A New Analysis of LALR Formalisms

JOSEPH C. H. PARK, K. M. CHOE, AND C. H. CHANG
Korea Advanced Institute of Science and Technology

The traditional LALR analysis is reexamined using a new operator and an associated graph. An improved method that allows factoring out a crucial part of the computation for defining states of LR(0) canonical collection and for computing LALR(1) lookahead sets is presented. This factorization leads to significantly improved algorithms with respect to execution time as well as storage requirements. Experimental results including comparison with other known methods are presented.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*syntax*; D.3.4 [**Programming Languages**]: Processors—*compilers, parsing, translator writing systems and compiler generators*; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems—*context free, parsing*

General Terms: Algorithms, Experimentation, Languages, Theory

Additional Key Words and Phrases: LALR(*k*) grammar, LALR(1) lookahead sets, parser generating system

1. INTRODUCTION

Since the original invention of LR parsing by Knuth [8] many years ago, and the subsequent realization of its practicality in the LALR form by DeRemer [5], LALR analysis has been extensively investigated [1], and has come to be accepted as the most powerful practical technique known for syntax analysis of programming languages modelled as context-free languages.

The essential parts of the traditional LALR formalism, namely, those pertaining to the definition of states and computation of lookahead sets, are recursive. This, unfortunately, hinders derivation of efficient algorithms needed in implementing a practical parser generating system. In the new approach, to be described, an operator δ is introduced to convert recursion into explicit iteration, where appropriate, and, in so doing, obtain efficient algorithms together with easily comprehensible proofs. The underlying ideas in an initial form have been described in a working paper [12], and are refined in the present report on the basis of extensive experimentation, lasting more than a year.

This work was funded in part by the Korea Traders Scholarship Foundation.
Authors' present addresses: J. C. H. Park, Braegen Corp., 525 Los Coches St., Milpitas, CA 95035; K. M. Choe and C. H. Chang, Project Micro Laboratory, Dept. of Computer Science, Korea Advanced Institute of Science and Technology, P.O. Box 150, Chongyangni, Seoul, 131, Korea.

Section 2 is a summary of the basic terminology and definitions used throughout this paper. A new operator δ and the associated graph (called the L-graph) are introduced in Section 3. On the basis of this approach, an explicit formula for computing LALR(k) lookahead sets is derived in Section 4. In Section 5, other known algorithms dealing with LALR(1) lookahead sets are recast and modified using our formalism. In each case examined, significant improvement in performance is obtained, culminating in new algorithms that are superior with respect to computing time as well as storage requirements. Experimental results supporting this fact are presented in Section 6.

2. TERMINOLOGY AND DEFINITIONS

The basic terminology and definitions in this paper are consistent with those of Aho and Ullman [2, 3]. Notational conventions are also stated.

A context-free grammar (CFG) is a quadruple (N, T, P, S), where N, T, P, and S stand, respectively, for a set of nonterminal symbols, a set of terminal symbols, a set of productions (each of which is of the form $A \rightarrow \omega$), and a start symbol in N. Given a grammar G, V (the vocabulary) stands for $N \cup T$ and V^* for the reflexive-transitive closure of V; the transitive closure is indicated by the superscript $^+$.

Lower-case Greek letters such as α, β, γ, and ω denote strings in V^*, lower-case Roman letters toward the beginning of the alphabet (a, b, and c) are terminals, whereas those near the end (u, v, and w) are strings in T^*; upper-case letters in the beginning of the alphabet (A, B, and C) are nonterminals, whereas those near the end (X, Y, and Z) are symbols in V. An empty string is denoted by Λ.

Familiarity with LR parsing, particularly LALR parsing, is assumed. We take for granted concepts such as FIRST_k, $\oplus_k$, LR(k) item, and the canonical collection of the set of LR(k) items. Also, a given grammar G is assumed to be *augmented* with a new rule, $S' \rightarrow S\$$, to P, with a new start symbol S' not in N and an endmarker \$ not in T. In an LR(k) item, the symbol after the dot is referred to as *marked*.

Let C_k be the canonical collection of states of LR(k) items. A state p in C_k is characterized by its *kernel* K_p of certain LR(k) items such that

$$p = \text{CLOSURE}\ (K_p). \tag{2.1}$$

Given a set K of LR(k) items, the CLOSURE [2] is defined as the smallest set satisfying

$$\begin{aligned}\text{CLOSURE}(K) = K \cup \{[X \rightarrow \cdot\omega, \text{FIRST}_k(\beta u)] \mid [A \rightarrow \alpha\cdot X\beta, u] \\ \in \text{CLOSURE}(K), X \rightarrow \omega \in P\}.\end{aligned} \tag{2.2}$$

The traditional LR(k) formalism makes use of another operation, called GOTO, defined as follows. Let $p \in C_k$ and $X \in V$. Then

$$\text{GOTO}(p, X) = \text{CLOSURE}(\{[A \rightarrow \alpha X\cdot\beta, u] \mid [A \rightarrow \alpha\cdot X\beta, u] \in p\}). \tag{2.3}$$

For subsequent discussion, it is convenient to introduce another function, PRED [9], whose effect is akin to an inverse of GOTO. Let $p, q \in C_k$, and α be any string in V^*. Then,

$$\text{PRED}(p, \alpha) = \{q \mid p \in \text{GOTO}(q, \alpha)\}. \tag{2.4}$$

In (2.4) the definition of GOTO has been extended to a string such that

$$\begin{aligned} \text{GOTO}(p, X\alpha) &= \text{GOTO}(\text{GOTO}(p, X), \alpha) \\ \text{GOTO}(p, \Lambda) &= p. \end{aligned} \tag{2.5}$$

A salient feature of the LR(k) formalism is that the canonical collection of states, together with GOTO, defines a deterministic-finite automation (DFA) for recognizing viable prefixes associated with the given grammar.

CLOSURE, an essential entity of the LR formalism, has been defined "recursively," as in (2.2). An understanding of the underlying formalism as well as derivation of efficient algorithms is aided by devising an "iterative" formulation of CLOSURE, as described in the next section.

3. A NEW OPERATOR δ AND THE L-GRAPH

The operator δ is a mapping in the power set of LR(k) items, defined as follows. Let $[A \to \alpha \cdot X\beta, u]$ be an LR(k) item for a context-free grammar $G = (N, T, P, S)$. Then,

$$\delta\{[A \to \alpha \cdot X\beta, u]\} = \begin{cases} \{\}, \text{ if } X \in T \text{ or } X\beta \text{ does not exist} \\ \{[X \to \cdot\omega, y] \mid y \in \text{FIRST}_k(\beta u), X \to \omega \in P\}. \end{cases} \tag{3.1}$$

For brevity, we simply write:

$$\delta\{[A \to \alpha \cdot X\beta, u]\} = \{[X \to \cdot\omega, \text{FIRST}_k(\beta u)] \mid X \to \omega \in P\}. \tag{3.2}$$

It turns out that the CLOSURE operation can be defined "iteratively" as the reflexive-transitive completion δ^* of δ:

$$\text{CLOSURE}(\{[A \to \alpha \cdot X\beta, u]\}) = \delta^*\{[A \to \alpha \cdot X\beta, u]\}. \tag{3.3}$$

In pursuing properties of δ, it soon becomes obvious that it essentially reflects a left dependency relation $L \subseteq N \times N$:

$$B \; L \; C \qquad \text{iff} \quad B \to C\gamma \in P. \tag{3.4}$$

We also call the directed graph associated with relation L the L-graph. It is constructed by representing each instance (3.4) as a pair of vertices connected by a directed edge:

$$\begin{tikzcd} \textcircled{B} \arrow[r, "\gamma"'] & \textcircled{C} \end{tikzcd}$$

Note that the edge has been labeled with the remaining part of the production right-hand side (RHS) that follows C.

A useful property of δ is established by the following lemma, a straightforward proof for which is given in Appendix A.1.

LEMMA 3.1.

$$\delta\{[A \rightarrow \alpha \cdot B\beta,\ u]\} = \{[C \rightarrow \cdot\gamma,\ Path_k(B,\ C) \oplus_k FIRST_k(\beta u)] \mid BL^{n-1}C,\ C \rightarrow \alpha \in P\}, \tag{3.5}$$

with the notation:

$$Path_k(B,\ C) = \bigcup \{FIRST_k(\beta_n \ldots \beta_2\beta_1) \mid B_0 = B,\ B_n = C,\ N \geq 0,\ B_0 \rightarrow B_1\beta_1 \in P,\ B_1 \rightarrow B_2\beta_2 \in P,\ \ldots,\ B_{n-1} \rightarrow B_n\beta_n \in P\}, \tag{3.6}$$

where the sequence $\beta_1 \ldots, \beta_n$ describes a path from B to C in the L-graph of the form

$$\begin{tikzcd} \textcircled{B} \arrow[r, "\beta_1"'] \arrow[d, phantom, "\|"] & \bigcirc \arrow[r, "\beta_2"'] & \cdots \arrow[r, "\beta_n"'] & \textcircled{C} \arrow[d, phantom, "\|"] \\ B_0 & B_1 & \cdots & B_n \end{tikzcd}$$

In (3.6) the $\cup$ stands for union over all such paths from B to C.

Using Lemma 3.1, it is easy in turn to prove the following result, which essentially captures the CLOSURE operation as recursively stated in (2.2).

LEMMA 3.2.

$$\delta^{+}\ \{[A \rightarrow \alpha \cdot B\beta,\ u]\} = \{[C \rightarrow \cdot\gamma,\ Path_k(B,\ C) \oplus_k FIRST_k(\beta u)] \mid BL^{*}C,\ C \rightarrow \gamma \in P\}. \tag{3.7}$$

Observe that, even when cycles are present in the L-graph, only paths of finite length are necessary in computing (3.6), due to FIRST_k, and that the total number of paths involved is finite. Thus the union in (3.6) is finite, leading to a well-defined $\text{Path}_k(B,\ C)$ and the transitive closure of δ in turn. For example, when $k = 1$, a cycle never need be traversed more than once.

With δ, a state p in a canonical collection is characterized by its kernel K_p of certain LR items such that

$$p = \delta^{*}K_p = K_p \cup \delta^{+}K_p, \tag{3.8}$$

where the two sets K_p and $\delta^{+}K_p$ are disjoint.

It is apparent in this formalism that both the construction of the GOTO function of the underlying DFA and the propagation of lookahead sets depend only on *kernel* items and *one* L-graph. The need for dealing with a large number of items (due to CLOSURE operations) is thus explicitly eliminated, as suggested by Aho and Ullman [2]. Furthermore, repeated traversals of the given L-graph (which must equivalently be performed in all other known algorithms) are also easily avoided in this approach, as elaborated subsequently.

In addition, the following lemma is useful in establishing Theorem 4.1, of the subsequent section, dealing with lookahead sets. A proof is found in Appendix A.2.

LEMMA 3.3. *Let $p, q \in C_k$ such that $q \in PRED(p, \alpha_1)$, $[A \rightarrow \alpha_1 \cdot \alpha_2, x] \in p$, and $A \neq S'$. Then there must be at least one kernel item $[B \rightarrow \beta_1 \cdot A'\beta_2, u] \in K_q$, satisfying*

$$x \in \{z \mid z \in Path_k(A', A) \oplus {}_k FIRST_k(\beta_2 u), \quad q \in PRED(p, \alpha_1), A'L^*A, [B \rightarrow \beta_1 \cdot A'\beta_2, u] \in K_q\}. \tag{3.9}$$

4. LALR LOOKAHEAD SETS

Having characterized a state and lookahead sets through Eqs. (3.7) to (3.9), a formula for LALR(k) lookahead sets is derived in a form amenable to efficient computation. The starting point is the following theorem:

THEOREM 4.1. *Let C_0 be the canonical collection of states of $LR(0)$ items, and let $p \in C_0$, $[A \rightarrow \alpha_1 \cdot \alpha_2] \in p$, and $A \neq S'$. Then the $LALR(k)$ lookahead set is the smallest set satisfying*

$$LA_k(p, [A \rightarrow \alpha_1 \cdot \alpha_2]) = \{x \mid x \in Path_k(A', A) \oplus_k FIRST_k(\beta_2) \oplus_k LA_k(q, [B \rightarrow \beta_1 \cdot A'\beta_2]), \quad q \in PRED(p, \alpha_1), A'L^*A, [B \rightarrow \beta_1 \cdot A'\beta_2] \in K_q\}. \tag{4.1}$$

PROOF. Since the LALR(k) lookahead set of an LR(0) item $[A \rightarrow \alpha_1 \cdot \alpha_2]$ in state p of C_0 is the union of *all* lookahead sets x of LR(k) item $[A \rightarrow \alpha_1 \cdot \alpha_2, x]$ in C_k [9, Definition 2.5], the theorem follows as a consequence of Lemma 3.3. □

Equation (4.1) contains two union-loops. The first union in (4.1) is over all states q in C_0 reachable from p in the transition (GOTO) graph of the underlying DFA by a backward traversal of edges (through the PRED function as defined in (2.4)). Given such q, there may be more than one item $[B \rightarrow \beta_1 \cdot A'\beta_2]$ in the kernel of q such that $A'L^*A$. The second loop in (4.1) deals with such cases. Equation (4.1) can thus be recast in the following form:

$$LA_k(p, [A \rightarrow \alpha_1 \cdot \alpha_2]) = \bigcup_{\substack{q \\ q \in \mathrm{PRED}(p,\alpha_1)}} \bigcup_{[B \rightarrow \beta_1 \cdot A'A'L^*A\beta_2] \in K_q} \mathrm{Path}_k(A', A) \oplus_k \mathrm{FIRST}_k(\beta_2) \oplus_k LA_k(q, [B \rightarrow \beta_1 \cdot A'\beta_2]). \tag{4.2}$$

Equation (4.2) explicitly exhibits why this approach is more efficient than other known methods. It is illuminating to digress and pursue this point. All known methods in other work [2, 4, 6, 9, 10] hinge on a formula that can be

$$\begin{tikzcd}
q \arrow[dd, "\alpha_1"] & {[B \to \beta_1 \cdot A'\beta_2] \in K_q} \\
 & {[A \to \cdot\alpha_1\alpha_2] \in \partial^{+}K_q} \\
p & {[A \to \alpha_1 \cdot \alpha_2] \in p}
\end{tikzcd}$$

Fig. 1. The GOTO DFA and Items.

Fig. 2. The L-graph.

$$\begin{tikzcd}
A_0 \arrow[r] \arrow[d, equal] & A_1 \arrow[r] & \cdots \arrow[r] & A_n \arrow[d, equal] \\
A'\ \omega_1 & \omega_2 & & \omega_n\ A
\end{tikzcd}$$

$$\begin{aligned}
A_0 &\to A_1\omega_1\\
A_1 &\to A_2\omega_2\\
&\vdots\\
A_{n-1} &\to A_n\omega_n.
\end{aligned}$$

Fig. 3. The rules.

equivalently expressed in our notation as follows [9]:

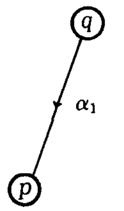
$$LA_k(p,\ [A \to \alpha_1 \cdot \alpha_2]) = \bigcup_{\substack{q\\ q \in \mathrm{PRED}(p,\alpha\ )}} \ \bigcup_{[C \to \gamma_1 \cdot A\gamma_2] \in q} \mathrm{FIRST}_k(\gamma_2) \oplus_k LA_k(q,\ [C \to \gamma_1 \cdot A\gamma_2]). \tag{4.3}$$

Formula (4.3) involves all items $[C \to \gamma_1 \cdot A\gamma_2]$ in a state q, whereas formula (4.2) only calls for items $[B \to \beta_1 \cdot A'\beta_2]$ in the kernel of a given state q such that $A'L^{*}A$. Thus the need for lookahead computation by recursion, involving items due to a CLOSURE operation, viz., from $[B \to \beta_1 \cdot A'\beta_2]$ to $[C \to \gamma_1 \cdot A\gamma_2]$ in the state q, is eliminated in our case by the use of a single quantity, $\mathrm{Path}_k(A', A)$. Note also that since $\mathrm{Path}_k(A', A)$ depends only on nonterminals (not on states), it is to be calculated once and for all, in contrast to other known methods, which involve repeated computations of an equivalent quantity for each state.

Consider a path from A' to A in the L-graph, and the corresponding CLOSURE (δ^{+}) operation. For $n \geq 0$, let the vertices from A' to A be $A_0, A_1 \ldots A_n$ (where $A' = A_0$ and $A = A_n$), and the edges be $\omega_1, \omega_2 \ldots \omega_n$, as shown in Figure 2. Due to the definition of the L-graph, there are corresponding productions, see Figure 3.

The state q of Theorem 4.1 has the kernel item whose "marked" symbol is A' ($=A_0$), and $p = \mathrm{GOTO}(q, \alpha_1)$ is shown as the Figure 4.

First, LA_k on the RHS of Eq. (4.2) comes into play (calling for recursive evaluation), only if both $\mathrm{Path}_k(A', A)$ and $\mathrm{FIRST}_k(\beta_2)$ contain Λ when $k = 1$. This is true when β_2 derives Λ and there exists at least one path labelled $\omega_1 \ldots \omega_n$ from A' to A in the L-graph such that all ω_i derive Λ. In that case, recursive



$$\begin{array}{lll}
[B & \rightarrow \beta_1 \cdot A'\beta_2] \in K_q & \\
[A_0 & \rightarrow \cdot A_1\omega_1] \in \partial^+ K_q & \text{where } A_0 = A' \\
[A_1 & \rightarrow \cdot A_2\omega_2] \in \partial^+ K_q & \\
[A_{n-1} & \rightarrow \cdot A_n\omega_n] \in \partial^+ K_q & \text{where } A_n = A \\
[A & \rightarrow \cdot \alpha_1\alpha_2] \in \partial^+ K_q &
\end{array}$$

$$[A \rightarrow \alpha_1 \cdot \alpha_2] \in p$$

Fig. 4. The GOTO DFA and Items.

use of formula (4.3) is needed n times in the state q, and once in the β_1-predecessor state of q in the formalisms of others. In contrast, no recursion in the state q (by $\text{Path}_k(A', A)$) or recursion to the β_1-predecessor state is needed in our formalism. The lookahead sets added by the CLOSURE (δ^+) operation in a state do not require recursion but only a single term, $\text{Path}_k(A', A)$.

Second, since Path, which represents the lookahead sets contributed by the CLOSURE operation, is independent of states, another substantial reduction in recursion is obtained by computing it once for all states that have the same "marked" symbol A' in the associated kernel. Equivalently stated, the need for a repeated recursive traversal of the L-graph (viz., the transitive completion of δ) found in the traditional method is eliminated explicitly by $\text{Path}_k(A', A)$ in our method, since the latter is independent of states.

Finally, since our formalism (4.2) deals only with items in the kernel K of a state, space and/or time overhead for dealing with items due to δ^+ are eliminated.

5. ALGORITHMS FOR LALR(1) LOOKAHEAD SETS

Having described the framework of our formalism, the purpose of this section is to derive explicit algorithms for computing LALR(1) lookahead sets, required in realizing an efficient LALR parser generator. A series of algorithms is given, including those of others, which are recast in our formalism and substantially improved. This discussion also serves as a basis for an experimental comparison, described in the next section.

From various known methods for computing LALR lookahead sets [2, 4, 6, 8, 10], it is instructive to select two representative approaches: one based on recursive calls and another on traversing a digraph, due to DeRemer and Pennello [6]. The former is conceptually the simplest, while the latter represents the most time-efficient method known prior to our result. Since our formalism is applicable to both types, we are led to distinguish four methods: (1) recursion only (R); (2) modified recursion using the new formalism (NR); (3) digraph traversal only (D); and, finally, the best method, (4) modified digraph traversal (ND). All of these algorithms are stated in a Pascal-like language.

5.1 Computing LALR(1) Lookahead Sets by Recursive Calls

Using recursive calls, Eq. (4.3) yields the simplest algorithm R [9] for computing the LALR lookahead sets; it is straightforward and is omitted. This algorithm can be improved substantially by using our result (4.2).

Algorithm NR. Computing Lookahead Sets by Recursive Calls Using the L-Graph

```
function LALR($p$:state, $I$:item): set of $T$;
  assume $I = [A \rightarrow \alpha_1 \cdot \alpha_2]$;
  LALR := {};
  for $q \in$ PRED($p$, $\alpha_1$) do
    for $[B \rightarrow \beta_1 \cdot A'\beta_2] \in K_q$ where $A'L^*A$ do
        LALR := LALR $\cup$ Path($A'$, $A$);
        if $\Lambda \in$ Path($A'$, $A$) then
          begin
          LALR := LALR $\cup$ FIRST($\beta_2$);
          if $\Lambda \in$ FIRST($\beta_2$) then LALR := LALR $\cup$ LALR($q$, $[B \rightarrow \beta_1 \cdot A'\beta_2]$)
          end
    end for
  end for
end function.
```

Note 1. Algorithm NR requires an additional mechanism for avoiding pathological cases of recursive calls to the same state and item, giving rise to infinite loops. Details are omitted for simplicity; but the mechanism should be included in actual programs.

Note 2. FIRST(β_2) in NR plays a role equivalent to TRANS(GOTO(q, A')) of Kristensen and Madsen [9].

Algorithm NR is an improvement over algorithm R because (1) the former involves only items in the kernel for each state, eliminating items due to a CLOSURE operation (this is the case even after taking into account the effects of L-graph and Path); and (2) the if test for Λ in Path(A, A'), which prevents recursive calls to items in the same state, effectively reduces recursion.

These observations are quantitatively verified in actual experiments, as discussed in Section 6.

5.2 Computing LALR(1) Lookahead Sets by Traversing a Digraph

In an improved method, suggested by DeRemer and Pennello [6], instead of recursive calls, a partial ordering, called an *includes* relation, is introduced for sets determined to be giving rise to a digraph. A generalized topological sorting a la Tarjan [15] then yields a linear ordering of sets, to be evaluated from the smallest to the largest through "union" operations. The number of "union" operations is at least that of the edges in the digraph, which is essentially the minimum number of recursive calls required; this represents the best possible behavior of algorithm R.

To recast DeRemer and Pennello's approach in our formalism, it is convenient to redefine their Follow. Let $q \in C_0$, $A \in N$, and $a \in T$. Then

$$\text{Follow}(q, A) = \{a \mid a \in \text{FIRST}(\beta_2) \oplus LA(q, [B \rightarrow \beta_1 \cdot A'\beta_2]), [B \rightarrow \beta_1 \cdot A'\beta_2] \in q\}. \quad (5.1)$$

With this definition of Follow, DeRemer and Pennello's method is compactly recaptured by the following three results, whose proofs are given in Appendix A.3. Parenthetically, DeRemer and Pennello's concept "Read" is not necessary in our presentation, since it is effectively handled by FIRST. Note that computation of FIRST is also required in constructing LR(0) states. Hence, the results are available at no additional cost.

Let $p, q \in C_0$. Then,

$$LA(p, [A \rightarrow \alpha_1 \cdot \alpha_2]) = \{a \mid a \in \text{Follow}(q, A), q \in \text{PRED}(p, \alpha_1)\}. \tag{5.2}$$

Let $q, r \in C_0$. Then,

$$\text{Follow}(q, A) = \{a \mid a \in \text{FIRST}(\beta_2), \oplus \text{Follow}(r, B), [B \rightarrow \beta_1 \cdot A\beta_2\} \in q, \\ r \in \text{PRED}(q, \beta_1), [C \rightarrow \gamma_1 \cdot B\gamma_2] \in r\}. \tag{5.3}$$

Let $q, r \in C_0$, $[B \rightarrow \beta_1 \cdot A\beta_2] \in q$, $[C \rightarrow \gamma_1 \cdot b\gamma_2] \in r$, $A, B \in N$. If $r \in \text{PRED}(a, \beta_1)$ and $\Lambda \in \text{FIRST}(\beta_2)$, then,

$$\text{Follow}(q, A) \supseteq \text{Follow}(r, B) \quad \text{or} \quad \text{Follow}(q, A) \text{ “includes” } \text{Follow}(r, B). \tag{5.4}$$

Evaluation of lookahead sets then proceeds in three steps:

(1) Construct the digraph of vertices, each representing Follow(q, A) initialized with the value FIRST(β_2), as in (5.1), and of edges corresponding to an “includes” relation, as in (5.4).
(2) Compute a Follow(q, A) traversing digraph (according to Tarjan's algorithm).
(3) Finally, compute lookahead sets using Follow and PRED as prescribed by (5.2).

Algorithm D1 deals with construction of the digraph as sketched in step (1), and Algorithm D3 performs step (3). Details for step (2) are omitted, since it is the well-known problem of ubiquitous generalized topological sorting [6, 15].

Algorithm D1. Construction of the Digraph

```
for $q \in C_0$ do
  for $[B \rightarrow \beta_1 \cdot A\beta_2] \in q$ where $A \in N$ do
    Follow($q$, $A$) := FIRST($\beta_2$);
    if $\Lambda \in$ FIRST($\beta_2$) then
      for $r \in$ PRED($q$, $\beta_1$) do
        for $[C \rightarrow \beta_1 \cdot B\beta_2] \in r$ where $B \in N$ do
          Follow($q$, $A$) “includes” Follow($r$, $B$)
        end for
      end for
  end for
end for.
```

Algorithm D3. Lookahead Computation Using Follow

```
for $p \in C_0$ do
  for $[A \rightarrow \alpha.] \in p$ do
    $LA(p, [A \rightarrow \alpha.]) := \{\}$;
    for $q \in$ PRED($p$, $\alpha$) do
      $LA(p, [A \rightarrow \alpha.]) := LA(p, [A \rightarrow \alpha.]) \cup$ Follow($q$, $A$)
    end for
  end for
end for.
```

DeRemer and Pennello's strategy, as paraphrased above, results in a striking improvement in computing time when compared with methods based on recursive calls. Unfortunately, it considers all items—those due to δ^+ as well as kernels. Thus it is not surprising that the digraph of Follow vertices and “includes” edges is large. A substantial reduction in size results if this method is modified to deal with kernel items only, as in our formulation. The result, as presented below, enhances storage requirements, while gaining time efficiency.

Expressions (5.2) through (5.4) are modified as in (5.5) through (5.7), respectively, with the use of Path and the L-graph so that only kernel items are involved. The proofs are given in Appendix A.4, and the corresponding algorithms are given as ND1 and ND3.

Let $p, q \in C_0$. Then,

$$LA(p, [A \rightarrow \alpha_1 \cdot \alpha_2]) = \{a \mid a \in \text{Path}(A', A) \oplus \text{Follow}(q, A'),\ q \in \text{PRED}(p, \alpha_1),\ A' L^* A,\ [B \rightarrow \beta_1 \cdot A' \beta_2] \in K_q\}. \tag{5.5}$$

Let $p, q \in C_0$. Then,

$$\text{Follow}(q, A') = \{a \mid a \in \text{FIRST}(\beta_2) \oplus \text{Path}(B', B) \oplus \text{Follow}(r, B'),\ [B \rightarrow \beta_1 \cdot A' \beta_2] \in K_q,\ r \in \text{PRED}(q, \beta_1),\ [C \rightarrow \gamma_1 \cdot B' \gamma_2] \in K_r,\ B' L^* B\}. \tag{5.6}$$

Let $q, r, \in C_0$, $[B \rightarrow \beta_1 \cdot A' \beta_2] \in K_q$, $[C \rightarrow \gamma_1 \cdot B' \gamma_2] \in K_r$. If $r \in \text{PRED}(q, \beta_1)$, $\Lambda \in \text{FIRST}(\beta_2)$, and $\Lambda \in \text{Path}(B', B)$. Then,

$$\text{Follow}(q, A') \supseteq \text{Follow}(r, B') \text{ or } \text{Follow}(q, A') \text{ “includes” } \text{Follow}(r, B'). \tag{5.7}$$

Algorithm ND1. Construction of the Digraph-Using Kernel Items Only

```
for $q \in C_0$ do
  for $[B \rightarrow \beta_1 \cdot A'\beta_2] \in K_q$ where $A' \in N$ do
    Follow($q, A'$) := FIRST($\beta_2$);
    if $\Lambda \in$ FIRST($\beta_2$) then
      for $r \in$ PRED($q, \beta_1$) do
        for $[C \rightarrow \beta_1 \cdot B'\beta_2\{ \in K_r$ where $B'L^* B$ do
          Follow($q, A'$) := Follow($q, A'$) $\cup$ Path($B', B$);
          if $\Lambda \in$ Path($B', B$)
            then Follow($q, A'$) “includes” Follow($r, B'$)
        end for
      end for
  end for
end for.
```

Algorithm ND3. Lookahead Computation Using Follow

```
for $p \in C_0$ do
  for $[A \rightarrow \alpha \cdot] \in p$ do
      $LA(p, [A \rightarrow \alpha \cdot]) := \{\}$;
  for $q \in$ PRED($p, \alpha$) where $[B \rightarrow \beta_1 \cdot A'\beta_2] \in K_q$ do
      $LA(p, [A \rightarrow \alpha \cdot]) := LA(p, [A \rightarrow \alpha \cdot]) \cup$ Path($A'$ $A$);
      if $\Lambda \in$ Path($A', A$)
        then $LA(p, [A \rightarrow \alpha \cdot]) := LA(p, [A \rightarrow \alpha \cdot]) \cup$ Follow($q, A'$)
    end for
  end for
end for.
```

The number of Follow vertices in the digraph is reduced significantly in the new method, since, as mentioned earlier, only those of kernels are involved, with

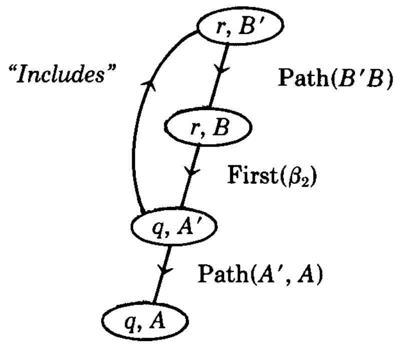


Fig. 5. The "includes" relations in the digraph and lookahead propagation.

the effect of vertices due to items δ^+ taken care of by the L-graph. Consider a situation depicted in Figure 5. If Follow($q \cdot A'$) "includes" Follow(r, B'), then the number of edges (i.e., "includes" in the original approach) is $n + 1$, when n is the path length in L-graph from B' to B. In the modified approach for the same case, only one edge appears, as opposed to $n + 1$. Thus not only the number of vertices, but also that of edges, is substantially reduced.

Furthermore, for all other states whose kernels contain the "marked" symbol B', the same reduction phenomenon occurs, since the lookahead sets are computed using the Path of (5.5) through (5.7), in a manner independent of states. Thus the overall reduction in number of edges, and hence union costs, is substantial.

It is straightforward to calculate Path(A', A), as defined in (3.6), by traversing the associated L-graph, with the direction of edges reversed for convenience. Tarjan's algorithm, mentioned earlier, for traversing the Follow digraph is modified to traverse the L-graph effectively, twice for this purpose. The saving in union costs, mentioned above, is partially offset by union operations required in calculating Path in this manner.

This traversal yields the transitive closure L^* and linked lists representing Path (A', A). The scheme used involves searching such a list for a given nonterminal (via., for A' starting for an element $H[A]$ of an array of list headers). The experimental data reported in the next section include such time overhead.

The advantage in our method, however, does not hinge on clever implementation details. For example, in processing Ada with 182 nonterminals, DeRemer and Pennello's method must deal with 2022 Follow(q, A) sets, whereas our method requires only 322 Follow sets and 768 nonempty Path sets. The order of magnitude reduction in Follow sets comes about because our method requires those due to kernel items only. The remaining Follow sets (i.e., those due to CLOSURE items) are effectively taken into account by Path sets. Note that in general far fewer Path sets are required than Follow sets, since the former depend only on nonterminals, in contrast to the latter, involving states. A theoretical assessment of this saving requires a formula relating the number of states to that of nonterminals for a given LALR(1) grammar, the derivation of which appears to be quite difficult, and is beyond the scope of the present paper. Experimental investigations are discussed in the next section.

6. EXPERIMENTAL RESULTS

The four methods for computing LALR(1) lookahead sets, namely algorithms R, NR, D, and ND have been implemented using Standard Pascal in an LALR(1) parser-generating system PGS82 [13], running on a midsize computing facility based on an IBM System/370-145, and used to experimentally evaluate their (the algorithms') performance.

LALR grammars for five languages, namely ADA [14], Pascal [7], PL360 [16], XPL [11], and PAL [3], have been investigated using PGS82. The results are summarized in Tables I and II.

In Table I, Part 1 pertains to data common to all four methods considered (number of rules(P), nonterminals(N), terminals(T), and LR(0) states(S)). Part 2 refers to the number of items; the top entry represents total numbers, including those of δ^+K, whereas the bottom represents those of kernels K only. Under Parts 3 and 4, the number of recursive calls and properties (number of vertices and edges) of the Follow digraph, respectively, are given; upper (lower) numbers represent unmodified (modified) results. Under Part 5, the properties of the L-graph and the number of nonempty sets of Path, needed in the modified methods only, are summarized.

In Table II, the time characteristics of all four methods are given. The numbers are actual execution times in seconds on an IBM system 370/145 (under OS/VS1). The data show the relative performances. The PGS82 [13] used in these experiments consists of two passes. Pass 1 deals with initialization such as grammar input, validity checks of the grammar, and the conversion of the grammar into the internal representation, etc.; this part is common to all four methods studied. Pass 2 is logically divided into four or five subsections dealing with initialization, computation of FIRST, L-graph and Path (needed in modified methods only), LR(0) states, and, finally, lookahead sets. Actual time spent in each subsection is listed; in each entry the upper (lower) refers to the unmodified (modified) case.

As shown in Part 2 of Table I, the total number of items to be dealt with is reduced in the modified version by a factor ranging from 3 to 9, compared to the unmodified case. This reduction in the size of data to be processed contributes to an overall gain in time/space efficiency for CFSM computation.

Table II indicates that, for the method based on recursion, the modified approach (NR) in pass 2 achieves a 20 to 50 percent reduction in CPU time when compared with the traditional approach (R). This gain is attributable to the 10 to 20 percent reduction in the computation of LR(0) state sets and the 40 to 60 percent reduction in computation of LALR(1) lookahead sets.

As expected, the more elaborate schemes D and ND exhibit better time characteristics than the corresponding R and NR in computing lookahead sets. As listed in Table II, the reduction in time of ND compared to that of D ranges from 15 to 20 percent. This is attributable to the 10 to 20 percent reduction in computing LR(0) states and the 20 to 30 percent in computing LALR(1) lookahead sets; the latter includes time taken for Path computation.

As a direct consequence of the reduced time for computing lookahead sets (in D and ND), the time for computing LR(0) states becomes significant. Here, again, our approach (lower entries under the LR(0) heading of Table II) consis-

Table I. Performance Comparison

	Part 1				Part 2	Part 3	Part 4 Follow		Part 5		
					Item all kernels	Recursions R NR	Vertex D ND	Edge D ND	L-graph vertex	Edge	Path
	P	N	T	S					(NR, ND)		
Ada	398	182	93	823	5943	13237	2022	2857			
					1305	4599	322	1177	182	205	768
Pascal	201	99	65	374	2732	5465	831	1361			
					526	1840	111	753	99	102	308
PL360	153	64	61	227	1067	987	289	280			
					317	337	52	129	64	74	316
XPL	112	52	47	186	1290	2001	484	564			
					257	515	62	260	52	63	231
PAL	80	32	47	156	1770	8710	590	1336			
					205	1045	63	827	32	39	280

Table II. Time Characteristics

		Pass 2							
		Init.	First	L-gr.	LR(0)	LRLR(1)		Total	
	Pass 1	R, D	R, D	R, D	R, D	R	D	R	D
Ada	71.69	3.56	0.50	NA	100.43	396.15	36.10	500.64	140.59
	71.69	3.56	0.50	1.95	91.03	150.41	22.25	247.45	119.29
Pascal	28.41	2.34	0.30	NA	36.22	91.94	12.03	130.80	50.89
	28.41	2.34	0.30	0.74	32.74	60.37	7.75	96.49	43.87
XPL	12.83	1.69	0.14	NA	11.01	20.97	5.76	33.81	18.60
	12.83	1.69	0.14	0.58	9.02	12.53	3.47	23.96	14.90
PL360	17.83	1.87	0.27	NA	9.90	8.31	4.87	20.35	16.91
	17.83	1.87	0.27	0.61	7.96	5.21	3.35	15.92	14.06
PAL	9.43	1.51	0.16	NA	12.88	169.73	8.36	184.28	22.91
	9.43	1.51	0.16	0.68	10.18	101.50	5.54	114.03	18.07

tently shows improvement. Note that the time taken for Path computation (listed separately under L-gr. in Table I) is insignificant.

Finally, as summarized in Parts 4 and 5 of Table I, the number of Follow vertices and entries for Path in ND is about half that of Follow vertices required in D (with the exception of PL360), implying the corresponding reduction in storage requirements in our method (ND).

7. CONCLUSIONS

In our research, originally motivated by a desire to understand the LALR(1) analysis, we were led to a formalism that (1) makes the underlying analysis more transparent, in that fewer definitions and lesser levels of argument are needed, and one that is easily comprehensible, in that the arguments needed are to a large degree reduced to formula manipulations; and (2) that by providing explicit formulas, leads to efficient algorithms both in space and time for constructing LR(0) states and computing LALR(1) lookahead sets.

Specifically, in traditional methods of computing LR(0) states and LALR(1) lookahead sets, certain relationships between nonterminals in the grammar are repeatedly reexamined, and the values that depend on these relationships only are repeatedly recomputed. A new method is devised in which (by factoring out Path) the relationships are examined once and the values are precomputed. This results in time and/or space saving for the LR(0) states and LALR(1) lookahead set computation.

Experiments have been performed involving comparisons with other known methods to verify the ensuing advantages of our approach in actual use.

ACKNOWLEDGMENTS

The authors are grateful to the referees for suggesting various improvements pertaining to writing style. In particular, one of the concluding paragraphs is almost verbatim that of one of the referees.

This research would not have been possible if the computing facility of the Computer Science Research Center, established as a collaborative effort with IBM, had not been available.

APPENDIX A

A.1. Proof of Lemma 3.1. Induction on n.

(1) For $n = 1$, from (3.2), $\delta\{[A \rightarrow \alpha \cdot B\beta, u]\} = \{[B \rightarrow \cdot \gamma, \text{FIRST}_k(\beta u)] \mid B \rightarrow \gamma \in P\}$. Since $B = C$, $\Lambda \in \text{Path}_k(B, C)$, and BL^0C. Lemma 3.1 holds for $n = 1$.

(2) Assume Lemma 3.1 holds for $n = i$,

$$\delta^i\{[A \rightarrow \alpha \cdot B\beta, u]\} = \{[C \rightarrow \cdot \gamma, \text{Path}_k(B, C) \oplus_k \text{FIRST}(\beta u)] \mid BL^{i-1}C, C \rightarrow \gamma \in P\}.$$

Now,

$$\delta^{i+1}\{[A \rightarrow \alpha \cdot B\beta, u]\} = \delta\{[C \rightarrow \cdot \gamma, \text{Path}_k(B, C) \oplus_k \text{FIRST}_k(\beta u)] \mid BL^{i-1}C, C \rightarrow \gamma \in P\}.$$

But, according to (3.2), property of $\oplus_k$, and definition of Path_k

$$\begin{aligned} &= \{[C' \rightarrow \cdot \gamma', \text{FIRST}(\gamma') \oplus_k \text{Path}_k(B, C) \oplus_k \text{FIRST}_k(\beta u)] \\ &\qquad \mid BL^{i-1}C, C \rightarrow \gamma \in P, \gamma = C'\beta', C' \rightarrow \gamma' \in P\}. \\ &= \{[C' \rightarrow \cdot \gamma', \text{Path}_k(B, C') \oplus_k \text{FIRST}_k(\beta u)] \mid BL^iC', C' \rightarrow \gamma' \in P\}. \end{aligned}$$

A.2. Proof of Lemma 3.3. (See also Fig. 6.). Since $q \in \text{PRED}(p, \alpha_1)$. $[A \rightarrow \cdot \alpha_1\alpha_2, x'] \in q$, where

$$x \in \{x' \mid [A \rightarrow \cdot \alpha_1\alpha_2, x'] \in q, q \in \text{PRED}(p, \alpha_1)\}.$$

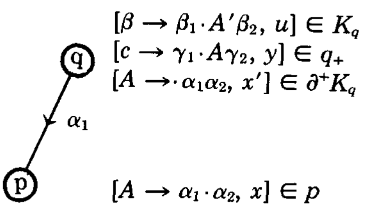
Since $A = S'$, $[A \rightarrow \cdot \alpha_1\alpha_2, x'] \in \delta^+K_q$. Hence, there must be at least *one* item $[C \rightarrow \gamma_1 \cdot A\gamma_2, y] \in q$. And

$$\begin{aligned} x &\in \{x' \mid [A \rightarrow \cdot \alpha_1\alpha_2, x'] \in \delta^+K_q, q \in \text{PRED}(p, \alpha_1)\}. \\ &= \{z \mid z \in \text{FIRST}_k(\gamma_2 y), q \in \text{PRED}(p, \alpha_1), [C \rightarrow \gamma_1 \cdot A\gamma_2, y] \in q\}, \end{aligned} \tag{A.1}$$

by Eq. (3.2).

Case 1. $[C \rightarrow \gamma_1 \cdot A\gamma_2, y] \in K_q$. Since the item $[C \rightarrow \gamma_1 \cdot A\gamma_2, y]$ is in the kernel K_q, it can be regarded as the same item as item $[B \rightarrow \beta_1 \cdot A'\beta_2, u]$ that is in the

Fig. 6. The GOTO DFA and Items.



$[\beta \rightarrow \beta_1 \cdot A'\beta_2,\ u] \in K_q$
$[c \rightarrow \gamma_1 \cdot A\gamma_2,\ y] \in q_+$
$[A \rightarrow \cdot \alpha_1\alpha_2,\ x'] \in \partial^+ K_q$
q
α_1
p
$[A \rightarrow \alpha_1 \cdot \alpha_2,\ x] \in p$

condition of Lemma 3.3 Thus, from Eq. (A.1),

$$\begin{aligned} x \in\ & \{z \mid z \in \mathrm{FIRST}_k(\gamma_2 y),\ q \in \mathrm{PRED}(p, \alpha_1),\ [C \rightarrow \gamma_1 \cdot A\gamma_2,\ y] \in K_q\}. \\ =\ & \{z \mid z \in \mathrm{FIRST}_k(\beta_2 u), \\ & \quad q \in \mathrm{PRED}(p, \beta_1),\ A'L^*A,\ [B \rightarrow \beta_1 \cdot A'\beta_2,\ y] \in K_q\}. \\ =\ & \{z \mid z \in \mathrm{Path}_k(A', A) \oplus_k \mathrm{FIRST}_k(\beta_2 u), \\ & \quad q \in \mathrm{PRED}(p, \alpha_1),\ A'L^*A,\ [B \rightarrow \beta_1 \cdot A'\beta_2,\ y] \in K_q\}, \end{aligned}$$

since $\Lambda \in \mathrm{Path}(A', A)$, and if $\mathrm{Path}(A', A)$ contains nonempty lookahead sets, it must be equivalently considered in Case 2.

Case 2. $[C \rightarrow \gamma_1 \cdot A\gamma_2,\ y] \in \delta^+ K_q \gamma_1 = \Lambda$. By Lemma 3.2, there must be at least one *kernel* item $[B \rightarrow \beta_1 \cdot A'\beta_2,\ u] \in K_q$. And, from Eq. (A.1),

$$\begin{aligned} x \in\ & \{z \mid z \in \mathrm{FIRST}_k(\gamma_2 y),\ q \in \mathrm{PRED}(p, \alpha_1),\ [C \rightarrow \cdot A\gamma_2,\ y] \in \delta^+ K_q\}. \\ =\ & \{z \mid z \in \mathrm{FIRST}_k(\gamma_2) \oplus_k \mathrm{Path}_k(A', C) \oplus_k \mathrm{FIRST}_k(\beta_2 u),\ q \in \mathrm{PRED}(p, \alpha_1), \\ & A'L^*A,\ [C \rightarrow \cdot A\gamma_2,\ y] \in \delta^+ Kq,\ [B \rightarrow \beta_1 \cdot A'\beta_2,\ u] \in K_q\}, \end{aligned}$$

according to the definition of the FIRST_k, $\oplus_k$, and Lemma 3.2,

$$\begin{aligned} =\ & \{z \mid z \in \mathrm{Path}_k(A', A) \oplus_k \mathrm{FIRST}_k(\beta_2 u), \\ & \quad q \in \mathrm{PRED}(p, \alpha_1),\ A'L^*A,\ [B \rightarrow \beta_1 \cdot A'\beta_2,\ u] \in K_q\}, \end{aligned}$$

according to the definition of Path.

A.3. Proofs of (5.2), (5.3), and (5.4). From (4.3),

$$\begin{aligned} & LA(p, [A \rightarrow \alpha_1 \cdot \alpha_2]) \\ & \qquad = \{a \mid a \in \mathrm{FIRST}(\beta_2) \oplus LA(q, [B \rightarrow \beta_1 \cdot A\beta_2]), \\ & \qquad\qquad q \in \mathrm{PRED}(p, \alpha_1),\ [B \rightarrow \beta_1 \cdot A\beta_2] \in q\}, \end{aligned}$$

which can be rewritten using (5.1) as

$$= \{a \mid a \in \mathrm{Follow}(q, A),\ q \in \mathrm{PRED}(p, \alpha_1)\}.$$

End of proof for (5.2). □

According to (5.1), for $A \in N$,

$\mathrm{Follow}(q, A)$

$$= \{a \mid a \in \mathrm{FIRST}(\beta_2) \oplus LA(q, [B \rightarrow \beta_1 \cdot A\beta_2]),\ [B \rightarrow \beta_1 \cdot A\beta_2] \in q\},$$

which can be rewritten using (4.3) as

$$\begin{aligned} =\ & \{a \mid a \in \mathrm{FIRST}(\beta_2) \oplus \mathrm{FIRST}(\gamma_2) \oplus LA(r, [C \rightarrow \gamma_1 \cdot B\gamma_2]), \\ & [B \rightarrow \beta_1 \cdot A\beta_2] \in q,\ r \in \mathrm{PRED}(q, \beta_1),\ [C \rightarrow \gamma_1 \cdot B\gamma_2] \in r\}, \end{aligned}$$

which, according to (5.1), is

$$= \{a \mid a \in \text{FIRST}(\beta_2) \oplus \text{Follow}(r, B), \\ [B \rightarrow \beta_1 \cdot A\beta_2] \in q,\ r \in \text{PRED}(q, \beta_1),\ [C \rightarrow \gamma_1 \cdot B\gamma_2] \in r\}.$$

End of proof for (5.3). □

According to (5.3), if

$$\Lambda \in \text{FIRST}(\beta_2) \quad \text{then} \quad \text{Follow}(q, A) \supseteq \text{Follow}(r, B).$$

End of proof for (5.4). □

A.4. Proofs of (5.5), (5.6), and (5.7). From (4.1),

$LA(p, [A \rightarrow \alpha_1 \cdot \alpha_2])$

$$= \{a \mid a \in \text{Path}(A', A \oplus \text{FIRST}(\beta_2) \oplus LA(q, [B \rightarrow \beta_1 \cdot A'\beta_2]), \\ q \in \text{PRED}(p, \alpha_1),\ A'L^*A,\ [B \rightarrow \beta_1 \cdot A'\beta_2] \in K_q\},$$

which can be rewritten using (5.1) as

$$= \{a \mid a \in \text{Path}(A', A) \oplus \text{Follow}(q, A'), \\ q \in \text{PRED}(p, \alpha_1),\ A'L^*A,\ [B \rightarrow \beta_1 \cdot A'\beta_2] \in K_q\}.$$

End of proof for (5.5). □

According to (5.1), for $A' \in N$,

Follow(q, A)

$$= \{a \mid a \in \text{FIRST}(\beta_2) \oplus LA(q, [B \rightarrow \beta_1 \cdot A'\beta_2]),\ [B \rightarrow \beta_1 \cdot A'\beta_2] \in K_q\},$$

which can be rewritten using (4.1) as

$$= \{a \mid a \in \text{FIRST}(\beta_2) \oplus \text{Path}(B', B) \oplus \text{FIRST}(\gamma_2) \oplus LA(r, [C \rightarrow \gamma_1 \cdot B'\gamma_2]), \\ B'L^*B,\ [B \rightarrow \beta_1 \cdot A'\beta_2] \in K_q,\ r\beta\text{PRED}(q, b_1),\ [C \rightarrow \gamma_1 \cdot B'\gamma_2] \in K_r\},$$

which, according to (5.1), is

$$= \{a \mid a \in \text{FIRST}(\beta_2) \oplus \text{Path}(B', B) \oplus \text{Follow}(r, B'),\ B'L^*B, \\ [B \rightarrow \beta_1 \cdot A'\beta_2] \in K_q,\ r \in \text{PRED}(q, \beta_1),\ [C \rightarrow \gamma_1 \cdot B'\gamma_2] \in K_r\}.$$

End of proof for (5.6). □

According to (5.6), if

$$\Lambda \in \text{FIRST}(\beta_2) \quad \text{and} \quad \Lambda \in \text{Path}(B', B), \quad \text{then} \quad \text{Follow}(q, A) \supseteq \text{Follow}(r, B).$$

End of proof for (5.7). □

REFERENCES

1. AHO, A.V. Translator writing systems: Where do they now stand? *IEEE Comput. Mag. 13*, 8 (Aug. 1980), 9–14.
2. AHO, A.V., AND ULLMAN, J.D. *Principles of Compiler Design.* Addison-Wesley, Reading, Mass., 1977.
3. AHO, A.V., AND ULLMAN, J.D. *The Theory of Parsing, Translation, and Compiling*; *vol.* 1, *Parsing*; *vol.* 2. *Compiling.* Prentice-Hall, Englewood Cliffs, N. J., 1972, 1973.

4. ANDERSON, T., EVE, J., AND HORNING, J.J. Efficient LR(1) parser. *Acta Inf. 2* (1973), 12–39.
5. DEREMER, F.L. Simple LR(k) grammars. *Commun. ACM 14*, 7 (July 1971), 453–460.
6. DEREMER, F.L., AND PENNELLO, T.J. Efficient computation of LALR(1) lookahead sets. *ACM Trans. Program. Lang. Syst. 4*, 4 (Oct. 1982), 615–649; also *ACM SIGPLAN Not. 14*, 8 (Aug. 1979), 176–187.
7. JENSEN, K., AND WIRTH, N. *Pascal User Manual and Report.* 2nd ed., Springer-Verlag, New York, 1975.
8. KNUTH, D.E. On the translation of languages from left to right. *Inf. Control 8*, 6 (Dec. 1965), 607–639.
9. KRISTENSEN, B.B., AND MADSEN, O.L. Methods for computing LALR(k) lookahead. *ACM Trans. Program. Lang. Syst. 3*, 1 (Jan. 1981), 60–82.
10. LALONDE, W.R. An efficient LALR parser generator. Tech. Rep., Computer Systems Research Group, Univ. of Toronto, 1971.
11. MCKEEMAN, W.M., HORNING, J.J., AND WORTMAN, D.B. *A Compiler Compiler.* Prentice-Hall, Englewood Cliffs, N. J., 1977.
12. PARK, J.C.H. A new LALR formalism. *ACM SIGPLAN Not. 17*, 7 (July 1982), 47–61; CSRC Tech. Rep. TR81-0002-0, Computer Science Research Center, Korea Advanced Institute of Science and Technology, Seoul, July 1981.
13. PARK, J.C.H., CHOE, K.M., CHANG, C.H., YOO, J.W., AND OH, S.M. *User's Manual for PGS82—An Efficient LALR Parser Generating System.* Computer Science Research Center, Korea Advanced Institute of Science and Technology, Seoul. In preparation.
14. PERSCH, G., WINTERSTEIN, G., DROSSOPOULOU, A., AND DAUSSMAN, M. An LALR(1) grammar for (revised) Ada. *ACM SIGPLAN Not. 16*, 3 (Mar. 1981), 85–98; see also the *Reference Manual for the ADA Programming Language*, DOD, July 1980.
15. TARJAN, R.E. Depth-first search and linear graph algorithms. *SIAM J. Comput. 1*, 2 (1972), 146–160.
16. WIRTH, N. PL360, a programming language for the 360 computers. *J. ACM 15*, 1 (Jan. 1968), 37–74.

Received August 1982; revised November 1983; accepted January 1984