印の左側は1つの文法記号であるので，Aから生成される文字列は文形式のAの両側の文字に影響されない．しかし，文脈自由文法より広い場合は生成規則の左側に複数の文字列がくる可能性があるので，文形式の両側の文字に影響される．そこで，始めに非終端記号に・をつけて，元々始めに・のついていた非終端記号から展開された文字列を明らかにする．例えば，$a_1\beta_1\dot{A}\beta_2\alpha_2$の$A$に関する展開を調べたい時は，記号$\overset{\cdot}{\Longrightarrow}$を用いて，次のように表す．

$$\alpha_1\beta_1\dot{A}\beta_2\alpha_2 \overset{\cdot}{\Longrightarrow} \alpha_3\dot{\beta}_3\dot{B}\dot{C}\dot{D}\dot{\beta}_4\alpha_4,$$
$$\beta_1 A\beta_2 \to \beta_3 BCD\beta_4 \in P,$$
$$\alpha_1,\alpha_2,\alpha_3,\alpha_4,\beta_1,\beta_2,\beta_3,\beta_4 \in (T\cup N)^*.$$

また，0回以上の展開の繰り返しを$\dot{A} \overset{*}{\underset{\cdot}{\Longrightarrow}} \dot{\gamma}, \gamma \in (T\cup N)^*$と表す．

すべての展開に対して，文形式中の文法記号Aから最左導出によって展開し，Aから生成された文字列の直後のk個の文字列を$FIRST'_k$で定義する．

$$FIRST'_k(A,\alpha_2) = \left\{ head_k(\gamma_2) \;\middle|\; \begin{array}{l} S \overset{*}{\Longrightarrow} \alpha_1 A \alpha_2 \overset{*}{\underset{\cdot}{\Longrightarrow}} \\ \gamma_1 \beta \gamma_2 \\ \alpha_1, \alpha_2, \beta \in (T \cup N)^* \\ \gamma_1, \gamma_2 \in T^* \end{array} \right\}$$

3.2 $ELC(k)$ 文法の定義

$ELC(k)$ 文法は生成規則の矢印の右側(木の左隅) の始めの文字が決定されると，その生成規則が適用される文法である．

任意の非終端記号 A, $S \overset{*}{\underset{lc}{\Longrightarrow}} wA\delta$ と長さ kの先読み $u \in T^*$に対して，以下のような生成規則$\alpha_1 \to \alpha_2$，$(\alpha_1\alpha_2 \in (T \cup N)^*)$，が常に高々 1 個存在し，かつ $A \overset{*}{\Longrightarrow} \alpha_1\gamma$，$(\gamma \in (T \cup N)^*)$，で次のいずれかが成り立つ．

1. (i) $\alpha_2 = C\beta$なら，uは$FIRST'_k(C, \beta\gamma\delta)$の要素である．
   (ii) さらに，$C = A$なら，uは$FIRST'_k(A, \delta)$の要素でない．
2. α_2が非終端記号で始まらないなら，uは$FIRST_k(\alpha\gamma\delta)$の要素である．
3. 終端記号を生成規則中に持つのは，$A \to a$の形式だけである．

ただし，3. は本質的な制約ではない．構文解析の処理において，終端記号の取り扱い手順を簡単にするための制約である．

3.3 解析表 LC

構文解析を進めていく上で，次に適用する生成規則を決定するのに，先頭記号，現在処理している文法記号，先読み文字列の引数が必要となる．それを表の形でまとめたのが解析表 LCである．解析表 LCは次の手順で作成する．

[解析表 LCの作成]

(i) $\alpha_2 = C\beta$の場合，uは$First'_k(C, \beta\gamma\delta)$の要素で $S \overset{*}{\underset{lc}{\Longrightarrow}} wA\delta$, $A \Longrightarrow \alpha_1\gamma$ なら

$LC(A, C, u) :=$ “$\alpha_1 \to \alpha_2$”とする．

(ii) α_2の先頭が終端記号で，その終端記号をaとする場合，uは$FIRST'_k(A, \beta\gamma\delta)$の要素で，$S \overset{*}{\underset{lc}{\Longrightarrow}} wA\delta$, $A \Longrightarrow \alpha_1\gamma$なら，$LC(A, a, u) :=$ “$\alpha_1 \to \alpha_2$ ”とする．

(iii) α_2の先頭が終端記号で，その終端記号をaとする場合，uは$FIRST_k(\delta)$の要素で，$S \overset{*}{\underset{lc}{\Longrightarrow}} wA\delta$, $A \Longrightarrow \alpha_1\gamma$ なら，$LC(A, a, u)$を未定義とする．

例えば，文法 $G = \{N, T, P, S\}$，$N = \{S, A, B, C, D\}$，$T = \{a, b, c\}$，$P = \{S \to ASCD, S \to ABD, DC \to CD, BC \to BB, A \to a, B \to b, D \to c\}$ は $ELC(1)$ の文法である．この文法に対する解析表 LCを表 1 に示す．

表 1: 解析表 LCの例

$LC(S, A, a) =$ “$S \to ASCD$”
$LC(S, A, b) =$ “$S \to ABD$”
$LC(S, a, b) =$ “$A \to a$”
$LC(S, a, a) =$ “$A \to a$”
$LC(S, b, b) =$ “$B \to b$”
$LC(B, b, b) =$ “$B \to b$”
$LC(B, b, c) =$ “$B \to b$”
$LC(B, B, b) =$ “$BC \to BB$”
$LC(D, C, c) =$ “$DC \to CD$”
$LC(D, c, c) =$ “$D \to c$”
$LC(D, c, \$) =$ “$D \to c$”

4 $ELC(k)$ 構文解析の手順

構文解析の処理ではinput_stack, ancestor_stack, closure_stackの3つのスタックを利用する. 以下にそれぞれのデータ構造を定義する.

1. input_stackは, 入力文字列及び生成規則の左側の文字列を積むスタック. input_stack= $(T \cup N) \times (T \cup N) \times \cdots \times (T \cup N)$.

2. ancestor_stackは, 先祖と予測される文法記号を積むスタック. ancestor_stack= $N \times N \times \cdots \times N$.

3. closure_stackは, 解析が終了した位置を表すために生成規則に・をつけた項を積むスタック. closure_stack= $([\alpha_1 \rightarrow \alpha_2 \cdot \alpha_3] \times [\alpha_4 \rightarrow \alpha_5 \cdot \alpha_6] \times \cdots \times [\alpha_7 \rightarrow \alpha_8 \cdot \alpha_9])$, $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7 \in (T \cup N)^*$.

次に構文解析の手順を示す.

Step 1: ancestor_stackに開始記号 Sを積む.

Step 2: input_stackの先頭が a_i, 先読みが $a_{i+1} \ldots a_{i+k+1}$, ancestor_stackの先頭を Hとする.

Step 2.1: もし $LC(H, a_i, a_{i+1} \ldots a_{i+k+1})$ = "$A \rightarrow a_i$"なら

Step 2.1.1: input_stackから a_iを降ろす.

Step 2.1.2: "$A \rightarrow a_i$"を出力する.

Step 2.1.3: A をinput_stackに積む.

Step 3: input_stackの先頭を非終端記号 A, 先読みを $a_{i+1} \ldots a_{i+k+1}$, ancestor_stackの先頭を Hとする.

Step 3.1: もしclosure_stackの先頭が $[\alpha_1 \rightarrow \alpha_2 \cdot AB\alpha_3]$ なら

Step 3.1.1: closure_stackの先頭 $[\alpha_1 \rightarrow \alpha_2 \cdot AB\alpha_3]$ を $[\alpha_1 \rightarrow \alpha_2 A \cdot B\alpha_3]$ に入れ換える.

Step 3.1.2: ancestor_stackに B を積む.

Step 3.1.3: input_stackから A を降ろす.

Step 3.2: もしclosure_stackの先頭が $[\alpha_1 \rightarrow \alpha_2 \cdot A]$ なら

Step 3.2.1: $\alpha_1 \rightarrow \alpha_2 A$ を出力する.

Step 3.2.2: closure_stackの先頭 $[\alpha_1 \rightarrow \alpha_2 \cdot A]$ を降ろす.

Step 3.2.3: input_stackから A を降ろし, α_1の逆順で積む.

Step 3.2.4: ancestor_stackから1つ降ろす.

Step 3.3: もし $LC(H, A, a_{i+1}a_{i+2} \ldots a_{i+k+1})$ = "$\alpha_1 \rightarrow A\alpha_2$"なら

Step 3.3.1: closure_stackに $[\alpha_1 \rightarrow A \cdot \alpha_2]$ を積む.

Step 3.3.2: ancestor_stackにα_2の先頭の文字を積む.

Step 3.4: それ以外はエラーを出力して終了する.

Step 4: **Step 2**から**Step 3**を繰り返す. input_stackの内容が $S\$$, closure_stackが空であるなら, "accept"を出力して終了する.

5 構文解析の動作例

入力を $I = aabbcc$ とした時の構文解析の動作を表2に，また作成される構文解析木を図5に示す．

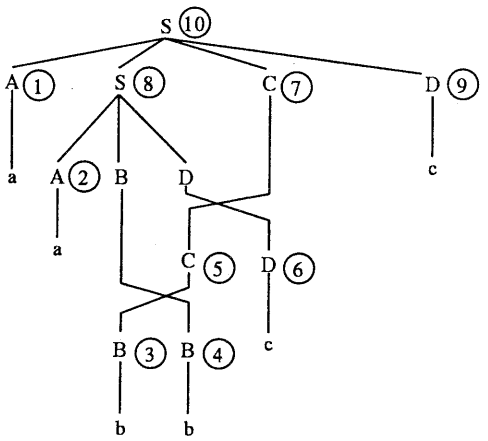

図4: 入力 $I = aabbcc$ に対する構文解析木

6 むすび

本研究では，構文解析木の文法記号が支配する文字列の記号の範囲を定義することにより，先読み文字列の定義を拡張した．それにより，LC構文解析法を文脈自由言語のクラスより大きいクラスに属する言語も構文解析ができる $ELC(k)$ 構文解析法に拡張し，また $ELC(k)$ 文法を提案した．今後は，$ELC(k)$ 文法の性質を調査する予定である．

表2: ELC構文解析の動作例

	input	ancestor	closure	action
1	$aabbcc\$$	S		$LC(S,a,a) =$ "$A \to a$" $A \to a$ を出力 ①
2	$Aabbcc\$$	S		$LC(S,A,a) =$ "$S \to ASCD$"
3	$abbcc\$$	SS	$[S \to A \cdot SCD]$	$LC(S,a,b) =$ "$A \to a$" $A \to a$ を出力 ②
4	$Abbcc\$$	SS	$[S \to A \cdot SCD]$	$LC(S,A,b) =$ "$S \to ABD$"
5	$bbcc\$$	BSS	$[S \to A \cdot BD]$ $[S \to A \cdot SCD]$	$LC(B,b,b) =$ "$B \to b$" $B \to b$ を出力 ③
6	$Bbcc\$$	BSS	$[S \to A \cdot BD]$ $[S \to A \cdot SCD]$	$LC(B,B,b) =$ "$BC \to BB$"
7	$bcc\$$	BBSS	$[S \to A \cdot BD]$ $[S \to A \cdot SCD]$	$LC(B,b,c) =$ "$B \to b$" $B \to b$ を出力 ④
8	$Bcc\$$	BSS	$[BC \to B \cdot B]$ $[S \to A \cdot BD]$ $[S \to A \cdot SCD]$	$BC \to BB$を出力 ⑤
9	$BCcc\$$	SS	$[S \to A \cdot BD]$ $[S \to A \cdot SCD]$	
10	$Ccc\$$	DSS	$[S \to AB \cdot D]$ $[S \to A \cdot SCD]$	$LC(D,C,c) =$ "$DC \to CD$"
11	$cc\$$	DDSS	$[S \to AB \cdot D]$ $[S \to A \cdot SCD]$	$LC(D,c,c) =$ "$D \to c$" $D \to c$ を出力 ⑥
12	$Dc\$$	DSS	$[DC \to C \cdot D]$ $[S \to AB \cdot D]$ $[S \to A \cdot SCD]$	$DC \to CD$を出力 ⑦
13	$DCc\$$	SS	$[S \to AB \cdot D]$ $[S \to A \cdot SCD]$	$S \to ABD$を出力 ⑧
14	$SCc\$$	S	$[S \to A \cdot SCD]$	
15	$Cc\$$	S	$[S \to AS \cdot CD]$	
16	$c\$$	DS	$[S \to ASC \cdot D]$	$LC(D,c,\$) =$ "$D \to c$" $D \to c$ を出力 ⑨
17	$D\$$	DS	$[S \to ASC \cdot D]$	$S \to ASCD$を出力 ⑩
18	$S\$$			accept

参考文献

[1] S. Y. Kuroda, *Classes of languages and linear-bounded automata.* Inform. Control, 7, pp207-223, 1967.

[2] J. Earley, *An efficient context-free parsing algorithm*, Commun. ACM, 13, 2, pp94-102, 1970.

[3] J. Loeckx, *The parsing for general phrase-structure grammars*, Inform. Control, 16, pp443-464, 1970.

[4] G. Sh. Vold'man, *A parsing algorithm for context-sensiteve grammars*, Program. Comput. Software, 7, pp302-307, 1981.

[5] L.A. Harris, *SLR(1) and LALR(1) parsing for unrestricted grammar*, Acta Inform., 24, pp191-209, 1987.

[6] M. Tomita, *An Efficient Augmented -Context-Free Parsing Algorithm*, Computational Linguistics, 13,1-2, pp31-46, 1987.

[7] M. Tomita ed, *Generalized LR parsing*, Kluwer Academic Publisher, 1991.

[8] 徳田 雄洋, 言語の構文解析, 共立出版, 1995.

[9] K. Sikkel, *Parsing Schemata*, Springer, 1997.

[10] H. Shiina and S. Masuyama, *Proposal of the unrestricted LR(k) grammar and its parser*, Mathematica Japonica, 46, 1, pp129-142, 1997.