1Q-5

線形インデックス文法の上向き構文解析法

夏目 雄介† 中村 克彦‡

†東京電機大学理工学部 ‡東京電機大学理工学部

1 まえがき

Aho [1] により提案された線形インデックス文法（LIG: linear indexed grammar）はプッシュダウンオートマトン（PDA : push down automaton）の拡張であり，弱文脈依存文法と呼ばれるクラスのなかで木接合文法（TAG : tree adjoining grammar）と等価であるという特長をもっている．LIG によって，文脈自由言語（CFL : context free language) に加えてコピー言語 $\{ww \mid w = (a \cup b)^*,\ a, b \in T\}$ や $\{a^n b^n c^n \mid a, b, c \in T, n > 0\}$ などの非文脈自由言語 (弱文脈依存言語) を導出することができる．

われわれは正負の例から LIG の学習をおこなうために LIG の構文解析法について調べている．下向きの構文解析では，非決定性により一般に指数時間を要してしまうが，上向きの構文解析では多項式時間により構文解析を行うことができる．この報告では，PDA と LIG の上向き構文解析法について述べる．

2 PDA と LIG

LIG は $M = (Q, T, \Gamma, P, f, s)$ で表わされる．ここで，Q は状態の有限集合，T は入力記号の有限集合，Γ はスタック記号の有限集合，P は以下に示す形式の規則の有限集合 ($p,\ q \in Q,\ \sigma \in \Gamma,\ a \in T$)，$f \in Q$ は受理状態，$s \in Q$ は初期状態である．

1. $p \to a\ q[\sigma]$,
2. $p \to q[\sigma]\ a$,
3. $p \to a\ q$,
4. $p \to q\ a$,
5. $p[\sigma] \to a\ q$,
6. $p[\sigma] \to q\ a$.

PDA の規則は，上記 LIG の規則形式の 1, 3, 5 の 3 種類 (以下 “ f 型 ” と略) に制限したものである．

以下，規則 1, 2 を “ push 規則 ”，規則 3, 4 を “ 推移規則 ”，規則 5, 6 を “ pop 規則 ” と呼ぶ．また，“ f 型 ” に対し 2, 4, 6 の規則を “ r 型 ” と呼び，これらを組み合わせて 1 の規則を “ f - push 規則 ” のように呼ぶ．

次に，LIG が言語をどのように導出（受理）するかを非文脈自由言語の例によって示す．

例 1 : コピー言語 : この言語は次の 6 つの規則により導出される．ただし，初期状態は s，受理状態は p である．

- 規則 : $s \to a\,s[a],\ s \to b\,s\,[b],\ s\,[a] \to p\,a.$
  $s\,[b] \to p\,b,\ p\,[a] \to p\,a,\ p\,[b] \to p\,b.$
- 導出の例 : $s\,[\,] \Rightarrow a\,s\,[a] \Rightarrow aa\,s\,[a, a]$
  $\Rightarrow aab\,s\,[b, a, a] \Rightarrow aab\,p\,[a, a]\,b$
  $\Rightarrow aab\,p\,[a] \Rightarrow aab\,p\,[\,]\,aab \Rightarrow aabaab.$

例 2 : $\{a^n b^n c^n \mid n \geq 1\}$: この言語は次の 4 つの規則により導出される．ただし，初期状態は s，受理状態は q である．

- 規則 : $s \to a\,p\,[a],\ s\,[a] \to b\,q,\ p \to s\,c,\ q\,[a] \to b\,q.$
- 導出の例 : $s\,[\,] \Rightarrow a\,p\,[a] \Rightarrow a\,s\,[a]\,c \Rightarrow aa\,p\,[a, a]\,c$
  $\Rightarrow aa\,s\,[a, a]\,cc \Rightarrow aab\,q\,[a]\,cc$
  $\Rightarrow aabb\,q\,[\,]\,cc \Rightarrow aabbcc.$

3 LIG の上向き構文解析

CFG に対する CYK アルゴリズムを基礎として，まず PDA の上向き構文解析アルゴリズムを考える．PDA では，現在の状態とスタックのトップにある記号により非決定的に状態を推移する．PDA が記号列を導出 (受理) するときスタックは空でなければならないため，導出過程でスタックに push された記号は pop されることとなる．ある push 規則とある pop 規則で扱うスタック記号が同じ場合，その規則対に注目する．

PDA では，記号列 $a_1 a_2 \ldots a_n$ $(n \geq 1)$ は a_1 から a_n の順に導出される．しかし，LIG では $a_1 \ldots a_j$ は f 型規則，$a_{j+1} \ldots a_n$ は r 型規則で導出され，j の値を知ることはできない．また，$a_{j+1} \ldots a_n$ の各文字が $a_1 \ldots a_j$ のどの文字の次に導出されるのかも知ることができない．

PDA の上向き構文解析アルゴリズムを基礎とし LIG の上向き構文解析アルゴリズムを考える．LIG を上向き

Bottom-up Parsing of Linear Indexed Grammars

Yusuke Natsume†, Katsuhiko Nakamura‡

†School of Science and Engineering, Tokyo Denki University

‡School of Science and Engineering, Tokyo Denki University

†09rd152@ms.dendai.ac.jp, ‡nakamura@rd.dendai.ac.jp

に構文解析するため，r 型規則で導出される記号列の情報を $a_i \ldots a_j$ を導出する状態に補足する形式で取り扱う．ここで，LIG は対称的な規則で表されるので，扱う文法に r - push 規則はないと仮定している．PDA の上向き構文解析アルゴリズムを図 1 に，$\{a^n b^n c^n\}$ において $n = 2$ の場合の構文解析の様子を表す導出木を図 2 に，LIG の構文解析アルゴリズムを図 3 に示す．図 2 は，a_1の次にr 型の規則によって c_6，a_2の次にr 型の規則によって c_5 が導出され，$(s-q) \stackrel{*}{\Rightarrow} a_2 b_3, c_5$ であり，その前後の規則が対であることから $(s-q) \stackrel{*}{\Rightarrow} a_1 a_2 b_3 b_4, c_5 c_6$ であることを表している．

[**入力**] 記号列 $a_1 a_2 \ldots a_n \ \ (n \geq 1)$.
[**変数**] T : $n \times n$ の次の要素 $T_{i,j} (i \leq j)$ をもつ 2 次元配列. ここで $T_{i,j}$ は $(p-q) \stackrel{*}{\Rightarrow} a_i \ldots a_j$ なる状態の対 $(p-q)$ の集合である.
Step 1 : $T_{i,i} = \{ \ (p-q) \mid p \rightarrow a_i \ q \ \}$;
Step 2 : $p \rightarrow a_i \ q[\alpha] \cap q[\alpha] \rightarrow a_j \ r$ ならば,
$T_{i,i+1} = \{ \ (p-r) \ \}$;
Step 3 : 下の条件 1 または 2 を満たす場合,
$T_{i,j} = \{ \ (p-r) \ \}$;
1. $(p-q) \in T_{i,k} \cap (q-r) \in T_{k+1,j}$
2. $p \rightarrow a_i \ q[\alpha] \cap (q-q') \in T_{i+1,j-1} \cap$
$q'[\alpha] \rightarrow a_j \ r$

Step 4 : $(s-f) \in T_{1,n}$ ならば成功，それ以外は失敗.

図 1 : PDA の上向き構文解析アルゴリズム

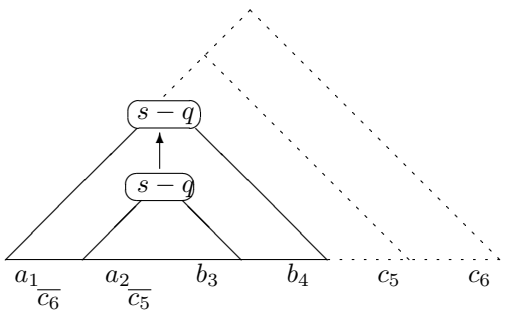

図 2 : *aabbcc* を構文解析する導出木

[**入力**] 記号列 $a_1 a_2 \ldots a_n \ \ (n \geq 1)$.
[**変数**] T : $n \times n$ の要素 $T_{i,j}$ をもつ 2 次元配列. ここで $T_{i,j} (i \leq j)$ は次のような状態の対 $(p-q)$ の集合である.
$(l, f, (p-q)) \stackrel{*}{\Rightarrow} a_i \ldots a_j, a_{l+1} \ldots a_x (x \leq n)$
$(n, f, (p-q)) \stackrel{*}{\Rightarrow} a_i \ldots a_j, \epsilon$
$(n, r, (p-q)) \stackrel{*}{\Rightarrow} \epsilon, a_i \ldots a_j$
Step 1 : $T_{i,i} = \{ \ (p-q) \mid p \rightarrow a_i \ q \ \}$;
$T_{i,i} = \{ \ (p-q) \mid p \rightarrow q \ a_i \ \}$;

Step 2 : $(p-q) \stackrel{*}{\Rightarrow} a_i \cap (q-r) \stackrel{*}{\Rightarrow} \epsilon, a_{j+1}$ ならば,
$T_{i,i} = \{ \ (j, \ f, \ (p-r) \ ) \ \}$;

Step 3 : 図 1 Step 2, 3 に示した条件，または下の条件 1 または 2 を満たす場合,
$T_{i,j} = \{ \ (l, \ f, \ (p-r) \ ) \ \}$;
1. $(p-q) \stackrel{*}{\Rightarrow} a_i \ldots a_k, a_{x'+1} \ldots a_x \cap$
$(q-r) \stackrel{*}{\Rightarrow} a_{k+1} \ldots a_j, a_{l+1} \ldots a_{x'}$
2. $p \rightarrow a_i \ q[\alpha] \cap q'[\alpha] \rightarrow r \ a_{l+1} \cap$
$(q-q') \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_j, a_{l+2} \ldots a_x$

Step 4 : $A \in T_{1,flen}$, $(s-f) \in A$,
$(s-f) \stackrel{*}{\Rightarrow} a_1 \ldots a_{flen}, a_{flen+1} \ldots a_n$
ならば成功，それ以外は失敗.

図 3 : LIG の上向き構文解析アルゴリズム

4 むすび

CYK アルゴリズムを基礎とした PDA および LIG の上向き構文解析について述べた．この方法によって，すでに提案された方法 [2] よりも簡潔に構文解析をおこなうことができる．

残された課題は，この構文解析法を用いて LIG の文法推論 [3] をさらに改良することである．

参考文献

[1] A. V. Aho, Indexed grammars: An extension of context-free grammars , 1968

[2] M. A. Alonso, Relating Tabular Parsing Algorithm for LIG and TAG, 2005

[3] Keita Imada and Katsuhiko Nakamura, Towards Machine Learning of Grammars and Compilers of Programming Languages, 2008