$LR(1)$ 文法の諸性質とそれに基づく parser の構成法*

関　本　彰　次**

Abstract

A method of constructing parsers for $LR(k)$ grammars has been given first by D. E. Knuth. But the parsers by knuth's method have generally too large sizes because of thier parsing tables. So, the several efforts for giving small and practical parsers have been reported by many authors. A. K. Korenjak has decreased the size of processor by partitioning the given grammar into a number of smaller parts. Another successful example has been given by T. Hayashi. In his method, by a graphical approach, a given grammar is translaled from CFG to PL.

In this paper, we show that $LR(1)$ grammars have some peculiar properties which $LR(k)$ grammars have not generally if k is not equal to 1 and that using those properties, a method of constructing practical $LR(1)$ parsers can be introduced.

1. はじめに

$LR(k)$ 文法については，Knuth[1] によって最初にその決定法と parser の構成法が与えられたが，そこで作られる parser の使用する表の大きさは，一般的には非常に大きなものになる．これに対し Korenjak[2] は Knuth の方法に手を加え，文法をいくつかの部分文法に分割し，もとの文法をそれらの合成形で表現することによって表を縮少させることにある程度成功している．しかし Korenjak の方法では文法をどのような部分文法によって分割するのが最適かという点については，その手法では触れていないこと，また，表の縮少の仕方でも，他により徹底した手法が見いだしうる余地がある．

これに対して，林[3] は Knuth の状態を Z-状態と呼ばれるものに変換することによって状態数を減らし，これに左文脈を考慮に入れること，グラフによる表現を導入することによって縮少された processor の構成に成功している．この Z-状態への変換は本文でも用い，ここでは $LR(1)$ 文法が $k>1$ についての $LR(k)$ とは異なって，個有の特性をもつことを変換特性を通して明らかにし，この特性に従って縮少した $LR(1)$ parser を得る方法を示す．

2. 記号および用語

文脈独立文法を $G=(V_N, V_T, P, S_0)$ と表わす．

ここで，V_N および V_T は文法 G でのそれぞれ非端記号および端記号の集合である。また P は $A\to\omega$ なる形をした生成規則の有限個からなる集合である．S_0 は V_N の要素の一つで，特に出発記号と呼ばれる．

$V=V_N\cup V_T$ で表わされる V に対し，そのうえでの有限長の記号列のすべてからなる集合を V^* で表わす．V^+ は V^* から空記号列 ε を除いた集合，また，$k0\geq$ について，V^k は V^* での長さ k の記号列のすべてからなる集合を表わす．

特にことわらない限り V_N の元は英大文字 (A, B, C, …) を用い，V_T の元には英小文字の初めの部分 (a, b, c, …) を，また，V_T^* の元には英小文字の尾部 (t, u, v, …) を用いる．また V^* の元にはギリシア小文字 (α, β, γ, …) を用いることにする．

生成規則 $A\to\omega$ が文法 G に含まれているとき，任意の α, β に対して，$\alpha A\beta\to\alpha\omega\beta$ と書くことにし，もし，$n\geq1$ について，$\alpha_1\to\alpha_2\to\cdots\to\alpha_n$ ならば，$\alpha_1\Rightarrow\alpha_n$ と書き α_1 は α_n を導き出す，または α_n は α_1 に還元されるという．また，$S_0\Rightarrow\omega$ によって ω が導かれるとき ω を文形という．特に $S_0\Rightarrow t$, $t\in V_T^*$ のとき，t は G での文という．G の文のすべてを $L(G)$ と

* A practical method of constructing $LR(1)$ parser based on the properties peculiar to $LR(1)$ grammars, by Shoji Sekimoto (Mitsubishi Electric Co.)

** 三菱電機鎌倉製作所電機器研究部

書き，G によって生成される言語と呼ぶ．

3. $LR(k)$ 文法に対する Knuth の方法

Knuth[1] および Korenjak[2] に従って $LR(k)$ 文法に対する processor について，その構成法と操作法を示す．

文法 $G=(V_N, V_T, P, S_0)$ において，P に含まれる生成規則の個数を π とする．

それらに対して1から π までの番号を割り当て，第 p 番目の生成規則を，

$$A_p \to X_{p1} \cdots X_{pnp}, \quad (n_p \geq 0)$$

と表わす．

非負整数定数 k ($k \geq 0$) と S_0'，$\# \notin V$ なる記号に対して，文法，

$$G'=(V_N', V_T', P', S_0'),$$
$$V_N'=V_N \cup \{S_0'\}, \quad V_T'=V_T \cup \{\#\},$$

かつ，

$$P'=P \cup \{S_0' \to S_0 \#^k\},$$

を定義する．ここで，$S_0' \to S_0\#^k$ に対しては G' のなかでは第0番を割り当てる．ここで記号 # を終記号 (endmarker)，そして，こうして作られた G' を G の k-増大文法 (k-augmented grammar) と呼ぶ．

以後，この G' を $G=(V_N, V_T, P, S_0)$ と改めて書き表わしこれを使用する．

3.1 $LR(k)$ processor の構成法

文法 G が $LR(k)$ ならば，G についての $LR(k)$ processor を記述する parsing table $T(G)$ を作りだすことができる．

$T(G)$ は processor の状態 (state) にそれぞれ対応する有限個の項からなる．状態 s に対応する $T(G)$ の項の内容は表 3.1 のようになっている．

Lookahead string $y_1, \cdots, y_l$ ($l \geq 1$) はいずれも V_T^k の要素で互いに異なる記号列，また，Stack Symbol $Y_1, \cdots, Y_m$ ($m \geq 1$) はいずれも V の要素で互いに異なる記号からなる．状態 s のときに入力として現われうる Lookahead String の集合を $L(s)$，また $y \in L(s)$ なる y に対応する Action を $A_s(y)$ と表わす．

表 3.1 状態 s に対する parsing table の内容

State name	Lookahead Symbol	Action	Stack Symbol	Goto state
s	y_1	shift	Y_1	s_1
			Y_2	s_2
	⋮	⋮	⋮	⋮
	y_l	reduce p	Y_m	s_m

Stack Symlol $Y_1, \cdots, Y_m$ の集合は $S(s)$ と表わされ，そのおのおのの要素 $Y \in S(s)$ に対して状態 $G_s(Y)$ が唯一に対応している．これを状態 s で Stack Symbol Y のときの行先状態 (Goto State) と呼ぶ．

Action の種類には shift, reduce p ($1 \leq p \leq \pi$) および accept があって，表中では $y_1, \cdots, y_l$ に対して，それぞれに唯一にどれかの Action が対応している．

$T(G)$ の作り方について述べるまえに，V^* の要素に対して V_T^k の部分集合または空集合を対応させる二つの関数を定義する．

定義 3.1

文法 $G=(V_N, V_T, P, S_0)$，$\alpha \in V^+$ および $k \geq 0$ に対して，$H(\alpha)$，$H'(\alpha)$ を次のように定める．

(a) $H(\alpha)=\{t \in V_T^k \mid \exists \beta \in V^*$ に対して，
G において $\alpha \Rightarrow t\beta\}$

(b) $H'(\alpha)=\{t \in V_T^k \mid \exists \beta \in V^*$ に対して，
G において $\alpha \Rightarrow t\beta$，
ただし，$\alpha \Rightarrow t\beta$ には $Ar \to r$ ($A \to \varepsilon$ による) なる step は含まない$\}$

Knuth による $LR(k)$ processor での状態とは，部分状態 (partial state) と呼ばれる $[p, j, w]$ なる triplet の集まりとみなされる概念である．

parsing table $T(G)$ は次のようにして作られる．

(i) 初めに，初期状態 (start state) として，
$s_0=\{[0, 0, \#^k]\}$
が定義される．

(ii) 次に，状態 s に対して，
$s'=s \cup \{[q, 0, x] \mid \varepsilon[p, j, w] \in s'$
に対して，$j<n_p$，$X_{p(j+1)}=A_q$ で，しかも，
$x \in H(X_{p(j+2)} \cdots X_{pnp}w)\}$
を定義する．

(iii) s' を用いて，$L(s)$ の要素およびそれらに対応する Action を決定する．すなわち，

① s' に含まれる $[p, j, w]$ があって，$j<n_p$ でしかも，$y \in H'(X_{p(j+1)} \cdots X_{pnp}w)$ のとき，
$y \in L(s)$，かつ，$A_s(y)=$shift，
とする．

② s' が $[p, n_p, y]$ を含むならば，
$y \in L(s)$，かつ，$A_s(y)=$reduce p，
とする．

G が $LR(k)$ である限り，$L(s)$ の各要素にはそれぞれ唯一の Action が対応する．$L(s)$ のある要素に異なる二つ以上の Action が対応するこ

とが起こったならば，G はもともと $LR(k)$ でなかったということである．

(iv) 次に，状態 s に対する $S(s)$ とそのおのおのの要素に対応する行先状態を決定する．すなわち，

$S(s)=\{Y|{}^{\exists}[p,\ j,\ w]\in s',\ Y=X_{p(j+1)}\}$，

${}^{\forall}Y\in S(s)$ について，

$G_s(Y)=\{[p,\ j+1,\ w]|{}^{\exists}[p,\ j,\ w]\in s',\ Y=X_{p(j+1)}\}$

(v) 最後に，s についての行先状態のおのおのについて，いまだ parsing table 中に項目が設けられていないものについては，おのおのに新しく項目を設ける．しかるのち，(i)～(iv)の操作によって $L(s)$, $A_s(y)$, $S(s)$, $G_s(Y)$ が決定していない状態について，(i)～(v)の操作を行なう．

部分状態 $[0,\ 1,\ \#^k]$ を含む状態はただ一つ存在し，それを s_F と表わす．$A_{sF}(\#^k)$=accept である．また，accept はこのとき以外はとられない．

3.2 processor の操作法

processor は 3.1 で作られた $T(G)$ に従い，ほかにここでは制御の目的で使用される2層の push-down stack を用いて，入力記号列を最左端還元によって S_0 に還元する．

入力テープの位置と stack の内容を次のように書き表わすことにする．

$$\begin{matrix}s_0,\ s_1\cdots s_n\\ \bullet X_1\cdots X_n\end{matrix}\Bigg|a_1\cdots a_k t \qquad (3.2.1)$$

すなわち，もとの入力記号列が，

$$xa_1\cdots a_k t\in V_T^*\#^k$$

のとき，x が $X_1\cdots X_n$ に還元されて，$a_1\cdots a_k$ がこのときの Lookahead string であり，t はこの時点ではまだ処理過程にはいってこない部分を示しているものとする．

push-down stack の上段の track には状態名 s_0, …, s_n がはいっているとしている．

processor は次のように動作する．

まず初めに，出発状態 s_0 が stack に置かれ（下段の track に置かれる記号 ● は単なる track の初期位置を示すものにすぎない），長さ k の入力記号列が最初の Lookahead string となる．この初期操作のあと，一般に processor が式 (3.2.1) に示した環境にあるとき，

(i) 記号列 $a_1\cdots a_k$ が $L(s_n)$ に含まれるかどうか検査する．その結果，操作は次の四つの場合のいずれかがとられる．

(a) $A_{sn}(a_1\cdots a_k)$=shift ならば，a_1 は下段 track に pushed on される．

$$\begin{matrix}s_0s_1\cdots s_n\\ \bullet X_1\cdots X_n a_1\end{matrix}\Bigg|a_2\cdots a_k t$$

(b) $A_{sn}(a_1\cdots a_k)$=reduce p ならば，上下段 track の右端から n_p 個の要素が popped off され，A_p が下段 track に pushed on される．

$$\begin{matrix}s_0s_1\cdots s_{n-n_p}\\ \bullet X_1\cdots X_{n-n_p}A_p\end{matrix}\Bigg|a_1\cdots a_k t$$

(c) $A_{sn}(a_1\cdots a_k)$=accept ならば，ここで process は終了する．

(d) $a_1\cdots a_k\notin L(s_n)$ のときは，入力に誤りがあったのであり，拒否される．

(ii) 次に，(i)において，

$A_{sn}(a_1\cdots a_k)$=shift または reduce p

のときには parsing table からそれぞれの場合についての行先状態が決定され，これを stack の上段 track の右端に入れる．この場合，$A_{sn}(a_1\cdots a_k)$=shift であれば，$a_1\in S(s_n)$ で，このときの行先状態は $G_{sn}(a_1)$ である．また，$A_{sn}(a_1\cdots a_k)$=reduce p では，$A_p\in S(s_{n-n_p})$ で，このときの行先状態は $G_{sn-np}(A_p)$ である．

(ii) の操作によって stack の環境は式 (3.2.1) に帰ったことになり，ふたたび (i) がくり返される．この操作は入力が誤りであると発見されるか，または，

$$\begin{matrix}s_0s_F\\ \bullet S_0\end{matrix}\Bigg|\#^k$$

で $A_{sF}(\#^k)$=accept が起こるまで続けられる．

4. Z-状態の定義

$LR(k)$ において，Knuth の状態（以後これを K-状態と呼ぶ）に対して，これらのある要素への mapping を考える．K-状態に対して，この mapping によって対応づけられる要素を Z-状態[3] と呼ぶことにする．

$LR(k)$ 文法 G において，Knuth の状態の集合を，

$$\boldsymbol{S}=\{s_i|0\leq i\leq n\}$$

とする．

$s_i\in\boldsymbol{S}$ について，$[p,\ j,\ w]\in s_i$ なる部分状態に対して，

$$f:[p,\ j,\ w]\rightarrow[p,\ j],$$

なる mapping を考える．ここで集合 Γ を，

$$\Gamma=\{[p,\ j]|[p,\ j,\ w]\rightarrow[p,\ j],$$

$\forall[p, j, w] \in s_i, \forall s_i \in \boldsymbol{S}\}$,

とし，また，Γ の部分集合の集合を $\boldsymbol{Z}$ として，$f(s_i)$ を，

$$f(s_i) = \{[p, j] \mid [p, j, w] \to [p, j], \forall[p, j, w] \in s_i\},$$

と定義する．定義から，s_i から $f(s_i)$ への対応は，$\boldsymbol{S}$ から $\boldsymbol{Z}$ のなかへの mapping である．$\boldsymbol{Z}$ の部分集合 $\boldsymbol{Z}_0$ を，

$$\boldsymbol{Z}_0 = \{f(s_i) \mid \forall s_i \in \boldsymbol{S}\},$$

と定義する．

f による s_i から $f(s_i)$ への対応は一意であるが，逆射像，すなわち，

$$f^{-1}(Z_k) = \{s_i \mid f(s_i) = Z_k, \exists s_i \in \boldsymbol{S}, Z_k \in \boldsymbol{Z}_0$$

は，$Z_k (Z_k \in \boldsymbol{Z}_0)$ に対して，必ずしも，唯一の要素が対応するとは限らない．

5. LR(1) における諸性質

定　義

$LR(k)$ において，Knuth の状態 s_i に対する集合 $S(s_i)$ の要素 Y による s_i の行先状態 $Gs_i(Y)$ を s_i の Y による直後続点とも呼ぶ．また s_i を Y による $Gs_i(Y)$ の直前点とも呼ぶ．

$LR(k)$ では s_i の $Y(Y \in S(s_i))$ による直後続点は唯一に定まる．

定　理 1

$LR(1)$ において，$f(s_i) = f(s'_i)$ のとき，

$a \in V_T$, $a \in L(s_i)$, かつ，$As_i(a) = \text{shift}$

ならば，

$a \in L(s'_i)$, かつ，$As'_i(a) = \text{shift}$

である．

証　明

$As_i(a) = \text{shift}$ から，$\exists[p, j, w] \in s_i$, $j < n_p$ であって，

$$a \in H'(X_{p(j+1)} \cdots X_{pn_p} w)$$

が成立する．

ここで $As_i(a) = \text{shift}$ および $LR(1)$ ということから a は w でなく，しかも，互いにその一部を共有することもない．

すなわち，$X_{p(j+1)} = a$ または $A_q \to X_{q1} \cdots X_{qn_q}$ なる生成規則が存在して $X_{q1} = a$，かつ，$\exists \alpha \in V^*$ に対し $X_{p(j+1)} \Rightarrow A_q \alpha$ とすることができる．

一方，$f(s_i) = f(s'_i)$ から，$\exists[p, j, w'] \in s'_i$ であって，また，$X_{p(j+1)} = a$ または $A_q \to X_{q1} \cdots X_{qn_q}$ なる生成規則があって $X_{q1} = a$，かつ，$X_{p(j+1)} \Rightarrow A_q \alpha$ であって，

$$a \in H'(X_{p(j+1)} \cdots X_{pn_p} w')$$

も成りたつ．よって，

$a \in L(s'_i)$ および $As'_i(a) = \text{shift}$

がいえる．

系 1.1

$LR(1)$ では，$f(s_i) = f(s'_i)$ のとき，$a \in L(s_i)$ であって，$As_i(a)$ がreduce p または accept とすれば，もし，$a \in L(s'_i)$ が成立すれば $As'_i(a)$ は reduce p' または accept である．p' については，$\exists b \in L(s_i)$, $As_i(b) = \text{reduce } p'$ がいえる．

系 1.1 は定理 1 の対偶である．

系 1.2

$LR(1)$ では，$f(s_i) = f(s'_i)$ であつて，$a \in L(s_i)$, かつ，$As_i(a) = \text{shift}$ のとき，s_i の a による直後続点 $Gs_i(a)$ とともに s'_i の a による直後続点 $Gs'_i(a)$ が存在し，それらは mapping f によって $\boldsymbol{Z}_0$ の同一要素に射像される．

$$f(Gs_i(a)) = f(Gs'_i(a))$$

証　明

定理 1 から $a \in L(s'_i)$, $As'_i(a) = \text{shift}$ がいえ，したがって $Gs'_i(a)$ の存在は明らかである．

$[p, j, w] \in Gs_i(a)$, $1 < j \leq n_p$,

ならば，

$\exists[p, j-1, w] \in s_i$, $X_{pj} = a$,

であって，かつ，$f(s_i) = f(s'_i)$ および定理 1 から，

$\exists[p, j-1, w'] \in s'_i$, $X_{pj} = a$,

したがって，$[p, j, w'] \in Gs'_i(a)$,

また，$[q, 1, u] \in Gs_i(a)$, $X_{q1} = a$ ならば，

$\exists[p, j, w] \in s_i$, $1 \leq j < n_p$, $\exists \alpha \in V^*$, $X_{p(j+1)} \Rightarrow A_q \alpha$

がいえる．

一方，$f(s_i) = f(s'_i)$ から，$\exists[p, j, w'] \in s'_i$

したがって，$a \in L(s'_i)$, $As'_i(a) = \text{shift}$ に結びついた行先状態 $Gs'_i(a)$ に $[q, 1, u']$ なる形の部分状態が含まれなくてはならない．よって $f(Gs_i(a)) \subseteq f(Gs'_i(a))$ であって，逆も同様にいえる．

系 1.3

$LR(1)$ では，$f(s_i) = f(s'_i)$ のとき，

$Y \in S(s_i)$ ならば $Y \in S(s'_i)$,

かつ，$f(Gs_i(Y)) = f(Gs'_i(Y))$

である．

これは，系 1.2 の証明から明らかである．

定 義

$f(s_i)\neq f(s'_i)$ であって，s_i の $Y\in S(s_i)$ による直後続点と s'_i の $Y\in S(s'_i)$ による直後続点が，

$$f(Gs_i(Y))=f(Gs'_i(Y))=Z_k,\quad Z_k\in \boldsymbol{Z}_0,$$

を満足するとき，s_i と s'_i は mapping f によって Z_k に併合されるという．また，Z_k を s_i と s'_i の mapping f による $\boldsymbol{Z}_0$ での併合点という．

定 義

$f(s_{i1})=f(s_{i2})=\cdots=f(s_{ie})=Z_k$，$Z_k\in\boldsymbol{Z}_0$ のとき，Z_k を $s_{i1}, s_{i2}, \cdots, s_{ie}$ の mapping f による $\boldsymbol{Z}_0$ での多重点という．

定 義

生成規則 $A_p\rightarrow X_{p1}X_{p2}\cdots X_{pnp}$，および K-状態列 $s^*, s_1, \cdots, s_{np}$ があって，

s_1 は s^* の X_{p1} による直後続点，

s_2 は s_1 の X_{p2} による直後続点，

……

s_{np} は s_{np-1} の X_{pnp} による直後続点

とし，$a\in L(s_{np})$，$As_{np}(a)=$reduce p であるとき，この K-状態列を a による p 生成列という．

定 義

$s_1, \cdots, s_i, s_{i+1}, \cdots, s_{i+np}$ なる K-状態列で，$1<l\leq i+n_p$ について s_l は s_{l-1} の Y_l による直後続点とする．ただし，Y_i はいずれも V の要素とする．

この K-状態列の部分列 $s_i, s_{i+1}, \cdots, s_{i+np}$ が端記号 a による p 生成列であって，$Gs_i(A_p)=s^*$ を満足し，かつ，$a\in L(s^*)$ のとき，この K-状態列を a による p 可約な列という．また K-状態列 $s_1, \cdots, s_i$，s^* をその還元列（生成規則 p による）という．

定 理 2

$LR(1)$ で，端記号 a について p 可約，かつ，端記号 b について p' 可約な K-状態列 $s_0, s_1, \cdots, s_i$ があって，mapping f によるその射像の列を，

$$Z_0, Z_1, \cdots, Z_i \qquad (5.1)$$

とし，Z_i が s_i と s'_i の多重点であったとする．さらに，

$$AG_{s_i-np}(A_p)(a)=\text{shift},$$

$$a\in L(s'_i),\ As'_i(a)=\text{reduce } p',\ p\neq p'$$

のとき，a による p' 可約な列 $s_0, s'_1, \cdots, s'_i$ で，そのおのおのの要素の mapping f による射像の列が式(5.1)と等しく，かつ，$Gs'_{i-np'}(A_{p'})=s^{**}$，$As^{**}(a)=$shift となるような K-状態列は存在しない．

ただし，s_0 は出発状態とする．

証 明

a による p' 可約な K-状態列 $s_0, s'_1, \cdots, s'_i$ で，そのおのおのの要素が mapping f によって式(5.1)と等しく，$Gs'_{i-np'}(A_{p'})=s^{**}$，$As^{**}(a)=$shift となるものが存在したとすると，$Z_{i-np'}$ は $s_{i-np'}$ と $s'_{i-np'}$ との mapping f による多重点であるか，$s_{i-np'}=s'_{i-np'}$ となる．

前者の場合，$Gs_{i-np'}(A_{p'})=s^*$ とするとき，系 1.3 によって，$f(s^*)=f(s^{**})$

定理 1 により，$a\in L(s^*)$，かつ，$As^*(a)=$shift となる．これは a による p 可約列 $s_0, s_1, \cdots, s_i$ が同時に a による p' 可約列であることを示しており，

$$As_i(a)=\text{reduce } p'$$

となって $LR(k)$ の定義と矛盾する．

$s_{i-np'}=s'_{i-np'}$

の場合も同じように矛盾がいえる．

系 2.1

$LR(1)$ で，ある端記号 a および b についてそれぞれ p および p' 可約な K-状態列 $s_0, s_1, \cdots, s_i$ があってその mapping f による射像列を $Z_0, Z_1, \cdots, Z_i$ とし，Z_i が s_i と s'_i との mapping f による多重点とする．

いま，$Gs_{i-np}(A_p)=s^*$，$As^*(a)=$shift，

$Gs_{i-np'}(A_{p'})=s'^*$，$As'^*(b)=$shift，

であって，$a\in Ls'_i(a)$，$As'_i(a)=$reduce p' とするとき，上記の Z-状態列に対し，mapping f による s'_i を導く併合点は部分列 $Z_{i-r^*+1}, \cdots, Z_i$ のなかに含まれる．ここでは，$p'\neq p$，かつ，$r^*=\max(n_p, n_{p'})$ とする．

なお，Z-状態列 $Z_0, Z_1, \cdots, Z_i$ に対して，mapping f による s'_i を導く併合点とは，K-状態列 s'_{i-l}，$s'_{i-l+1}, \cdots, s'_i$ でとなり合う2要素間で右側のものが左側のものの直後続点という関係にあり，かつ，

$$f(s'_{i-m})=Z_{i-m},\ 0\leq m\leq l-1,\ f(s'_{i-l})\neq Z_{i-l}$$

なるものについての Z_{i-l+1} のことをいう．

証明は定理 2 から明らかである．

系 2.2

ある端記号 a について，p_1 可約な K-状態列 $s_0, s_1, \cdots, s_i$ があって，生成規則 p_1 による還元列 $s_0, s_1, \cdots, s_{i-np_1}$，$s^{(1)*}$ がまた a について p_2 可約とする．これをくり返し，一般に m 回の還元列，$s_0, s_1, \cdots, s_{i-r}$，$s^{(m)*}$ がもはや a について可約でなく，

$a\in L(s^{(m)*})$，$As^{(m)*}(a)=$shift とする．ただし r は

$$r=n_{p_1}+n_{p_2}+\cdots+n_{pm}-m+1$$

なるものとする．

このとき，もとの K-状態列についての mapping f

による射像の列 $Z_0, Z_1, \cdots, Z_i$ の Z_i が s_i と s'_i の f による多重点であって，$a \in L(s'_i)$，$As'_i(a)$=reduce p_1'，$p_1' \neq p_1$ であって，$b \in L(s_i)$，$As_i(b)$=reduce p_1' ならば，この Z-状態列に対し，f による s'_i を導く併合点は，部分列 $Z_{i-r^*+1}, Z_{i-r^*+2}, \cdots, Z_i$ のなかに含まれる．ただし r^* は $r^*=\max(r, r')$，かつ，$r'=n_{p_1}'+\cdots+n_{p_1}'-m'+1$ であって，もとの K-状態列が b についても p_1' 可約であり，その還元列 $s_0, s_1, \cdots s_{i-n_{p_1}'}, s'^{(1)*}$ がまた b について p_2' 可約とし，これをくり返し $p_{m'}'$ による還元列がもはや可約でなく，$b \in L(s'^{(m')*})$，$As'^{(m')*}(b)$=shift が成立するものとする．

証明は定理 2 および系 2.1 から明らかである．

6. Z-状態を用いた $LR(1)$ parser

5章で示した $LR(1)$ の諸性質に基づいて Z-状態を基礎にした $LR(1)$ parser について述べる．Z-状態を基礎にした parser については $LR(k)$ の一般の形（文献 3)）での例がある．そこでは，$LR(k)$ 一般を扱い，ある意味での左文脈その他の道具だてのもとでこれを実現している．ここでは $LR(1)$ parser ということに限って取り上げ，$LR(1)$ の諸性質に基づいた手法を示す．

6.1 parsing table

3.1 に示された Knuth および Korenjak による K-状態に基づいた $LR(1)$ parser を基礎にして話しを進める．

(i) $\boldsymbol{Z}_0=\{f(s_i)|\forall s_i \in \boldsymbol{S}\}$ によって，K-状態から Z-状態を求める．

(ii) $\forall Z_k \in \boldsymbol{Z}_0$，$Z_k=f(s_{i_1})=\cdots=f(s_{i_n})$，について Z_k に対する Lookahead Symbol 集合 $L(Z_k)$ を，

$$\boldsymbol{L}(Z_k)=\bigcup_{l=1}^{n} L(s_{i_l})$$

とする．

また，$a \in \boldsymbol{L}(Z_k)$ なる a に対する Action の集合 $\boldsymbol{A}_{Z_k}(a)$ を，

$$\boldsymbol{A}_{Z_k}(a)=\bigcup_{l=1}^{n} As_{i_l}(a)$$

とする．

Z_k に対する Stack Symbol の集合 $\boldsymbol{S}(Z_k)$ を，

$$\boldsymbol{S}(Z_k)=S(s_{i_1})=\cdots=S(s_{i_n}),$$

とする．そして，$Y \in \boldsymbol{S}(Z_k)$ についての行先状態 $\boldsymbol{G}_{Z_k}(Y)$ を，

$$\boldsymbol{G}_{Z_k}(Y)=f(Gs_{i_1}(Y))=\cdots\cdots=f(Gs_{i_n}(Y)),$$

とする．

ここで，$\boldsymbol{L}(Z_k)$ についてみると，定理 1 によって，$a \in L(s_{i_1})$，$As_{i_1}(a)$=shift ならば，

$a \in L(s_{i_l})$，$As_{i_l}(a)$=shift，$2 \leq l \leq n$，

であるから，$b \in L(s_{i_l})$，$As_{i_l}(b)$=reduce p であって，$b \notin L(s_{i_1})$ なるものだけが余分に $L(s_{i_1})$ に付加されたものである．

$a \in \boldsymbol{L}(Z_k)$ に対する $\boldsymbol{A}_{Z_k}(a)$ が shift でないときは Action が複数個対応することもありうる．

さらに，$\boldsymbol{S}(Z_k)$ および $\boldsymbol{G}_{Z_k}(Y)$ についての定義の仕方の妥当性は定理 1 およびその系からいえる．

6.2 parsing の手続き

3.2 で述べたところの手続きで K-状態を表わす箇所を Z-状態のそれで置き換えることによって，ここでは基本的な部分は，そのまま踏襲するものとする．

(i) parsing の過程で stack の状態が，

$$\begin{array}{l} Z_0 Z_1 \cdots Z_m \\ \bullet X_1 \cdots X_m \end{array} \Big| a \qquad (6.2.1)$$

であるとき，$a \in \boldsymbol{L}(Z_m)$ であって，$r \geq 1$ に対し，

$$\boldsymbol{A}_{Z_m}(a)=\{\text{reduce } p_{m_1}, \cdots, \text{reduce } p_{m_r}\} \qquad (6.2.2)$$

のときは stack の状態から唯一の Action を選び出す．

そのために parser はいくつかの新しい stack を用意する．

そこで式 (6.2.1) に現われるこれまでのものを M-stack と呼ぶことにする．このほかに新しく R-stack として次のような機構のものを用いる．これは push-down stack 形式（リスト形式とみてもよい）のもので $\boldsymbol{A}_{Z_m}(a)$ が式 (6.2.2) で表わされるものであるときその stack の内容が push-down され，逆に式 (6.2.2) の右辺に示される Action がいずれも妥当でないとわかったとき pop-up が起こる．R-stack の各要素は内容として式 (6.2.2) の右辺とその何番目かの要素である p_{m_k} をさすポインタ，M-stack の先頭から還元のくり返しによって達する M-stack 中の特定要素までの距離および下記の C-stack 中から除去される要素列からなる．

C-stack は parsing のある過程で M-stack の特定の箇所から連結して使用する M-stack と同質の push-down stack である．

$\boldsymbol{A}_{Z_m}(a)$ が式 (6.2.2) で表わされるときは (a)

以下を行なう．また $A_{Z_m}(a)$ が shift または accept のときは(ii)を行なう．

(a) R-stack が初期条件にあれば式 (6.2.2) の右辺およびその最初の要素 p_{m_1} を指すポインタ，さらに M-stack の先頭からの還元列長 0 を R-stack に格納する．また，このとき C-stack は初期条件にあり，空であるが，その基底と Z_m の位置とで連結する．

R-stack が空でないときは，その先頭要素中の M-stack の先頭からの還元列長を取り出したのち R-stack を push-down する．そして R-stack に式 (6.2.2) の右辺，p_{m_1} をさすポインタおよび先に取り出しておいた M-stack の先頭からの還元列長を格納する．

parsing では，このほかに，還元木（分枝のない直線的なもの）と呼ぶものを用いる．また(a)の手続きで，R-stack が初期のときから始めた場合，parser はそれを認識しているものとする．

(b) R-stack の先頭要素中のポインタが p_{m_k} をさしているとする．ただし，$1 \leq k \leq r$ である．

C-stack の先頭から np_{m_k} 隔たった要素 Z_l についてみる．

R および C-stack の初期の状態から始めるときは，C-stack は M-stack の先頭 Z_m の位置で連結しているから C-stack の先頭要素は Z_m とみなし，以下 M-stack の内容が C-stack のそれとなる．

まず，

$$Ap_{m_k} \in \boldsymbol{S}(Z_l),\quad a \in \boldsymbol{L}(\boldsymbol{G}_{Z_l}(Ap_{m_k})) \qquad (6.2.3)$$

が成立するかをみる．これが成立しないときは(d)を行なう．式 (6.2.3) が成立するときは還元木に新しい節点を付加し，前の端節点と枝で結び，新しい節点に Ap_{m_k} および生成規則番号 p_{m_k} を格納する．

次に，C-stack の先頭から M-stack との連結点までの距離を h (C-stack の初期の状態から始めたときは $h=0$ である) として，$np_{m_k} > h$ ならば，その差 $np_{m_k} - h$ だけ連結点を M-stack の左側に移動させる．連結点の右側にある M-stack の内容はそのまま残される．R-stack の先頭要素中の M-stack からの還元列長に $np_{m_k} - h$ の値を加える．しかるのち，C-stack について先頭から M-stack の連結点に至るまでの要素を取り出し R-stack の先頭要素に移し，C-stack からはそれらを除去する．また新たに C-stack の先頭には $\boldsymbol{G}_{Z_l}(Ap_{m_k})$，$Ap_{m_k}$ を上下段に格納し，続いて(c)を実行する．

$np_{m_k} \leq h$ のときは，単に C-stack の先頭から np_{m_k} 個の要素を取り出し R-stack の先頭要素に移してからそれらを C-stack から除去する．また C-stack の先頭には $\boldsymbol{G}_{Z_l}(Ap_{m_k})$，$Ap_{m_k}$ を上下段に格納し，続いて(c)を実行する．

(c) 式 (6.2.3) が成立し，$A\boldsymbol{G}_{Z_l}(Ap_{m_k})(a)$ が式 (6.2.2) の形式のときは(a)をくり返し行なう．

$A\boldsymbol{G}_{Z_l}(Ap_{m_k})(a)$ が shift または accept のときは還元木に現われた parsing は確定したのであって，この還元木を出力する．また，M-stack と C-stack との連結点より右側にある M-stack の要素を除去するとともに，連結点の右側に C-stack に残された要素を移す．

M-stack を除く他の stack を初期条件にもどして(ii)を行なう．

(d) R-stack の先頭要素中のポインタを count up して次の p_{m_k} を指すようにする．

$k \leq r$ ならば(b)を，また $k > r$ ならば(e)を行なう．

(e) R-stack および C-stack の先頭要素を pop-up によって除去するとともに，還元木の端節点も切り離して除去する．R-stack が空でないならば R-stack の先頭から 2 番目の要素中にある M-stack の先頭からの還元列長に示される位置に C-stack との連結点を移動させる．R-stack の先頭から 2 番目の要素がないときは，連結点を M-stack の先頭の位置にもどす．R-stack の先頭要素中にあるポインタによってさされる p_{m_k} による還元のとき C-stack から移してきた要素列を C-stack の頭部にもどす．そののち(d)を行なう．

(e)の初めで R-stack を pop-up した結果，それが空になったときは入力文章の誤りを告げ，parsing を中断する．

(ii) $A_{Z_m}(a)$=shift または $A_{Z_m}(a)$=accept のときは 3 章に述べたと同じ手続きをとる．

6.3 制限された $LR(1)$ の場合

6.2 に述べた手続きで C-stack を用いたり，R-stack 中の要素に C-stack からの要素列を移してきて保持するといったことは，$LR(1)$ における生成規則中に，

$A\to\varepsilon$, (ただし, ε は空を示す)

なるタイプのものが存在することに由来している.

生成規則中に, $A\to\varepsilon$ タイプのものが存在しないか, または存在しても, このタイプの生成規則によって連続的に還元が行なわれることがないことが保証されているならば, 6.2 で述べた手続きの還元の過程では C-stack にはいる要素は, 実はいつも1個と限られている. このことからC-stack とそれに伴う手続きは必要とせず, また R-stack も簡単になる. さらに, parsing table 中に現われる任意の Z についての Action のうちで式 (6.2.2) の形式のもののすべてが,

$$A_Z(a)=\{\text{reduce } p\}$$

の形をしているとき, すなわち, Action が唯一で, 判別の手続きが不用のときは, parser は 3.2 で述べたところのものとまったく同じ機構のものでよい. ただし, この場合は 3.2 で K-状態 s を表わしているところは Z-状態のそれに置き替わるものとする.

表 6.1 文法 G に対する K-状態に基づいた parsing table

State name	s	s'-s	Lookahead Symbol	Action	Stack Symbol	Goto State
s_0	00#	10# 20*bc*	*a*	shift	S_0 S_1 *a*	s_1 s_2 s_3
s_1	01#		#	accept		
s_2	11#	30*a* 40*a*	*b* *c*	shift shift	*A* *b* *c*	s_4 s_5 s_6
s_3	21*bc*	70*bce* 80*bce*	*e*	shift	*C* *e*	s_7 s_7
s_4	12#	20*cd*	*a*	shift	S_1 *a*	s_9 s_{10}
s_5	31*a*		*a*	reduce 3		
s_6	41*a*		*a*	reduce 4		
s_7	22*bc* 71*bce*		*b* *c* *e*	reduce 2 reduce 2 shift	*e*	s_{11}
s_8	81*bce*		*b* *c* *e*	reduce 8 reduce 8 reduce 8		
s_9	13#	50# 60#	*c* *d*	shift shift	*B* *c* *d*	s_{12} s_{13} s_{14}
s_{10}	21*cd*	70*cde* 80*cde*	*e*	shift	*C* *e*	s_{15} s_{16}
s_{11}	71*bce*		*b* *c* *e*	reduce 7 reduce 7 reduce 7		
s_{12}	14#		#	reduce 1		
s_{13}	51#		#	reduce 5		
s_{14}	61#		#	reduce 6		
s_{15}	22*cd* 71*cde*		*c* *d* *e*	reduce 2 reduce 2 shift	*e*	s_{17}
s_{16}	81*cde*		*c* *d* *e*	reduce 8 reduce 8 reduce 8		
s_{17}	72*cde*		*c* *d* *e*	reduce 7 reduce 7 reduce 7		

6.4 例

6.2 で述べた手続きによる $LR(1)$ parser での parsing table は 6.1 で示したが, 3.1 で述べられた K-状態に基づく parsing table との比較例を示す. $LR(1)$ 文法およびその K-状態に基づく parsing table の例として文献 2) で Korenjak によってあげられた

表 6.2 文法 G に対する Z-状態に基づいた parsing table

State name	Z	$f^{-1}(Z)$	Lookahead Symbol	Action	Stack Symbol	Goto State
Z_0	00	s_0	*a*	shift	S_0 S_1 *a*	Z_1 Z_2 Z_3
Z_1	01	s_1	#	accept		
Z_2	11	s_2	*b* *c*	shift shift	*A* *b* *c*	Z_4 Z_5 Z_6
Z_3	21	s_2, s_{10}	*e*	shift	*C* *e*	Z_7 Z_8
Z_4	12	s_4	*a*	shift	S_1 *a*	Z_9 Z_3
Z_5	31	s_5	*a*	reduce 3		
Z_6	41	s_6	*a*	reduce 4		
Z_7	22 71	s_7, s_{15}	*b* *c* *d* *e*	reduce 2 reduce 2 reduce 2 shift	*e*	Z_{10}
Z_8	81	s_8, s_{16}	*b* *c* *d* *e*	reduce 8 reduce 8 reduce 8 reduce 8		
Z_9	13	s_9	*c* *d*	shift shift	*B* *c* *d*	Z_{11} Z_{12} Z_{13}
Z_{10}	72	s_{11}, s_{17}	*b* *c* *d* *e*	reduce 7 reduce 7 reduce 7 reduce 7		
Z_{11}	14	s_{12}	#	reduce 1		
Z_{12}	51	s_{13}	#	reduce 5		
Z_{13}	61	s_{14}	#	reduce 6		

ものをここでも用いることにする.

$LR(1)$ 文法例

$G=(\{S_0, S_1, A, B, C\}, \{a, b, c, d, e\}, P, S_0)$

1. $S_0 \to S_1AS_1B$,
2. $S_1 \to aC$,
3. $A \to b$,
4. $A \to c$
5. $B \to c$
6. $B \to d$
7. $C \to Ce$
8. $C \to e$

これに対して，6.1で述べられた parsing table は表6.2に示すものとなる．この例によっても明らかなように表 6.1 に比べて表 6.2 は状態数は減少している．

しかも，この文法例では，$A \to \varepsilon$ なる形の生成規則が存在せず，また，どの Z-状態においても同一 Lookahead Symbol に対して単一の Action しか対応していないから 6.3 で述べたように 3.2 で示した parser を用いて（ただし，このときは Z-状態を K-状態とみる）処理しうることを示している．

7. あとがき

本論で示した手法は $LR(1)$ についてのみいえる文法の諸性質に基づいている．

このことは $LR(1)$, $k>1$ なる一般についてはこの手法そのまま適用することはできない．一般の k に対する $LR(k)$ については特別の条件下で適用しうるが，今後は任意 $LR(k)$ の諸性質を考慮することで条件を弛めることあるいは手法の修正を考えたい．

$LR(1)$ に限ってみるとき，本手法はその特性をとらえた点で有利にはたらきうること，実用上の適用範囲としては十分の広がりをもつものであるとみることができる．

謝辞 本論文をまとめるにあたり，多くの援助をいただいた当所電子機器研究部首藤勝氏に深く感謝をします．

参 考 文 献

1) D.E. Knuth: "On the translation of languages from left to right", *Inform. Contr.*, No. 8 (Dec. 1966).
2) A. J. Korenjak: "A practical Method for Construting $LR(k)$ Procesors", *CACM*, Vol. 12, No. 11 (1969).
3) 林 達也: "CFG-PL 変換について", 情報処理, Vol. 12, No. 3 (1971).
4) J. Feldman and D. Gries: "Translator writing system", *CACM*, Vol. 11, No. 2 (1968).
5) R. W. Floyd: "Bounded Context syntactic analysis", *CACM*, Vol. 7. No. 2 (1964).

(昭和46年9月11日受付，12月14日再受付)