Information Processing in Japan Vol. 12, 1972

A Practical Method of Constructing LR (1) Parsers and on the Properties Peculiar to LR (1) Grammars

SHOJI SEKIMOTO*

1. *Introduction*

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 their parsing tables. So, many cases of giving small and practical parsers for *LR* (*k*) grammars have been reported. For example, A. K. Korenjak has decreased the size of the processor by partitioning a given grammar into a number of smaller part. A next example has been given by Hayashi. Using a simple mapping of states of Knuth which will be used in this paper, the number of entry states has been decreased. His graphical method may need extra informations of the left contexts and path functions to parse a sentence. For another example, D. Pager has given a solution to the problem of the minimization of the number of Knuth parsers (DPDA).

We will show that some properties peculiar to *LR* (1) grammars exist, which may not hold in *LR* (*k*) grammars ($k>1$), and that using these properties, a method of constructing practical *LR* (1) parsers driven only by a reduced parsing table is introduced.

2. *Knuth's Method for LR (k) Grammars*

We show the method of constructing *LR* (*k*) parsers given by Knuth and Korenjak.

We suppose that a context free grammar $G=(V_N, V_T, P, S_0)$ is given, where the number of production rules in P is π and they are numbered from 1 to π. The p-th production rule is represented as $A_p \rightarrow X_{p1}\ldots X_{pn_p}(n_p \geq 0)$.

Then we give the k-augumented grammar G' of G, such as

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

$V_T'=V_T \cup \{\#\}(\# \notin V_N \cup V_T)$ and $P'=P \cup \{S_0' \rightarrow S_0\#^k\}$.

We assign a number O to $S_0' \rightarrow S_0\#^k$ and call $\#$ the endmarker. Rewriting G' to $G=(V_N, V_T, P, S_0)$, we will use this G hereafter. Definition: Following the two functions are defined for

$G=(V_N, V_T, P, S_0)$ and $k \geq 0$.

(a) $H(\alpha)=\{t \in V_T^k \mid \alpha \Rightarrow t\beta$ for some $\beta \in V^*\}$

This paper first appeared in Japanese in Joho-Shori (Journal of the Information Processing Society of Japan), Vol. 13, No. 3 (1972), pp. 170~178.

* Mitsubishi Electric Corporation

(b) $H'(\alpha)=\{t\in V_T{}^k \mid \alpha\Rightarrow t\beta$ for some $\beta\in V^*$, but $\alpha\Rightarrow t\beta$ does not contain a step of the form $A\gamma\Rightarrow\gamma$ (due to $A\rightarrow\varepsilon$ in $P)\}$

Each state s of the $LR(k)$ parser by Knuth consists of triplets $[p, j, w]$ called 'partial states' and the parsing table $T(G)$ [3] which drives the $LR(k)$ parser is generated for a given $LR(k)$ grammar G as follows:

(i) First, we define the start state $s_0=\{[0, 0, \sharp^k]\}$.

(ii) Second, we define the set s' for a state s as follows: s' is the smallest set satisfying the next equation.

$s'=s\cup\{[q, o, x]\mid[p, j, w]\in s$ such that $j<n_p$, $X_{p(j+1)}A_q$ and $x\in H(X_{p(j+2)}\ldots X_{pn_p}w)\}$

(iii) Using s', the set of lookahead strings of s, i. e. $L(s)$ and an action $As(y)$ for each $y\in L(s)$ are determined as follows:

(1) $y\in L(s)$ and $As(y)=$shift if $[p, j, w]\in s'$, $j<n_p$ and $y\in II'(X_{p(j+1)}\ldots X_{pn_p}w)$.

(2) $y\in L(s)$ and $As(y)=$reduce p if $[p, n_p, y]\in s'$ and $p\neq 0$.

(3) $y\in L(s)$ and $As(y)=$accept if $y=\sharp^k$ and $[0, 1, \sharp^k]\in s'$.

(vi) Next, the set of stack symbols of s, i. e. $S(s)$, and goto symbol $Gs(Y)$ for each $Y\in S(s)$ are defined as follows:

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

$Gs(Y)=\{[p, j+1, w]\mid[p, j, w]\in s',\ Y=X_{p(j+1)}\}$

(v) Finally we choose a $Gs(Y)$ for some s and Y, which has been given through the above process and is not yet entered in $T(G)$, and repeat above steps for the state $Gs(Y)$ until the parsing table $T(G)$ is completed.

Using the table $T(G)$ and a pushdown stack with 2 tracks for control, the $LR(k)$ parser examines an input string from left to right and attempt to reduce it to S_0.

We denote the position of the input tape and the contents of the stack by the notation:

$$\begin{array}{l} s_0s_1\ldots s_n \\ \bullet x_1\ldots x_n \end{array}\ \Big|\ a_1\ldots a_kt \qquad (*)$$

Let the original input string be $xa_1\ldots a_k{}^t\in V_T{}^*\sharp^k$, then this notation indicates that:

(1) substring x was reduced to $x_1\ldots x_n$,

(2) the lookahead string is $a_1\ldots a_k$,

(3) t is not yet handled at this point and

(4) the upper track of the stack contains the state names, $s_0, s_1,\ldots,s_n$.

The parser operates as follows.

Initially, the start state s_0 is placed on the stack and the fitst input string with length k is the initial lookahead string. The symbol $\bullet$ is just the start marker.

After this initial operation, assume that the parser has reached the configuration shown above, then:

(i) The string $a_1 \ldots a_k$ is examined whether it is in $L(s_n)$ or not. According to the result, the parser takes one of the following operations.

(a) If $a_1 \ldots a_k$ is not in $L(s)$ then this is rejected by the reason that the input string is not in $L(G)$.

(b) If $As_n(a_1 \ldots a_k)$ is shift then a_1 is pushed on the lower track.

$$\begin{matrix} s_0 s_1 \ldots s_n \\ \bullet x_1 \ldots x_n a_1 \end{matrix} \Bigg| \; a_2 \ldots a_k t$$

(c) If $As_n(a_1 \ldots a_k)$ is reduce p then the rightmost n_p elements in the both tracks are popped off and Ap is pushed on the lower track.

$$\begin{matrix} s_0 s_1 \ldots s_{n-n_p} \\ \bullet x_1 \ldots x_{n-n_p} A_p \end{matrix} \Bigg| \; a_1 \ldots a_k t$$

(d) If $As_n(a_1 \ldots a_k)$ is accept then this process terminates.

(ii) Next, if $As_n(a_1 \ldots a_k)$ is shift or reduce p then a goto state $Gs_n(a_1)$ or $Gs_{n-n_p}(A_p)$ is respectively determined by the parsing table and it is placed in the rightmost empty position of the upper track.

By the operation (ii), the configuration of the stack returns to (*) and the above operation is repeated until the input is found out as erroneous, or $As_F(\#^k)$=accept occurs at the configuration of

$$\begin{matrix} s_0 s_F \\ \bullet s_0 \end{matrix} \Bigg| \; \#^k$$

3. *Definition of Z-state*

We will define a mapping from Knuth's states (K-states) into Z-states. We assume that a set of all K-states of a given $LR(k)$ grammar G is presented by $\boldsymbol{S}=\{s_i \mid 0 \leq i \leq n\}$. Then we define a mapping $f(s_i)$ and a set of its images, as follows:

$$f(s_i)=\{[p, j] \mid [p, j, w] \in s_i\}$$
$$\boldsymbol{Z}_0=\{f(s_i) \mid s_i \in \boldsymbol{S}\}$$

We call each element of $\boldsymbol{Z}_0$ to be Z-state.

4. *Some Properties of $LR(1)$ Grammars*

We show that there are some properties peculiar to $LR(1)$ grammars. Now we present some theorems without proofs.

Theorem 1: Let a given grammar be $LR(1)$ and assume that $f(s_i)=f(s_j)$ is satisfied for some K-states s_i and s_j.

Then if the conditions $a \in V_T$ and $As_i(a)$=shift hold, $a \in L(s_j)$ and $As_j(a)$=shift are both satisfied.

Corollary 1: Let a given grammar be $LR(1)$ and assume that $f(s_i)=f(s_j)$ is satisfied for some K-states s_i and s_j.

Then if the condition $Y\in S(s_i)$ holds for some $Y\in V$, $Y\in S(s_j)$ and $f(Gs_i(Y))=f(Gs_j(Y))$ are both satisfied.

Definition: For some K-states s_i and s_j, if the conditions $f(s_j)\neq f(s_j)$ and $f(Gs_i(Y))=f(Gs_j(Y))=Z_k$ hold for $Z_k\in\boldsymbol{Z}_0$ and $Y\in S(s_i)\cap S(s_j)$ then s_i and s_j are said to be 'annexed to Z_k by mapping f' and Z_k is said to be 'annexed point of s_i and s_j by mapping f'.

Definition: If the equation $f(s_{i_1})=f(s_{i_2})=\ldots=f(s_{i_l})=Z_k$ holds for some K-states $s_{i_1}, s_{i_2},\ldots, s_{i_l}$, then Z_k is said to be 'heap point of $s_{i_1}, s_{i_2},\ldots, s_{i_l}$ by mapping f'.

Definition: A sequence of K-states, $s_0, s_1,\ldots, s_i,\ldots, s_{n_{p+1}}$, is given and if it satisfies the following conditions:

1) s_0 is the start state,
2) $s_l=Gs_{l-1}(Y_l)$ for $1\leq l\leq i+n_p$ and $\exists Y_l\in V$,
3) $As_{i+n_p}(a)=$reduce p for $a\in L(s_{i+n_p})$ and $A_p\rightarrow X_{p1}\ldots X_{pn_p}$, where $X_{p1}=Y_{i+l}$ for $1\leq l\leq n$,
4) $a\in L(s^*)$ for $Gs_i(Ap)=s^*$,

then we call the K-states sequence $s_0, s_1,\ldots, s_i,\ldots, s_{i+n_p}$ to be 'p-reducible for a' and also the sequence $s_o, s_1,\ldots, s_i, s^*$ to be 'reduced sequence (by p)'.

Theorem 2: Let $\zeta=\{s_j\}0\leq j\leq i$ and $\zeta'=\{s_j'\}0\leq j\leq i$ be two K-states sequences such that (1) $f(sj)=f(sj')$ for $0\leq j\leq i$,
(2) ζ is p-reducible by a and
(3) ζ' is p'-reducible by a.
If $A_{G_{sn-n_p}}(A_{np})$ (a)$=A_{G_{sn-n_p}'}(A_{np}')$ (a)=shift,
then $p=p'$.

Corollary 2: Assume that
(1) $s_0, s_1,\ldots, s_i$: K-states sequence in a given $LR(1)$ parser,
(2) $Z_0, Z_1,\ldots, Z_i$: Z-states sequence such that $f(s_j)=Z_j$ for $1\leq j\leq i$,
(3) the above K-states sequence is p_1-reducible for $a(\in L(s_i))$, its p_1-reduced sequence $s_0,\ldots, s_{i-n_p}, s^{(1)*}$ is also p_2-reducible for a and repeating this process m times we get K-state sequence $s_0, s_1,\ldots, s_{i-r}, s^{(m)*}$ which is no longer reducible for a,
where $r=n_{p_1}+n_{p_2}+\ldots+n_{p_m}$,
(4) $A_{s^{(m)*}}$(a)=shift.
If Zi is the heap point of s_i and s_j by f and if $a\in L(s_j)$, As_j (a)= reduce p_1' and $p_1'\neq p_2$ hold, then the annexed point leading to s_i by f is included in the subsequence $Z_{i-r+1}, Z_{i-r+2},\ldots, Z_i$.

5. *LR(1) Parser Using Z-states*

We will give the parsing table $T(G)_z$ for a given $LR(1)$ grammar, using Z-states instead of K-states.

(i) We get Z-states from K-states by $\boldsymbol{Z}_0=\{f(s_i)\,|\,s_i\in\boldsymbol{S}\}$.

(ii) The set $\boldsymbol{L}(Z_k)$ of lookahead symbols for $Z_k \in \boldsymbol{Z}_0$ is defined by $\boldsymbol{L}(Z_k)=\bigcup_{l=1}^{n} L(s_{i_l})$, where $Z_k=f(s_{i_1})=\ldots=f(s_{i_n})$.

(iii) The set A_{Z_k} (a) of actions for $a \in \boldsymbol{L}(Z_k)$ is defined by $A_{Z_k}(a)=\bigcup_{l=1}^{n} A_{s_{i_l}}$ (a) for all s_{i_l} such as $f(s_{i_l})=Z_k$.

(iv) The set $S(Z_k)$ of stack symbols for Z_k is defined by $S(Z_k)=S(s_{i_1})=\ldots=S(s_{i_n})$. The goto state $G_{Z_k}(Y)$ for $Y \in S(Z_k)$ is defined by
$G_{Z_k}(Y)=f(G_{s_{i_1}}(Y))=\ldots=f(G_{s_{i_n}}(Y))$.

The definitions $\boldsymbol{L}(Z_k)$, $A_{Z_k}(a)$, $S(Z_k)$ and $G_{Z_k}(Y)$ are well-defined in consideration of Theorem 1 and its corollary.

Now, we assume that there is the same main stack as mentioned above, except that K-states are replaced by Z-states as follows:

$$\begin{array}{l|l} Z_0 Z_1 \ldots Z_m & \\ \bullet x_1 \ldots x_m & \mathrm{a} \end{array}$$

By the results of the previous section, we may decide a next action or actions by means of $T(G)_z$.

If $a \in \boldsymbol{L}(Z_m)$ and $A_{Z_m}(a)$=shift in the above configuration then the next action is shift and the next goto atate is $G_{Z_m}(a)=Z_{m+1}$.

If $a \in \boldsymbol{L}(Z_m)$ and $A_{Z_m}(a)$ is a set of reduce $p_i(1)$ then there should be at most one sequence of reductions $p_{i_l}{}^{(1)},\ldots, p_{i_n}{}^{(n)}$ such that after the completion of the successive reductions,

$a \in \boldsymbol{L}(Z^*)$ and $A_Z{}^*(a)$=shift or accept

are satisfied for the recent goto state Z^* which is in the top of the stack and may be got by the result of the last reduction. If there is such a sequence of reductions then it is the true sequence of actions else the original input a is false.

If $a \notin \boldsymbol{L}(Z_m)$, the input a is false.

We can construct $LR(1)$ parsers using $T(G)_Z$ and 3 push-down stacks.

6. *Conclusion*

The method proposed in this paper is based on the properties peculiar to $LR(1)$ grammars respecting the image of a simple mapping f of K-states. After this work, we have got a family of mappings, which is a proper subset of equivalence relations of all K-states of a given $LR(k)$ grammar. We call each mapping in the family analytical and this family of mappings is a direct extension of f in this paper to the case of $LR(k)$ grammars.

We have also got an algorithm to find the analytic functions for a given $LR(k)$ grammar, which give the minimum number of image states of K-states.

References

[1] Knuth D. E., 'On the translation of languages from left to right', *Information and Control 8*, 607-639, 1965.

[2] Pager D., 'A solution to an open problem by Knuth', *Information and Control 17*, 462-473, 1970.
[3] Korenjak A. S., 'A practical method for constructing LR (k) processors', *Comm. ACM* 12, 613-623, 1969.
[4] Hayashi T., 'Development of LR (k) analyzer', *Proc.* IPSJ 11 Nat' l Conf., 1971.
[5] Sekimoto S., *et al.*, 'Simplification of LR (k) parser', A 71-116/I 771-101, IECE, 1972.