Vol. 27 No. 1 情報処理学会論文誌 Jan. 1986

ショートノート

正規右辺文法のLRパーサの簡単な実現法†

佐々政孝†† 中田育男††

正規右辺文法とは，生成規則の右辺に文法記号の正規表現を許すような文脈自由文法のことである．本稿では，これに対するLRパーサを簡単に構成する方法を提案する．その基本は，構文解析スタックと並行に，生成規則の右辺から生成される記号の列の長さをカウントするためのスタックを設けるものである．この方法は，構文解析の効率は最良ではないが，パーサの作成が簡単で，通常のLRパーサに対する方法を若干精密化するだけですみ，作成時に文法の変換やlookback状態[6]等の計算が不要であるという特徴をもつ．

1. はじめに

正規右辺文法（または拡張文脈自由文法）とは，文脈自由文法の生成規則の右辺の記述に文法記号の正規表現を許したものである[1)~7)]．これは，プログラミング言語の構文を自然に表現するのに有効であり，Pascalの規格などで使われている．

正規右辺文法は，k記号の先読みにより左から右へほぼ線形時間でLR構文解析できるとき，ELR (k) 文法であると言われる．（以下では $k=1$ の場合を扱う．またELR(1)文法，LR(1)文法/パーサのことを単にELR文法，LR文法/パーサと呼ぶこともある．）

ELR文法のLR構文解析で問題となるのは，生成規則の右辺を認識したときの還元動作である．その理由は，通常のLR文法では，生成規則の右辺の長さがあらかじめ定まっているのに対し，ELR文法では生成規則の右辺の正規表現で生成された文形式の長さが一般には定まっていないからである．

ELR文法のLR構文解析法として，これまでに提案されている方法には大きく分けて三つある．

（1） ELR文法を等価なLR文法に変換してから通常の方法でLRパーサを作成する[2),3)]．

（2） ELR文法のまま，直接LRパーサを作成する[2の別法),4)~6)]．

（3） （2）の方法で処理できない部分だけ，別のELR文法に変換してから，直接LRパーサを作成する[7)]．

（1）や（3）の方法では，変換された文法に新しい非終端記号が追加され，もとの文法に比べて複雑な文法構造となる．コンパイラの生成に適用する際は意味規則との整合性が崩れる．これらの点から，（2）の方法が望ましい．

本稿では，（2）の分類に属する一つの方法として，構文解析用のスタックと並行に，生成規則の右辺の長さをカウントするためのスタックを設けることにより，通常のLR構文解析の方法を若干精密化するだけでLRパーサを作成できる簡単な方法を提案する．

2. 本方法の概要

本方法の概要を，次の文法[4),7)]により説明する．

$G1$: #0: $S' \rightarrow S\,\$$
#1: $S \rightarrow \{a\}\ b$
#2: $S \rightarrow a\ A\ c$
#3: $A \rightarrow \{a\}$

（$\{a\}$ は，a の0回以上の繰り返しを表す．）

入力として「$a\ a\ a\ b\ \$$」（入力1）が与えられたとしよう．これは，

$$S' \overset{\#0}{\Rightarrow} S\,\$ \overset{\#1}{\Rightarrow} \{a\}\,b\,\$$$

により生成される．この入力を構文解析するには，各入力信号が，対応する生成規則の右辺の先頭から数えて何番目に当たるかをカウントしていけばよい．通常の解析スタックのほかに，カウントスタックというスタックを設け，そこにカウント値を覚えておくと，

解析スタック　$a\ a\ a\ b$　残り入力 $\$$
カウントスタック　1 2 3 4

となる．ここで生成規則#1により還元することになるが，カウントスタックのトップにある個数(4)だけ

† A Simple Realization of LR Parsers for Regular Right Part Grammars by MASATAKA SASSA and IKUO NAKATA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学電子・情報工学系

解析スタックからポップして還元すればよい．

一方，入力が「$a\ a\ a\ c\ \$$」（入力２）であったとする．これは

$$S' \overset{\#0}{\Rightarrow} S\$ \overset{\#2}{\Rightarrow} aAc \overset{\#3}{\Rightarrow} a\{a\}c\,\$$$

によって生成されたものであるから，構文解析では

$$\underbrace{a\ a}_{A}\ \ldots$$

```
a a a c $
  |_|
   ↓
   A
```

と還元しなければならない．つまり，入力２でのカウント値は入力１と異なり，

解析スタック	$a\ a\ a$	残り入力	$c\$$
カウントスタック	1 1 2		

となっていかなければならない．入力１も入力２も「$a\ a\ a$」を読んでいる間は区別がつかないので，カウント値に場合分けが必要となり，カウントスタックを必要な本数だけ用意することにする．この例では

解析スタック	$a\ a\ a$	残り入力	$b\,\$$
カウントスタック１	1 2 3		または
カウントスタック２	1 1 2		$c\,\$$

とする（カウントスタックの本数が組合せ的に増大する心配がないことは後述する）．

次の入力記号が c であれば，この時点で生成規則 #3 により還元すべきであることがわかる．#3 に対応するカウント値は，カウントスタック２の方のトップにあるので，二つポップをして還元すればよい．

3. LR パーサの構成法

ELR 文法に対する LR パーサは，通常の LR パーサ[8] と同様に LR 項や LR 状態を作成して構成できる．詳細な定式化は文献 7),9) などと同様に行えるが，以下では主に例を用いて説明する．

まず，ELR 文法の慣例にならい，生成規則の右辺の正規表現を，それに対応する有限状態オートマトン（fsa）で置き換えて表す（本稿では決定性有限状態オートマトンとする）．文法 $G1$ に対するものを**図１**に示す．

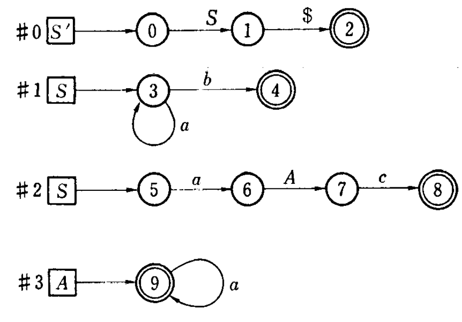


図 1 正規右辺文法 $G1$ の有限状態オートマトン表現
Fig. 1 Representation of regular right part grammar $G1$ by fsa.

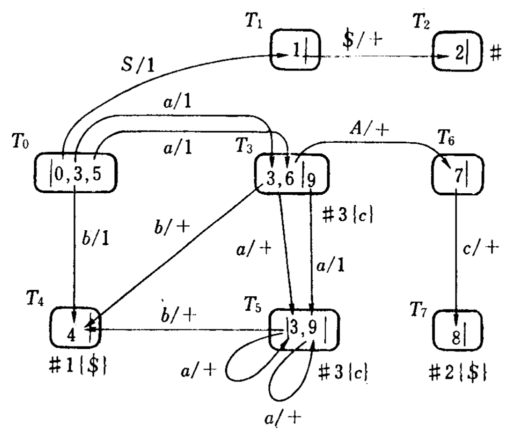


図 2 $G1$ の LR オートマトン
Fig. 2 LR automaton of $G1$.

同様にして，LR 項も fsa の状態に対応させて表す．たとえば，LR 項

$$S \to \cdot\ \{a\}b$$

は，LR マーカ「・」が図１の３の位置にあるので「３」と表す（ここでは説明の便宜のため，LR 項のうち先読み部分を除いた core[8] を示す）．

ELR 文法の LR 状態は，通常のものと同様に LR 項の閉包（closure）として定義される．たとえば，

$$S \to \cdot\ \{a\}b \text{ と}$$
$$S \to a \cdot Ac$$

がカーネル（kernel）に入っていたとするとき，閉包をとると

$$A \to \cdot\ \{a\}$$

が非カーネルとしてこの LR 状態に入る．これらも fsa の状態を用いて

$$\{3,\ 6\,|\,9\}$$

と表す．ここで，LR 状態のカーネルと非カーネルの間を「｜」で区切って示すことにする．

次に，通常どおり GOTO 関係と閉包演算を用いて，LR オートマトンを作成する．文法 $G1$ に対する LR オートマトンを**図２**に示す．

ここで，GOTO 関係は，通常，LR 状態から LR 状態への遷移関係を示すが，カウントの場合分けを扱えるよう，これを精密化した GOTO１ という関係を導入する．GOTO１ 関係は，（LR 状態，LR 項）の組から（LR 状態，LR 項）の組への遷移関係を示し，さらに出所がカーネル内か否かの情報をふくんだもの

である．たとえば，LR 状態

{3，6｜9}（図2の T_3）

から a による遷移をすると，カーネル内の LR 項（T_3, 3）が（T_5, 3）に，非カーネル内の LR 項（T_3, 9）が（T_5, 9）に移るが，これを，

$$(T_3,\ 3) \xrightarrow{a/+} (T_5,\ 3)$$

$$(T_3,\ 9) \xrightarrow{a/1} (T_5,\ 9)$$

のように表す．ここで，GOTO 1 関係

$\xrightarrow{X/1}$ と $\xrightarrow{X/+}$

の違いは，X による遷移の際に，対応するカウント値を1に初期化するか，あるいは，1だけ増やすかの違いである．カウントの初期化は，右辺の先頭の記号を読むときに行えばよいので，非カーネル内の LR 項から遷移するときに行う．

GOTO 1 関係は詳しくは次のように定義される．

T_i, T_j が LR 状態，q_k, q_l が LR 項，X が記号で，(T_i, q_k) が X により (T_j, q_l) に遷移するとき，

- q_k が T_i のカーネル内にあるならば，

  GOTO 1$((T_i,\ q_k),\ X/+)=(T_j,\ q_l)$

  または，$(T_i,\ q_k) \xrightarrow{X/+} (T_j,\ q_l)$
- q_k が T_i の非カーネル内にあるならば，

  GOTO 1$((T_i,\ q_k),\ X/1)=(T_j,\ q_l)$

  または，$(T_i,\ q_k) \xrightarrow{X/1} (T_j,\ q_l)$

とする．図2は，実はこの GOTO 1 関係を用いて作成したものである．

図2において，たとえば T_3 から T_5 への GOTO 1の矢印が2本あることは，ちょうど，状態 T_5 でカウント値の場合分けが二つあることに対応している．

また，通常と異なり，初期状態

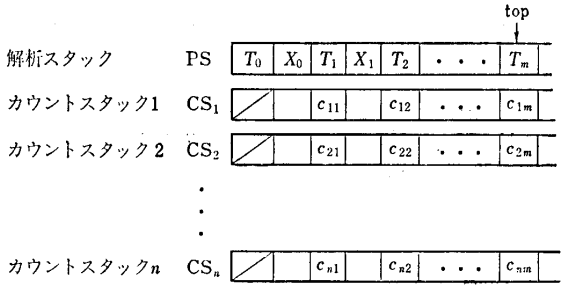
{｜0, 3, 5}（図2の T_0）

の LR 項はすべて非カーネル内にあるとみなしているが，これは，初期状態からある記号 X によって他の状態に遷移するときには，必ずカウント値の初期化がはかられるようにするためである．

一方，還元状態は通常どおりに得られる．図2で状態に付加した「$\#i$ $\{l_1, \cdots\}$」は，LR パーサがその状態にあって先読み記号が $\{l_1, \cdots\}$ に含まれるとき，生成規則 $\#i$ で還元することを意味する．

4. LR パーサの動作

LR パーサは，通常と同様の動作のほかにカウントスタックの処理を行う．LR パーサのスタックの構成を**図3**に示す．カウントスタックは解析スタックと同期する．慣例に従い，解析スタックには LR 状態と文法記号を交互にしまっておくことにする．これに伴い，カウントスタックは一つおきに使うこととする．



ただし，nは，LR状態のカーネル中のLR項の最大数である．
c_{ij}は，カウント値で空のこともある．

図 3 LR パーサのスタックの構成
Fig. 3 Stack configuration of the LR parser.

スタック	残り入力	動作
PS T_0 CS1 CS2	$a\ a\ a\ c\ \$$	aをシフト
⋮		
$T_0aT_3aT_3aT_5$ 1 2 3 1 1 2	$c\ \$$	#3で還元． 2×2だけポップ(注) してAをプッシュ．
$T_0aT_3AT_6$ 1 2 1	$c\ \$$	cをシフト
$T_0aT_3AT_6cT_7$ 1 2 3 1	$\$$	#2で還元． 3×2だけポップして Sをプッシュ．
T_0ST_1 1	$\$$	$\$$をシフト
$T_0ST_1\$T_2$ 1 2		#0で還元． (accept)

入力は「$a\ a\ a\ c\ \$$」とする．
PS は解析スタック
(注) $(T_5,\ 9)$ で還元するので，CS2 [top]×2 だけポップ

図 4 *G1* の構文解析の例
Fig. 4 An example parsing of a sentence generated by *G1*.

（1） 記号をプッシュするときの動作

LR 状態 T_i で記号 X を解析スタックにプッシュして状態 T_j へ遷移するとする．このときのカウントスタックの扱いは次のとおりである．

- GOTO 1$((T_i,\ q_k),\ X/+)=(T_j,\ q_l)$ なる q_k, q_l があり，q_k が T_i のカーネル中 u 番目，q_l が T_j のカーネル中 v 番目のとき，

  CSv[top]:=CSu[top−2]+1

を行う.

- GOTO 1 $((T_i, q_k), X/1)=(T_j, q_l)$ なる q_k, q_l があり, q_l が T_j のカーネル中 v 番目 (q_k は非カーネル中に含まれているはず) のとき,

  CSv[top]:=1

  を行う.

これでわかるように, カウントは, 解析スタック中の記号よりもLR状態に同期していると考えるのがよい. また, カウントスタックの本数は, LR状態のカーネル中のLR項の最大数だけあればよい.

(2) 還元時の動作

LR状態 T_i で還元をおこすとき, 解析スタックとカウントスタックをポップするが, その数は, T_i 中で還元に対応するLR項 q_k がカーネル中 u 番目であるとき, CSu[top]×2である. ただし, ε-規則による還元のときはスタックのポップは行わない.

文法 *G1* に対する構文解析の例を図4に示す.

5. おわりに

正規右辺文法のLRパーサの簡単な実現法について述べた.

この方法は, 1章で述べた三つの分類のうち, ELR文法から直接LRパーサを作成するクラスに属し, 文法の変換を必要としない. このクラスに属する従来の方法では, 右辺の長さを知るための仕組みとして, readback 状態を加えたり[4)-6)], lookback 状態かどうかを調べたりしていたが[6)], これらを求めるアルゴリズムはやや複雑であった. ここで提案した方法では, これらの計算は不要で, 通常のLRパーサの手法を若干精密化するだけで, 簡単にELR文法のパーサが作成できる.

本方法が適用できるためには, 4章の(1)で示した状態遷移時のカウントスタックの処理が, 与えられた文法に対して一意的に決まる必要がある. つまり, 4章(1)における T_i と (T_j, q_l) に対して

GOTO 1 $((T_i, q_k), X/+)=(T_j, q_l)$ または

GOTO 1 $((T_i, q_k), X/1)=(T_j, q_l)$

を満たす q_k が唯一つだけ存在しなければならない.

この条件は, 文献 6), 9) で示された条件と同等であり, ELR文法から直接LRパーサを作成できる文法クラスの本質的な条件であると思われる. この条件さえ成り立っていれば, 本方法は,

$A\to\alpha\{\beta\}\gamma\{\delta\}\zeta$ や $B\to\eta(\mu|\nu)\xi$

(α, β 等は記号列) のような形の生成規則を含む正規右辺文法に対して一般的に適用できる.

本稿では述べられなかったが, 右辺の長さが一定であるような生成規則では, カウントスタックの処理をすべて省略することができる.

実際の場面で正規表現が生成規則にどの程度現れるかを, コンパイラ生成系GAGによるPascalの記述にみると, 生成規則141個中, 右辺に正規表現を用いているのは20個 (14%) であった[10)] (これが少ないのは, 正規右辺文法に対する意味解析の手法がまだ確立していないからであると思われる).

本方法は構文解析の効率の点では最良ではないが, これらのことを考えると, 実際上のオーバヘッドはさほど大きくなく, 有望な方法であると期待される.

参考文献

1) 稲垣, 大山口：構文解析法からみた最近の計算機言語理論の動向, 情報処理, Vol. 23, No. 1, pp. 53-61 (1982).
2) Madsen, O. L. and Kristensen, B. B.: LR-Parsing of Extended Context Free Grammars, *Acta Inf.*, Vol. 7, pp. 61-73 (1976).
3) Heilbrunner, S.: On the Definition of ELR (k) and ELL(k) Grammars, *Acta Inf.*, Vol. 11, pp. 169-176 (1979).
4) LaLonde, W. R.: Regular Right Part Grammars and Their Parsers, *Comm. ACM*, Vol. 20, No. 10, pp. 731-741 (1977).
5) LaLonde, W. R.: Constructing LR Parsers for Regular Right Part Grammars, *Acta Inf.*, Vol. 11, pp. 177-193 (1979).
6) Chapman, N. P.: LALR (1, 1) Parser Generation for Regular Right Part Grammars, *Acta Inf.*, Vol. 21, pp. 29-45 (1984).
7) Purdom, P. W. and Brown, C. A.: Parsing Extended LR(k) Grammars, *Acta Inf.*, Vol. 15, pp. 115-127 (1981).
8) Aho, A. V., Sethi, R. and Ullman, J. D.: Compilers- Principles, Techniques and Tools, Addison-Wesley, Reading (1985).
9) Nakata, I. and Sassa, M.: Generation of Efficient LALR Parsers for Regular Right Part Grammars, to appear in *Acta Inf.* (1986).
10) Kastens, U., Hutt, B. and Zimmermann, E.: GAG: A Practical Compiler Generator, *Lec. Notes in Comp. Sci.*, 141, Springer, Berlin (1982).

(昭和60年5月29日受付)
(昭和60年9月19日採録)