正規右辺文法の効率のよいLRパーサの簡単な実現法†

張 又普†† 中田育男†† 佐々政孝††

正規右辺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。正規右辺文法はLR構文解析できるとき，ELR文法であると言われる．文献6）では効率のいいELRパーサを作成するアルゴリズムを提案したが，Lookback状態の計算が必要であるので，アルゴリズムはやや複雑である．文献8）ではELRパーサを簡単に構成する方法を提案したが，構文解析の効率に多少問題がある．本稿では文献6）と文献8）の方法の利点を組み合わせて，効率のいいELRパーサを簡単に構成する方法を提案する．その基本は次のようである．還元の際，その対象となる右辺を認識するために，非カーネル内のLR項からの遷移に対しては，状態記号をパーサスタックにプッシュするが，カーネル内のLR項からの遷移に対しては，状態記号をパーサスタックにプッシュしない．非カーネルとカーネルからの遷移がともにあったら（それをスタック競合と呼ぶ），状態記号をパーサスタックにプッシュするが，パーサスタックと並行にスタック競合の回数をカウントするためのスタックを設ける．この方法は，パーサの作成が簡単で，構文解析の効率がいい，作成時に文法の変換やLookback状態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[2]．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．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．ELR文法のLR構文解析法として，これまでいろいろ研究された[3),6)~8),10)]が，文献6）はELR文法のまま直接LRパーサを作成する効率のいいアルゴリズムを提案した．3章で述べるように，文献6）は多少問題があるので，本稿では，文献8），10）で提案するカウンタ方法により，文献6）で提案するスタック方法のスタック競合を解決する方法を提案する．詳細は3章から述べる．

2. 術語の定義

$G=(N, T, P, S)$ は文脈自由文法 (CFG)，N は非終端記号の有限集合，T は終端記号の有限集合，P は生成規則の有限集合，$S\in N$ は出発記号である．$G'=(N', T', P', S')$ は G に生成規則「$S'\to S\$$」を加えて得られる拡大文法とする．特に断わりのないかぎり，以下のような記号を使うことにする．$V=N\cup T$, A, B, $C\in N$, a, b, $c\in T$, x, y, $z\in T^*$, X, Y, $Z\in V$, α, β, $\gamma\in V^*$, ϕ は空集合とする．

以下の定義は文献1），5）とほぼ同じであるので，その詳細を文献1），5）に譲る．

定義：LR項 (LR item)

$G=(N, T, P, S)$ はCFGで，$A\to\beta_1\beta_2\in P$ のとき，$[A\to\beta_1.\ \beta_2]$ をLR (0) 項と定義する．以下に単にLR項と呼ぶことにする．記号「.」はLRマーカと呼ばれる．最右導出 $S'\Rightarrow\alpha Az\Rightarrow\alpha\beta_1\beta_2 z$ が存在するとき，LR項 $[A\to\beta_1.\ \beta_2]$ は成長可能な語頭 $\alpha\beta_1$ に対して正当であると定義する．$[A\to\beta.]$ のようにLRマーカが生成規則の右辺の最終位置にあるLR項を還元LR項と呼ぶことにする．

定義：LR状態 (LR state)

$G=(N, T, P, S)$ はCFGとする．$\gamma\in V^*$ は成長可能な語頭としたとき，語頭 γ に対して正当なすべてのLR項の集合をLR状態と定義する．

定義：I を文法 G に対するLR項の集合としたとき，LR項の集合 succ(I, X) を次のように定義する．

$$\text{succ}(I, X)=\{[A\to\alpha X.\beta]\mid[A\to\alpha.X\beta]\in I\}$$

定義：I を文法 G に対するLR項の集合としたとき，LR項の集合 closure(I, X) を以下の規則によって I から構成する．

1) I に含まれるLR項はすべて closure(I) に含まれる．

2) $\forall[A\to\alpha.B\beta]\in\text{closure}(I)$ に対して，「$B\to\gamma$」$\in P$ なら，$[B\to.\gamma]$ は closure(I) に含まれる．

定義：文法 G の初期状態を $I_0=\text{closure}(\{[S'\to.S]\})$ と定義する．I を文法 G のLR状態としたとき，

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

†† 筑波大学電子・情報工学系

LR 状態 Goto(I, X) を次のように定義する.

Goto(I, X)=closure(succ(I, X))

文法 G のすべての状態は次のようにカーネル (kernel) と非カーネル (nonkernel) に分けられる.

kernel(I_0)=ϕ

kernel(Goto(I, X))=succ(I, X)

nonkernel(I)=I−kernel(I)

以下に, LR 状態のカーネルに属する LR 項をカーネル項, 非カーネルに属する LR 項を非カーネル項と呼ぶことにする.

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) で置き換えて表すことにする. 以下では例1を用いて説明する.

例 1: $G1$: $S\rightarrow\{a\}A$; $A\rightarrow a\{bc\}B$; $B\rightarrow bcd$

文法 $G1$ に対するものを**図1**に示す. 同様にして, LR 項(以下に $t_1, t_2, \cdots$ のような記号で示す)も FSA の状態に対応させて表す. たとえば, LR 項 $A\rightarrow.a\{bc\}B$ は LR マーカ "." が図1の3の位置にあるので, "3" と表す. ELR 文法の LR 状態 (以下に $I_1, I_2, \cdots$, $T_1, T_2, \cdots$ のような記号で示す) 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, $A\rightarrow a.\{bc\}B$ と $B\rightarrow bc.d$ がカーネルに入っていたとするとき, 閉包をとると, $B\rightarrow.bcd$ が非カーネル項として, この LR 状態に入る. これらも FSA の状態を用いて {4, 9|7} 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, Goto 関係と閉包演算を用いて, 文法 $G1$ に対する LR オートマトンを作成し, **図2**に示す. 図2の ss(A) と s(A) のような記号は3章で述べるようにそれぞれ "A によるスタックシフト動作" と "A によるシフト動作" を意味する. $C_3(1):=0$ のような記号は4章で述べるようにカウンタ動作である.

ここで, Goto 関係は LR 状態から LR 状態への遷移関係を示す. Goto(I_i, a)=I_j なら, $I_i\rightarrow^a I_j$ のような遷移図で示すことにする. これを精密化して (LR 状態, LR 項) の組から (LR 状態, LR 項) の組への遷移関係を導入し, さらに出所がカーネル内か否かの情報を含む記法を導入する. 例えば, LR 状態 {1, 4|3, 7} (図2の T_2) から b による遷移をすると, カーネル内の LR 項 (T_2, 4) が (T_4, 5) に, 非カーネル内の LR 項 (T_2, 7) が (T_4, 8) に移るが, これを (T_2, 4)$\rightarrow^{b/K}$(T_4, 5) と (T_2, 7)$\rightarrow^{b/N}$(T_4, 8) のように表す. ここで, Kはカーネル, Nは非カーネルを表す.

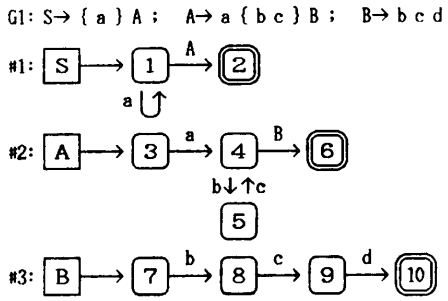

図1 $G1$ の正規右辺文法
Fig. 1 Representation of regular right part grammar $G1$.

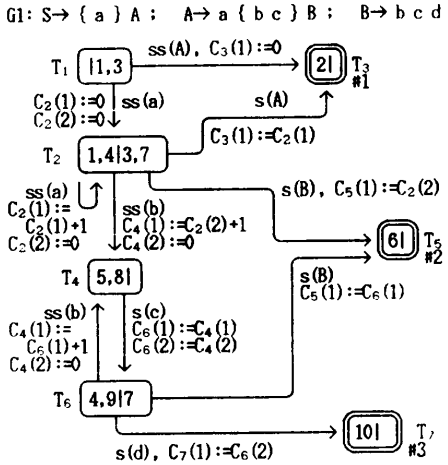

図2 $G1$ の LR オートマトン (ss(a) と s(a) はそれぞれ a による stack shift と shift を意味する)
Fig. 2 LR-Automaton for $G1$ (ss(a) and s(a) means stack shift and shift by a, respectively).

3. 文献 6) のスタック方法の概要

通常の LR 構文解析のシフト動作は入力文字列の先頭の一記号をパーサスタックにプッシュしてから, 状態記号もパーサスタックにプッシュすることである. 一般に, 非カーネル内の LR 項からの遷移は生成規則の右辺の先頭文字を読むことに対応し, カーネル内の LR 項からの遷移は生成規則の右辺の2番目以後の文字を読むことに対応する. 文献 6) の ELR 構文解析法では生成規則の右辺の先頭位置を認識するために, 非カーネル内の LR 項からの遷移に対し, 状態記号をパーサスタックにプッシュするが, カーネル内の LR 項からの遷移に対しては, 状態記号をパーサスタックにプッシュしない. 言い換えれば, パーサスタックの内の状態はある生成規則の右辺の先頭位置を示している.

アルゴリズム 3.1：文献 6) の ELR (1) 構文解析の基本動作

ELR (1) 構文解析系の状況 (configuration) とは最初の成分がパーサスタックの内容であり，2番目の成分が現状態と呼ばれる記号であり，3番目の成分が残っている入力列である次のような三つ組のことである．

$(I_0\beta_0I_1\beta_1\cdots I_{n-1}\beta_{n-1},\ I_n,\ Xz)$.

ここで，$\beta_i\in V^*$, $X\in V$, $z\in T^*$, I_i はパーサスタックの左から $i+1$ 番目の状態記号である．

ELR (1) 構文解析系の次の動作は現在の入力記号 X と現状態 I_n とを見て，構文解析動作表の記入欄 Action $[I_n, X]$ を調べることによって決まる．3種類の動作を行った後の状況はそれぞれ次のようになる．

(1) $I_n\rightarrow^{X/N}I_{n+1}$ なら，

Action $[I_n, X]$:="stack shift I_{n+1}"

と定義する．現状態 I_n をパーサスタックにプッシュしてから，X もパーサスタックにプッシュする．I_{n+1} を新しい現状態にする．この動作をスタックシフト動作と呼ぶことにする．次のような状況になる．

$(I_0\beta_0I_1\beta_1\cdots I_{n-1}\beta_{n-1}I_nX,\ I_{n+1},\ z)$

(2) $I_n\rightarrow^{X/K}I_{n+1}$ なら，Action $[I_n, X]$:="shift I_{n+1}" と定義する．X だけパーサスタックにプッシュする．I_{n+1} を新しい現状態にする．この動作をシフト動作と呼ぶことにする．次のような状況になる．

$(I_0\beta_0I_1\beta_1\cdots I_{n-1}\beta_{n-1}X,\ I_{n+1},\ z)$

(3) $A\rightarrow\beta_{n-1}$ が p 番目の生成規則で，$[A\rightarrow\beta_{n-1}., X]\in I_n$ なら，Action $[I_n, X]$:="reduce p" と定義する．パーサスタックの先頭にある状態記号 I_{n-1} の右側にある記号をパーサスタックから消去し，それを A に還元し，I_{n-1} を現状態にする．A を入力の先頭に付ける．この動作を還元動作と呼ぶことにする．次のような状況になる．

$(I_0\beta_0I_1\beta_1\cdots\beta_{n-2},\ I_{n-1},\ AXz)$ ■

この方法の問題は，状態 I_n に対し，$I_n\rightarrow^{X/N}I_{n+1}$ と $I_n\rightarrow^{X/K}I_{n+1}$ がともにあったら，Action $[I_n, X]$ がシフトかスタックシフトかを一意に決められないことである．これをスタック競合と呼ぶことにする．例えば，図2は，$T_2\rightarrow^{b/K}T_4$ と $T_2\rightarrow^{b/N}T_4$ がともにあるから，Action $[T_2, b]$ でスタック競合がある．

文献 6) における解決方法は次のようである．$t\in I_n$ が還元項なら，次のように (I_n, t) の Lookback 状態の集合を定義する．$LB(I_n, t)=\{I'\mid{}^{\exists}\beta\in V^*$ がある生成規則の右辺で，$(I', t')\rightarrow\cdots^{\beta}\cdots\rightarrow(I_n, t)$, t' が I' の非カーネル項である$\}$.

シフトするときに，遷移 $I_n\rightarrow^{X}I_{n+1}$ でスタック競合があったら，Action $[I_n, X]$:="stack shift I_{n+1}" と定義する．

還元するときに，現状態の Lookback 状態がパーサスタックの先頭になるまでパーサスタックの状態をポップする．

Lookback 方法はアルゴリズムがやや複雑で，効率の点では多少問題があると思う．

文献 8) の ELR (1) 構文解析法は各入力記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することにより右辺の長さを覚える方法である．

4. スタック競合の解決

本章では文献 8) で提案するカウンタ方法で文献 6) のスタック競合を解決する方法を提案する．ただし，右辺の長さをカウントするのでなく，スタック競合の回数をカウントする方法である．

解決方針：1) シフトするときに，遷移 $I_n\rightarrow^{X}I_{n+1}$ でスタック競合があったら，文献 6) と同じように Action $[I_n, X]$:="stack shift I_{n+1}" と定義する．例えば，図2の遷移 $T_2\rightarrow^{b}T_4$ に対し，Action $[T_2, b]$:="stack shift T_4" と定義する．このとき，遷移 $(T_2, 4)\rightarrow^{b/K}(T_4, 5)$ に対しては，余計な状態がパーサスタックに入れられる．

2) 1)で生じる余計な状態をカウントしていけばよい．すべての状態にカーネル項ごとにカウンタを付ける．言い換えれば，各状態にその属性としてカウンタの配列を付け，状態記号と一緒にパーサスタックにプッシュかポップし，値を計算する．カウンタ配列は1次元の整数型配列で，要素数は下限が1，上限が各状態のカーネル項の数である．以下に C_n で状態 I_n に付くカウンタ配列を示す．$C_n(i)$ は I_n の i 番目のカーネル項に対応するカウント値である．I_n の i 番目のカーネル項が還元項なら，カウント値 $C_n(i)$ は余計にプッシュされた状態の数を示す．ELR パーサのスタ

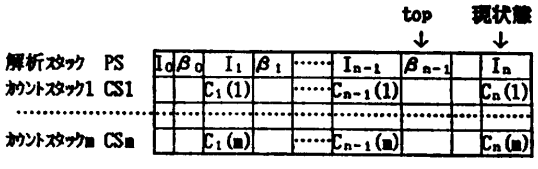

図 3 ELR 構文解析のパーサスタックの構成
Fig. 3 The structure of ELR parser stack.

ックの構成を図3に示す.

カウンタの動作は三つある.

1) 初期設定: $(I_{n-1}, t_{i1})\to^{X/N}(I_n, t_{i2})$ のように非カーネル内のLR項から遷移するときは,余計にプッシュされた状態は0個であるから,スタックシフトを実行するときに,$C_n(i_2):=0$ のように初期化する.ここで,t_{i2} は状態 I_n の i_2 番目のカーネル項とする.

2) $(I_{n-1}, t_{i1})\to^{X/N}(I_n, t_{i2})$, $(I_{n-1}, t_{j1})\to^{X/K}(I_n, t_{j2})$ のようにスタック競合が発生するときには,強制的にAction $[I_{n-1}, X]$:="stack shift I_n" と定義するから,I_{n-1} のカーネル内のLR項 t_{j1} からの遷移に対しては,状態が一つ余計にプッシュされることになる.ゆえに,I_n のカーネル内のLR項 t_{j2} と対応するカウンタを1だけ増やす.すなわちスタックシフトを実行するときに,$C_n(i_2):=0$; $C_n(j_2):=C_{n-1}(j_1)+1$ も実行する.ここで,t_{i2} と t_{j2} はそれぞれ状態 I_n の i_2 と j_2 番目のカーネル項である.

3) $(I_{n-1}, t_{j1})\to^{X/K}(I_n, t_{j2})$ のようなカーネル内のLR項からの遷移しかなければ,Action $[I_{n-1}, X]$:="shift I_n" と定義するから,余計にプッシュされた状態の数は変わらない.そこで,カウンタをそのまま転送する.すなわちシフト動作を実行するときに,$C_n(j_2):=C_{n-1}(j_1)$ も実行する.

4) 還元するときには,カウント値で余計な状態の数が分るから,右辺の先頭位置を簡単に認識できる.

以上のことをまとめて,構文解析系の動作を述べる.

アルゴリズム4.1: ELR (1) 構文解析のカウンタ動作

ELR (1) 構文解析の動作ではアルゴリズム3.1と同様な動作のほかに,シフトかスタックシフトされるときに,カウンタの処理も行う.ELR (1) 構文解析系の状況とは次のような三つ組のことである.

$(I_0\beta_0C_1I_1\beta_1\cdots C_{n-1}I_{n-1}\beta_{n-1}, C_nI_n, Xz)$

ここで,$\beta_i\in V^*$, $X\in V$, $z\in T^*$ で,I_i はパーサスタックの左から $i+1$ 番目の状態記号である.C_i は状態 I_i に付いているカウンタ配列である.

1) スタックシフト動作:

a) $(I_n, t_{i1})\to^{X/N}(I_{n+1}, t_{j1})$ なる t_{i1}, t_{j1} があり,t_{j1} は I_{n+1} のカーネル中 j_1 番目の項のとき,$C_{n+1}(j_1):=0$ を行う.

b) $(I_n, t_{i1})\to^{X/K}(I_{n+1}, t_{j1})$, $(I_n, t_{i2})\to^{X/N}(I_{n+1}, t_{j2})$ なる t_{i1}, t_{i2} と t_{j1}, t_{j2} があり,t_{i1} と t_{j1} がそれぞれ I_n と I_{n+1} のカーネル中 i_1 と j_1 番目項のとき,$C_{n+1}(j_1):=C_n(i_1)+1$ を行う.

2) シフト動作:

$(I_n, t_{i1})\to^{X/K}(I_{n+1}, t_{j1})$ なる t_{i1}, t_{j1} があり,t_{i1} と t_{j1} がそれぞれ I_n と I_{n+1} のカーネル中 i_1 と j_1 番目の項のとき,$C_{n+1}(j_1):=C_n(i_1)$ を行う.

3) 還元のときの動作:

$[A\to\alpha., X]$ が I_n のカーネル中の i 番の項で,$A\to\alpha$ は p 番目の生成規則で,$C_n(i)=h$ のとき,I_{n-h-1} の右側にある記号をパーサスタックから消去し,それを A に還元し,I_{n-h-1} を現状態にする.A を入力の先頭に付ける. ■

文法 $G1$ に対するカウンタ動作付きELRオートマトンを図2に示す.文法 $G1$ に対する構文解析の例を図4に示す.

なお本方法が適用できるためには,$(I_{n-1}, t_{i1})\to^{X/N}(I_n, t_{i2})$, $(I_{n-1}, t_{j1})\to^{X/K}(I_n, t_{j2})$, $i_1\neq j_1$ というスタック競合に対して,$i_2=j_2$ であってはならない.これは

スタック	現状態	残り入力	動作
PS CS1 CS2	T_1	abcbcd	aをスタックシフト
T_1a	T_2 0 0	bcbcd$	bをスタックシフト
T_1aT_2b 0 0	T_4 1 0	cbcd$	cをシフト
T_1aT_2bc 0 0	T_6 1 0	bcd$	bをスタックシフト
$T_1aT_2bcT_6b$ 0 1 0 0	T_4 2 0	cd$	cをシフト
$T_1aT_2bcT_6bc$ 0 1 0 0	T_6 2 0	d$	dをシフト
$T_1aT_2bcT_6bcd$ 0 1 0 0	T_7 0	$	#3で還元.余計にプッシュされた状態は0であるので,T_6の右側の文字をポップする
T_1aT_2bc 0 0	T_6 1 0	B$	Bをシフト
T_1aT_2bcB 0 0	T_5 1	$	#2で還元.余計にプッシュされた状態は1であるので,T_1の右側の文字をポップする
	T_1	A$	Aをスタックシフト
T_1A	T_3 0	$	受理する

図4 文法 $G1$ に対する構文解析の例
Fig. 4 An example of parsing for grammar $G1$.

文献 6), 8) での条件と本質的に同じであり，共通の制限条件である．

5. カウンタ動作の最適化

アルゴリズム 4.1 はカウンタの基本動作であるが，スタック競合のない場合にもカウントしなければならないので，全体として効率がよくない．スタック競合のない場合は文献 6) が一番効率がよい．したがって，スタック競合のあるところだけカウンタ動作をとることにしたい．アルゴリズム 4.1 の (1b) によると，スタック競合が発生するとき，カーネル LR 項と対応するカウンタはすべて 1 増やす．これは無駄な動作ではない．ほかのカウンタ動作をできるかぎり省略したい．

アルゴリズム 4.1 の (1a) によると，カウンタの初期値はすべて 0 だから，次のように最適化する．現状態 I_n をパーサスタックにプッシュしてから，新しい現状態 I_{n+1} に付くカウンタ配列の全領域を 0 に設定する．その後，(1b) のカウンタ動作を実行する．0 の設定はすべてブロックリセット命令のような効率のいい命令で設定できるので，アルゴリズム 4.1 の (1a) は全部効率よくできるはずである．

アルゴリズム 4.1 の (2) については，カウンタの転送動作は現状態から新しい現状態への転送であるから，カーネル内の項の順序を適当に入れ替えれば，カウンタの転送動作は省略できるものが多い．たとえば，図 2 の $T_4 \rightarrow T_6$ のカウンタ転送動作は省略できる．図 4 の 3 行と 4 行に示したように，新しい現状態 T_6 でもとの現状態 T_4 を置き換え，カウンタをそのままにしてもよい．一方，図 2 の $T_6 \rightarrow T_7$ のカウンタ転送動作は省略できないが，状態 T_7 の一番目の項を空とし，二番目の項を "10" とすれば，$T_6 \rightarrow T_7$ のカウンタ転送動作は省略できる．一般に，$I \rightarrow^{X/N} J$ のような遷移がなく，$(I, t_i) \rightarrow^{X/K} (J, s_j)$ の $j=i$ なら，カウンタ転送動作 "$C_J(j) := C_I(i)$" を省略できる．ここで，t_i, s_j はそれぞれ状態 I, J の i, j 番目のカーネル項である．できるかぎりカウンタ転送動作を省略するために，文法の LR オートマトンを作るときに各状態のカーネル項の順序を考慮しなければならない．アルゴリズム 5.1 はそのようなアルゴリズムである．ここで，"項 (I, t_i) をマークする" とは "カーネル項 t_i が状態 I の i 番目の項に決まった" ことを意味する．

アルゴリズム 5.1：カウンタ動作付き LR オートマトンの作り方：

入力：$G=(N, T, P, S)$

出力：G のカウンタ動作付き LR オートマトン

初期設定：$C := \{\text{closure}(\{[S' \rightarrow .S, \$]\})\}$.

Step 1：以下を新たな集合が C に付け加えられなくなるまで繰り返す．

任意の $I \in C$ と $X \in V$ に対し，$J := \text{Goto}(I, X)$ が空集合でないときに，

a) 状態 I の非カーネルからの遷移があったら，Action $[I, X]$:="stack shift J"；さもなければ，Action $[I, X]$:="shift J"

b) J が C に入っていなかった場合：

任意の $(I, t_i) \rightarrow^{X/K} (J, s_j)$ に対し，J のカーネル項の順序を $j=i$ となるように入れ替える．項 (I, t_i) と (J, s_j) をマークして，J を C に加える．

c) J が C に入っていた場合：

任意の $(I, t_i) \rightarrow^{X/K} (J, s_j)$ に対し，

$i=j$ なら，項 (I, t_i) と (J, s_j) をマークする；

$i \neq j$ で，項 (J, s_i), (J, s_j) がマークされていなかったら，J の i と j 番目のカーネル項を交換し，(J, s_i) をマークする；

$i \neq j$ で，項 (J, s_i), (J, s_j) のどちらか一つがマークされていたら，$I \rightarrow^{X/N} J$ のような遷移がないかぎり，カウンタ転送動作 "$C_J(j) := C_I(i)$" を Action $[I, X]$ に加える．

Step 2：任意の $I, J \in C$, $X \in V$, Action $[I, X]$ = "stack shift J" に対し，$(I, t_i) \rightarrow^{X/K} (J, s_j)$ があったら（スタック競合があったら），カウンタ動作 "$C_J(j)$

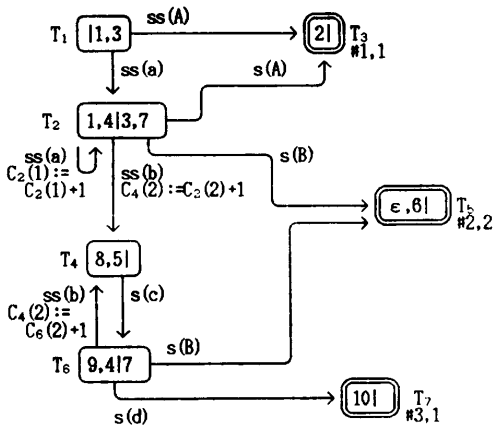

図 5 アルゴリズム 5.1 による $G1$ の LR オートマトン (ss(a) と s(a) はそれぞれ a による stack shift と shift を意味する)

Fig. 5 LR-Automaton for $G1$ by Algorithm 5.1 (ss(a) and s(a) means stack shift and shift by a, respectively).

:=$C_I(i)+1$" を Action [I, X] に加える.

Step 3: スタック競合が一つもなかったら, すべてのカウンタ動作を削除し, 文献 6) の基本方法のアルゴリズム 3.1 で解析する. ■

図5 はアルゴリズム 5.1 による $G1$ の LR オートマトンである. 無駄なカウンタ動作が一つもない. 図5の状態 T_7 の "#3, 1" とは "1番目のカーネル項は3番目の生成規則で還元する項である" を意味する. **図6** は図5による構文解析の例である.

6. おわりに

正規右辺文法の LR 構文解析の実現法について述べた. この方法は文献 6) と文献 8) の方法の利点を組み合わせたものである. ELR 文法から直接 LR パーサを作成できるので, 文法の変換を必要としない. 文献 6) の方法では, スタック競合が生じたときには右辺の長さを知るための仕組として, Lookback 状態かどうかを調べなければならないが, 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, これらの計算は不要で, 簡単に ELR 文法のパーサが作成できる. 一方, カウンタの動作は文献 8) の提案する方法では右辺の長さをカウントするのに対し, 本方法ではスタック競合の回数をカウントするだけであるから, 本方法のほうが少ない. ゆえに, 構文解析の効率は文献 8) の方法よりよいと言える. 処理系はまだ作成していないが, 以上のことを考えると, 実際上のオーバヘッドはほとんどなく, 有望な方法であると期待される. 今後は本方法と属性文法[9]を組み合わせる応用も考えられる.

スタック	現状態	残り入力	動作
PS CS1 CS2	T_1	abcbcd	aをスタックシフト
T_1a	T_2 0 0	bcbcd$	bをスタックシフト
T_1aT_2b 0 0	T_4 0 1	cbcd$	cをシフト
T_1aT_2bc 0 0	T_6 0 1	bcd$	bをスタックシフト
$T_1aT_2bcT_6$b 0 0 0 1	T_4 0 2	cd$	cをシフト
$T_1aT_2bcT_6bc$ 0 0 0 1	T_6 0 2	d$	dをシフト
$T_1aT_2bcT_6bcd$ 0 0 0 1	T_7 0 2	$	#3で還元. 余計にプッシュされた状態は0であるので, T_6の右側の文字をポップする
T_1aT_2bc 0 0	T_6 0 1	B$	Bをシフト
T_1aT_2bcB 0 0	T_5 0 1	$	#2で還元. 余計にプッシュされた状態は1であるので, T_1の右側の文字をポップする
	T_1	A$	Aをスタックシフト
T_1A	T_3 0 0	$	受理する

図 6 図5による文法 $G1$ に対する構文解析の例
Fig. 6 An example of parsing for grammar $G1$ based on Fig. 5.

参考文献

1) Aho, A. V., Sethi, R. and Ullman, J. D.: *Compilers-Principles, Techniques, and Tools*, Addison-Wesley, Reading, Massachusetts (1986).
2) Chapman, N. P.: LALR (1, 1) Parser Generation for Regular Right Part Grammars, *Acta Inf.*, Vol. 21, pp. 29-45 (1984).
3) Madsen, O. L. and Kristensen, B. B.: LR Parsing of Extended Context Free Grammars, *Acta Inf.*, Vol. 7, pp. 61-73 (1976).
4) Nakata, I.: On Compiling Algorithms for Arithmetic Expressions, *Comm. ACM*, Vol. 10, No. 8, pp. 492-494 (1967).
5) 中田育男: コンパイラ, 産業図書, 東京 (1982).
6) Nakata, I. and Sassa, M.: Generation of Efficient LALR Parsers for Regular Right Part Grammars, *Acta Inf.*, Vol. 23, pp. 149-162 (1986).
7) Purdom, P. W. and Brown, C. A.: Parsing Extended LR (k) Grammars, *Acta Inf.*, Vol. 15, pp. 115-127 (1981).
8) 佐々政孝, 中田育男: 正規右辺文法の LR パーサの簡単な実現法, 情報処理学会論文誌, Vol. 27, No. 1, pp. 124-127 (1986).
9) 佐々政孝: 属性文法チュートリアル, ソフトウェア基礎論, 16-1 (1986).
10) Sassa, M. and Nakata, I.: A Simple Realization of LR-parsers for Regular Right Part Grammars, *Information Processing Letters*, Vol. 24, pp. 113-120 (1987).

(昭和62年5月6日受付)
(昭和62年9月9日採録)

張　又普（正会員）

1953年生．1982年2月中国西北大学電子計算机系卒業．1985年電気通信大学大学院修士課程修了．工学修士．現在筑波大学大学院博士課程工学研究科に在学中．プログラミング言語，属性文法，コンパイラの生成系の研究に興味を持っている．ソフトウェア科学会会員．

中田　育男（正会員）

1935年生．1958年東京大学理学部数学科卒業．1960年同大学院修士課程修了．1960～1979年(株)日立製作所中央研究所，同システム開発研究所勤務．1979年4月より筑波大学電子・情報工学系教授．理学博士．プログラム言語，言語処理系，ソフトウェア工学などに興味を持っている．著書「コンパイラ」(産業図書)．日本ソフトウェア科学会，電子情報通信学会，ACM，IEEE 各会員．

佐々　政孝（正会員）

1948年生．1970年東京大学理学部物理学科卒業．1974年同理学系研究科博士課程中退，東京工業大学理学部情報科学科助手となる．1981年筑波大学電子情報工学系講師，1986年より同助教授．理学博士．プログラミング言語，属性文法，コンパイラ，コンパイラ生成系，プログラミング支援系に興味を持っている．1981年本学会論文賞受賞．ソフトウェア基礎論研究会幹事．ソフトウェア科学会，ACM，IEEE 各会員．