4-25-1973

A General LR(k) Parser Building Algorithm

Thomas Joshua Sager

A GENERAL
LR (k)
PARSER
BUILDING
ALGORITHM

SAGER

THE UNIVERSITY OF NEW MEXICO
ALBUQUERQUE, NEW MEXICO 87106

Policy on Use of Theses and Dissertations

Unpublished theses and dissertations accepted for master's and doctor's degrees and deposited in the University of New Mexico Library are open to the public for inspection and reference work. *They are to be used only with due regard to the rights of the authors.* The work of other authors should always be given full credit. Avoid quoting in amounts, over and beyond scholarly needs, such as might impair or destroy the property rights and financial benefits of another author.

To afford reasonable safeguards to authors, and consistent with the above principles, anyone quoting from theses and dissertations must observe the following conditions:

1. Direct quotations during the first two years after completion may be made only with the written permission of the author.

2. After a lapse of two years, theses and dissertations may be quoted without specific prior permission in works of original scholarship provided appropriate credit is given in the case of each quotation.

3. Quotations that are complete units in themselves (e.g., complete chapters or sections) in whatever form they may be reproduced and quotations of whatever length presented as primary material for their own sake (as in anthologies or books of readings) ALWAYS require consent of the authors.

4. The quoting author is responsible for determining "fair use" of material he uses.

This thesis/dissertation by Thomas Joshua Sager has been used by the following persons whose signatures attest their acceptance of the above conditions. (A library which borrows this thesis/dissertation for use by its patrons is expected to secure the signature of each user.)

NAME AND ADDRESS	DATE

3/1/73—1M

This dissertation, directed and approved by the candidate's committee, has been accepted by the Graduate Committee of The University of New Mexico in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

A GENERAL LR(k) PARSER BUILDING ALGORITHM

Title

THOMAS JOSHUA SAGER

Candidate

DEPARTMENT OF MATHEMATICS

Department

Ray D. Johnson

Dean

April 25, 1973

Date

Committee

Edgar J. Gilbert

Chairman

R. G. Stanton

Nancy M. Moler

John Wade Ward

A GENERAL LR(k) PARSER BUILDING ALGORITHM

BY

THOMAS JOSHUA SAGER

B.A., California State College
At San Bernardino, 1969

M.A., University of New Mexico, 1971

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy in Mathematics
in the Graduate School of
The University of New Mexico
Albuquerque, New Mexico
May, 1973

ACKNOWLEDGMENT

I would like to express my gratitude to Professor Edgar J. Gilbert who has been a constant source of inspiration and to Professors Nancy M. Moler and John W. Ulrich for their helpful suggestions. I also wish to thank Linda Thorne and Carolyn Quinn for typing the final draft of this dissertation. Lastly I wish to thank Barbara, Sammy and Wallabean without whom this dissertation could never have been completed.

A GENERAL LR(k) PARSER BUILDING ALGORITHM

BY

THOMAS JOSHUA SAGER

ABSTRACT OF DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy in Mathematics
in the Graduate School of
The University of New Mexico
Albuquerque, New Mexico
May, 1973

ABSTRACT

The problem is to find an efficient algorithm that, given the productions of a context-free grammar G , will discover whether G is LR(k) for given k and if it is build an efficient parser for G . The algorithm is given in Section 8. It is essentially a synthesis of the best parts of Knuth's and DeRemer's algorithms. On simple LR(k) grammars it yields a result equivalent to DeRemer's algorithm, and like Knuth's algorithm it will work on all LR(k) grammars.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS, TERMS AND ABBREVIATIONS

1. INTRODUCTION

This dissertation is concerned with LR(k) grammars in general and in particular with the building of LR(k) parsers. Intuitively, a parse of a sentence is an indication of the derivation of the sentence. Intuitively, a grammar G is LR(k) if and only if every sentence in the language L(G) defined by G may be parsed by reading it once from left to right scanning ahead at most k symbols. Practically, this means that the LR(k) grammars are precisely those grammars for which there exists an upper bound on the time required to parse a sentence which is proportional to the length of the sentence. A more precise definition of LR(k) is given later.

Knuth (Knu 65) has shown that any LR(k) grammar may be parsed by a deterministic push-down automaton (DPDA) with the added capability of scanning the next k symbols of the input string at any time.[1] Furthermore Knuth gives an algorithm which, given the productions of a context-free grammar G , will find for given k whether G is an LR(k) grammar and if so find an LR(k) parser for G . Knuth's algorithm, however, has certain drawbacks. For one, it is impractical for any large grammar. The algorithm must do a lot of work which in the final analysis is unnecessary. The number of states and the size of the states are both much larger than necessary. Knuth mentions methods for combining states in order to decrease the size and increase the efficiency of his parser. However, the question of how best to do this, he leaves as a subject for further research.

Korenjak (Kor 69) and Pager (Pag 70) have both attempted to answer this question. Korenjak found a method of reducing the number of states through splitting the original grammar G into m small interconnected grammar's and applying Knuth's method separately to each one. Unfortunately, how best to split up a large grammar is still an open question. Pager demonstrated that known automata theory methods could be applied to the result of Knuth's algorithm in order to reduce it. But unfortunately the resulting parser although it will correctly parse any sentence in the grammar might also parse sentences not in the grammar. Pager's results must therefore also be considered impractical.

DeRemer (DeR 69 and DeR 71) found a practical modification of Knuth's algorithm which, for a large subclass of LR(k) grammars which he calls simple LR(k) (SLR(k)) grammars, will with much less effort produce a much smaller parser than Knuth's unmodified algorithm. DeRemer has found that most unambiguous context-free grammars in practical use are SLR(1) and that his method gives better results than other known parsing methods (DeR 71).

What I propose to do in this dissertation is describe a practical parser building algorithm which will work on all LR(k) grammars. For SLR(k) grammars this algorithm will yield a parser essentially equivalent to DeRemers SLR(k) parser.

2. TERMINOLOGY

A context-free grammar is a 4-tuple (V_T, V_N, S, P).

The set $V = V_T \cup V_N$ is called the vocabulary.

$*$ is the closure operator.

V^* is the set of all strings of finite length over V including the empty string which we will denote by ϵ.

V_T is a finite set of symbols called the terminal vocabulary.

V_N is a finite set of symbols called the non-terminal vocabulary.

$$V_T \cap V_N = \emptyset$$

S is a specific element of V_N called the start symbol.

P is a finite set of ordered pairs $(A, \omega) \in V_N \times V^*$ which we will write $A \longrightarrow \omega$.[2]

Henceforth unless otherwise stated grammar will be used synonymously with context-free grammar in this paper.

When speaking of elements of V^*, unless otherwise stated or obvious from context, capital letters will denote elements of V_N, small letters will denote elements of V_T and Greek letters will denote strings over V.

We will define a grammar $G = (V_N, V_T, S, P)$ only by listing the elements of P. It will be understood that

$S = A$ where $A \longrightarrow \omega$ is the first production in the list

$V_N = \left\{ A \mid A \longrightarrow \omega \in P \text{ for some } \omega \right\}$

$V_T = \left\{ a \mid \text{for some } A \longrightarrow \omega \in P,\ a \text{ is a symbol in the string } \omega \right\} \setminus V_N$

For example let the grammar G_0 be defined by the set of productions listed below.

1. $S \longrightarrow SST$
2. $S \longrightarrow T$
3. $T \longrightarrow a$
4. $A \longrightarrow b$

$G_0 = (V_N, V_T, S, P)$ where

$S = S$

$V_N = \{S, T, A\}$

$V_T = \{a, b\}$

$\alpha\beta$ always denotes the concatenation of α and β. If

$$\alpha = ab \qquad \text{and} \qquad \beta = bab$$

then $\alpha\beta = abbab$.

α^n where n is a non-negative integer denotes the concatenation of α with itself n times. For example $\alpha^0 = \epsilon$ $\alpha^3 = ababab$.

$|\alpha|$ = number of symbols in the string α. Thus $|\alpha| = 2$ for α defined above. If there exists $\gamma \in V^* \setminus \{\epsilon\}$ such that $\gamma\alpha = \beta$, then α is a suffix of β. We write this α suf β . We write $\alpha \longrightarrow \beta$ and say α directly produces β if and only if

$$\alpha = \gamma A \delta \ , \ \beta = \gamma \omega \delta \qquad \text{and} \qquad A \longrightarrow \omega \in P.$$

Thus in grammar G_0 we may write

$$SSTST \longrightarrow SSaST$$

We write $\alpha \longrightarrow^* \beta$ and say α produces β if and only if for some

integer $n \geq 0$

$$\forall i \,|\, 0 \leq i \leq n \quad \exists \alpha_i \,|\, \alpha = \alpha_0,\ \beta = \alpha_n \quad \text{and where}$$

$$i \neq n \quad \alpha_i \longrightarrow \alpha_{i+1}$$

Thus in grammar G_0

$$S \longrightarrow^* SSaST$$

Since $S \longrightarrow SST$

$SST \longrightarrow SSTST$

and $SSTST \longrightarrow SSaST$.

Trivially $\forall \omega \in V^* \quad \omega \longrightarrow^* \omega$

A sentential form is a string $\omega \in V^*$ such that

$$S \longrightarrow^* \omega$$

In grammar G_0 $\omega = SSaST$ is a sentential form. The language $L(G)$ is defined as the set of all terminal sentential forms or sentences

$$L(G_0) = \left\{ a^{2n+1} \,\middle|\, n \geq 0 \right\}$$

A production $A \longrightarrow \omega$ is useful if and only if

$$\exists \alpha, \beta, \gamma \,|\, S \longrightarrow^* \alpha A \beta \longrightarrow \alpha \omega \beta \longrightarrow^* \gamma \quad \text{where} \quad \gamma \in L(G).$$

A production that is not useful is useless. $A \longrightarrow b$ is the only useless production in G_0. Unless otherwise stated we will assume that a grammar G contains no useless productions.

A derivation of a sentential form ω is an ordered $n+1$ tuple of sentential form $(\alpha_0, \ldots, \alpha_n)$ for some $n \geq 0$ where

$$\alpha_0 = S, \quad \alpha_n = \omega \quad \text{and} \quad \forall i \mid 0 \leq i < n \quad \alpha_i \longrightarrow \alpha_{i+1} .$$

Thus (S, SST, SSTST, SSaST) is a derivation of the sentential form SSaST in grammar G_0.

A right derivation of a sentential form is a derivation $(\alpha_0, \ldots, \alpha_n)$ such that

$$\forall i \mid 0 \leq i < n \quad \alpha_i = \beta_i A_i \gamma_i, \quad \alpha_{i+1} = \beta_i \omega_i \gamma_i , \quad A_i \longrightarrow \omega_i \in P$$

$$\text{and} \quad \gamma_i \in V_T^* .$$

In other words we right derive a sentential form by always applying some production to the right most non-terminal symbol in the preceding sentential form in the derivation. A sentential form is a right sentential form if and only if it possesses at least one right derivation.

SSaST is not a right sentential form. However SSTaa is a right sentential form.

(S, SST, SSa, SSSTa, SSSaa, SSTaa)

is a right derivation of SSTaa.

We will only be interested in right derivations. Therefore unless otherwise stated derivation means right derivation, $\alpha \longrightarrow^* \beta$ ($\alpha \longrightarrow \beta$) means α (directly) produces β by right derivations and sentential form means right sentential form.

A sentential form that possesses more than one right derivation is called ambiguous. A grammar G is ambiguous if and only if there is at

least one ambiguous sentence in $L(G)$. G_0 is ambiguous since

(S, SST, SSa, STa, Saa, SSTaa)

is another right derivation of SSTaa . We must make a distinction between grammar and language. A language is a set of strings over some terminal vocabulary V_T . Let G_0' be defined by the productions

1 $S \longrightarrow aaS$ 2 $S \longrightarrow a$

$$L(G_0) = L(G_0') = \left\{ a^{2n+1} \,\middle|\, n \geq 0 \right\}$$

G_0' is an unambiguous grammar. In this paper we are interested in grammars rather than languages.

Let k be a non-negative integer. Define the following three functions on the domain V^* :

$k{:}\,\alpha = \alpha$ if $|\alpha| \leq k$

$k{:}\,\alpha = \beta \,|\, \exists \gamma \in V^*$ for which $\beta\gamma = \alpha$ and $|\beta| = k$ if $|\alpha| > k$.

Thus $k{:}\,\alpha$ is the first k symbols of α if $|\alpha| > k$. Otherwise $k{:}\,\alpha$ is α .

$$H_k(\alpha) = \left\{ \omega \,\middle|\, \text{for some } \beta \in V_T^* \;\; \alpha \longrightarrow^* \beta, \text{ and } k{:}\,\beta = \omega \right\}$$

$$H_k'(\alpha) = \left\{ \omega \,\middle|\, \exists\, \beta \text{ and } \beta' \;\; \alpha \longrightarrow^* \beta' \longrightarrow \beta \,,\; \beta \in V_T^*,\; k{:}\,\beta = \omega \right.$$

$$\left. \text{and } \beta' \neq A\beta \text{ for any } A \in V_N \right\}$$

Definition 1:

A push down automata (PDA) is a 7-tuple

$(K, V_T, \Gamma, \delta, S_0, T_0, F)$ where

K is a finite set of states.

V_T is a finite set called the input vocabulary.

Γ is a finite set called the push down stack vocabulary.

S_0 is an element of K called the start state.

T_0 is an element of Γ called the start symbol.

F is a subset of K called the set of final states.

δ is a function from $K \times (V_T \cup \{\epsilon\}) \times \Gamma$ into the collection of finite subsets of $K \times \Gamma^*$.

An interpretation of a PDA is that of a machine, M, which is started in state S_0 with a stack of T_0 and is fed a string of symbols $\omega \in V_T^*$. An input head is placed over the first symbol of ω.

If M is in state S, T is the element on top of the stack and the input head is over a symbol $a \in V_T$, then if $(S', \gamma) \in \delta(S, a, T)$ where $\gamma \in \Gamma^*$, then M may move to state S' while moving the input head to the next symbol in ω, popping T off the stack and then pushing γ on top of the stack. If $(S', \gamma) \in \delta(S, \epsilon, T)$, then M may move to state S' without advancing the input head while popping T off the stack and then pushing γ on the top of the stack.

We say that M accepts ω if and only if it is possible for M to be in some state $S \in F$ after reading the whole input string ω.

A PDA M is deterministic if and only if

$\delta(S, \epsilon, T) \cap \delta(S, a, T) = \emptyset \quad \forall S$, a and T and $\delta(S, \alpha, T)$ contains at most one element $\forall S, \alpha$, and T .[3]

We are interested in machines which are slightly modified push down automata. We will call them parsing machines (PM).

Definition 2:

A parsing machine is an 8-tuple

$$(K, V_T', S_0, F, P, N, k, \delta)$$

K is a finite set of states and is also the push down stack vocabulary.

V_T' is a finite set called the input vocabulary. The symbol $\dashv$ called the end symbol is in V_T' .

S_0 is an element of K called the start state.

F is a subset of K called the set of final states.

P is a finite set of symbols called the output vocabulary.

N is a function from P into the non-negative integers.

k is a non-negative integer.

δ is a function from $K \times W_k \times K$ into the collection of subsets of $K \times (P \cup \{\epsilon\})$ where for $k = 0$ $W_0 = V_T'$, and for $k > 0$ $W_k = \{\alpha \in V_T'^* \mid |\alpha| = k \text{ or } (|\alpha| < k \text{ and } \dashv \text{ suf } \alpha)\}$.

If $k = 0$ then if $(U, p) \in (S, a, T)$ then $(U, p) \in (S, b, T)\ \forall b \in V_T'$.

An interpretation of a PM is that of a machine M which is started in state S_0 with a stack of S_0 and is fed a string of symbols $\omega \in V_T'^*$.

An input head is placed over the first k symbols of ω.

If M is in state S with T the $N(p)$ symbol from the top of the stack, (we consider the symbol on top of the stack as zeroth from the top) and α is under the input head, then if

$$(T' , p) \in \delta (S, \alpha, T) ,$$

then M may move to state T', replace the top $N(p)$ symbols on the stack by T' and output p without moving the input head.

If $(T', \epsilon) \in \delta (S, \alpha, T)$,

then M may move to state T', push T' on top of the stack and output ϵ while moving the input head forward one symbol.

We say that M accepts ω if and only if it is possible for M to be in some state $S \in F$ after reading the input string ω. M is deterministic if and only if $\forall S, \omega, T$, the set $\delta(S, \omega, T)$ contains at most one element.

Finally for our purposes a graph is a finite set of nodes Φ and a set of ordered pairs in $\Phi \times \Phi$ called transitions. If (ρ, σ) is a transition in a graph GR, we write $\rho \longrightarrow \sigma$ and say GR has a transition from ρ to σ.

3. KNUTH'S ALGORITHM

We now give a formal definition of an LR(k) grammar.

Definition 3:

A context-free grammar G is LR(k) if and only if for all right derivations of the form

$$S \longrightarrow^{*} \alpha A \gamma \longrightarrow \alpha \beta \gamma$$

and $$S \longrightarrow^{*} \alpha' A' \gamma' \longrightarrow \alpha' \beta' \gamma'$$

if $$(\lfloor \alpha \beta \rfloor + k) : \alpha \beta \gamma = (\lfloor \alpha \beta \rfloor + k) : \alpha' \beta' \gamma'$$

then $\alpha = \alpha'$, $\beta = \beta'$ and $A = A'$ (Knu 71) [4]

Now for a given grammar $G = (V_T, V_N, S, P)$ form a new grammar $G' = (V_T', V_N', S' P')$ where

$$V_T' = V_T \cup \{ \dashv \} \quad \text{where} \quad \dashv \notin V$$

$$V_N' = V_N \cup \{ S' \} \quad \text{where} \quad S' \notin V$$

$$P' = \begin{cases} P \cup \{ S' \longrightarrow S \dashv^{k} \} & \text{if} \quad k \geq 1 \\ P \cup \{ S' \longrightarrow S \dashv \} & \text{if} \quad k = 0 \end{cases} .$$

$L(G') = \lfloor L(G) \rfloor \dashv^{k}$ if $k \geq 1$ and $L(G') = \lfloor L(G) \rfloor \dashv$ if $k = 0$.

Intuitively G is LR(k) if and only if G' is LR(k) and G' is LR(k) if and only if

if $$S' \longrightarrow^{*} \alpha A \gamma \longrightarrow \alpha \beta \gamma$$

and $$S' \longrightarrow^{*} \omega \longrightarrow \alpha \beta \lfloor k{:}\gamma \rfloor \delta$$

are derivations in G'

then $\omega = \alpha A \lfloor k{:}\gamma \rfloor \delta$.

In other words if we are reading a sentential form $\alpha\beta\gamma$ in G' from left to right, after reading $\alpha\beta$ we need scan at most k:γ in order to tell if

$$S' \longrightarrow^* \alpha A \gamma \longrightarrow \alpha\beta\gamma$$

is the derivation of $\alpha\beta\gamma$. Thus if G is LR(k) then a derivation of a sentential form in G is unique. That is LR(k) implies unambiguity. Knuth (Knu 65) proves that in general whether a grammar G is LR(k) for some k is an undedicable question. However, if k is given a-priori, Knuth gives an algorithm for deciding if a grammar G is LR(k) and building an LR(k) parser if it is.

Briefly, Knuth's algorithm is this: (Knu 65)
Number the productions of G ; 1,2, ..., r . Add production number 0 , $S' \longrightarrow S \dashv^k$ ($S' \longrightarrow S \dashv$ if k = 0) where $|S, \dashv| \cap V = \emptyset$. Let n_p be the number of symbols on the right hand side of production p . Let p_j be the j^{th} symbol on the right hand side of production p and p_0 be the symbol on the left hand side of production p .

Define a configuration as a 3-tuple (p, j, α) where p is the number of a production in G' , j is an integer such that $0 \leq j \leq n_p$ and $\alpha \in V_T'^*$ with $|\alpha| = k$. Define the closure T* of a set of configurations T as the smallest set of configurations such that T* $\supset$ T and if (p, j, α) $\in$ T* with $j < n_p$ then (q, 0, β) $\in$ T* $\forall$ q, β | production q is $p_{j+1} \longrightarrow \omega$ for some $\omega \in V^*$ and $\beta \in H_k(p_{j+2} \ldots p_{n_p} \alpha)$.

Define a state S as a closed set of configurations arrived at in the following way. The start state S_0 is $|(0, 0, \dashv^k)|^*$, ($|(0, 0, \dashv)|^*$ if k = 0) . If S_n is a state and (p, j, α) $\in$ S_n

with $j < n_p$ and $p_{j+1} \neq +$ then

$S_m = \left\{ (q, i+1, \beta) \;\middle|\; (q, i, \beta) \in S_n \text{ , } i < n_q \text{ and } q_{i+1} = p_{j+1} \right\}^*$

is a state. In this case we will say that there is a p_{j+1} - read transition from state S_n to state S_m and write $S_n \xrightarrow{p_{j+1}} S_m$.

If $(p, n_p, \alpha) \in S_n$ for some α then we say that there is a reduce-p transition from state S_n and write $S_n \xrightarrow{\#p}$.

If $ord(V') = m$, $ord(P') = h$ and $n = \max_p(n_p)$, then an upper bound on the number of configurations is nhm^k . Thus 2^{nhm^k} is an upper bound on the number of states. Thus for any grammar G , the number of states although possibly very large is finite.

It can now be shown that

if $S' \longrightarrow^* \alpha_0 \ldots \alpha_n A \gamma \longrightarrow \alpha_0 \ldots \alpha_m \gamma$ with $m \geq n$

where $\alpha_i \in V \qquad \forall i \mid 0 \leq i \leq m$

then there exists a sequence of states

$S_0 \ldots S_{m+1}$ with S_0 = start state such that

$S_i \xrightarrow{\alpha_i} S_{i+1} \qquad \forall i \mid 0 \leq i \leq m$.

Furthermore if p is the production $A \longrightarrow \alpha_{n+1} \ldots \alpha_m$

(if $n = m$ then p is $A \longrightarrow \epsilon$) then $(p, n_p, k{:}\gamma) \in S_{m+1}$.

And if $S' \longrightarrow^* \alpha' A' \gamma' \longrightarrow \alpha' \beta' \gamma'$

where $\alpha' \beta' \gamma' = \alpha_0 \ldots \alpha_m \gamma$,

γ' suf γ and $\gamma \in V_T'^*$

then $\exists\, q, i, \beta$ with $i < n_q \mid (q, i, \beta) \in S_{m+1}$

and $k{:}\gamma \in H_k'(q_{i+1} \ldots q_{n_q} \beta)$.

(we use H_k' instead of H_k since γ' suf γ)

This tells us that G is LR(k) if and only if for all states S, $(p, n_p, \gamma) \in S$ and $(q, i, \beta) \in S$ with $p \neq q$ or $n_p \neq i$ implies

$$\gamma \notin H'_k (q_{i+1} \cdots q_{n_q} \beta) .$$

Now suppose we find that G is LR(k). We can use these states and transitions to define a PM, $M_k(G)$ with the power to scan the next k symbols of the input string which will parse the grammar L(G) .

Let $M = (K, V', S_0, F, P, N, k, \delta)$ where

K = the set of states as defined above,

$P = \{p \in I \mid 1 \le p \le r\}$, $N(p) = n_p$,

$F = \{S \in K \mid (0, 1, \dashv^k) \in S\}$ and

δ is defined by the following procedure.

Let α = the next k symbols of the input string. If M is in state S and $(p, n_p, \alpha) \in S$ for some p then do not advance the input head, pop the top n_p items off the stack, look at the symbol on top of the stack T , enter the state $T' \mid T \xrightarrow{p_0} T'$, push T' on the stack and output p . Call this action taking a reduce -p transition from state S . If $(p, n_p, \alpha) \notin S \ \forall p$ then enter the state $S' \mid S \xrightarrow{1:\omega} S'$, push S' on the stack and advance the input head. Call this action taking a read transition from state S . If $\omega = \dashv^k$ ($\omega = \dashv$ if k = 0) and $(0, 1, \dashv^k) \in S$, then accept the input string. Here ω = the unread portion of the input string.[5] If it becomes impossible to make a transition, then reject the input string. Loosely speaking, whenever the parser is in state S_n , it may be working on production p , it has already found the first j symbols on the right hand side of production p and this instance of the production p may be followed by the string α if and only if $(p, j, \alpha) \in S_n$.[6]

Knuth (Knu 65) proves that if we set the input string $\omega = \alpha \dashv^k$ ($\alpha \dashv$ if $k = 0$) where α is the string we wish to parse, then the machine defined above is a parser for G. Furthermore the machine $M_k(G)$ conforms with the concept of parsing machine given in the preceding section.

4. THREE LR(k) GRAMMARS

Let's start with a trivial example. The grammar G_1 is defined by the two productions listed below:

1. $S \longrightarrow aS$ 2. $S \longrightarrow a$

We may represent $L(G_1)$ as $\left\{a^n \mid n>0\right\}$. Intuitively G_1 is not LR(0) but is LR(1) since we can always answer the question "is this the last symbol in a string" by looking to see if it is followed by another symbol. To build Knuth's LR(1) parser for G_1 add the zeroth production $S' \longrightarrow S\dashv$. The states and transitions are listed below in Table 1.

Table 1 States and Transitions for $M_1(G_1)$

Number	State	Transitions To
0	(0, 0, ⊣), (1, 0, ⊣), (2, 0, ⊣)	a 2 S 1
1	(0, 1, ⊣)	⊣ accept
2	(1, 1, ⊣), (2, 1, ⊣)	reduce -2
	(1, 0, ⊣), (2, 0, ⊣)	a 2 S 3
3	(1, 2, ⊣)	reduce 1

Since $H_1'(S\dashv) = H_1'(a\dashv) = H_1'(aS\dashv) = a$

G_1 is LR(1).

To convert Table 1 into the parsing machine $M_1(G_1)$ let

K be the set of states in Table 1

$V_T' = \{a, \dashv\}$

$S_0 =$ state 0

$F = \{$state 1$\}$

$P = \{1, 2\}$

$N(1) = 2$ and $N(2) = 1$

$\delta(0, a, 0) = \delta(2, a, 2) = \{(2, \epsilon)\}$

$\delta(2, \dashv, 2) = \{(3, 2)\}$

$\delta(2, \dashv, 0) = \{(1, 2)\}$

$\delta(3, \dashv, 2) = \{(3, 1)\}$

$\delta(3, \dashv, 0) = \{(1, 1)\}$

δ maps all other members of $K \times W_k \times K$ into $\emptyset$.

Since the parsing machine is uniquely determined by the grammar and the table of states and transitions and since it's easier to see what is going on from the table than from the function δ, in subsequent examples we will not convert the table of states and transitions into a parsing machine.

Now we wish to find an LR(1) parser for the grammar G_2 of arithmetic expressions defined below:

1. $E \longrightarrow E + T$
2. $E \longrightarrow T$
3. $T \longrightarrow P \uparrow T$
4. $T \longrightarrow P$
5. $P \longrightarrow (E)$
6. $P \longrightarrow i$

Here $S = E$, $V_N = \{E, T, P\}$ and $V_T = \{+, \uparrow, (,), i\}$.

We start by adding to P the zeroth production

$$0. \quad S \longrightarrow E \dashv \quad .$$

The states and transitions are given in Table 2. We use the shorthand $(5, 0, \dashv + \uparrow)$ for $(5, 0, \dashv)$, $(5, 0, +)$, $(5, 0, \uparrow)$.

The only states which contain configurations (p, n_p, α) and (q, i, β) with $p \neq q$ or $n_p \neq i$ are states 4 and 12. Since

$$H_1' (\uparrow T \dashv) = H_1' (\uparrow T +) = H_1' (\uparrow T)) = \uparrow$$

G_2 is LR(1). We will see later that many of the 21 states in Table 2 are actually unnecessary.

Suppose we wish to parse the sentence $i + (i \uparrow i)$ in G_2. First let the input string

$$\omega = i + (i \uparrow i) \dashv \quad .$$

Start the parser in state 0 with 0 on the stack. We write $0 * i + (i \uparrow i) \dashv$ meaning the portion of the input string to be read in on the right of the marker $*$ and the stack is to the left. The parser is in the state immediately to the left of the marker $*$.

Table 2 States and Transitions For $M_1(G_2)$

Number	State	Transition To
0	(0, 0, ⊣), (1, 0, ⊣ +), (2, 0, ⊣ +)	E 1
		T 21
	(3, 0, ⊣ +), (4, 0, ⊣ +)	P 4
		(7
	(5, 0, ⊣ +↑), (6, 0, ⊣ +↑)	i 20
1	(0, 1, ⊣) (1, 1, ⊣ +)	⊣ accept
		+ 2
2	(1, 2, ⊣ +), (3, 0, ⊣ +), (4, 0, ⊣ +),	T 3
		P 4
	(5, 0, ⊣ +↑), (6, 0, ⊣ +↑)	(7
		i 20
3	(1, 3, ⊣ +)	reduce 1
4	(3, 1, ⊣ +) (4, 1, ⊣ +)	↑ 5
		reduce 4
5	(3, 2, ⊣ +), (3, 0, ⊣ +), (4, 0, ⊣ +)	T 6
		P 4
	(5, 0, ⊣ +↑), (6, 0, ⊣ +↑)	(7
		i 20
6	(3, 3, ⊣ +)	reduce 3
7	(5, 1, ⊣ +↑), (1, 0,) +), (2, 0,) +)	E 8
		T 18
	(3, 0,) +), (4, 0,) +)	P 12
		) 15
	(5, 0,) +↑), (6, 0,) +↑)	i 19
8	(5, 2, ⊣ +↑), (1, 1,) +)	+ 10
		) 9
9	(5, 3, ⊣ +↑)	reduce 5

Table 2 Continued

Number	State	Transition To
10	(1, 2,) +), (3, 0,) +), (4, 0,) +) (5, 0,) + ↑), (6, 0,) + ↑)	T 11 P 12 (15 i 16
11	(1, 3,) +)	reduce 1
12	(3, 1,) +), (4, 1,) +)	↑ 13 reduce 4
13	(3, 2,) +), (3, 0,) +), (4, 0,) +) (5, 0,) + ↑), (6, 0,) + ↑)	T 14 P 12) 15 i 16
14	(3, 3,) +)	reduce 3
15	(5, 1,) + ↑), (1, 0,) +), (2, 0,) +), (3, 0,) +), (4, 0,) +), (5, 0,) + ↑), (6, 0,) + ↑)	E 16 T 18 P 12) 15 i 16
16	(5, 2,) + ↑), (1, 1,) +)	) 17 + 10
17	(5, 3,) + ↑)	reduce 5
18	(2, 1,) +)	reduce 2
19	(6, 1,) + ↑)	reduce 6
20	(6, 1, + + ↑)	reduce 6
21	(2, 1, + + ↑)	reduce 2

We scan the first i and enter state 20.

We now have 0, 20 * + (i ↑ i) ⊣ .

Next reduce production 6.

We now have	0, 4 * + (i ↑ i) ⊣	output 6
Continuing	0, 21 * + (i ↑ i) ⊣	output 4
	0, 1 * + (i ↑ i) ⊣	output 2
	0, 1, 2 * (i ↑ i) ⊣	
	0, 1, 2, 7 * i ↑ i) ⊣	
	0, 1, 2, 7, 19 * ↑ i) ⊣	
	0, 1, 2, 7, 12 * ↑ i) ⊣	output 6
	0, 1, 2, 7, 12, 13, i) ⊣	
	0, 1, 2, 7, 12, 13, 16 *) ⊣	
	0, 1, 2, 7, 12, 13, 12 *) ⊣	output 6
	0, 1, 2, 7, 12, 13, 14 *) ⊣	output 4
	0, 1, 2, 7, 18 *) ⊣	output 3
	0, 1, 2, 7, 8 *) ⊣	output 2
	0, 1, 2, 7, 8, 9 * ⊣	
	0, 1, 2, 4 * ⊣	output 5
	0, 1, 2, 3 * ⊣	output 4
	0, 1 * ⊣	output 1

accept

If we start with E and apply the productions obtained in the reverse order and always to the right most non-terminal symbol, we get

	E	4	E + (P ↑ P)
1	E + T	6	E + (P ↑ i)
4	E + P	6	E + (i ↑ i)
5	E + (E)	2	T + (i ↑ i)
2	E + (T)	4	P + (i ↑ i)
3	E + (P ↑ T)	6	i + (i ↑ i)

This is precisely what we mean by a parse of a sentence ω, an ordered list of the productions used in a right derivation of the sentence ω.

Our next example is more complicated. Find an LR(1) parser for the grammar G_3 defined by the productions listed below:

1. $S \longrightarrow T_1 w$
2. $S \longrightarrow T_2 x$
3. $S \longrightarrow T_3 w$
4. $S \longrightarrow T_4 x$
5. $T_1 \longrightarrow a$
6. $T_2 \longrightarrow a$
7. $T_1 \longrightarrow bU_1$
8. $T_2 \longrightarrow bU_2$
9. $T_3 \longrightarrow bU_3 p$
10. $T_3 \longrightarrow bU_3 q$
11. $T_4 \longrightarrow bU_4 r$
12. $T_4 \longrightarrow bU_4 s$
13. $T_1 \longrightarrow cU_1 p$
14. $T_1 \longrightarrow cU_1 r$
15. $T_2 \longrightarrow cU_2 q$
16. $T_2 \longrightarrow cU_2 s$
17. $T_3 \longrightarrow cU_3$
18. $T_4 \longrightarrow cU_4$
19. $U_1 \longrightarrow dT_3$
20. $U_2 \longrightarrow dT_4$
21. $U_3 \longrightarrow dT_1$
22. $U_4 \longrightarrow dT_2$

Here $S = S$ $\qquad V_N = \{ S \} \cup \{ T_i, U_i \mid 1 \le i \le 4 \}$

$V_T = \{ a, b, c, d, p, q, r, s, w, x \}$

The states and transitions for $M_1(G_3)$ are listed in Table 3. In Table 3 we use the shorthand $(\lfloor 9, 10, 17 \rfloor, 0, pq)$ to stand for $(9, 0, p)$, $(9, 0, q)$, $(10, 0, p)$, $(10, 0, q)$, $(17, 0, p)$, $(17, 0, q)$. States 1, 4 and 13 are the only states containing configurations (p, n_p, α) and (q, i, β) where $p \neq q$ or $n_p \neq i$. Since $\lfloor w \rfloor \cap \lfloor x \rfloor = \lfloor p, q \rfloor \cap \lfloor r, s \rfloor = \lfloor p, r \rfloor \cap \lfloor q, s \rfloor = \emptyset$, grammar G_3 is LR(1). G_3 was included because it is an extremely "messy" grammar. However in spite of this, it is LR(1). We will see shortly that well over half of the 90 states in Table 3 are unnecessary and may be eliminated from the parser.

Table 3 States and Transitions For $M_1(G_3)$

Number	State	Transition		To
0	(0, 0, +), (5, 7, 13, 14 , 0, w)	S 81	a	1
		T_1 82		
	($\lfloor$6, 8, 15, 16$\rfloor$, 0, x), ($\lfloor$9, 10, 17$\rfloor$, 0, w)	T_2 84	b	2
		T_3 86		
	($\lfloor$11, 12, 18$\rfloor$, 0, x), ($\lfloor$1, 2, 3, 4$\rfloor$, 0, +)	T_4 88	c	20
1	(5, 1, w), (6, 1, x)	reduce 5 reduce 6		
2	(7, 1, w), (8, 1, x), ($\lfloor$9, 10$\rfloor$, 1, w), ($\lfloor$11, 12$\rfloor$, 1, x)	U_1 45	U_4	66
		U_2 46		
	(19, 0, w), (20, 0, x), (21, 0, pq), (20, 0, rs)	U_3 63	d	3
3	(19, 1, w), (20, 1, x), (21, 1, pq), (22, 1, rs)	T_1 79	a	4
		T_2 80		
	($\lfloor$5, 7, 12, 14$\rfloor$, 0, pq), ($\lfloor$6, 8, 15, 16$\rfloor$, 0, rs)	T_3 75	b	5
	($\lfloor$9, 10, 17$\rfloor$, 0, w), ($\lfloor$11, 12, 18$\rfloor$, 0, x)	T_4 76	c	16
4	(5, 1, pq), (6, 1, rs)	reduce 5 reduce 6		
5	(7, 1, pq), (8, 1, rs), ($\lfloor$9, 10$\rfloor$, 1, w), ($\lfloor$11, 12$\rfloor$, 1, x)	U_1 55	a	4
		U_2 56		
	(19, 0, pq), (20, 0, rs), (21, 0, pq), (22, 0, rs)	U_3 63	d	6
6	(19, 1, pq), (20, 1, rs), (21, 1, pq), (22, 1, rs)	T_1 77	a	4
		T_2 78		
	($\lfloor$5, 7, 13, 14$\rfloor$, 0, pq), ($\lfloor$6, 8, 15, 16$\rfloor$, 0, rs)	T_3 75	b	7
	($\lfloor$9, 10, 17$\rfloor$, 0, pq), ($\lfloor$11, 12, 18$\rfloor$, 0, rs)	T_4 76	c	8
7	(7, 1, pq), (8, 1, rs), ($\lfloor$9, 10$\rfloor$, 1, pq), (11, 12, 1, rs)	U_1 55		
		U_2 56	U_4	60
	(19, 0, pq), (20, 0, rs), (21, 0, pq), (22, 0, rs)	U_3 57	d	6

Table 3 Continued

Number	State	Transition	To
8	([13, 14], 1, pq), ([15, 16], 1, rs), (17, 1, pq), (18, 1, rs) (19, 0, pr), (20, 0, qs), (21, 0, pq), (22, 0, rs)	U_1 34 U_2 31 U_4 43 U_3 44 d 9	
9	(19, 1, pr), (20, 1, qs), (21, 1, pq), (22, 0, rs) ([5, 7, 13, 14], 0, pq), ([6, 8, 15, 16], 0, rs) ([9, 10, 17], 0, pr), ([11, 12, 18], 0, qs)	T_1 69 a 4 T_2 70 T_3 75 b 10 T_4 76 c 11	
10	(7, 1, pq), (8, 1, rs), ([9, 10], 1, pr), ([11, 12], 1, qs) (19, 0, pq), (20, 0, rs), (21, 0, pq), (22, 0, rs)	U_1 55 U_2 56 U_4 50 U_3 47 d 6	
11	([13, 14], 1, pq), ([15, 16], 1, rs), (17, 1, pr), (18, 1, qs) (19, 0, pr), (20, 0, qs), (21, 0, pr), 22, 0, qs)	U_1 34 U_4 29 U_2 31 U_3 30 d 12	
12	(19, 1, pr), (20, 1, qs), (21, 1, pr), (22, 1, qs) ([5, 7, 13, 14], 0, pr), ([6, 8, 15, 16], 0, qs) ([9, 10, 17], 0, pr), ([11, 12, 18], 0, qs)	T_1 69 a 13 T_2 70 b 14 T_3 73 T_4 74 c 15	
13	(5, 1, pr), (6, 1, qs)	reduce 5 reduce 6	
14	(7, 1, pr), (8, 1, qs), ([9, 10], 1, pr), ([11, 12], 1, qs) (19, 0, pr), (20, 0, qs), (21, 0, pq), (22, 0, rs)	U_1 53 U_4 50 U_2 54 U_3 47 d 9	

Table 3 Continued

Number	State	Transition	To
15	([13, 14], 1, pr), ([15, 16], 1, qs), (17, 1, pr), (18, 1, qs) (19, 0, pr), (20, 0, qs), (21, 0, pr), (22, 0, qs)	U_1 40 U_2 37 U_3 30	U_4 29 d 12
16	([13, 14], 1, pq), ([15, 16], 1, rs), (17, 1, w), (18, 1, x) (19, 0, pr), (20, 0, qs), (21, 0, w), (22, 0, x)	U_1 34 U_2 31 U_3 22	U_4 21 d 17
17	(19, 1, pr), (20, 1, qs), (21, 1, w), (22, 1, x) ([5, 7, 13, 14], 0, w), ([6, 8, 15, 16], 0, x) ([9, 10, 17], 0, pr), ([11, 12, 18], 0, qs)	T_1 69 T_2 70 T_3 71 T_4 72	a 1 b 18 c 19
18	(7, 1, w), (8, 1, x), ([9, 10], 1, pr), ([11, 12], 1, qs) (19, 0, w), (20, 0, x), (21, 0, pq), (22, 0, rs)	U_1 45 U_2 46 U_3 47	U_4 50 d 3
19	([13, 14], 1, w), ([15, 16], 1, x), (17, 1, pr), (18, 1, qs) (19, 0, pr), (20, 0, qs), (21, 0, pr), (22, 0, qs)	U_1 26 U_2 23 U_3 30	U_4 29 d 12
20	([13, 14], 1, w), ([15, 16], 1, x), (17, 1, w), (18, 1, x) (19, 0, pr), (20, 0, qs), (21, 0, w), (22, 0, x)	U_1 26 U_2 23 U_3 22	U_4 21 d 17

Table 3 Continued

Number	State	Transition To
21	(18, 2, x)	reduce 18
22	(17, 2, w)	reduce 17
23	([15, 16], 2, x)	q 24 s 25
24	(15, 3, x)	reduce 15
25	(16, 3, x)	reduce 16
26	([13, 14], 2, w)	p 27 r 28
27	(13, 3, w)	reduce 13
28	(14, 3, w)	reduce 14
29	(18, 2, qs)	reduce 18
30	(17, 2, pr)	reduce 17
31	([15, 16], 2, rs)	q 32 s 33
32	(15, 3, rs)	reduce 15
33	(16, 3, rs)	reduce 16
34	([13, 14], 2, pq)	p 35 r 36
35	(13, 3, pq)	reduce 13
36	(14, 3, pq)	reduce 14
37	([15, 16], 2, qs)	q 38 s 39
38	(15, 3, qs)	reduce 15
39	(16, 3, qs)	reduce 16
40	([13, 14], 2, pr)	p 41 r 42
41	(13, 3, pr)	reduce 13
42	(14, 3, pr)	reduce 14
43	(18, 2, rs)	reduce 18
44	(17, 2, rs)	reduce 17
45	(7, 2, w)	reduce 7
46	(8, 2, x)	reduce 8
47	([9, 10], 2, pr)	p 48 q 49
48	(9, 3, pr)	reduce 9
49	(10, 3, pr)	reduce 10
50	([11, 12], 2, qs)	r 51 s 52
51	(11, 3, qs)	reduce 11
52	(12, 3, qs)	reduce 12
53	(7, 2, pr)	reduce 7
54	(8, 2, qs)	reduce 8
55	(7, 2, pq)	reduce 7
56	(8, 2, rs)	reduce 8
57	([9, 10], 2, pq)	p 58 q 59
58	(9, 3, pq)	reduce 9
59	(10, 3, pq)	reduce 10
60	([11, 12], 2, rs)	r 61 s 62
61	(11, 3, rs)	reduce 11
62	(12, 3, rs)	reduce 12
63	([9, 10], 2, w)	p 64 q 65
64	(9, 3, w)	reduce 9
65	(10, 3, w)	reduce 10

Table 3 Continued

Number	State	Transition	To
66	([11, 12], 2, x)	r 67	s 68
67	(11, 3, x)	reduce 11	
68	(12, 3, x)	reduce 12	
69	(19, 2, pr)	reduce 19	
70	(20, 2, qs)	reduce 20	
71	(21, 2, w)	reduce 21	
72	(22, 2, x)	reduce 22	
73	(21, 2, pr)	reduce 21	
74	(22, 2, qs)	reduce 22	
75	(21, 2, pq)	reduce 21	
76	(22, 2, rs)	reduce 22	
77	(19, 2, pq)	reduce 19	
78	(20, 2, rs)	reduce 20	
79	(19, 2, w)	reduce 19	
80	(20, 2, x)	reduce 20	
81	(0, 1, +)	+	accept
82	(1, 1, +)	w	83
83	(1, 2, +)	reduce 1	
84	(2, 1, +)	x	85
85	(2, 2, +)	reduce 2	
86	(3, 1, +)	w	87
87	(3, 2, +)	reduce 3	
88	(4, 1, +)	x	88
89	(4, 2, +)	reduce 4	

5. DEREMER'S ALGORITHM

DeRemer has an algorithm which will work on a large proper subset of LR(k) grammars called simple LR(k) (SLR(k)) grammars and will yield considerable better results than Knuth's algorithm.

Briefly DeRemer's algorithm is this: (DeR 69 and DeR 71)
Number the productions of G ; 1, 2, ... , r . Add production number

0. $S' \longrightarrow \vdash S \dashv$ where $\left| S, \dashv, \vdash \right| \cap V = \emptyset$

$\vdash$ denotes the beginning of an input string. $\dashv$ denotes the end of an input string. Now apply Knuth's algorithm for $k = 0$.[7] If G is LR(0) , then convert to an LR(0) parsing machine $M_0'(G)$, as in Knuth's algorithm, and we are done. If G is not LR(0) then there exist states $I_1 \ldots I_n$ where for each $i \mid 1 \le i \le n$ $\exists p$ and $q \mid$

$(p , n_p , \epsilon) \in I_i$ and $(q , j, \epsilon) \in I_i$

with $p \neq q$ or $n_p \neq j$. Call a state of this kind an inadequate state.

If $M_0'(G)$ has an inadequate state, then $M_0'(G)$ is nondeterministic. We could use $M_0'(G)$ to parse G , however each time $M_0'(G)$ entered an inadequate state it would have to try all possible actions to find out which ones[8] if any were correct. We wish to make $M_0'(G)$ deterministic by giving it the power to scan the next k symbols of the input string.

For each inadequate state I , define sets Z_p $\forall p \mid 0 \le p \le r$ as follows

$$Z_p = \left| k : \eta \;\middle|\; S' \longrightarrow^* \zeta A \eta \text{ with } \eta \in V_T^* \right.$$
$$\left. \text{where } p_0 = A \right| \text{ if } \exists \text{ a reduce -p transition from state } I .$$

$Z_p = \emptyset$ if $p > 0$ and $\nexists$ a reduce -p transition from state I .

$$Z_0 = \Big\{ k : a_0 \ldots a_n \eta \;\Big|\; \exists \text{ an } a_0\text{-read transition from state I to state } T_0 \text{ , } \exists \text{ an } a_i\text{-read transition from state } T_{i-1} \text{ to state } T_i \;\forall i \;\Big|\; 1 \leq i \leq n \text{ , } \exists \text{ a reduce -p transition from state } T_n \text{ and } S' \longrightarrow^{*} \zeta A \eta \text{ with } \eta \in V_T^{*} \text{ where } p_0 = A \Big\}$$

Now suppose $\omega \in L(G)$ and $M_0'(G)$ acts on the input string $\vdash\omega\dashv$ following some path which parses ω correctly and arrives in state I with ω' = unread portion of the input string. If at this point reduce -p is a correct transition in parsing ω, then $k : \omega' \in Z_p$. If read -1 : ω is a correct transition then $k : \omega' \in Z_0$. Therefore if for each inadequate state I $Z_p \cap Z_q = \emptyset$ for $p \neq q$, then let the transition function δ from state I be as follows:

Let ω' be as above and $\alpha = k : \omega'$. If $\alpha \in Z_p$ with $p > 0$, then take a reduce -p transition from state I . If $\alpha \in Z_0$ take a read -1 : α transition from I . Otherwise, reject the input string. $M_0'(G)$ with the above modification is now a deterministic LR(k) parser for G if and only if for each inadequate state I , $p \neq q$ implies $Z_p \cap Z_q = \emptyset$. Call this parsing machine $M_k'(G)$.

It should be noted that Z_p may contain strings α such that for no $\omega \in L(G)$ will $M_0(G)$ acting upon the input string $\vdash\omega\dashv$ arrive

in state I with $\alpha = k : \omega'$ where ω' = the unread portion of $\vdash\omega\dashv$. We say that G is SLR(k) if and only if this technique yields a deterministic parsing machine.

Definition 4

G is SLR(k) if and only if all derivations of the form

$$S' \longrightarrow^* \alpha A \gamma \longrightarrow \alpha \beta \gamma$$

$$S' \longrightarrow^* \alpha' A' \gamma' \longrightarrow \alpha' \beta' \gamma'$$

$$S' \longrightarrow^* \zeta A \eta \qquad \text{with} \quad \eta \in V_T^*$$

and

$$S' \longrightarrow^* \zeta' A' \eta' \qquad \text{with} \quad \eta' \in V_T^*$$

$$\alpha' \beta' = \alpha \beta \delta \quad \text{with} \quad \delta \in V_T^* \quad \text{implies} \quad k : \delta \eta' \neq k : \eta$$

If G is SLR(k) then G is LR(k). However, the converse of this statement is false.

If $k = 1$, then it is a simple matter to compute Z_p. Given an inadequate state I, $a \in Z_0$ if and only if $\exists$ an a-read transition from state I. To compute Z_p for $p > 0$ where $p_0 = A$, first compute $\lfloor A \rfloor$ where

$\lfloor A \rfloor$ = smallest subset of V_N such that $A \in \lfloor A \rfloor$ and if $C \in \lfloor A \rfloor$, $B \longrightarrow \alpha C \beta \in P$ and $\beta \longrightarrow^* \epsilon$ [9] then $B \in \lfloor A \rfloor$.

$\lfloor A \rfloor$ may be computed directly from the grammar. Then

$$Z_p = \bigcup_{B \in \lfloor A \rfloor} \left| H_1 \beta \;\middle|\; C \longrightarrow \alpha B \beta \in P \text{ for some } C, \alpha, \beta \right| \frown |\epsilon|$$

or alternately

$$Z_p = \bigcup_{B \in \lfloor A \rfloor} \left| a \;\middle|\; \text{for some state S, } \exists \text{ a B-read transition to state S and an a-read transition from state S} \right| .$$

Unfortunately for $k > 1$ the sets Z_p are not in general so easily computed. The techniques discussed in subsequent sections of this dissertation may be employed.

DeRemer uses a slightly different notation for a configuration. Instead of a 3-tuple (p, i, α), DeRemer writes out production p and places a marker $*$ after the symbol p_i where $i > 0$ and after the $\longrightarrow$ where $i = 0$. Since DeRemer is building an LR(0) parser, the final member of the 3-tuple is always ϵ and is omitted. Thus if production p is $A \longrightarrow a_1\ a_2\ a_3$ then

$A \longrightarrow * a_1\ a_2\ a_3$ means $(p, 0, \epsilon)$

$A \longrightarrow a_1 * a_2\ a_3$ means $(p, 1, \epsilon)$

and $A \longrightarrow a_1\ a_2\ a_3 *$ means $(p, 3, \epsilon)$.

We will employ DeRemer's notation wherever it is convenient.

Now let's employ DeRemer's algorithm on grammar G_1 defined on page 16. Add the zeroth production $S' \longrightarrow \vdash S \dashv$. The states and transitions for the LR(0) parser for grammar G_1 are listed in Table 4.

Table 4 States and Transitions for $M'_0(G_1)$

Number	State		Transitions To
0	$S' \longrightarrow * \vdash S \dashv$		$\vdash$ 1
1	$S' \longrightarrow \vdash * S \dashv$	$S \longrightarrow * a S$	S 2
		$S \longrightarrow * a$	a 3
2	$S' \longrightarrow \vdash S * \dashv$		$\dashv$ accept
3	$S \longrightarrow a * S$	$S \longrightarrow * a S$	S 4 a 3
	$S \longrightarrow a *$	$S \longrightarrow * a$	reduce 2
4	$S \longrightarrow a S *$		reduce 1

The only inadequate state is state 3. For state 3, $Z_0 = \left| a \right|$. Since production 2 is $S \longrightarrow a *$. We must find $\left[S \right]$.

$$\left[S \right] = \left| S \right|$$

There are S-read transition to states 2 and 4. There is an $\dashv$ -read transition from state 2. Therefore $Z_2 = \left| \dashv \right|$. Since $Z_2 \cap Z_0 = \emptyset$ G_1 is SLR(1) . DeRemer's algorithm actually yields a parser with one more state than Knuth's. But this is entirely due to the use of the symbol $\vdash$.

In Table 5 we list the states and transitions for the LR(0) parser for grammar G_2 defined on page 17.

Table 5 States and Transitions for $M_0'(G_2)$

Number	State	Transitions To
0	S' ⟶ * ⊢ E ⊣	⊢ 1
1	S' ⟶ ⊢ * E ⊣ T ⟶ * P ↑T P ⟶ * i	E 2 (8
	E ⟶ * E + T T ⟶ * P	T 11 i 12
	E ⟶ * T P ⟶ * (E)	P 5 + 3
2	S ⟶ ⊢ E * ⊣ E ⟶ E * + T	⊣ accept
3	E ⟶ E + * T T ⟶ * P P ⟶ * i	T 4 (8
	T ⟶ * P ↑T P ⟶ * (E)	P 5 i 12
4	E ⟶ E + T *	reduce-1
5	T ⟶ P * ↑T T ⟶ P *	↑ 6 reduce-4
6	T ⟶ P ↑ * T T ⟶ * P P ⟶ * i	T 7 (8
	T ⟶ * P ↑T P ⟶ * (E)	P 5 i 12
7	T ⟶ P ↑T *	reduce-3
8	P ⟶ (* E) T ⟶ * P ↑T P ⟶ * i	E 9 (8
	E ⟶ * E + T T ⟶ * P	T 11 i 12
	E ⟶ * T P ⟶ * (E)	P 5
9	P ⟶ (E *) P ⟶ E * + T	) 10 + 3
10	P ⟶ (E) *	reduce-5
11	E ⟶ T *	reduce-2
12	P ⟶ i *	reduce-6

Table 5 contains one inadequate state, state 5. For state 5

$$Z_0 = \{ \uparrow \}$$
$$[T] = \{ T, E \} .$$

There are T-read transitions to states 7 and 11. There are E-read transitions to states 2 and 9. There are + and ⊣ read transitions from state 2. There are + and) read transitions from state 9. There are no read transitions from states 7 and 11.

Therefore $Z_4 = \{ +, \dashv,) \}$.

And since $Z_4 \cap Z_0 = \emptyset$ G_2 is SLR(1) . The only time the parser will have to scan the next symbol is when it is in state 5. If it scans an element of Z_4 it performs the action reduce-4. If it scans $\uparrow$ it performs the action read- 1 : ω' .

Notice that Knuth's 22 states have been reduced to 13.

We will conclude this section by attempting to find an LR(1) parser for grammar G_3 defined on page 22 by using DeRemer's algorithm. The states and transitions for the LR(0) parser for grammar G_3 are given in Table 6. In Table 6 we use the shorthand

$$S \longrightarrow * T_1 w \mid * T_2 x \mid * T_3 w \mid * T_4 x$$

for $S \longrightarrow * T_1 w$, $S \longrightarrow * T_2 x$, $S \longrightarrow * T_3 w$, $S \longrightarrow * T_4 x$

Table 6 States and Transitions for $M_0'(G_3)$

Number	State	Transition	To
0	$S' \longrightarrow * \vdash S \dashv$	$\vdash$	1
1	$S' \longrightarrow \vdash * S \dashv$		
	$S \longrightarrow * T_1 w \mid * T_2 x \mid * T_3 w \mid * T_4 x$	T_1	26
	$T_1 \longrightarrow * a \mid * b\ U_1 \mid * c\ U_1\ p \mid * c\ U_1\ r$	T_2	28
	$T_2 \longrightarrow * a \mid * b\ U_2 \mid * c\ U_2\ q \mid * c\ U_2\ s$	T	30
	$T_3 \longrightarrow * b\ U_3\ p \mid * b\ U_3\ q \mid * c\ U_3$	T_4	32
	$T_4 \longrightarrow * b\ U_4\ r \mid * b\ U_4\ s \mid * c\ U_4$		
2	$T_1 \longrightarrow a *$	reduce-5	
	$T_2 \longrightarrow a *$	reduce-6	
3	$T_1 \longrightarrow b * U_1$		
	$T_3 \longrightarrow b * U_3\ p \mid b * U_3\ q$	U_1	18
	$T_2 \longrightarrow b * U_2$		
	$T_4 \longrightarrow b * U_4\ r \mid b * U_4\ s$	U_2	19
	$U_1 \longrightarrow * d\ T_3$ $\quad U_3 \longrightarrow d\ T_1$	U_3	20
	$U_2 \longrightarrow * d\ T_4$ $\quad U_4 \longrightarrow d\ T_2$	U_4	23
4	$T_1 \longrightarrow c * U_1\ p \mid c * U_1\ r$ $\quad T_3 \longrightarrow c * U_3$	U_1	10
	$T_2 \longrightarrow c * U_2\ q \mid c * U\ s$ $\quad T_4 \longrightarrow c * U_4$	U_2	13
	$U_1 \longrightarrow * d\ T_3$ $\quad U_3 \longrightarrow * d\ T_1$	U_3	16
	$U_2 \longrightarrow * d\ T_4$ $\quad U_4 \longrightarrow * d\ T_2$	U_4	17

	a	2
	b	3
	c	4
	S	34
	d	5
	d	5

Table 6 Continued

Number	State	Transition To
5	$U_1 \longrightarrow d * T_3$ $\quad U_3 \longrightarrow d * T_1$ $U_2 \longrightarrow d * T_4$ $\quad U_4 \longrightarrow d * T_2$ $T_1 \longrightarrow * a \mid * b\ U_1 \mid * c\ U_1\ p \mid * c\ U_1\ r$ $T_2 \longrightarrow * a \mid * b\ U_2 \mid * c\ U_2\ q \mid * c\ U_2\ s$ $T_3 \longrightarrow * b\ U_3\ p \mid * b\ U_3\ q \mid * c\ U_3$ $T_4 \longrightarrow * b\ U_4\ r \mid * b\ U_4\ s \mid * c\ U_4$	T_1 6 a 2 T_2 7 b 3 T_3 8 c 4 T_4 9
6	$U_3 \longrightarrow d\ T_1\ *$	reduce-21
7	$U_4 \longrightarrow d\ T_2\ *$	reduce-22
8	$U_1 \longrightarrow d\ T_3\ *$	reduce-19
9	$U_2 \longrightarrow d\ T_4\ *$	reduce-20
10	$T_1 \longrightarrow c\ U_1 * p$ $\quad T_1 \longrightarrow c\ U_1 * r$	p 11 r 12
11	$T_1 \longrightarrow c\ U_1\ p\ *$	reduce-13
12	$T_1 \longrightarrow c\ U_1\ r\ *$	reduce-14
13	$T_2 \longrightarrow c\ U_2 * q$ $\quad T_2 \longrightarrow c\ U_2 * s$	q 14 s 15
14	$T_2 \longrightarrow c\ U_2\ q\ *$	reduce-15
15	$T_2 \longrightarrow c\ U_2\ s\ *$	reduce-16
16	$T_3 \longrightarrow c\ U_3\ *$	reduce-17

Table 6 Continued

Number	State		Transition To
17	$T_4 \longrightarrow c\ U_4\ *$		reduce-18
18	$T_1 \longrightarrow b\ U_1\ *$		reduce-7
19	$T_2 \longrightarrow b\ U_2\ *$		reduce-8
20	$T_3 \longrightarrow b\ U_3\ *\ p$	$T_3 \longrightarrow b\ U_3\ *\ q$	p 21 q 22
21	$T_3 \longrightarrow b\ U_3\ p\ *$		reduce-9
22	$T_3 \longrightarrow b\ U_3\ q\ *$		reduce-10
23	$T_4 \longrightarrow b\ U_4\ *\ r$	$T_4 \longrightarrow b\ U_4\ *\ s$	r 24 s 25
24	$T_4 \longrightarrow b\ U_4\ r\ *$		reduce-11
25	$T_4 \longrightarrow b\ U_4\ s\ *$		reduce-12
26	$S \longrightarrow T_1\ *\ w$		w 27
27	$S \longrightarrow T_1\ w\ *$		reduce-1
28	$S \longrightarrow T_2\ *\ x$		x 29
29	$S \longrightarrow T_2\ x\ *$		reduce-2
30	$S \longrightarrow T_3\ *\ w$		w 31
31	$S \longrightarrow T_3\ w\ *$		reduce-3

Table 6 Continued

Number	State	Transition To
32	$S \longrightarrow T_4 * x$	x 33
33	$S \longrightarrow T_4 x *$	reduce-4
34	$S' \longrightarrow \vdash S * \dashv$	$\dashv$ accept

Table 6 contains one inadequate state, state 2.

$$\left[T_1\right] = \left[T_2\right] = \left\{T_1, T_2, T_3, T_4, U_1, U_2, U_3, U_4\right\}$$

and for state-2

$$Z_5 = Z_6 = \left\{p, q, r, s\right\}$$

Therefore G_3 is not SLR(1) . In fact it is easy to see that G_3 will not be SLR(k) for any k . G_3 is therefore an LR(1) grammar for which DeRemer's algorithm will not work.

6. SOME EXAMPLES

In the next sections I will reverse the usual order of things by applying my algorithm to several grammars before actually stating the algorithm and proving that it works. Anyone who objects to this format should skip this section now and return to it later.

Informally, this algorithm starts, as does DeRemer's, by applying Knuth's algorithm to a grammar G with $k = 0$. If G is LR(0) then build the parser $M'_0(G)$ and we are done. If G is not LR(0) and we wish to find out if G is LR(k) for a given k, look at each inadequate state. What we wish to do is look backwards and forwards through the table of states for $M'_0(G)$ and find for each possible stack sequence or string of states on the stack, σ, for each possible inadequate state, I, and for each possible action from state I, z, the set $Y_k(\sigma, I, z) = \{k : \omega' \mid$ for some $\omega \in L(G)$, $M'_0(G)$ acting correctly upon the input string $\vdash\omega\dashv$ will arrive at state I with the string $\sigma \in \Gamma^*$ on the stack, ω' = the unread portion of $\vdash\omega\dashv$ and z being a correct parsing action from I$\}$. We will use z_0 to denote the action read and z_p to denote the action reduce-p. We will call $Y_k(\sigma, I, z)$ the k-look ahead set associated with σ, I and z. G is LR(k) if and only if $z \neq z'$ implies that

$$Y_k(\sigma, I, z) \cap Y_k(\sigma, I, z') = \emptyset \qquad \forall \sigma, I, z, z' .$$

If $z \neq z'$ implies that

$$Y_k(\sigma, I, z) \cap Y_k(\sigma', I, z') = \emptyset \quad \forall I, \sigma, \sigma', z, z'$$

then for each state I, let

$$Z_p = \bigcup_{\sigma \in \Gamma^*} Y_k(\sigma, I, z_p)$$

and build the LR(k) parser from the table of states and transitions and the sets Z_p exactly as in DeRemer's algorithm.

If $\exists I, \sigma, \sigma', z, z' \mid z \neq z'$ and

$$Y_k(\sigma, I, z) \cap Y_k(\sigma', I, z') \neq \emptyset$$

then we wish to split certain states in order that the parser may "remember" how it reached state I. Exactly how this is to be done will become clearer after working a few examples.

Now let's use this algorithm on grammar G_1 which is defined on page 16. The table of states and transitions for the LR(0) parser for G_1 is on page 33. The only inadequate state is state 3. Since the only read transition from state 3 is under a,

$$Y(\sigma, 3, z_0) = \left| a \right| \quad \forall \sigma \quad .$$

Before we can find out where the machine $M_0(G)$ can go from state 3 on taking a reduce-2 transition we must find out what states could precede state 3 on the stack. Therefore, associate with this instance of state 3 the set of k-look ahead configurations (k-LAC's) (Definition 9, page 60) $\left| (n_p, p_0) \mid p = 2 \right| = \left| (1, S) \right|$.

The meaning of $(1, S)$ is that we must pop the top item off the stack and enter the state T' such that if T is now the item on top of the stack then $T \xrightarrow{S} T'$.

From Table 4 we see that states 1 and 3 both have a-read transitions to state 3. From state 1 we have $1 \xrightarrow{S} 2$ and from state 2 the only transition is $\dashv$-accept. Therefore associate the set of k-LAC configurations $\{\dashv\}$ with state 1. From state 3 we have $3 \xrightarrow{S} 4$, and the only transition from state 4 is reduce 1. Since

$$n_1 = 2$$

we must now pop state 4 and one more item off the stack. Therefore associate with this instance or state 3 the set of k-LAC's

$$\{(n_p - 1), p_0) \mid p = 1\} = \{(1, S)\} \quad .$$

Thus we have the graph in Figure 1. We have numbered each node. The meaning of the graph is that if $M_0'(G)$ is in state 3 the stack must be of the form $\sigma, 1, 3^n, 3$ for some $\sigma \in \Gamma^*$ and $n \geq 0$. Since we have associated $\{(1, S)\}$ with state 3 in both instances, certainly

$$Y_1([\sigma, 1, 3^n, 3], 3, z_2) = \{\dashv\} \quad \forall n \geq 0 \text{ and } \forall \sigma .$$

This is reflected in Table 7.

Figure 1 Possible Stack Sequences for State 3, $M_0'(G_1)$

Table 7 Sets of k-LAC'S for State 3, $M_0'(G_1)$

Terminal	Node	z_0	z_2
	2	$\{a\}$	$\{(1, S)\}$
Yes	1	$\{a\}$	$\{\dashv\}$

A node i is terminal if and only if the set of k-LAC's associated with node i and action z_j for all j is a set of strings in V'^*. In this case we shall also refer to the set as being terminal.

Thus letting

$$Z_0 = \bigcup_{\sigma \in \Gamma^*} Y_1(\sigma, 3, z_0) = \{\dashv\}$$

and

$$Z_2 = \bigcup_{\sigma \in \Gamma^*} Y_1(\sigma, 3, z_2) = \{a\}$$

we now build our parser from Table 4 and the Z_p's defined above as described in Section 5. Notice that we end up with exactly the same parser as DeRemer's.

Now let's try this method on grammar G_2 defined on page 17. The table of states and transitions for $M_0'(G_2)$ is on page 34. The only inadequate state is state 5.

$$Y_1(\sigma, 5, z_0) = \left\{\uparrow\uparrow\right\} \qquad \forall\sigma$$

Before we can find out where the machine $M_0'(G_2)$ can go from state 5 on taking a reduce-4 transition, we must find out which states could precede state 5 on the stack. Therefore associate with this instance of state 5 the set of k-LAC's

$$\left\{(n_p, p_0) \mid p = 4\right\} = \left\{(1, T)\right\}$$

states 1, 3, 6 and 8 all have transitions to state 5. From state 1 we have $1 \xrightarrow{T} 11$. From 11 we have only a reduce-2 transition. Pop state 11 off the stack and go to state 2. From state 2 we have $2 \xrightarrow{+} 3$ and $2 \xrightarrow{\dashv}$ accept . Therefore associate with this instance of state 1 the set of k-LAC's $\left\{+, \dashv\right\}$. From state 3 we have $3 \xrightarrow{T} 4$ and from state 4 we have reduce-1. Therefore, associate with this instance of state 3 the set of k-LAC's

$$\left\{(n_p - 1, p_0) \mid p = 1\right\} = \left\{(2, E)\right\} .$$

We wish to continue this process of looking backwards and forwards along all possible paths until we can associate with a node a set of terminal k-LAC's for each action or else a set of k-LAC's that has already been associated with some other node on the path which is occupied by the same state. The result of this process is shown in

Figure 2 and Table 8. The loop (5, 6) in the graph means that

$$Y_k(\lfloor\sigma, (5, 6)^n, 5\rfloor, 5, z) = Y_k(\lfloor\sigma, (5, 6)^m, 5\rfloor, 5, z) \forall\sigma, z, n \text{ and } m .$$

For now we need only consider the terminal nodes of the graph in Figure 2, nodes 1, 3, 6 and 8. For example

$$Y_k(\lfloor\sigma, 1, 2, 3, (5, 6)^n, 5\rfloor, 5, z_4) = \lfloor +, \dashv\rfloor \qquad \forall\sigma, n .$$

Since

$$\bigcup_{\sigma} Y_k(\sigma, 5, z_4) = \lfloor +,), \dashv\rfloor$$

and

$$\bigcup_{\sigma} Y_k(\sigma, 5, z_0) = \lfloor \uparrow\rfloor$$

we may let

$$Z_0 = \lfloor \uparrow\uparrow\rfloor \qquad \text{and} \qquad Z_4 = \lfloor +,), \dashv\rfloor .$$

We may now build our parser from Table 5 and the sets Z_p defined above as described in Section 5. Again our final parser is precisely DeRemer's SLR(1) parser.

Now let's take a harder example. Grammar G_3 is defined on page 22. The states and transitions for $M_0'(G_3)$ are listed in Table 6 on page 36.

State 2 is the only inadequate state. Two actions are possible from state 2; reduce-5 and reduce-6. Figure 3 and Table 9 show the possible stack sequences and the associated k-LAC's for each node of the graph.

Figure 2 Possible Stack Sequences For State 5, $M_0'(G_2)$

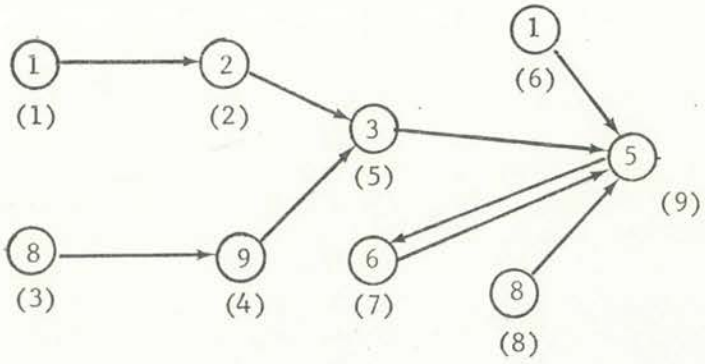


Table 8 Sets of k-LAC's For State 5, $M_0'(G_2)$

Terminal	Node	z_0	z_4
Yes	1	↑	), +
	2	↑	(1, E)
Yes	3	↑	1, +
	4	↑	(1, E)
	5	↑	(2, E)
Yes	6	↑	1, +
	7	↑	(2, T)
Yes	8	↑	), +
	9	↑	(1, T)

Figure 3 Possible Stack Sequences For State 2 $M_0'(G_3)$

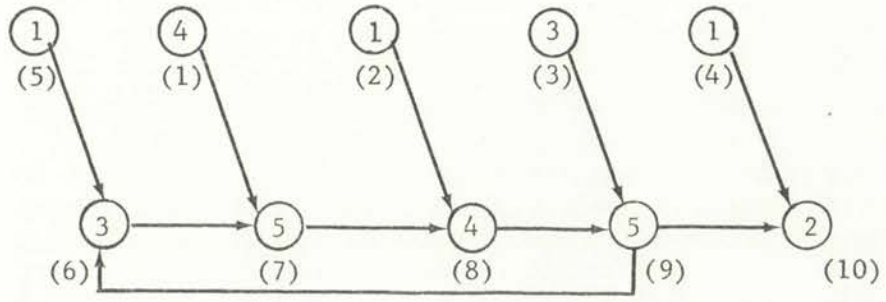


Table 9 Sets of k-LAC's For State 2 $M_0'(G_3)$

Terminal	Node	z_5	z_6
Yes	1	{p, r}	{q, s}
Yes	2	{w}	{x}
Yes	3	{p, q}	{r, s}
Yes	4	{w}	{x}
Yes	5	{w}	{x}
	6	$\{(1, T_1)\}$	$\{(1, T_2)\}$
	7	$\{(1, U_1)\}$	$\{(1, U_2)\}$
	8	$\{(1, T_3)\}$	$\{(1, T_4)\}$
	9	$\{(1, U_3)\}$	$\{(1, U_4)\}$
	10	$\{(1, T_1)\}$	$\{(1, T_2)\}$

Unfortunately in this case it is clear from Table 9 that

$$\left[\bigcup_{\sigma \in \Gamma^*} Y_1(\sigma, 2, z_5) \right] \cap \left[\bigcup_{\sigma \in \Gamma^*} Y_1(\sigma, 2, z_6) \right] \neq \emptyset \quad .$$

Therefore the tactics used in the preceding two examples will not work here. It is also clear from Table 9 that given σ

$$Y_1(\sigma, 2, z_5) \cap Y_1(\sigma, 2, z_6) = \emptyset$$

and therefore G_3 is LR(1) . What we need to do is this:

Let $B_{(i, j)}$ = the set associated with terminal node i and action z_j in Table 9

Let $A_{(1, 5)} = \{p, r, w\} = \bigcup_{i \in J_1} B(i, 5)$ where $J_1 = \{1, 2, 4, 5\}$

$A_{(1, 6)} = \{q, s, x\} = \bigcup_{i \in J_1} B(i, 6)$

$A_{(2, 5)} = \{p, q\} = \bigcup_{i \in J_2} B(i, 5)$ where $J_2 = \{3\}$

$A_{(2, 6)} = \{r, s\} = \bigcup_{i \in J_2} B(i, 6)$

We have $j \neq j'$ implies $A_{(h, j)} \cap A_{(h, j')} = \emptyset$ $\forall h$

and that $\forall h$ and $h' \mid h \neq h'$, $\exists j$ and $j' \mid j \neq j'$ and

$$A_{(h, j)} \cap A_{(h', j')} \neq \emptyset .$$

Now to each terminal node i and each action z_j associate the appropriate set $A_{(h, j)} \mid i \in J_h$ as in Table 10.

If S does appear on the graph create a new state which will be an 11-tuple. For example, create state

(3, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0)

and number it state 36. Here again the three means that this state was created from state 3, the zeros mean that state 3 does not occupy these nodes in the graph. The 2 means that state 3 occupies terminal node 3 and that for each j, $A_{(2, j)}$ is associated with node 3 and action z_j in Table 10. The one in the sixth position is because state 35 has a one in the fifth position and

$$5 \longrightarrow 6$$

in the graph in Figure 3. Now let

$$35 \xrightarrow{b} 36 \quad .$$

For the same reason let

$$35 \xrightarrow{a} 37$$

where state 37 is represented by the 11-tuple

(2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1) .

Continue this process until done. Now change all transitions to state 1 to go to state 35 and delete all states which appear in Figure 3, namely states 1 through 5. We now have the states and transitions listed in Table 11 .

Table 11 States And Transitions For Grammar G_3 Revised

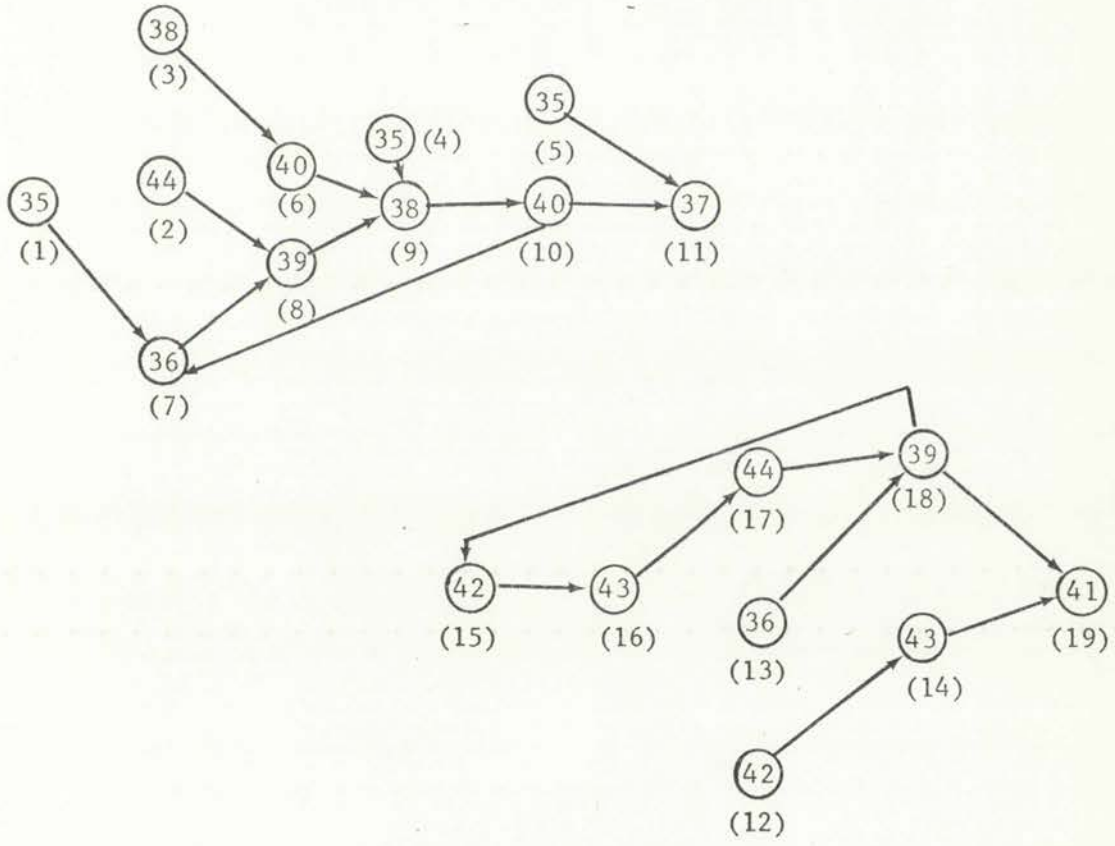
Number	State	Transition To
0	S' ⟶ * ⊢ S ⊣	⊣ 35
	States 6 Through 34 As In Table 6	
35	(1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0)	T_1 26 T_4 32 a 37 T_2 28 S 34 b 36 T_3 30 c 38
36	(3, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0)	U_1 18 U_3 20 d 39 U_2 19 U_4 23
37	(2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)	reduce-5 reduce-6
38	(4, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0)	U_1 10 U_3 16 d 40 U_2 13 U_4 17
39	(5, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0)	T_1 6 T_3 8 a 41 c 38 T_2 7 T_4 9 b 42
40	(5, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0)	T_1 6 T_3 8 a 37 c 38
41	(2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2)	reduce-5 reduce-6
42	(3, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0)	U_1 18 U_3 20 d 43 U_2 19 U_4 23
43	(5, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0)	T_1 6 T_3 8 a 41 c 44 T_2 7 T_4 9 b 42
44	(4, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0)	U_1 10 U_3 16 d 39 U_2 13 U_4 17

In Table 11, for states 35 through 44 the configurations are the same as for the state in Table 6 whose number corresponds to the zeroth member of the 11-tuple in the state column.

Now if we redraw the graph in Figure 3 to reflect the changes we have just made we get the graphs in Figure 4.

Figure 4 Revised Graph of Stack Sequences For Grammar G_3

States 37 and 41



We now have two inadequate states, states 37 and 41. Revising Table 9 we get Table 12.

Table 12 Revised k-LAC's For States 37 and 41 Grammar G_3

Terminal	Node	z_5	z_6
Yes	1	w	x
Yes	2	p, r	q, s
Yes	3	p, r	q, s
Yes	4	w	x
Yes	5	w.	x
	6	$(1, U_1)$	$(1, U_2)$
	7	$(1, T_1)$	$(1, T_2)$
	8	$(1, U_1)$	$(1, U_2)$
	9	$(1, T_3)$	$(1, T_4)$
	10	$(1, U_3)$	$(1, U_4)$
	11	$(1, T_1)$	$(1, T_2)$
Yes	12	p, q	r, s
Yes	13	p, q	r, s
	14	$(1, U_3)$	$(1, U_4)$
	15	$(1, T_1)$	$(1, T_2)$
	16	$(1, U_1)$	$(1, U_2)$
	17	$(1, T_3)$	$(1, T_4)$
	18	$(1, U_3)$	$(1, U_4)$
	19	$(1, T_1)$	$(1, T_2)$

Table 12 tells us that for state 37 we may let

$$Z_5 = \{p, r, w\} \qquad \text{and}$$
$$Z_6 = \{q, s, x\}$$

and for state 41 we may let

$$Z_5 = \{p, q\} \qquad \text{and}$$
$$Z_6 = \{r, s\} \ .$$

Now since for each inadequate state

$$Z_5 \cap Z_6 = \emptyset$$

we may build our parser from Table 11 and the sets Z_p defined above exactly as in Section 5. Note that for state 37 $Z_j = A_{(1, j)} \quad \forall_j$ and for state 41 $Z_j = A_{(2, j)} \quad \forall_j$. This is no coincidence. Our final parser has 40 states. This compares favorably with the 90 state parser built with Knuth's algorithm.

This section was neither complete nor rigorous. However hopefully the reader now has an intuitive feel for what is going on here and will be in a better position to follow the more rigorous exposition in the subsequent sections.

7. DEFINITIONS AND THEOREMS

We wish now to define a subclass of parsing machines (Definition 2, page 9), which we will call LR(k) parsing machines for the grammar G (LR(k)PM(G)), which will parse (possibly non-deterministicly) the grammar $G = (V_T, V_N, S, P)$. Number the productions in P 1 through r and let $S' \longrightarrow \vdash S \dashv$ (or $S' \longrightarrow S \dashv^k$) be the zeroth production where $\left| S, \vdash, \dashv \right| \cap V = \emptyset$. Define a configuration as an ordered pair (p, j) where $0 \leq j \leq n_p$ and $0 \leq p \leq r$. Here again, if production p is $A \longrightarrow \alpha_1 \ldots \alpha_m$ with $\alpha_i \in V \ \forall i \mid 1 \leq i \leq m$, then

$$n_p = m \ , \ p_0 = A \quad \text{and}$$

$$p_i = \alpha_i \ \forall i \mid 1 \leq i \leq m \ .$$

If C is a non empty set of configurations, then C* or the closure of C is the smallest set of configurations $\left| C^* \supset C \right.$ and if $(p, j) \in C^*$ with $j < n_p$ and $p_{j+1} \in V_N$ then $(q, 0) \in C^* \ \forall q \mid q_0 = p_{j+1}$.

Definition 5a

A parsing machine $M = (K, V'_T, S_0, P, N, F, \delta, k)$ is a member of the subclass, Incomplete LR(k) Parsing Machines For G (ILR(k)PM(G)) if and only if

$$V'_T = V_T \cup \left| \vdash, \dashv \right|$$

$$P = \left| 1, \ldots, r \right|$$

$$N(p) = n_p$$

(a) Conditions on K

1. Each element $S \in K$ is an $r + 3$-tuple

   $S = (i, C_i^*, Z_0^i, \ldots, Z_r^i)$ where

   i is a non-negative integer which uniquely defines S,

   C_i^* is a closed set of configurations and

   $\forall_p \mid 0 \leq p \leq r \quad Z_p^i \subset W_k$ where

   $W_0 = V_T'$ and

   $\forall k > 0 \quad W_k = \left| \omega \in V_T'^* \mid |\omega| = k \text{ or } (|\omega| < k \text{ with } \dashv \text{ suf } \omega) \right|$

2. $\exists (q, j) \in C_i^* \mid q_{j+1} = a$ if and only if $\exists \alpha \in Z_0^i \mid 1 : \alpha = a$

3. If $k = 0$ and $(p, n_p) \in C_i^*$ then $Z_p^i = W_0$

4. $\exists (p, n_p) \in C_i^*$ if and only if $Z_p^i \neq \emptyset$

5. $S_0 = (0, \left|(0, 0)\right|^*, Z_0^0, \ldots Z_r^0)$

6. If $(0, 0) \in C_i^*$ then $i = 0$

7. $S = (i, C_i^*, Z_0^i, \ldots Z_r^i) \in F$ if and only if $(0, 2) \in C_i$

(b) Conditions on δ

Let $S_i = (i, C_i^*, Z_0^i, \ldots Z_r^i)$

1. δ is a function from $(K \times W_k \times K)$ into the collection of subsets of $K \times (P \cup \left| \epsilon \right|)$

2. If $\exists (q, j) \in C_i^* \mid q_{j+1} = a$ where $a \in V_T \cup \left| \dashv \right|$ then $\forall \alpha \in Z_0^i \mid 1 : \alpha = a \; \exists ! \; S_n = (n, C_n^*, Z_0^n, \ldots Z_r^n)$ $(S_n, \epsilon) \in \delta (S_i, \alpha, S_i)$ and furthermore n is the same for all such α and $C_n = \left| (q, j+1) \mid (q, j) \in C_i^* \text{ and } q_{j+1} = a \right|$

3. If $(S, \epsilon) \in \delta (S_j, \alpha, S_i)$ then $i = j$ and $\alpha \in Z_0^i$

4. If $\exists (q, j) \in C_i^* \mid q_{j+1} = p_0$
then $\forall m$ and $\alpha \mid S_m = (m, C_m^*, Z_0^m, \ldots . Z_r^m)$, $\alpha \in Z_p^m$
and $(p, n_p) \in C_m^*$ $\exists !$ $S_n = (n, C_n^*, Z_0^n, \ldots Z_r^n) \mid$
$(S_n, p) \in \delta (S_m, \alpha, S_i)$ and furthermore n is the same for
all such m and α and
$C_n = \left| (q, j + 1) \mid (q, j) \in C_i^* \text{ and } q_{j+1} = p_0 \right|$
5. If $(S, p) \in \delta (S_m, \alpha, S_i)$
then $\alpha \in Z_p^m$, $(p, n_p) \in C_m^*$ and $\exists (q, j) \in C_i^* \mid q_{j+1} = p_0$
6. M contains no useless states

We say that M is a deterministic member of ILR(k)PM(G) if and only if $\forall_i$ $j \neq j'$ implies that $Z_j^i \cap Z_{j'}^i = \emptyset$.
It is important to notice that if M is deterministic then

$(S, \Pi) \in \delta (S_m, \alpha, S_i)$ and $(S', \Pi') \in \delta (S_m, \alpha, S_{i'})$

where Π and $\Pi' \in (P \cup \{\epsilon\})$ implies that

$$\Pi = \Pi'$$

In other words Π is uniquely determined by m and α.
The meaning of this is that M is started in state S_0 with S_0 only on the stack and is fed an input string ω. The input head is placed over the first k symbols of ω. If M is in state $S = (i, C^*, Z_0^i, \ldots , Z_r^i)$ and the input head reads $\alpha \in Z_0$ then M may go to state S' such that $(S', \epsilon) \in \delta (S, \alpha, S)$ while pushing S' on the stack and advancing the input head one symbol. If $\alpha \in Z_p$ then M may pop the top n_p items off the stack and enter state S' such that $(S', p) \in \delta (S, \alpha, T)$ where T is the state on top of the stack, while pushing S' on the stack and outputting p . If M is

in state $S \in F$ and the input head reads $\dashv$ then accept the string. Otherwise reject it.

Definition 5b

Let $M \in ILR(k)PM(G)$ and $S = (i, C_i^*, Z_0^i, \ldots Z_r^i)$ be a state of M

The set Z_j^i is <u>maximal</u> if and only if

$$Z_0^i = \left\{ \omega \in W_k \;\middle|\; 1 : \omega = p_{j+1} \text{ where } (p, j) \in C_i^* \right\},$$

For $j > 0$ and $(p, n_p) \in C_i^*$, $Z_j^i = W_k$ and

For $j > 0$ and $(p, n_p) \notin C_i^*$, $Z_j^i = \emptyset$.

Definition 5c

Let $M = (K, V_T', S_0, P, N, F, \delta, k) \in ILR(k)PM(G)$

Let $M' = (K', V_T', S_0', P, N, F', \delta', k)$ where

$S_i' = (i, C_i^*, Z_0^{i'}, \ldots, Z_r^{i'}) \in K'$ if and only if

$S_i = (i, C_i^*, Z_0^i, \ldots Z_r^i) \in K$ where

each $Z_j^{i'}$ is maximal and F' and δ' correspond to F and δ respectively in the obvious manner.

$M \in \underline{LR(k)PM(G)}$ if and only if M accepts $\omega \in V_T'^*$ while outputting $\Pi \in P^*$ if and only if M' accepts ω while outputting Π.

Theorem 1:

$M \in LR(k)PM(G)$ if and only if $\omega \in L(G)$ and $\Pi \in P^*$ is a parse of ω then M will accept $\vdash\omega\dashv$ while outputting Π.

Proof: Suppose M accepts $\vdash\omega\dashv$ while outputting Π. Then since every transition in M corresponds to reading a symbol or

performing a reduction and M goes to a state $S \in F$ after reading $\vdash\omega$ if and only if M has reduced $\vdash\omega$ to $\vdash S$, $\omega \in L(G)$. Since M outputs p if and only if M uses production p to perform a reduction, Π is a parse of ω.

Now suppose $\omega \in L(G)$ and Π is a parse of ω, M will certainly accept ω while outputting π if the sets M_j^i are large enough. But this is precisely the condition imposed in Definition 5c.

Definition 6

Let $M \in LR(k)PM(G)$

We say that there exists an a-read transition from state S to state S' if and only if

$$\exists\ \alpha \mid 1 : \alpha = a \quad \text{and} \quad (S', \epsilon) \in \delta(S, \alpha, S)$$

we write $S \xrightarrow{a} S'$. If M is in state S, reads α and goes to state S', then we say that M has performed the action read which we will label z_0. We say that there exists an A-read transition from state S to state S' if and only if

$$\exists\ T, \alpha \text{ and } p \mid p_0 = A \quad \text{and} \quad (S', p) \in \delta(T, \alpha, S)$$

we write $S \xrightarrow{A} S'$.

We say that there exists a reduce-p transition from state T if and only if

$$\exists\ \alpha, S \text{ and } S' \mid (S', p) \in \delta(T, \alpha, S)$$

we write $T \xrightarrow{\#p}$.

If M is in state T and goes to state S' while outputting p, then we say that M has performed the action reduce-p which we will label z_p.

We should notice that the relation δ is uniquely defined by the set of all transitions and the sets Z_j^i .

Definition 7

$M_k(G)$ is the parsing machine built by Knuth's algorithm. $M_k'(G)$ is the parsing machine built by DeRemer's algorithm.

It is clear that $M_k(G)$ and $M_k'(G)$ are both elements of $LR(k)PM(G)$. From here on assume $M \in LR(k)PM(G)$.

Definition 8

A <u>stack sequence</u> (SS) in M is a sequence of states $\sigma = T_n \ldots T_0$ such that $T_i \xrightarrow{\alpha_i} T_{i-1}$ where $\alpha_i \in V'$. If $T_n = S_0$ then σ is a <u>complete stack sequence</u> (CSS) . We will say that σ <u>reads</u> the string $\alpha_n \ldots \alpha_1$.

Definition 9

Let $k > 0$. A <u>k-look ahead configuration</u> (k-LAC) is a string $\alpha \in W_k$ or $\alpha(n, A)$ where $\alpha \in V_T^*$, $|\alpha| < k$ and $\alpha \notin W_k$, n is a non-negative integer such that $0 \leq n \leq \max_p n_p$ and $A \in V_N$.

Definition 10

Let Σ be a sequence of states $T_0 \ldots T_n$ and Π a sequence of actions $\tau_0 \ldots \tau_{n-1}$. The ordered pair (Σ, Π) is a <u>path</u> through M , if and only if, if M is started in state T_0 with only state T_0 on the stack, then M may for all i such that $0 \leq i \leq n$ perform the action τ_i from state T_i and go to state T_{i+1} subject to the stack without ever popping state T_0 off the stack.

If T_0 is the start state and T_n is the final state, then (Σ, Π) is called a complete path.

Suppose $\sigma = T'_m \ldots T'_0$ is a SS with $T_0 = T'_0$ and $T'_i \xrightarrow{\beta_i} T'_{i-1}$ for some β_i $\forall i \mid 0 < i \leq m$. Then (Σ, Π, σ) is a conditional path if and only if, if M is started in state T_0 with σ on the stack, then M may for all i such that $0 \leq i < n$ take a τ_i transition from state T_i to state T_{i+1} without ever popping state T'_m off the stack. If T'_m is the start state and T_n is the final state then (Σ, Π, σ) is called a complete conditional path. Every path is a conditional path with $\sigma = T_0$.

We say that the (conditional) path (Σ, Π, σ) reads $\alpha = \alpha_0 \ldots \alpha_{n-1}$ where either τ_i is the action read and $T_i \xrightarrow{\alpha_i} T_{i+1}$ or τ_i is a reduce action and $\alpha_i = \epsilon$. If $T_n \in F$ then (Σ, Π, σ) also reads $\alpha\dashv$.

Every complete (conditional) path in M yields a parse to some sentence (sentential form) in G.

Definition 11

Let $\sigma = T_s \ldots T_0$ be a SS

n be an integer such that $0 \leq n \leq s$ and

z be an action

$X_k(\sigma, n, z)$ is a function into the set of all sets of k-LAC's defined recursively on n in the following manner.

Let $n = 0$

$z = z_p$ where production p is not of the form $A \longrightarrow \epsilon$

then $X_k(\sigma, n, z) = \left| (n_p, p_0) \right|$.

Let $n = 0$

$z = z_p$ where production p is of the from $A \longrightarrow \epsilon$

then let $h = k$

$$\sigma' = U_0 U_1 \mid U_0 = T_0 \text{ and } U_0 \xrightarrow{p_0} U_1$$

follow each conditional path (Σ, Π, σ') until one of the four conditions below occurs:

(a) 1. (Σ, Π, σ') has read α such that $\alpha \in W_h$

2. It becomes necessary to pop state U_0 off the stack after reading α and while performing a reduce transition

3. (Σ, Π, σ') goes from a state U_n where some σ_1 is the sequence on the stack to a state U_m such that

$U_m = U_n$ and

σ_1 is the sequence on the stack while reading ϵ

4. (Σ, Π, σ') goes from a state U_n to a state U_m such that $U_m = U_n$ while reading ϵ without popping state U_n off the stack

Now $\beta \in X_k(\sigma, n, z)$ if and only if

(b) For some (Σ, Π, σ') condition 1 in (a) occurs with $\alpha = \beta$

$\beta(m, B) \in X_k(\sigma, n, z)$ if and only if

(c) For some (Σ, Π, σ') condition 2 in (a) occurs with

$\alpha = \beta$

m = number of items to be popped off the stack including U_0 and

B = the transition to be taken after popping m more items

off the stack.

Let $n = 0$

$z = z_0$

then let $h = k$

$U_0 = T_0$ and $\sigma' = U_0$

follow each conditional path (Σ, Π, σ') until one of the four conditions in (a) above occurs.

$\beta \in X_k(\sigma, n, z)$ if and only if (b).

$\beta(m,B) \in X_k(\sigma, n, z)$ if and only if (c).

Now let $n > 0$

1. If $\gamma \in X_k(\sigma, n-1, z)$ then $\gamma \in X_k(\sigma, n, z)$
2. If $\gamma(m, A) \in X_k(\sigma, n-1, z)$ with $m > 1$
   then $\gamma(m-1, A) \in X_k(\sigma, n, z)$
3. If $\gamma(1, A) \in X_k(\sigma, n-1, z)$
   then let $h = k - |\gamma|$
   $\sigma' = U_0U_1 \mid U_0 = T_n$ and $U_0 \xrightarrow{A} U_1$
   follow each conditional path (Σ, Π, σ') until one of the four conditions in (a) above occurs.
   $\gamma\beta \in X_k(\sigma, n, z)$ if (b)
   $\gamma\beta\ (m, B) \in X_k(\sigma, n, z)$ if (c)
4. Otherwise a given k-LAC is not in $X_k(\sigma, n, z)$.

In Theorem 3 we will prove that one of the four conditions in (a) must occur within a bounded number of transitions depending only on M. Thus $X_k(\sigma, n, z)$ is well defined.

Notice that we did not have to define $X_k(\sigma, n, z)$ recursively on n. This choice was motivated by the fact that in the algorithm we will need to find $X_k(\sigma, n-1, z)$ as well as $X_k(\sigma, n, z)$, and there is less work involved in building $X_k(\sigma, n, z)$ from $X_k(\sigma, n-1, z)$ than in building $X_k(\sigma, n, z)$ from scratch.

The difference between the function X_k and the function Y_k from the previous section is that the argument σ of Y_k is a CSS whereas the argument σ of X_k is a SS. Let $\sigma = T_n \ldots T_0$ with $T_n = S_0$. Then, if G is LR(k), $X_k(\sigma, n, z) = Y_k(\sigma, T_0, z)$. In the previous section we used Y_k because we were being "intuitive." However, henceforth we will be concerned only with the function X_k.

Definition 12

$X_k(\sigma, n, z)$ is <u>terminal</u> if and only if every element of $X_k(\sigma, n, z)$ is a string in $V_T'^*$

Definition 13

$X_k(\sigma, n, z)$ is <u>complete</u> if and only if $\alpha \in X_k(\sigma, n, z)$

$\forall\ \alpha \in W_k \mid \exists$ a conditional path (Σ, Π, σ_n) which reads α where σ_n is the last $n+1$ items of the SS σ.

Theorem 2

Let $\sigma = T_n \ldots T_0$ and

$\sigma' = T'_m \ldots T'_0$ be SS's

$T_j = T'_j \ \forall j \mid 0 \le j \le h$ where $h \le n$ and $h \le m$

Then $X_k(\sigma, j, z) = X_k(\sigma', j, z)$ $\forall z$ and $\forall j \mid 0 \le j \le h$.

Proof: $X_k(\sigma, j, z)$ depends only on $T_j \ldots T_0$ and z.
(See Definition 11)

Theorem 3

The number of transitions a conditional path (Σ, Π, σ) where $\sigma = U_0 \ldots U_v$ may take before one of the four conditions in (a) of Definition 11 occurs is bounded by a bound W which depends only on M.

Proof: Let x = the number of states in M and $y = \text{ord}(V_N)$
We may have at most $k - 1$ read actions to a path else condition 1 occurs. Initially let $w = n$ where U_n is the state at the start of the path. If we take an a-read transition to a state U_m then let $w = m$. If we pop state U_w off the stack in taking a transition to state U_h then let $w = h$. The stack can never have more than $x - 1$ items beyond U_w else condition 4 occurs. The maximum length of the stack is therefore $k x + v$. But we can take at most $y - 1$ reduce transitions which pop everything beyond but not including some state U_n off the stack without taking an a-read transition else condition 3 occurs. But we can never pop U_0 off the stack else condition 2 occurs. Thus

$W = \sum_{j=0}^{k} \sum_{i=0}^{j x + v} y^i$ is certainly an upper bound.

It is intuitively obvious that any machine M for which there is a path even approaching this bound is "ridiculous."

Theorem 4

Every conditional path in $M \in LR(k)PM(G)$ has a complete extension.

Proof: This follows direct from the fact that M contains no useless states and that G contains no useless production. Thus every state is accessible from the start state and M contains no "dead ends."

Theorem 5

If condition 3 of (a) of Definition 11 occurs in following some conditional path $(\Sigma', \Pi', \sigma) \in M$, then G is ambiguous.

Proof: Let $\Sigma = T_0 \ldots T_n$ $\quad \Sigma' = T_0 \ldots T_m$

$\Pi = \tau_0 \ldots \tau_{n-1}$ $\quad \Pi' = \tau_0 \ldots \tau_{m-1}$ with $T_n = T_m$ and $n < m$

$\sigma = U_p \ldots U_0$ with $U_0 = T_0$ and condition 3 occurs in following (Σ', Π', σ) from T_n to T_m

Let $(\Sigma_1, \Pi_1, \sigma_1)$ be a complete extension of (Σ, Π, σ)

where $\Sigma_1 = T_0 \ldots T_n T'_0 \ldots T'_\ell$

and $\Pi_1 = \tau_0 \ldots \tau_{n-1} \tau'_{-1} \cdots \tau'_{\ell-1}$

$(\Sigma_1, \Pi_1, \sigma_1)$ yields a parse for some sentential form α.

But then $(\Sigma'_1, \Pi'_1, \sigma_1)$ with

$\Sigma'_1 = T_0 \ldots T_m T'_0 \ldots T'_\ell$

$\Pi'_1 = \tau_0 \ldots \tau_{m-1} \tau'_{-1} \cdots \tau'_{\ell-1}$ is complete and yields a different parse for the sentential form α. Therefore G is ambiguous.

Theorem 6

If condition 4 of (a) of Definition 11 occurs in following some conditional path $(\Sigma_1, \Pi_1, \sigma)$ then G is not LR(k) for any k .

Proof: Let $\Sigma_i = T_0 \ldots T_n (T_{n+1} \ldots T_m)^i$

$\Pi_i = \tau_0 \ldots \tau_{n-1} (\tau_n \ldots \tau_{m-1})^i$ with $T_n = T_m$ and $n < m$

$\sigma = U_p \ldots U_0$ with $U_0 = T_0$ and

condition 4 occurs in following $(\Sigma_1, \Pi_1, \sigma)$ from T_n to T_m . Since condition 4 occurs

$(\Sigma_i, \Pi_i, \sigma)$ is a conditional path $\forall i \mid i \geq 0$.

Let $(\Sigma'_i, \Pi'_i, \sigma')$ be a complete extension of

$(\Sigma_i, \Pi_i, \sigma)$ such that

$\Sigma'_i = T_0 \ldots T_n (T_{n+1} \ldots T_m)^i T_0^i \ldots T_h^i$

$\Pi'_i = \tau_0 \ldots \tau_{n-1} (\tau_n \ldots \tau_{m-1})^i \tau_{-1}^i \ldots \tau_{h-1}^i$ and

$\tau_n \ldots \tau_{m-1}$ is not a prefix of $\tau_{-1}^i \ldots \tau_{h-1}^i$.

For all $i \geq 0$, $(\Sigma'_i, \Pi'_i, \sigma')$ yields a parse of some sentential form α_i.

Let β_i be the portion of α_i read by $T_0^i \ldots T_h^i$.

Then given k, $\exists$ j and j' with $j' < j \mid k : \beta_j = k : \beta_{j'}$

and $\exists$ s $\mid -1 \leq s < m - n - 1$ and $\tau_s^j \neq \tau_{n+s+1}$ but

$\tau_{n+t+1} = \tau_t^j \quad \forall t \mid -1 \leq t \leq s$.

Let τ_{n+s+1} be the action reduce $A \longrightarrow \omega$. This means that $S' \longrightarrow^* \gamma A \beta_{j'} \longrightarrow \gamma \omega \beta_{j'}$ for some γ.

τ_s^j is a different action so $S' \longrightarrow^* \gamma' A' \beta \longrightarrow \gamma' \omega' \beta$

for some γ', A', β, ω' where $\gamma' \omega' \beta = \gamma \omega \beta_j$ with $\beta = \beta_j$

if τ_s^j is reduce $A' \longrightarrow \omega'$, and β suf β_j if τ_s^j is read. In either case G is not LR(k) .

Theorem 7

If condition 4 of (a) of Definition 11 is not encountered in building $X_k(\sigma, i, z)$ $\forall i \mid 0 \leq i \leq n$ then $X_k(\sigma, n, z)$ is complete.

Proof: Let $\sigma = T_m \ldots T_0$ and $\forall i \mid 0 \leq i \leq m$ let $\sigma_i = T_i \ldots T_0$. We build T_i by following all possible conditional paths (Σ, Π, σ_i) until:

(Σ, Π, σ_i) has read a string $\omega \in W_k$ (condition 1),

(Σ, Π, σ_i) must terminate (condition 2) or

(Σ, Π, σ_i) goes through a loop which neither reads anything nor changes the stack (condition 3).

Thus if there is a path (Σ, Π, σ_i) which reads ω and condition 3 occurs in following it then there is a path $(\Sigma', \Pi', \sigma_i)$ which reads ω and condition 3 does not occur in following it. Since by hypothesis condition 4 does not occur, it follows that $X_k(\sigma, n, z)$ is complete.

Theorem 8

If $\exists \sigma, n, z, z', \alpha$ with $z \neq z'$ and $\alpha \in W_k \mid$

$$\alpha \in X_k(\sigma, n, z) \cap X_k(\sigma, n, z')$$

then G is not LR(k).

Proof: Since $\alpha \in X_k(\sigma, n, z) \cap X_k(\sigma, n, z')$ there exists

$\Sigma_0, \Pi_0, \Sigma_1, \Pi_1 \mid (\Sigma_0, \Pi_0, \sigma)$ and $(\Sigma_1, \Pi_1, \sigma)$ both read α, $\Pi_0 = \tau_0 \ldots \tau_r$ and $\Pi_1 = \tau'_0 \ldots \tau'_s$ where $\tau_0 = z$ and $\tau'_0 = z'$. Assume without loss of generality that z is reduce $A \longrightarrow \omega$.

Let $(\Sigma'_0, \Pi'_0, \sigma')$ and $(\Sigma'_1, \Pi'_1, \sigma')$ be complete extentions of $(\Sigma_0, \Pi_0, \sigma)$ and $(\Sigma_1, \Pi_1, \sigma)$ respectively.

We have $S' \longrightarrow {}^*\beta\, A\alpha\delta \longrightarrow \beta\omega\alpha\delta$ for some β, δ

and $S' \longrightarrow {}^*\beta'\, A'\, \alpha' \longrightarrow \beta'\omega'\alpha'$

where $\beta'\omega'\alpha' = \beta\omega\alpha\delta'$ for some $\beta', A', \alpha', \omega'$ and δ'

and $\alpha' = \alpha\delta'$ if z' is reduce $A' \longrightarrow \omega'$

α' suf $\alpha\delta'$ if z' is read.

In either case G is not LR(k).

Theorem 9

If $\exists\, \sigma, n, z$ and z' with $z \neq z' \mid$

$\beta \in X_k(\sigma, n, z) \cap X_k(\sigma, n, z')$ with $\dashv$ suf β or

$\alpha(m, A) \in X_k(\sigma, n, z) \cap X_k(\sigma, n, z')$ for some α, m and A

Then G is ambiguous.

Proof: Suppose $\alpha(m, A) \in X_k(\sigma, n, z) \cap X_k(\sigma, n, z')$.

$\exists \quad \Sigma_0, \Pi_0, \Sigma_1, \Pi_1 \mid (\Sigma_0, \Pi_0, \sigma) \neq (\Sigma_1, \Pi_1, \sigma)$

and both conditional paths read α and upon ending both are doing precisely the same thing. This means we may complete both paths in exactly the same way. Thus let $(\Sigma'_0, \Pi'_0, \sigma')$ and $(\Sigma'_1, \Pi'_1, \sigma')$ be complete extension of $(\Sigma_0, \Pi_0, \sigma)$ and $(\Sigma_1, \Pi_1, \sigma)$ completed in the same way.

Then $(\Sigma_0', \Pi_0', \sigma')$ and $(\Sigma_1', \Pi_1', \sigma')$ each yield a different parse for some string $\beta \in L(G)$. Therefore G is ambiguous. If $\beta \in X_k(\sigma, n, z) \cap X_k(\sigma, n, z')$ with $\dashv$ suf β then there exists $\Sigma_0, \Pi_0, \Sigma_1, \Pi_1 \mid (\Sigma_0, \Pi_0, \sigma) \neq (\Sigma_1, \Pi_1, \sigma)$ and both read β. But then there exists $\sigma' \mid (\Sigma_0, \Pi_0, \sigma')$ and $(\Sigma_1, \Pi_1, \sigma')$ are complete extensions of $(\Sigma_0, \Pi_0, \sigma)$ and $(\Sigma_1, \Pi_1, \sigma)$ respectively and both read β. Therefore G is ambiguous.

Definition 14

Let $\sigma = T_\ell \ldots T_0$ with $\ell \geq m > n \geq 0$

$T_m \ldots T_n$ is a <u>k-loop</u> in σ if and only if

$T_m = T_n$ and $\forall z\ X_k(\sigma, n, z) = X_k(\sigma, m, z)$.

Theorem 10

Let $\sigma = T_n \ldots T_0$

If σ contains no k-loops and $X_k(\sigma, n, z)$ is not terminal, then n is bounded by some bound U which depends only on M.

Proof: Let $y = \text{ord } V_N$ $\quad x = \text{ord } V_T'$

$r = \max_p n_p$ $\quad w = \text{ord } K$

$s = \text{ord } P + 1$

There exist at most $z = x^{k-1}(x + yr)$ different k-LAC's, or at most 2^z different sets of k-LAC's. Therefore there exist no more than 2^{zs} different arrangements of s sets of k-LAC's. So any SS σ containing no k-loops is no longer than $w2^{zs}$.

Again it is intuitively obvious that any grammar G for which there exists a SS σ even approaching this bound is "ridiculous."

Theorem 11

If G is not LR(k) then there exists a SS $\sigma = T_n \ldots T_0$ in M where $T_0 = (i, C_i^*, Z_0^i, \ldots Z_r^i) \mid$

1. $X_k(\sigma, n, z)$ is terminal $\forall z$ but $\exists z \mid X_k(\sigma, n-1, z)$ is not terminal or $n = 0$
2. $\exists$ j and j' $\mid$ $j \neq j'$ and $Z_j^i \cap Z_{j'}^i \neq \emptyset$
3. σ contains no k-loops
4. $X_k(\sigma, n, z_j) \cap X_k(\sigma, n, z_{j'}) \neq \emptyset$ or $\exists z \mid$
5. $X_k(\sigma, n, z)$ is not complete
6. $X_k(\sigma, n-1, z)$ is not terminal or $n = 0$
7. $\exists$ j and j' $\mid$ $j \neq j'$ and $Z_j^i \cap Z_{j'}^i \neq \emptyset$
8. σ contains at most one k-loop and if it does it is of the form $T_n \ldots T_m$ where $n > m \geq 0$.

Proof: Since G is not LR(k) there exist derivations

$S' \longrightarrow^* \alpha A \gamma \longrightarrow \alpha \omega \gamma$ and

$S' \longrightarrow^* \alpha' A' \gamma' \longrightarrow \alpha' \omega' \gamma'$ where

$\alpha' \omega' \gamma' = \alpha \omega \lfloor k : \gamma \rfloor \delta, \quad \delta \in V_T'^*$ and

(γ' suf $\lfloor k : \gamma \rfloor \delta$ or ($\gamma' = \lfloor k : \gamma \rfloor \delta$ and $A \longrightarrow \omega \neq A' \longrightarrow \omega'$)).

Now since $M \in LR(k)PM(G)$ $\exists \sigma' = T_m \ldots T_0 \mid T_m = S_0$, σ' reads $\alpha \omega$ and $\exists \Sigma, \Sigma', \Pi = \tau_0 \ldots \tau_r$ and $\Pi_1 = \tau_0' \ldots \tau_s'$. τ_0 is the action z_p, reduce $A \longrightarrow \omega$. τ_0' is the

action z_q , reduce $A' \longrightarrow \omega'$, if $\gamma' = \lfloor k : \gamma \rfloor \delta$ or read if γ' suf $\lfloor k : \gamma \rfloor \delta$. (Σ, Π, σ') and (Σ', Π', σ') both read $k : \gamma$. Certainly $Z_p^i \cap Z_q^i \neq \emptyset$ is implied by the above statement.

Now assume $X_k(\sigma', m, z_p)$ and $X_k(\sigma', m, z_q)$ are complete. Then certainly $k : \gamma$ is in their intersection. But then $k : \gamma \in X_k(\rho, m', z_p) \cap X_k(\rho, m', z_q)$ where $\rho = U_{m'} \ldots U_0$ is σ' with all k-loops removed. Now since $U_{m'} = S_0$ and $U_0 = T_0$, $X_k(\rho, m', z)$ is terminal $\forall z$. But then $\exists$ a minimum integer $n \mid X_p(\rho, n, z)$ is terminal $\forall_z$. Let $\sigma = U_n \ldots U_0$. It follows that $k : \gamma \in X_k(\sigma, n, z_p) \cap X_k(\sigma, n, z_q)$ and σ contains no k-loops.

Now assume $X_k(\sigma', m, z)$ is not complete . Certainly $\exists$ a smallest integer m' such that $X_k(\sigma', m', z)$ is not complete. Let $\rho = T_{m'} \ldots T_0$. $X_k(\rho, m', z)$ is not complete and neither is $X_k(\sigma, n, z)$ where $\sigma = U_n \ldots U_0$ with $U_0 = T_0$ is ρ with all k-loops not involving $T_{m'}$ removed. Now σ contains at most one k-loop and it must be of the form $U_n \ldots U_{n'}$ with $n > n' \geq 0$. Furthermore $X_k(\sigma, n-1, z)$ is not terminal for if it were it would not be complete but that is impossible.

Definition 15

For each state T_0 define a graph $GR(T_0)$ in the following way:

$\sigma = T_n \ldots T_0$ is a node of $GR(T_0)$ if and only if σ contains

no k-loops and ($X_k(\sigma, n-1, z)$ is not terminal for some z or $n = 0$). Let σ and ρ be nodes of $GR(T_0)$. There is a <u>transition</u> from ρ to σ ($\rho \longrightarrow \sigma$) where $\sigma = T_n \ldots T_0$ if and only if $\rho = T_{n+1} \ldots T_0$ or ($\rho = T_m \ldots T_0$ with $n \geq m \geq 0$ and $T_{n+1} \ldots T_m$ is a k-loop in $\sigma' = T_{n+1} \ldots T_0$ where $T_{n+1} = T_m$). If σ is a node and $X_k(\sigma, n, z)$ is terminal $\forall z$ then σ is a <u>terminal node</u>. All other nodes are <u>non-terminal nodes</u>. The node $\sigma = T_0$ is called the <u>foot</u> of $GR(T_0)$. We say that state T_n occupies the node σ of $GR(T_0)$.

Theorem 12

Suppose $M \in LR(k)PM(G)$ and G is $LR(k)$

$T_0 = (i, C_i^*, Z_0^i, \ldots, Z_r^i) \in K$

$\sigma = T_n \ldots T_0$.

Let $T_0' = (i, C_i^*, \bigcup_\sigma X_k(\sigma, n, z_0), \ldots, \bigcup_\sigma X_k(\sigma, n, z_r))$

where the union is taken over all σ such that σ is a terminal node of $GR(T_0)$.

The machine M' where M' is M with state T_0 replaced by state T_0' is in $LR(k)PM(G)$.

Proof: Since G is $LR(k)$ $X_k(\sigma, n, z)$ is complete $\forall \sigma$, n and z

Now let (Σ, Π, ρ) be a complete conditional path such that $\rho = U_m \ldots U_0$ with $U_m = S_0$ and $U_0 = T_0$, $\Pi = \tau_0 \ldots \tau_r$ (Σ, Π, ρ) reads α and $k : \alpha \in W_k$. Certainly $\exists\ \Sigma', \Pi' \mid (\Sigma, \Pi, \rho)$ is an extension of (Σ', Π', ρ) and (Σ', Π', ρ) reads $k : \alpha$.

Let $\rho' = U_{m'} \ldots U_0$ with $m' \leq m$ be such that $X_k(\rho', m', z)$ is terminal $\forall z$ but $\exists z \mid X_k(\rho', m'-1, z)$ is not terminal. Certainly $k : \alpha \in X_k(\rho', m', \tau_0)$. Now let $\sigma = T_n \ldots T_0$ be ρ' with all k-loops removed. σ is a terminal node of $GR(T_0)$ and $k : \alpha \in X_k(\rho', m', \tau_0)$.

Therefore M' will act on any input string exactly as M will and thus $M' \in LR(k)PM(G)$.

Theorem 13

Let $M = \{ i \in I \mid 0 \leq i \leq m \}$

$J = \{ i \in I \mid 0 \leq i \leq r \}$

$\{ B_{(i, j)} \mid i \in M, j \in J \}$ be any collection of sets $\mid$ if $j \neq j'$ with j and $j' \in J$ then

$$B_{(i, j)} \cap B_{(i, j')} = \emptyset \quad \forall i .$$

Then $\exists$ at least one partition of M into set N_n where $1 \leq n \leq s \mid$

If $A_{(n, j)} = \bigcup_{i \in N_n} B_{(i, j)}$ then

if $j \neq j'$ with j and $j' \in J$ then $A_{(i, j)} \cap A_{(i, j')} = \emptyset$ and $\forall n, n' \mid n \neq n'$ and $1 \leq n, n' \leq s$ $\exists j, j'$ with $j \neq j'$ and j and $j' \in J \mid A_{(n, j)} \cap A_{(n', j')} \neq \emptyset$.

Proof: Certainly there exists some maximal subset N_1 of

$$M \mid \bigcup_{i \in N_1} B_{(i, j)} \cap \bigcup_{i \in N_1} B_{(i, j')} = \emptyset \quad \forall j, j' \in J \mid j \neq j'$$

If $M = N_1$ we are done. Otherwise find a maximal subset N_2 of $M \setminus N_1 \mid$

$$\bigcup_{i \in N_2} B_{(i, j)} \cap \bigcup_{i \in N_2} B_{(i, j')} = \emptyset \quad \forall j, j' \in J \mid j \neq j'.$$

If $M = N_1 \cup N_2$ we are done else continue until $M = \bigcup_{i=1}^{s} N_i$.
Certainly $s \leq m + 1$ and the sets $A_{(n, j)} = \bigcup_{i \in N_n} B_{(i, j)}$
satisfy the necessary requirements.

Theorem 14

Let $M \in LR(k)PM(G)$ and

Γ be a non-empty subset of K

Then $\exists$ at least one state $S \mid$

S occupies a terminal node of GR(I) for some $I \in \Gamma$

and S occupies no non-terminal node of GR(I) $\forall I \in \Gamma$.

Proof: The state S_0 can only occupy terminal nodes since

$\nexists\ S \in K \mid S \xrightarrow{\alpha} S_0$.

Let $I \in \Gamma$. Consider $\sigma = T_n \ldots T_0$ where $T_n = S_0$ and $T_0 = I$.
T_0 occupies the foot of GR(I) . Now suppose $\exists I \in \Gamma \mid T_i \in GR(I)$.
Then either T_i occupies no non-terminal nodes of GR(I) $\forall I \in \Gamma$
or else T_{i+1} occupies some node of GR(I) for some $I \in \Gamma$.
Therefore there exists at least one state S satisfying the
hypothesis.

8. AN ALGORITHM

In this section we describe an algorithm for answering the question is grammar G LR(k) and if the answer is yes building a deterministic LR(k) parser for G .

Step 1

Build the table of states and transitions for the parsing machine $M_0'(G)$. Convert to an element of LR(k)PM(G) by letting the sets Z_j^i be maximal. (Definition 5b)

Step 2

For each state $I = (i, C_i^*, Z_0^i, \ldots Z_r^i)$ such that

$\exists j$ and j' with $j \neq j' \mid Z_j^i \cap Z_{j'}^i \neq \emptyset$

(This is precisely the set of inadequate states)

find $X_k(\sigma, n, z)$ $\forall \sigma$, n and z $\mid \sigma = T_n \ldots T_0$, $T_0 = I$, σ contains at most one k-loop and it must be of the form $T_n \ldots T_m$ where $n > m \geq 0$ and $X_k(\sigma, n-1, z)$ is not terminal for some z. (Definition 11)

This is a bounded process by Theorems 3 and 10. If condition 3 or 4 of (a) of Definition 11 is encountered or $\exists$ z and z' $\mid X_k(\sigma, n, z) \cap X_k(\sigma, n, z') \neq \emptyset$ for some σ then G is not LR(k) by Theorems 5, 6, 8 and 9. Otherwise G is LR(k) by Theorems 7 and 11.

Step 3

If G is not LR(k) then stop, else build GR(I) for each state I in step 2. With each node σ on some graph GR(I) associate a unique positive integer i and for each terminal node σ, let $B_{(i, j)} = X_k(\sigma, n; z_j)$ where σ is the terminal node i. (Definition 15) Henceforth call node σ node i .

Step 4

Find sets $A_{(I, n, j)}$ $\forall I$ from step 2. $\forall n \mid 1 \leq n \leq s_I$ where s_I is a positive integer let

$A_{(I, n, j)} = \bigcup_{i \in N_{(I, n)}} B_{(i, j)}$ where $N_{(I, n)}$ is a partition of the set of all integers i associated with a terminal node in GR(I) $\mid$

$A_{(I, n, j)} \cap A_{(I, n, j')} = \emptyset$ if $j \neq j'$ and

$\forall n, n'$ and I, $\exists j$ and $j' \mid A_{(I, n, j)} \cap A_{(I, n', j')} \neq \emptyset$.

(There exist at least one such partition and an algorithm for finding it by Theorem 13)

Associate the p + 1 tuple of sets $(A_{(I, n, 0)}, \ldots, A_{(I, n, r)})$ with each node $i \in N_{(I, n)}$.

Step 5

$\forall I \mid N_{(I, 1)} = \{ i \mid i$ is a terminal node of GR(I)$\}$

if $I = \{ p, C^*_p, Z^p_0, \ldots, Z^p_r \}$ then

let $I = (p, C^*_p, A_{(I, n, 0)}, \ldots, A_{(I, n, r)})$.

(This new machine is an element of $LR(k)PM(G)$ by Theorem 12 and will parse G.)

$j \neq j'$ implies $A_{(I, n, j)} \cap A_{(I, n, j')} = \emptyset$.

Step 6

If $\forall I \; N_{(I, 1)} = \left| i \;\middle|\; i \text{ is a terminal node of } GR(I) \right|$, then stop. Else

let $\Gamma = \left| I \;\middle|\; N_{(I, 1)} \neq \left| i \;\middle|\; i \text{ is a terminal node of } GR(I) \right| \right|$

Δ = the set of all states $S \in K$ which occupy some terminal node of some $GR(I) \mid I \in \Gamma$ but occupy no non-terminal node of any $GR(I) \mid I \in \Gamma$

$J = \left| i \;\middle|\; i \text{ is a node of } GR(I) \text{ for some } I \in \Gamma \right|$.

(Theorem 14 says that Δ is not empty)

$\forall S \in \Delta$ create a state (S, H) where H is an m-tuple of integers where $m = ord(J)$. If S does not occupy node i then let the i^{th} element of H be 0. If S occupies node i where i is a node of $GR(I)$ then let the i^{th} element of H be n where $i \in N_{(I, n)}$. Let Θ be the set of all states (S, H) created in this step.

Step 7

Let $\Lambda = \left| S \in K \;\middle|\; S \text{ occupies some node } i \text{ of } GR(I) \text{ for some } I \in \Gamma \right|$. $\forall (S, H) \in \Theta$ if $S \xrightarrow{\#p}$ then $(S, H) \xrightarrow{\#p}$.

If $T \in K \setminus \Lambda$ then $\forall (S, H) \in \Theta$ If $S \xrightarrow{\alpha} T$ then $(S,H) \xrightarrow{\alpha} T$.

If $T \in \Lambda$ then $\forall (S, H) \in \Theta$

if $S \xrightarrow{\alpha} T$ then $(S, H) \xrightarrow{\alpha} (T, H')$

where H' is an m-tuple of integers such that if T does not occupy node i then the i^{th} element of H' is 0. If T occupies node i and i is a terminal node of GR(I) then the i^{th} element of H' is n such that $i \in N_{(I, n)}$. If T occupies node i and i is a non-terminal node of GR(I) then there exists some node j of GR(I) such that $j \longrightarrow i$ in GR(I) and S occupies node j. The i^{th} element of H' equals the j^{th} element of H. If $(T, H') \notin \Theta$ then create state (T, H') and let $\Theta = \Theta \cup \{(T, H')\}$.

Step 8

If some state (T, H') was created during the last application of step 7 then repeat step 7, else go on to step 9.

Step 9

For all states (S, H) from step 6 if $\exists T \in K \mid T \xrightarrow{\alpha} S$ then let $T \xrightarrow{\alpha} (S, H)$.

Step 10

If i is the node at the foot of GR(I) $\mid I \in \Gamma$ and n is the i^{th} element of H then if $I = (p, C^*_p, Z^p_0, \ldots, Z^p_r)$ then let

$$(I, H) = (q, C^*, A_{(I, n, 0)}, \ldots, A_{(I, n, r)}).$$

If $(S, H) \in \Theta$ but $S \notin \Gamma$ then

if $S = (p, C^*_p, Z^p_0, \ldots, Z^p_r)$ let

$$(S, H) = (q, C^*_p, Z^p_0, \ldots, Z^p_r)$$

where q is an integer not yet assigned to a state.

Step 11

Delete all states $S \in \Lambda$ and all transitions involving these states. Let $K = (K \setminus \Lambda) \cup \theta$. Let δ be the relation uniquely defined by the set of all transitions and the sets Z_j^i .

If $S_0 \in \Lambda$ let $S_0 = (S_0, H)$.

Let $F = \left\{(S, H) \mid S \in F)\right\} \cup (F \setminus \Lambda)$.

$M = (K, V_T', S_0, P, N, F, \delta, k)$ is now a deterministic element of $LR(k)PM(G)$.

Step 12

Stop.

This algorithm given any grammar G which is LR(k) will find a deterministic element of $LR(k)PM(G)$. If G is LR(0) the result will be equivalent to $M_0'(G)$. If G is not LR(0) , the algorithm will find if possible a deterministic parser whose states are in 1-1 correspondence with the states of $M_0'(G)$. If G is not LR(k) , the algorithm will discover this fact.

It should be noted that it is not necessary to apply steps 2 through 11 to all inadequate states simultaneously. We could apply steps 2 through 11 to a subset of the set of all inadequate states. The result will be a new member M' of $LR(k)PM(G)$. Then we may apply steps 2 through 11 to M' . We should however not take the inadequate states one by one as we have no assurance that we will not create inadequate states faster than we resolve them. A possible algorithm is to deal simultaneously with all states

$S_i = (i, C_i^*, Z_0^i, \ldots, Z_r^i) \mid C_i^* = C_j^*$ for a given j . The advantage of this method is that if Γ is large we don't have to deal with it all at once. The disadvantage is that it is longer and we may be doing some unnecessary work.

The resulting parser may if desired by optimized by any of the methods mentioned by DeRemer (DeR 69). However, in this paper, we will not be concerned with optimizations.

It is interesting to note that if we apply this algorithm to a grammar G first with k = 0 and then keep increasing k until we either find that G is LR(k) for a given k or that G is not LR(k) for any k . Then one of four situations must occur:

1. Eventually we find that G is LR(k) for some k .
2. Eventually Theorem 5 or 9 tells us that G is ambiguous and therefore not LR(k) for any k .
3. Eventually Theorem 6 tells us that G is not LR(k) for any k . In this case G may still be ambiguous.
4. For every k Theorem 8 tells us that G is not LR(k) for that particular k . In this case the procedure will not stop.

However if G contains no productions of the form $A \longrightarrow \epsilon$, then the situations described in Theorems 5 and 6 cannot occur and the procedure will halt if and only if G is ambiguous or else G is LR(k) for some k .

9. SOME MORE EXAMPLES

Let's look at grammar G_4 defined by the productions

1. $S \longrightarrow S\ S\ T$
2. $S \longrightarrow T$
3. $T \longrightarrow a$

This is just grammar G_0 with the useless production removed. Since we already know that this grammar is ambiguous, we would expect to find that G_4 is not LR(k) no matter which value of k we choose. Let k = 4. Step 1 of the algorithm yields the states and transitions in Table 13.

Table 13 States and Transitions For $M_0'(G_4)$

Number	State		Transition To
0	$S' \longrightarrow \ast \vdash S \dashv$		$\vdash$ 1
1	$S' \longrightarrow \vdash \ast S \dashv$	$S \longrightarrow \ast S\ S\ T$	S 2 a 5
	$T \longrightarrow \ast a$	$S \longrightarrow \ast T$	T 3
2	$S' \longrightarrow \vdash S \ast \dashv$	$S \longrightarrow \ast S\ S\ T$	S 4 a 5
	$S \longrightarrow S \ast S\ T$	$S \longrightarrow \ast T$ $T \longrightarrow \ast a$	T 3 $\dashv$ accept
3	$S \longrightarrow T \ast$		reduce 2
4	$S \longrightarrow S\ S \ast T$	$S \longrightarrow \ast S\ S\ T$	S 4 a 5
	$S \longrightarrow S \ast S\ T$	$S \longrightarrow \ast T$ $T \longrightarrow \ast a$	T 6
5	$T \longrightarrow a \ast$		reduce 3
6	$S \longrightarrow S\ S\ T \ast$		reduce 1
	$S \longrightarrow T \ast$		reduce 2

$M_0(G_4)$ has only one inadequate state, state 6.

$$X_4(6, 0, z_1) = \{(3, S)\} \quad \text{and}$$
$$X_4(6, 0, z_2) = \{(1, S)\}.$$

The only state with a transition to state 6 is state 4.

$$X_4([4, 6], 1, z_1) = \{(2, S)\} \quad \text{and}$$
$$X_4([4, 6], 1, z_2) = \{a\ a\ a\ a,\ a\ a\ a\ (1, S),\ a\ (1, S)\}.$$

Every conditional path $(\Sigma, \Pi, [4, 4])$ starts by reading a a a a except $\Sigma = 4, 5, 6$ and $\Pi = z_0 z_3 z_1$ which yields a (1, S), and $\Sigma = 4, 5, 6, 4, 5, 6, 4, 5, 6$ $\Pi = z_0 z_3 z_2 z_0 z_3 z_1 z_0 z_3 z_1$ which yields a a a (1, S).

$$X_4([4, 4, 6], 2, z_1) = \{(1, S)\} \quad \text{and}$$
$$X_4([4, 4, 6], 2, z_2) = \{a\ a\ a\ a,\ a\ a\ (1, S)\}$$

by the same reasoning as above.

And finally $X_4([4, 4, 4, 6], 3, z_1) = \{a\ a\ a\ a,\ a\ a\ a\ (1, S),\ a\ (1, S)\}$

and $X_4([4, 4, 4, 6], 3, z_2) = \{a\ a\ a\ a,\ a\ a\ a\ (1, S)\}.$

Thus G is not LR(4) since

$$X_4([4, 4, 4, 6], 3, z_1) \cap X_4([4, 4, 4, 6], 3, z_2) \neq \emptyset.$$

However Theorem 9 tells us more than this, namely that G_4 is ambiguous.

Now let's look at grammar G_5 defined by the productions

listed below:

1. $S \longrightarrow a\ A\ d$
2. $S \longrightarrow b\ A\ c$
3. $S \longrightarrow a\ e\ c$
4. $S \longrightarrow b\ e\ d$
5. $A \longrightarrow e\ A$
6. $A \longrightarrow e$

Step 1 of the algorithm yields the states and transition listed in Table 14.

Table 14 States and Transitions For $M_0'(G_5)$

Number	State			Transition To
0	$S' \longrightarrow * \vdash S \dashv$			$\vdash$ 1
1	$S' \longrightarrow \vdash * S \dashv$	$S \longrightarrow * a A d$	$S \longrightarrow * b A c$	S 2 a 3
		$S \longrightarrow * a e c$	$S \longrightarrow * b e d$	b 10
2	$S' \longrightarrow \vdash S * \dashv$			$\dashv$ accept
3	$S \longrightarrow a * A d$	$A \longrightarrow * e A$		A 4
	$S \longrightarrow a * e c$	$A \longrightarrow * e$		e 6
4	$S \longrightarrow a A * d$			d 5
5	$S \longrightarrow a A d *$			reduce 1
6	$S \longrightarrow a e * c$	$A \longrightarrow e * A$	$A \longrightarrow * e A$	A 7 c 8
		$A \longrightarrow e *$	$A \longrightarrow * e$	e 9 reduce 6
7	$A \longrightarrow e A *$			reduce 5
8	$S \longrightarrow a e c *$			reduce 3
9	$A \longrightarrow e * A$	$A \longrightarrow * e A$		A 7 e 9
	$A \longrightarrow e *$	$A \longrightarrow * e$		reduce 6
10	$S \longrightarrow b * A c$	$A \longrightarrow * e A$		A 11
	$S \longrightarrow b * e d$	$A \longrightarrow * e$		e 13
11	$S \longrightarrow b A * c$			c 12
12	$S \longrightarrow b A c *$			reduce 2
13	$S \longrightarrow b e * d$	$A \longrightarrow e * A$	$A \longrightarrow * e A$	A 7 d 14
		$A \longrightarrow e *$	$A \longrightarrow * e$	e 9 reduce 6
14	$S \longrightarrow b e d *$			reduce 4

$M_0(G_5)$ has three inadequate states, states 6, 9 and 13.

Letting k = 1 we get $X_1(6, 0, z_0) = \{c, e\}$ and

$$X_1(6, 0, z_6) = \{(1, A)\}.$$

Only state 3 has a transition to state 6 and

$$X_1(\lfloor 3, 6 \rfloor, 1, z_0) = \{c, e\} \quad \text{and}$$

$$X_1(\lfloor 3, 6 \rfloor, 1, z_6) = \{d\}.$$

For state 13 we get $X_1(\lfloor 10, 13 \rfloor, 1, z_0) = \{d, e\}$ and

$$X_1(\lfloor 10, 13 \rfloor, 1, z_6) = \{c\}.$$

For state 9 $X_1(9, 0, z_0) = \{e\}$

$$X_1(9, 0, z_6) = \{(1, A)\}.$$

States 6, 13 and 9 all have transitions to state 9.

$$X_1(\lfloor 9, 9 \rfloor, 1, z_6) = \{(1, A)\}.$$

Thus we have found a k-loop.

$$X_1(\lfloor 6, 9 \rfloor, 1, z_6) = \{(1, A)\} \quad \text{and}$$

$$X_1(\lfloor 3, 6, 9 \rfloor, 2, z_6) = \{d\} \quad \text{while}$$

$$X_1(\lfloor 13, 9 \rfloor, 1, z_6) = \{(1, A)\} \quad \text{and}$$

$$X_1(\lfloor 10, 13, 9 \rfloor, 2, z_6) = \{c\}.$$

Therefore G is LR(k) and changing states 6, 9 and 13 to

(6, $S \longrightarrow a\ e * c$ $A \longrightarrow e * A$ $A \longrightarrow * e\ A$, $\{c,e\}$, $\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,$ $\{d\}$)

$A \longrightarrow e *$ $A \longrightarrow * e$

(9, $A \longrightarrow e * A$ $A \longrightarrow * e\ A$, $\{e\}$, $\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,$ $\{c,d\}$)

$A \longrightarrow e *$ $A \longrightarrow * e$

(13, $S \longrightarrow b\ e * d$ $A \longrightarrow e * A$ $A \longrightarrow * e\ A$, $\{d,e\}$, $\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,$ $\{c\}$)

$A \longrightarrow e *$ $A \longrightarrow * e$

we now have a deterministic parser for grammar G_5 .

It is of interest to note that since

$$\left\lfloor A \right\rfloor = \left\lfloor A \right\rfloor$$

using DeRemer's algorithm we would get for

state 6 $\quad Z_0 = \left\lfloor e, c \right\rfloor \quad$ and $\quad Z_6 = \left\lfloor d, c \right\rfloor \quad$ and for

state 13 $\quad Z_0 = \left\lfloor e, d \right\rfloor \quad$ and $\quad Z_6 = \left\lfloor d, c \right\rfloor$.

Thus grammar G_5 is not SLR(1) . In fact it is not SLR(k) for any k . Yet it is LR(1) and we have found a parser for it without resorting to the state splitting techniques of steps 6 through 9 of the algorithm.[10]

For our final example take grammar G_6 defined by the productions listed below:

1. S ⟶ u A a
2. S ⟶ u B c
3. S ⟶ v A b
4. S ⟶ v B d
5. S ⟶ w C a
6. S ⟶ w D d
7. S ⟶ x C a
8. S ⟶ x D b
9. S ⟶ y E a
10. S ⟶ y F b
11. S ⟶ z E d
12. S ⟶ z F c
13. A ⟶ r G
14. B ⟶ r H
15. C ⟶ s G
16. D ⟶ s H
17. E ⟶ t G
18. F ⟶ t H
19. G ⟶ e
20. H ⟶ e

Applying the algorithm with k = 1 , step 1 yields the states and transitions listed in Table 15.

Table 15 States and Transitions For $M_0'(G_6)$

Number	State			Transition	To
0	S' ⟶ * ⊢ S ⊣			⊢ 1	
1	S' ⟶ ⊢ * S ⊣			S 3	
	S ⟶ * u A a	S ⟶ * w C a	S ⟶ * y E a	u 2	x 25
	S ⟶ * u B c	S ⟶ * w D d	S ⟶ * y F b	v 12	y 30
	S ⟶ * v A b	S ⟶ * x C a	S ⟶ * z E d	w 17	z 38
	S ⟶ * v B d	S ⟶ * x D b	S ⟶ * z F c		
2	S ⟶ u * A a	A ⟶ * r G		A 4	r 8
	S ⟶ u * B c	B ⟶ * r H		B 6	
3	S' ⟶ ⊢ S * ⊣			⊣ accept	
4	S ⟶ u A * a			a 5	
5	S ⟶ u A a *			reduce 1	
6	S ⟶ u B * c			c 7	
7	S ⟶ u B c *			reduce 2	
8	A ⟶ r * G	G ⟶ * e		G 9	e 11
	B ⟶ r * H	H ⟶ * e		H 10	
9	A ⟶ r G *			reduce 13	
10	B ⟶ r H *			reduce 14	
11	G ⟶ e *			reduce 19	
	H ⟶ e *			reduce 20	
12	S ⟶ v * A b	A ⟶ * r G		A 13	r 8
	S ⟶ v * B d	B ⟶ * r H		B 15	
13	S ⟶ v A * b			b 14	
14	S ⟶ v A b *			reduce 3	

Table 15 Continued

Number	State		Transition	To
15	S ⟶ v B * d		d 16	
16	S ⟶ v B d *		reduce 4	
17	S ⟶ w * C a	C ⟶ * s G	C 18	S 22
	S ⟶ w * D d	D ⟶ * s H	D 20	
18	S ⟶ w C * a		a 19	
19	S ⟶ w C a *		reduce 5	
20	S ⟶ w D * d		d 21	
21	S ⟶ w D d *		reduce 6	
22	C ⟶ s * G	G ⟶ * e	G 23	e 11
	D ⟶ s * H	H ⟶ * e	H 24	
23	C ⟶ s G *		reduce 15	
24	D ⟶ s H *		reduce 16	
25	S ⟶ x * C a	C ⟶ * s G	C 26	s 22
	S ⟶ x * D b	D ⟶ * s H	D 28	
26	S ⟶ x C * a		a 27	
27	S ⟶ x C a *		reduce 7	
28	S ⟶ x D * b		b 29	
29	S ⟶ x D b *		reduce 8	
30	S ⟶ y * E a	E ⟶ * t G	E 31	t 35
	S ⟶ y * F b	F ⟶ * t H	F 33	
31	S ⟶ y E * a		a 32	
32	S ⟶ y E a *		reduce 9	
33	S ⟶ y F * b		b 34	

Table 15 Continued

Number	State		Transition To
34	$S \longrightarrow y\ F\ B\ *$		reduce 10
35	$E \longrightarrow t\ *\ G$	$G \longrightarrow *\ e$	G 36 e 11
	$F \longrightarrow t\ *\ H$	$H \longrightarrow *\ e$	H 37
36	$E \longrightarrow t\ G\ *$		reduce 17
37	$F \longrightarrow t\ H\ *$		reduce 18
38	$S \longrightarrow z\ *\ E\ d$	$E \longrightarrow *\ t\ G$	E 39 t 35
	$S \longrightarrow z\ *\ F\ c$	$F \longrightarrow *\ t\ H$	F 41
39	$S \longrightarrow z\ E\ *\ d$		d 40
40	$S \longrightarrow z\ E\ d\ *$		reduce 11
41	$S \longrightarrow z\ F\ *\ c$		c 42
42	$S \longrightarrow z\ F\ c\ *$		reduce 12

$M_0'(G_6)$ has one inadequate state, state 11.

$$X_1(11,\ 0,\ z_{19}) = \left|(1,\ G)\right| \quad \text{and} \quad X_1(11,\ 0,\ z_{20}) = \left|(1,\ H)\right|.$$

States 8, 22 and 35 have transitions to state 11.

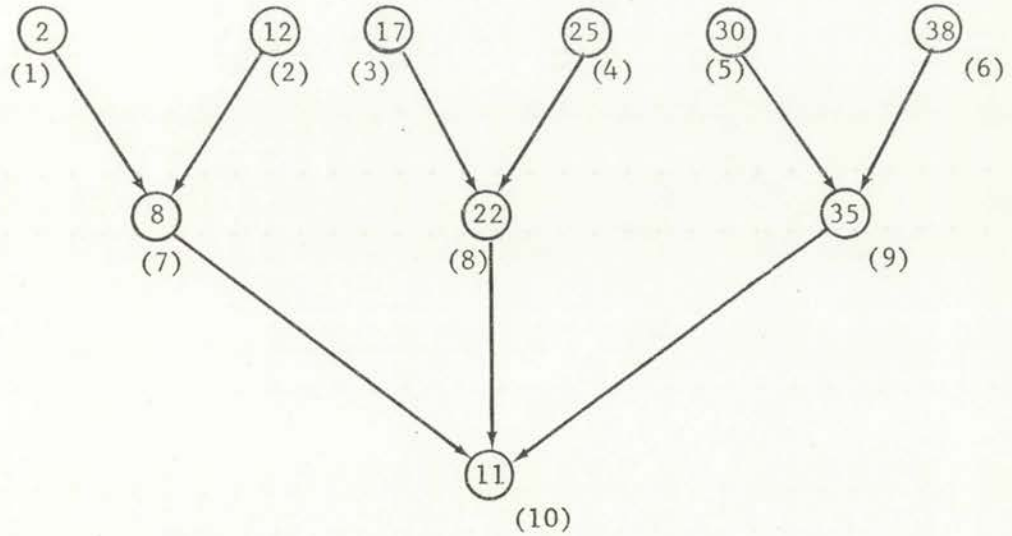
$$X_1(\left[8,\ 11\right],\ 1,\ z_{19}) = \left|(1,\ A)\right| \ ,\ X_1(\left[8,\ 11\right],\ 1,\ z_{20}) = \left|(1,B)\right|,$$
$$X_1(\left[22,\ 11\right],\ 1,\ z_{19}) = \left|\ (1,\ C)\right| \ ,\ X_1(\left[22,\ 11\right],\ 1,\ z_{20}) = \left|\ (1,D)\right|,$$
$$X_1(\left[35,\ 11\right],\ 1,\ z_{19}) = \left|\ (1,\ E)\right| \text{and } X_1(\left[35,\ 11\right],\ 1,\ z_{20}) = \left|\ (1,F)\right|.$$

Looking further back,

$$X_1(\lfloor 2, 8, 11 \rfloor, 2, z_{19}) = \lfloor a \rfloor \quad , \quad X_1(\lfloor 2, 8, 11 \rfloor, 2, z_{20}) = \lfloor c \rfloor ,$$
$$X_1(\lfloor 12, 8, 11 \rfloor, 2, z_{19}) = \lfloor b \rfloor \quad , \quad X_1(\lfloor 12, 8, 11 \rfloor, 2, z_{20}) = \lfloor d \rfloor ,$$
$$X_1(\lfloor 17, 22, 11 \rfloor, 2, z_{19}) = \lfloor a \rfloor \quad , \quad X_1(\lfloor 17, 23, 11 \rfloor, 2, z_{20}) = \lfloor d \rfloor ,$$
$$X_1(\lfloor 25, 22, 11 \rfloor, 2, z_{19}) = \lfloor a \rfloor \quad , \quad X_1(\lfloor 25, 22, 11 \rfloor, 2, z_{20}) = \lfloor b \rfloor ,$$
$$X_1(\lfloor 30, 35, 11 \rfloor, 2, z_{19}) = \lfloor a \rfloor \quad , \quad X_1(\lfloor 30, 35, 11 \rfloor, 2, z_{20}) = \lfloor b \rfloor ,$$
$$X_1(\lfloor 38, 35, 11 \rfloor, 2, z_{19}) = \lfloor d \rfloor \text{ and } X_1(\lfloor 38, 35, 11 \rfloor, 2, z_{20}) = \lfloor c \rfloor .$$

Thus grammar G_6 is LR(1) . Performing step 3 we get the graph GR(11) in Figure 5. The number of the node is below the circle. The state occupying the node is within the circle. The terminal nodes are nodes 1 through 6. Node 10 is the foot of GR(11) .

Figure 5 The Graph GR(11)



Step 4 may be done in several ways. We shall do it in two different ways and get two completely different parsers. The first way:

Let $N_{(11,\ 1)} = \{1, 2\}$, $N_{(11,\ 2)} = \{3, 4\}$ and $N_{(11,\ 3)} = \{5, 6\}$.

Then $A_{(11,\ 1,\ 19)} = \{a, b\}$, $A_{(11,\ 1,\ 20)} = \{c, d\}$,

$A_{(11,\ 2,\ 19)} = \{a\}$, $A_{(11,\ 2,\ 20)} = \{b, d\}$,

$A_{(11,\ 3,\ 19)} = \{a, d\}$ and $A_{(11,\ 3,\ 20)} = \{b, c\}$.

Continuing in step 6 we create states 43 through 48 in Table 16. In steps 7 and 8 we create states 49 through 54 and fill in the transitions in Table 16. In step 9 we change the transitions from state 1. In step 10 we change Z_{19} and Z_{20} for states 52, 53 and 54.

Table 16 New States For First Parser For Grammar G_6

Number	State	10-Tuple	Z	Transition	To
1	Same As In Table 15	——	–	s 3	
				u 43	v 44 w 45
				x 46	y 47 z 48
43(2)	$S \rightarrow u * A\ a$ $A \rightarrow * r\ G$	(1,0,0,0,0,0,0,0,0,0)	–	A 4	r 49
	$S \rightarrow u * B\ c$ $B \rightarrow * r\ H$			B 6	
44(12)	$S \rightarrow v * A\ b$ $A \rightarrow * r\ G$	(0,1,0,0,0,0,0,0,0,0)	–	A 13	r 49
	$S \rightarrow v * B\ d$ $B \rightarrow * r\ H$			B 15	
45(17)	$S \rightarrow w * C\ a$ $C \rightarrow * s\ G$	(0,0,2,0,0,0,0,0,0,0)	–	C 18	s 50
	$S \rightarrow w * D\ d$ $D \rightarrow * s\ H$			D 20	
46(25)	$S \rightarrow x * C\ a$ $C \rightarrow * s\ G$	(0,0,0,2,0,0,0,0,0,0)	–	C 26	s 50
	$S \rightarrow x * D\ b$ $D \rightarrow * s\ H$			D 28	
47(30)	$S \rightarrow y * E\ a$ $E \rightarrow * t\ G$	(0,0,0,0,3,0,0,0,0,0)	–	E 31	t 51
	$S \rightarrow y * F\ b$ $F \rightarrow * t\ H$			F 33	
48(38)	$S \rightarrow z * E\ d$ $E \rightarrow * t\ G$	(0,0,0,0,0,3,0,0,0,0)	–	E 39	t 51
	$S \rightarrow z * F\ c$ $F \rightarrow * t\ H$			F 41	
49(8)	$A \rightarrow r * G$ $B \rightarrow r * H$	(0,0,0,0,0,0,1,0,0,0)	–		
	$G \rightarrow e *$ $H \rightarrow e *$			G 9	H 10 e 52

Table 16 Continued

Number	State		10-Tuple	Z	Transition	To
50(22)	C ⟶ s * G	D ⟶ s * H	(0,0,0,0,0,0,0,2,0,0)	—		
	G ⟶ e *	H ⟶ e *			G 23 H 24	e 53
51(35)	E ⟶ t * G	G ⟶ * e	(0,0,0,0,0,0,0,0,3,0)	—	G 36	e 54
	F ⟶ t * H	H ⟶ * e			H 37	
52(11)	G ⟶ e *		(0,0,0,0,0,0,0,0,0,1)	$Z_{19} = \{a,b\}$	reduce 19	
	H ⟶ e *			$Z_{20} = \{c,d\}$	reduce 20	
53(11)	G ⟶ e *		(0,0,0,0,0,0,0,0,0,2)	$Z_{19} = \{a\}$	reduce 19	
	H ⟶ e *			$Z_{20} = \{b,d\}$	reduce 20	
54(11)	G ⟶ e *		(0,0,0,0,0,0,0,0,0,3)	$Z_{19} = \{a,d\}$	reduce 19	
	H ⟶ e *			$Z_{20} = \{b,c\}$	reduce 20	

Now deleting states 2, 8, 11, 12, 17, 22, 25, 30, 35 and 38 from Table 15 and changing the transitions from state 1, we get a parser for grammar G_6 which is represented by Tables 15 and 16 combined.

Going back to step 4, suppose we had let

$N_{(11,\ 1)} = \{1, 2, 3\}$ and $N_{(11,\ 2)} = \{4, 5, 6\}$. Then

$A_{(11,\ 1,\ 19)} = \{a, b\}$, $A_{(11,\ 1,\ 20)} = \{c, d\}$,

$A_{(11,\ 2,\ 19)} = \{a, d\}$ and $A_{(11,\ 1,\ 20)} = \{b, c\}$.

In step 6 we create states 43 through 48 in Table 17. In steps 7 and 8 we create states 49 through 54 and fill in the transitions in Table 17. The final parser is represented by Table 15 with states 1, 2, 8, 11, 12, 17, 22, 25, 30, 35 and 38 deleted combined with Table 17. This is a different parser for grammar G_6 than the first parser for G_6. The difference is shown in the graphs in Figures 6 and 7. It would be very difficult to say that either one of these parsers is better. They have the same number of states yet they are certainly not equivalent graphicly.

Table 17 New States For Second Parser For Grammar G_6

Number	State		10-Tuple	Z	Transition		To
1	Same As In Table 15		——	—	S 3		
					u 43	v 44	w 45
					x 46	y 47	z 48
43(2)	S ⟶ u * A a	A ⟶ * r G	(1,0,0,0,0,0,0,0,0,0)	—	A 4	r 49	
	S ⟶ u * B c	B ⟶ * r H			B 6		
44(12)	S ⟶ v * A b	A ⟶ * r G	(0,1,0,0,0,0,0,0,0,0)	—	A 13	r 49	
	S ⟶ v * B d	B ⟶ * r H			B 15		
45(17)	S ⟶ w * C a	C ⟶ * s G	(0,0,1,0,0,0,0,0,0,0)	—	C 18	s 50	
	S ⟶ w * D d	D ⟶ * s H			D 20		
46(25)	S ⟶ x * C a	C ⟶ * s G	(0,0,0,2,0,0,0,0,0,0)	—	C 26	s 51	
	S ⟶ x * D b	D ⟶ * s H			D 28		
47(30)	S ⟶ y * E a	E ⟶ * t G	(0,0,0,0,2,0,0,0,0,0)	—	E 31	t 52	
	S ⟶ y * F b	F ⟶ * t H			F 33		
48(38)	S ⟶ z * E d	E ⟶ * t G	(0,0,0,0,0,2,0,0,0,0)	—	E 39	t 52	
	S ⟶ z * F c	F ⟶ * t H			F 41		
49(8)	A ⟶ r * G	G ⟶ * e	(0,0,0,0,0,0,1,0,0,0)	—	G 9	e 53	
	B ⟶ r * H	H ⟶ * e			H 10		

Table 17 Continued

Number	State		10-Tuple	Z	Transition	To
50(22)	C $\longrightarrow$ s * G	G $\longrightarrow$ * e	(0,0,0,0,0,0,0,1,0,0)		G 23	e 53
	D $\longrightarrow$ s * H	H $\longrightarrow$ * e			H 24	
51(22)	C $\longrightarrow$ s * G	G $\longrightarrow$ * e	(0,0,0,0,0,0,0,2,0,0)		G 23	e 54
	D $\longrightarrow$ s * H	H $\longrightarrow$ * e			H 24	
52(35)	E $\longrightarrow$ t * G	G $\longrightarrow$ * e	(0,0,0,0,0,0,0,0,2,0)		G 36	e 54
	F $\longrightarrow$ t * H	H $\longrightarrow$ * e			H 37	
53(11)	G $\longrightarrow$ e *		(0,0,0,0,0,0,0,0,0,1)	$Z_{19} = \lvert a,b \rvert$	reduce 19	
	H $\longrightarrow$ e *			$Z_{20} = \lvert c,d \rvert$	reduce 20	
54(11)	G $\longrightarrow$ e *		(0,0,0,0,0,0,0,0,0,2)	$Z_{19} = \lvert a,d \rvert$	reduce 19	
	H $\longrightarrow$ e *			$Z_{20} = \lvert b,c \rvert$	reduce 20	

Figure 6 GR(52), GR(53) and GR(54) First Parser

For Grammar G_6

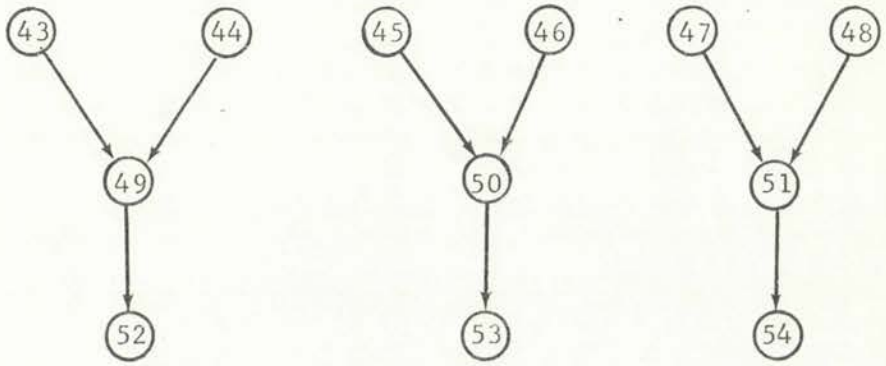
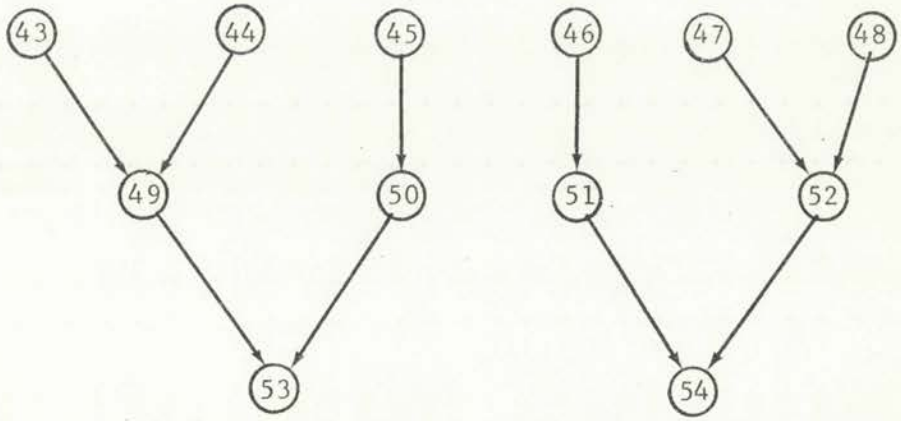


Figure 7 GR(53) and GR(54) Second Parser

For Grammar G_6



10. SUMMARY

In conclusion we have described and proved the validity of a general LR(k) parser building algorithm. This algorithm lies somewhere between Knuth's and DeRemer's. It will yield a result at least as good as Knuth's in all cases and on those grammars on which DeRemer's algorithm works it will yield a result equivalent to DeRemer's. The algorithm might involve somewhat more work than DeRemer's algorithm on most SLR(1) grammars. However this should not be a serious disadvantage.

FOOTNOTES

[1] A parser should not be confused with a recognizer. A recognizer merely answers the question "is this sentence in the language." Knuth (Knu 65) shows that any LR(k) language may be generated by a grammar which is LR(0) with the possible exception of needing to scan ahead one symbol in order to recognize the end of a sentence. Knuth proves that these are precisely the deterministic languages or those languages which may be recognized by a DPDA.

[2] Some authors such as Hopcroft and Ullman (H & U 69) define P as a finite set of ordered pairs in $V_N \times (V \setminus \{\epsilon\})$.

[3] See (H & U 69) for a more complete treatment of the concepts presented so far in this section.

[4] Knuth's original definition (Knu 65) is slightly different but equivalent. I prefer this one due to its conciseness.

[5] Knuth actually sets the stack vocabulary $\Gamma = K \cup V'$ and pushes the transition symbol as well as the state on the stack. DeRemer (DeR 71) attributes the observation that it is unnecessary to include the input vocabulary in the stack vocabulary to J. J. Horning.

[6] Even if G is not LR(k), that is there exists a state I such that $(p, n_p, \alpha) \in I$ and $(q, i, \beta) \in I$ with $\alpha \in H'(q_{i+1} \cdots q_{n_q} \beta)$ and $(p \neq q$ or $n_p \neq i)$, we can still build an LR(k) parsing machine by Knuth's algorithm. However, we must allow for more than one transition from I on scanning α. Thus the resulting machine will not be deterministic. We will use this fact in subsequent sections.

[7] The introduction of the symbol $\vdash$ to denote the beginning of a sentence is a convenience rather than a necessity. Nothing essential has been changed. We refer to the resulting machine as $M_0'(G)$.

[8] We use the plural here since G may be ambiguous.

[9] DeRemer does not consider productions of the form $A \longrightarrow \epsilon$. However the extension is almost trivial.

[10] This is a concocted example. However a grammar of this type was written by students in a compiler writing class this semester at the University of New Mexico. DeRemer's algorithm failed to yield a parser, but the algorithm in Section 8 found an LR(0) parser without resorting to state splitting.

BIBLIOGRAPHY

(DeR 69) DeRemer, F. L. "Practical Translations for LR(k) Languages," Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September, 1969.

(DeR 71) DeRemer, F. L. "Simple LR(k) Grammars," Comm. ACM., 14, 7 (July, 1971), pp. 453-460.

(H & U 69) Hopcroft, J. E., and Ullman, J. D. Formal Languages and Their Relation To Automata. Reading, Massachusetts: Addison-Wesley, Inc., 1969.

(Knu 65) Knuth, D. E. "On the Translation of Languages from Left to Right," Info. Contr., 8,3 (October, 1965), pp. 127-145.

(Knu 71) Knuth, D. E. "Top Down Syntax Analysis," Acta Informatica, 1, (1971), pp. 79-110.

(Kor 69) Korenjak, A. J. "A Practical Method for Constructing LR(k) Processors," Comm. ACM, 12, 11 (November, 1969), pp. 613-623.

(Pag 70) Pager, David. "A Solution To An Open Problem By Knuth," Info. Contr., 17, 5 (December, 1970), pp. 462-473.

CURRICULUM VITAE

Thomas Joshua Sager was born in New York City on March 5, 1942. He attended Reed College and San Francisco State College. He received a B.A. in Mathematics from California State College at San Bernardino in 1969 and an M.A. in Mathematics from the University of New Mexico in 1971. He has served as a graduate assistant and a teaching assistant at the University of New Mexico.