数理解析研究所講究録
1093 巻 1999 年 230-235

LC文法とその構文解析法の拡張について

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

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, 複数個の文法記号を親に持つ生成規則を含むため, 親が1つに限られない形態の構文解析木を持つ. 更に, 構文解析の実行においては, 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形式をしている. そのため, 構文解析時に最悪のケースのみ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法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, 予測した構文解析木を作成しないで処理する解析方法と 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, 及び, 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, 生成規則の矢印の右側の先頭の文字でない時, 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 $S \underset{\mathrm{lc}}{\overset{*}{\Longrightarrow}} wA\delta$ 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, $S \underset{\mathrm{lc}}{\overset{*}{\Longrightarrow}} wA\delta$ と長さ kの先読み文字列 u に対して, 常に以下のような生成規則 $B\rightarrow\alpha$ 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) $\alpha=C\beta$, $C\in N$, なら, uは $FIRST_k(\beta\gamma\delta)$ の要素である.
   (ii) さらに, $C=A$, $C\in N$, 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, $FIRST_k(w)$ は次のように定義する.

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

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部分木は下から上へ，第2部分木から最右部分木までは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 (図1)．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)).

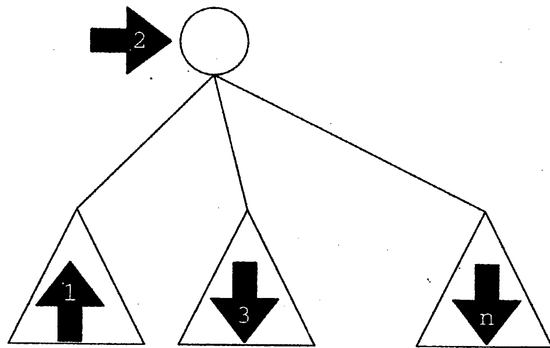

図 1: *LC*走査

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)$B \to b$ の順に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．

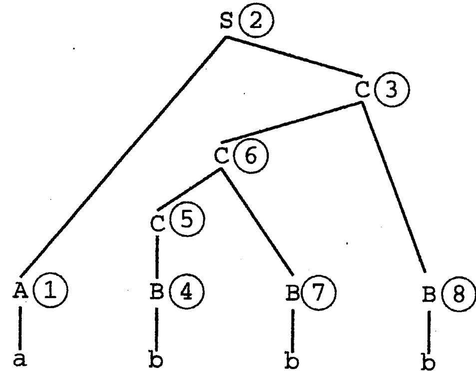

図 2: *LC*構文解析の例

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生成規則$\alpha \to \beta \in P$，$|\alpha| > 1$，$|\beta| > 0$ 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，構文解析木中に図3に示すような交差をもつ（図3，$AB \to CDE$）．そのため，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．

これらのいずれ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を拡張し，更に，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．

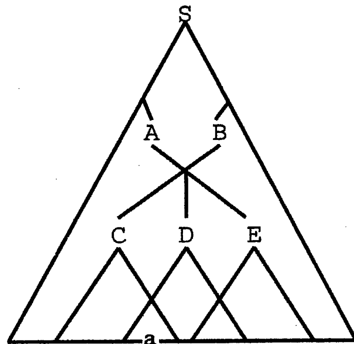

図 3: 構文解析木の交差

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，本稿では，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 $S \overset{*}{\underset{\mathrm{lc}}{\Longrightarrow}} wA\delta$ 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，生成する言語が文脈自由文法より広いクラスに属する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，始めに非終端記号の上に・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，$\alpha_1\beta_1\dot{A}\beta_2\alpha_2$の A に関する展開を調べたい時は，記号 $\underset{\cdot}{\Longrightarrow}$ を用いて，次のように表す.

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

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.

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

3.2 $ELC(k)$ 文法の定義

$ELC(k)$ 文法は生成規則の矢印の右側の始めの文字 (木の左隅) 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 $A \in N$，$S \overset{*}{\underset{\mathrm{lc}}{\Longrightarrow}} wA\delta$ 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$\alpha_1 \rightarrow \alpha_2$，$(\alpha_1\alpha_2 \in (T \cup N)^*)$，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) $\alpha_2 = C\beta$，$C \in N$，$\beta \in (T \cup N)^*$ なら，u は $FIRST'_k(C,\beta\gamma\delta)$ の要素である.
   (ii) さらに，$C = A$，$C \in N$，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 ($FIRST_k$は，**2.2** の定義参照).
3. 終端記号を生成規則中に持つのは，$A \rightarrow a$ の形式だけである.

ただし，3. は構文解析において，終端記号の取り扱い手順を簡単にするための制約であり，$ELC(k)$ 文法から生成される言語の大きさに関して本質的な制約ではない.

3.3 解析表 ELC

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，先頭記号と現在処理してい

る文法記号と先読み文字列の3つ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 ELC である．解析表 ELC は次の手順で作成する．

[解析表 ELC の作成]

(i) $\alpha_2 = C\beta$ の場合，u は $FIRST'_k(C,\beta\gamma\delta)$ の要素で $S \underset{lc}{\overset{*}{\Longrightarrow}} wA\delta$，$A \Longrightarrow \alpha_1\gamma$ なら $LC(A,C,u) :=$ "$\alpha_1 \rightarrow \alpha_2$" 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，$S \underset{lc}{\overset{*}{\Longrightarrow}} wA\delta$，$A \Longrightarrow \alpha_1\gamma$ なら，$LC(A,a,u) :=$ "$\alpha_1 \rightarrow \alpha_2$" 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，$S \underset{lc}{\overset{*}{\Longrightarrow}} wA\delta$，$A \Longrightarrow \alpha_1\gamma$ なら，$LC(A,a,u)$ を未定義とする．

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

表1: 解析表 ELC の例

$ELC(S,A,a) =$ "$S \rightarrow ASCD$"
$ELC(S,A,b) =$ "$S \rightarrow ABD$"
$ELC(S,a,b) =$ "$A \rightarrow a$"
$ELC(S,a,a) =$ "$A \rightarrow a$"
$ELC(S,b,b) =$ "$B \rightarrow b$"
$ELC(B,b,b) =$ "$B \rightarrow b$"
$ELC(B,b,c) =$ "$B \rightarrow b$"
$ELC(B,B,b) =$ "$BC \rightarrow BB$"
$ELC(D,C,c) =$ "$DC \rightarrow CD$"
$ELC(D,c,c) =$ "$D \rightarrow c$"
$ELC(D,c,\$) =$ "$D \rightarrow c$"

4 $ELC(k)$ 構文解析の手順

LC 構文解析器は，入力文字列をテープ (入力テープ)，左隅文字を積むスタックと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 (以下，LC 項) を積むスタックの3種類から構成される．それに対して，本稿で示す ELC 構文解析では，生成した生成規則の矢印の左側の文字列を再び入力として用いる．そのため，ELC 構文解析器は，入力テープの代わりとなる入力スタックと左隅文字と LC 項を積む3種類のスタック (input_stack, ancestor_stack, closure_stack) から構成される．以下にそれぞれのデータ構造を定義する．なお，LC 項は $[\alpha_1 \rightarrow \alpha_2 \cdot \alpha_3]$，$\alpha_1$，$\alpha_2$，$\alpha_3 \in (T \cup N)^*$ で定義され，LC 項の集合を I_{LC} とする．

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，左隅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，LC 項を積むスタック．
   closure_stack= $I_{LC} \times I_{LC} \times \ldots \times I_{LC}$．

次に構文解析の手順を示す．

Step 1(初期設定): ancestor_stack に開始記号 S を積む．

Step 2(終端記号処理): 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 $ELC(H, a_i, a_{i+1} \ldots a_{i+k+1})$ = "$A \rightarrow a_i$" 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(非終端記号処理): 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(LC項処理):
もし closure_stack の先頭が $[\alpha_1 \to \alpha_2 \cdot AB\alpha_3]$ なら,

Step 3.1.1: closure_stack の先頭 $[\alpha_1 \to \alpha_2 \cdot AB\alpha_3]$ を $[\alpha_1 \to \alpha_2 A \cdot B\alpha_3]$ 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(生成規則出力):
もし closure_stack の先頭が $[\alpha_1 \to \alpha_2 \cdot A]$ なら,

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

Step 3.2.2: closure_stack の先頭 $[\alpha_1 \to \alpha_2 \cdot A]$ 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 から先頭の Hを 1 つ降ろす.

Step 3.3(LC項の新規追加):
もし $ELC(H, A, a_{i+1}a_{i+2}\dots a_{i+k+1})$ = "$\alpha_1 \to A\alpha_2$"なら,

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

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

Step 3.4(エラー処理): それ以外はエラーを出力して終了する.

Step 4(終了処理):
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 ELC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 4に示す.

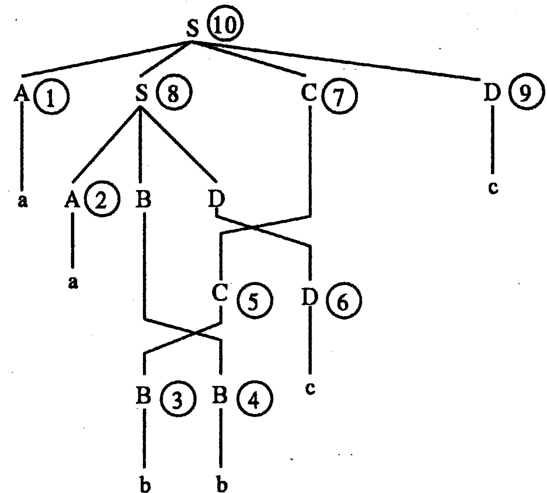

図 4: 入力 $I = aabbcc$ に対する構文解析木

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