Programming Languages

B. Wegbreit Editor

On the Complexity of LR(k) Testing

Harry B. Hunt III
Harvard University
Thomas G. Szymanski and Jeffrey D. Ullman
Princeton University

The problem of determining whether an arbitrary context-free grammar is a member of some easily parsed subclass of grammars such as the LR(k) grammars is considered. The time complexity of this problem is analyzed both when k is considered to be a fixed integer and when k is considered to be a parameter of the test. In the first case, it is shown that for every k there exists an $O(n^{k+2})$ algorithm for testing the LR(k) property, where n is the size of the grammar in question. On the other hand, if both k and the subject grammar are problem parameters, then the complexity of the problem depends very strongly on the representation chosen for k. More specifically, it is shown that this problem is NP-complete when k is expressed in unary. When k is expressed in binary the problem is complete for nondeterministic exponential time. These results carry over to many other parameterized classes of grammars, such as the LL(k), strong LL(k), SLR(k), LC(k), and strong LC(k) grammars.

Key Words and Phrases: computational complexity, context-free grammars, parsing, LR(k) grammars, NP-complete problems

CR Categories: 4.12, 5.23, 5.25

A version of this paper was presented at the Second ACM Symposium on Principles of Programming Languages, Palo Alto, Calif., Jan. 20–22, 1975.

The first author's work was supported by U.S. Army grant DA-31-124-ARO-D-462 and NSF grant GJ35570. The second and third authors' work was supported by NSF grant DCR74-21939.

Authors' addresses: H. B. Hunt III, Center for Research in Computing Technology, Harvard University, Cambridge, MA 02138; T.G. Szymanski and J.D. Ullman, Department of Electrical Engineering, Computer Science Laboratory, Princeton, NJ 08540.

1. Introduction

It has long been known that $LR(k)$ parsers can have a number of states which is exponential in the size of the grammar which they parse. Nevertheless, for any fixed value of k there is a polynomial time algorithm for testing whether an arbitrary grammar possesses the $LR(k)$ property. A class of $O(n^{4k+4})$ algorithms are implicit in [4], whereas a class of $O(n^{2k+2})$ algorithms are presented in [1]. In Section 2 of this paper we shall sharpen this bound by presenting a class of $O(n^{k+2})$ algorithms.

In Section 3, we shall consider lower bounds on the complexity of any uniform algorithm for $LR(k)$ testing, that is, any algorithm which accepts both k and the grammar in question as parameters. More specifically, we shall show that the problem of testing whether a grammar is $LR(k)$ is:

(1) complete for nondeterministic polynomial time if k is expressed in unary as an input to the algorithm;
(2) complete for nondeterministic exponential time if the "natural" representation of k as a binary integer is chosen.

Moreover, these results carry over to many other parameterized classes of grammars.

We shall first need some basic definitions pertaining to computational complexity.

Definition 1.1. A function $f(n)$ is said to be *polynomially bounded* (*exponentially bounded*) if there exists a polynomial $p(n)$ and integer n' such that $f(n) \leq p(n)$ ($f(n) \leq 2^{p(n)}$) for all $n > n'$.

Definition 1.2. P, NP, E, and NE are the classes of languages accepted by deterministic and nondeterministic, polynomially and exponentially, time bounded Turing machines respectively.

It is not currently known whether $P = NP$ or $E = NE$. Nevertheless certain languages called *complete* languages reflect the complexity of each entire class.

Definition 1.3. A language L_0 in NP (NE) is said to be *NP-complete*[1] (*NE-complete*) if the following condition is true: Given a deterministic polynomially (exponentially) time bounded Turing machine to recognize L_0 and a language L in NP (NE), we can effectively find a deterministic polynomially (exponentially) time bounded Turing machine to recognize L.

Definition 1.4. A language L is *polynomially transformable* to a language L_0 if there exists a function f,

[1] Although we use Cook's definition of completeness, our results are still true if Karp's definition is used instead.

computable by a polynomially time bounded Turing machine, such that $x \in L$ iff $f(x) \in L_0$.

A sufficient condition for completeness is described in the following lemma.

LEMMA 1.5. *L_0 is NP-complete (NE-complete) if* (a) *$L_0 \in NP$ (NE) and* (b) *every language $L \in NP$ (NE) is polynomially transformable to L_0.*

We shall next need some standard definitions relating to context-free grammars.

Definition 1.6. A *context-free grammar* (cfg) is a 4-tuple $G = (V, \Sigma, P, S)$, where V and Σ are finite sets called the *vocabulary* and *terminal alphabet* respectively, $\Sigma \subseteq V$, the set of *productions* P is a finite subset of $(V - \Sigma) \times V^*$, and the *start symbol* S is a member of $V - \Sigma$. Productions will be written in the form $A \to \alpha$ rather than (A, α).

We shall use the following naming conventions throughout this paper: members of Σ are denoted by $a, b, c, \ldots$; members of $V - \Sigma$ by $A, B, C, \ldots$; members of V by $\ldots, X, Y, Z$; members of Σ^* by $\ldots, x, y, z$; and members of V^* by $\alpha, \beta, \gamma, \ldots$. The empty string is denoted by λ and the length of a string α by $|\alpha|$.

Definition 1.7. Let $G = (V, \Sigma, P, S)$ be a cfg. For any production $A \to \alpha$ of P and strings β and γ we say that $\beta A\gamma$ *derives* $\beta\alpha\gamma$ (written $\beta A\gamma \Rightarrow_G \beta\alpha\gamma$). The subscript G will be omitted when the identity of the grammar to be used is clear from the context. In addition, if $\gamma \in \Sigma^*$, we say that $\beta A\gamma$ *right derives* $\beta\alpha\gamma$ (written $\beta A\gamma \Rightarrow_{\text{rm}} \beta\alpha\gamma$. An asterisk above the arrow denotes reflexive and transitive closure, that is, any finite number of applications of $\Rightarrow$ or $\Rightarrow_{\text{rm}}$.

Henceforth, we assume every cfg is *reduced*, that is, every symbol is used in some derivation of a terminal string from the start symbol.

In order to define the complexity of problems involving grammars, we need some notion of the size of a cfg.

Definition 1.8. Let $G = (V, \Sigma, P, S)$ be a cfg. The *size* of G (denoted $|G|$) is defined to be

$$\sum_{A \to \alpha \in P} |A\alpha| .$$

Thus the size of a grammar is nothing more than the number of distinct positions within the right sides of all the productions of the grammar. Notice that the initial and final positions are also counted.

Let us next review the basic concepts of $LR(k)$ parsing.

Definition 1.9. Let $G = (V, \Sigma, P, S)$ be a cfg. Then for any $\alpha \in V^*$, we define

$$FIRST_k(\alpha) = \{x \mid \alpha \Rightarrow^* xy \text{ and } |x| = k \text{ or } \alpha \Rightarrow^* x \text{ and } |x| < k\}.$$

Similarly, we define

$$EFF_k(\alpha) = \{x \mid \alpha \overset{*}{\Rightarrow}_{\text{rm}} z, x = FIRST_k(z) \text{ and the next to last step in the derivation of } z \text{ from } \alpha \text{, if it exists, is not } Az \Rightarrow_{\text{rm}} z \text{ for any nonterminal } A\}.$$

$FIRST_k$ and EFF_k may also be applied to sets of strings in the obvious fashion.

Definition 1.10. Let $G = (V, \Sigma, P, S)$ be a cfg and S' and \$ be new symbols not in V. We say that the cfg

$$G' = (V \cup \{S', \$\}, \Sigma \cup \{\$\}, \{S' \to S\$\} \cup P, S')$$

is the *augmented* grammar corresponding to G.

Henceforth we deal only with augmented grammars and reserve the symbol \$ to denote the endmarker introduced in the previous definition.

We may now define an $LR(k)$ grammar.

Definition 1.11. Let $G = (V, \Sigma, P, S)$ be a cfg, and k be a nonnegative integer. G is said to be $LR(k)$ if the three conditions

(1) $S \overset{*}{\Rightarrow}_{\text{rm}} \alpha Aw \Rightarrow_{\text{rm}} \alpha\beta w$,
(2) $S \overset{*}{\Rightarrow}_{\text{rm}} \gamma Bx \Rightarrow_{\text{rm}} \alpha\beta y$,
(3) $FIRST_k(w) = FIRST_k(y)$

imply that $\alpha Ay = \gamma Bx$.

Definition 1.12. Let $G = (V, \Sigma, P, S)$ be a cfg. If $A \to \beta_1\beta_2 \in P$ and $u \in \Sigma^* \cup \Sigma^*\$$ then we say that $[A \to \beta_1.\beta_2, u]$ is an *LR item*. If $u \in FIRST_k(\Sigma^*\$)$ then we say $[A \to \beta_1.\beta_2, u]$ is an *LR(k) item*.

If $S \overset{*}{\Rightarrow}_{\text{rm}} \alpha Aw \Rightarrow_{\text{rm}} \alpha\beta w$ then any prefix of $\alpha\beta$ is called a *viable prefix* of G. Moreover, if $\beta = \beta_1\beta_2$ and u is a prefix of w then the item $[A \to \beta_1.\beta_2, u]$ is said to be *valid* for $\alpha\beta_1$.

Note that the presence of the augmenting \$ in G allows the notion of valid item to be used without specification of a value for k. For example, in the grammar

$S \to A\$$
$A \to aAa \mid b$

the items $[A \to b., \lambda]$, $[A \to b., a]$, $[A \to b., aa]$ and $[A \to b., aa\$]$ are all valid items for the viable prefix $\gamma = aab$. However, only $[A \to b., aa]$ is a valid $LR(2)$ item for γ and only $[A \to b., aa\$]$ is a valid $LR(5)$ item for γ.

The following lemma provides an equivalent definition of an $LR(k)$ grammar.

LEMMA 1.13. [3] *Let G be a context-free grammar. Then G is not $LR(k)$ iff there exist*
(1) *a string $u \in FIRST_k(\Sigma^*\$)$,*
(2) *a viable prefix γ,*
(3) *two distinct $LR(k)$ items $I_1 = [A \to \alpha., u]$ and $I_2 = [B \to \beta_1.\beta_2, v]$ such that*
(a) *I_1 and I_2 are both valid for γ,*
(b) *$u \in EFF_k(\beta_2 v)$.*

One final piece of notation is necessary.

Definition 1.14. The collection $\mathcal{C}$ consists exactly of the following six subclasses of the context-free grammars:

(1) $\{G \mid \exists k$ such that G is $LR(k)\}$
(2) $\{G \mid \exists k$ such that G is $LC(k)\}$
(3) $\{G \mid \exists k$ such that G is $LL(k)\}$
(4) $\{G \mid \exists k$ such that G is $SLR(k)\}$
(5) $\{G \mid \exists k$ such that G is strong $LC(k)\}$
(6) $\{G \mid \exists k$ such that G is strong $LL(k)\}$

Definitions of these grammar classes may be found in [3].

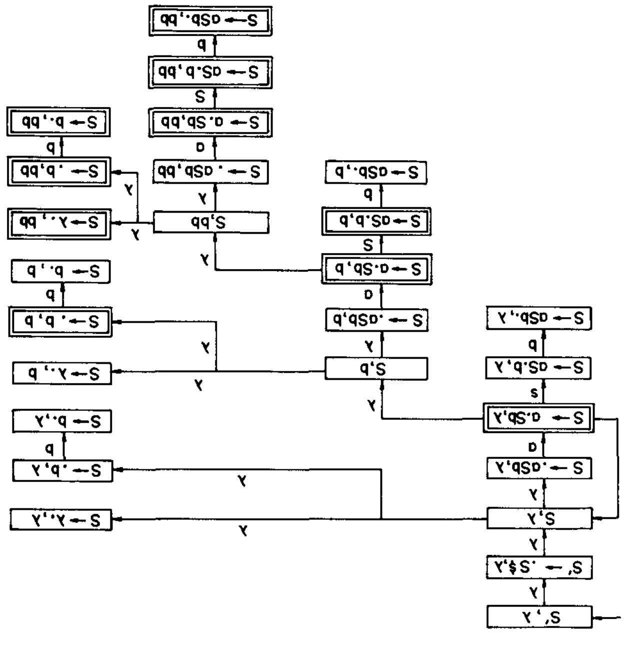
If C denotes some class in $\mathcal{C}$, then we may parameterize C to denote an appropriate subclass of C. For example, if C denotes the LL grammars, $C(3)$ denotes the $LL(3)$ grammars.

2. Upper Bounds

In this section we shall consider the problems of testing whether a grammar is $LR(k)$ both when k is fixed in advance and when k is a parameter of the decision procedure. The bounds which we obtain represent a marked improvement over those reported in [1].

Most of the results in this section depend on the efficient construction of a nondeterministic finite state automaton $M(G, u)$ which accepts those viable prefixes of G which may be followed by the terminal string u in some right sentential form. The state set of $M(G, u)$ will contain those LR items for G whose lookahead components are suffixes of u. Examination of $\delta(q_0, \gamma)$, where γ is any viable prefix of G, will thus reveal whether any LR conflict involving the lookahead string u occurs on γ.

Fig. 1.



The construction of $M(G, u)$ resembles the standard construction of an $LR(k)$ parser for G except we rely on nondeterminism to "discard" those portions of the lookahead string being accumulated which will never form a part of u.

Definition 2.1. Let $G = (V, \Sigma, P, S)$ be a cfg. Let $u \in \Sigma^* \cup \Sigma^*\$$. We define $M(G, u)$ to be the nondeterministic finite state automaton (Q, V, δ, q_0, F) in which

(1) Let

$$Q_1 = \{[X, v] \mid X \in V \text{ and } v \in SUFFIX(u)\},$$

$$Q_2 = \{[A \to \beta_1.\beta_2, v] \mid A \to \beta_1\beta_2 \in P \text{ and } v \in SUFFIX(u)\},$$

where $SUFFIX(u)$ is the set of suffixes of u. Then $Q = Q_1 \cup Q_2$.

(2) Define

(a) $\delta([A, z], \lambda) = \{[A \to .\beta, z] \mid A \to \beta \in P]\}$
(b) $\delta([A \to \beta_1.X\beta_2, z], X) = \{[A \to \beta_1 X.\beta_2, z]\}$
(c) $\delta([A \to \beta_1.X\beta_2, z], \lambda)$

$$= \{[X, wz] \in Q_2 \mid \beta_2 \Rightarrow^* w\} \cup \{[X, w] \in Q_2 \mid z = \lambda, w \neq \lambda \text{ and } w \in FIRST_{|w|}(\beta_2)\} \cup$$

(d) $\{[X, \lambda] \mid z = \lambda\}$

Call these type (a), (b), (c), and (d) transitions, respectively.

(3) Let $q_0 = [S, \lambda]$.

(4) Let

$$F_1 = \{[A \to \beta., u] \mid A \to \beta \in P\},$$
$$F_2 = \{[A \to \beta_1.\beta_2, v] \mid \beta_2 \neq \lambda,\ u \in EFF_{|u|}(\beta_2 v)\}.$$

Then $F = F_1 \cup F_2$.

Explained in terms of the standard $LR(k)$ parser construction, transitions of types (a) and (c) represent the operation of "closing" an $LR(k)$ item, whereas type (b) transitions represent a "shift" move. Type (d) transitions allow us to ignore any accumulated lookahead string until reaching a point at which we nondeterministically start to accumulate portions of u.

Example 2.2. Consider the grammar

$\bar{G}$: $S' \to S\$$
$\quad S \to aSb \mid b \mid \lambda$

$M(\bar{G}, bb)$ is shown in Fig. 1 with final states "double outlined."

Next we show that the set of states reachable from q_0 upon reading γ is the set of valid LR items for γ whose second components begin with suffixes of u.

LEMMA 2.3. *Let $G = (V,\Sigma, P, S)$ be a cfg, $u \in \Sigma^* \cup \Sigma^*\$$, $\gamma \in V^*$. Let $M(G, u)$ be as in Definition* 2.1. *Then*

$Q_2 \cap \delta(q_0, \gamma) = \{[B \to \beta_1.\beta_2, v] \in Q_2 \mid [B \to \beta_1.\beta_2, v]$ *is valid for* $\gamma\}$.

The proof appears in Appendix A.

Thus $M(G, u)$ is a representation of those viable prefixes of G which may be followed by u in a right sentential form. Moreover, this representation is very compact, as we show in the following lemma.

LEMMA 2.4. *Let G be a cfg and $u \in \Sigma^* \cup \Sigma^*\$$. Let $M(G, u) = (Q, V, \delta, q_0, F)$ be as in Definition 2.1, and let $k = |u|$ and $n = |G|$. Then*
(1) $|Q| \leq 2\cdot(k+1)\cdot n$ and
(2) $|\delta| \leq 4\cdot(k+1)^2\cdot n$.

PROOF. It is clear that

$|Q_1| \leq |V - \Sigma| \cdot (k+1) \leq n\cdot(k+1)$ and
$|Q_2| = n\cdot(k+1)$.

This establishes statement (1). Consider next the number of possible transitions of M. For each production of G there are exactly $k+1$ transitions of type (a), hence there are exactly

$|P|\cdot(k+1) \leq n\cdot(k+1)$

type (a) transitions. For each position within a production (except a final position such as $A \to \alpha.$) we have exactly $k+1$ transitions of type (b) and one of type (d) so the number of type (b) transitions is bounded by $n\cdot(k+1)$ and the number of type (d) transitions is bounded by n. Finally, observe that each state of Q_2 can have at most $k+1$ transitions of type (c) defined for it. Hence the total number of type (c) transitions is bounded by

$|Q_2|\cdot(k+1) = n\cdot(k+1)^2$.

Combining the bounds on the number of each type of transition yields an upper bound on the size of δ of

$n\cdot(k+1) + n\cdot(k+1) + n\cdot(k+1)^2 + n$,

which is clearly less than or equal to $4\cdot(k+1)^2\cdot$n. □

Let us next consider those viable prefixes of G which exhibit an LR conflict involving a given lookahead string u.

Definition 2.5. Let $G = (V, \Sigma, P, S)$ be a cfg and $u \in \Sigma^* \cup \Sigma^*\$$. Let $M(G, u)$ be as in Definition 2.1. We define

$$CONF(G, u) = \{\gamma \in V^* \mid \delta(q_0, \gamma) \text{ contains two or more states of } F_1\} \cup \{\gamma \in V^* \mid \delta(q_0, \gamma) \text{ contains at least one state from each of } F_1 \text{ and } F_2\}.$$

Since a grammar is $LR(k)$ iff no conflicts occur for any lookahead string, we see that $CONF(G, u)$ must be empty for all u of length k iff G is $LR(k)$.

LEMMA 2.6. *Let $G = (V, \Sigma, P, S)$ be a cfg and k be a non-negative integer. Then G is not $LR(k)$ iff there exists some $u \in FIRST_k(\Sigma^*\$)$ for which $CONF(G, u) \neq \varnothing$.*

PROOF. Suppose that G is not $LR(k)$. By Lemma 1.13, there exist
(a) some $u \in FIRST_k(\Sigma^*\$)$,
(b) a viable prefix γ,
(c) two distinct $LR(k)$ items $I_1 = [A \to \alpha., u]$ and $I_2 = [B \to \beta_1.\beta_2, v]$ such that
(d) I_1 and I_2 are both valid for γ and
(e) $u \in EFF_k(\beta_2 v)$.

Let $M(G, u)$ be as in Definition 2.1. Then by Lemma 2.3, $I_1 \in F_1 \cap \delta(q_0, \gamma)$. Two cases now arise, depending on whether β_2 is the empty string.

Case 1. $\beta_2 = \lambda$. Then $u = v$ and $I_2 = [B \to \beta_1., u]$. By Lemma 2.3, $I_2 \in F_1 \cap \delta(q_0, \gamma)$. Hence $\gamma \in CONF(G, u)$ and $CONF(G, u) \neq \varnothing$.

Case 2. $\beta_2 \neq \lambda$. Since $u \in EFF_k(\beta_2 v)$ we can write $v = v_1 v_2$ such that $u \in EFF_k(\beta_2 v_1)$. Thus v_1 is a (possibly null) suffix of u and $[B \to \beta_1.\beta_2, v_1]$ is valid for γ. Thus by Lemma 2.3, $I_2 = [B \to \beta_1.\beta_2, v_1] \in \delta(q_0, \gamma)$. Moreover $I_2 \in F_2$. Thus $\gamma \in CONF(G, u)$ and $CONF(G, u) \neq \varnothing$.

This completes the "only if" portion of the proof.

Suppose next that for some $u \in FIRST_k(\Sigma^*\$)$, $CONF(G, u) \neq \varnothing$. Let γ be any string in $CONF(G, u)$. Two cases arise, depending on the reason why $\gamma \in CONF(G, u)$.

Case 1. $\delta(q_0, \gamma)$ contains two members of F_1, say $[A \to \alpha., u]$ and $[B \to \beta., u]$. By Lemma 2.3, both $[A \to \alpha., u]$ and $[B \to \beta., u]$ are valid for γ. Since $u \in FIRST_k(\Sigma^*\$)$, these items are valid $LR(k)$ items for γ and thus by Lemma 1.13, G is not $LR(k)$.

Case 2. $\delta(q_0, \gamma)$ contains a state of F_1, say $[A \to \alpha., u]$ and a state of F_2, say $[B \to \beta_1.\beta_2, v]$. By Lemma 2.3 and the fact that $u \in FIRST_k(\Sigma^*\$)$, $[A \to \alpha., u]$ is a valid $LR(k)$ item for γ. By Lemma 2.3, $[B \to \beta_1.\beta_2, v]$ is a valid $LR(|v|)$ item for γ and accordingly there exists some string $\bar{v}$ (possibly null) such that $[B \to \beta_1.\beta_2, v\bar{v}]$ is a valid $LR(k)$ item for γ. By definition of F_2, $u \in EFF_k(\beta_2 v)$ and therefore u is also in $EFF_k(\beta_2 v\bar{v})$. Thus, we may apply Lemma 1.13 to conclude that G is not $LR(k)$.

This completes the "if" portion of the proof. □

Referring again to Example 2.2, we observe that for the automaton $M(\bar{G}, bb)$, $\delta(q_0, aa)$ contains two final states, namely $[S \to \lambda., bb]$ and $[S \to .b, bb]$. Thus $CONF(\bar{G}, aa)$ is not empty and G is not $LR(2)$.

COROLLARY 2.7. *If a cfg G of size n is not $LR(k)$, then there exists a viable prefix of G of length $\leq 2\cdot(k+1)\cdot n^2$ which exhibits an $LR(k)$ conflict.*

PROOF. By Lemma 2.6, there must exist some $u \in FIRST_k(\Sigma^*\$)$ such that $CONF(G, u)$ is nonempty. Let γ be the shortest string in $CONF(G, u)$.

Consider any two computations of M on γ[2] which accept γ by different final states. The standard "re-

[2] A *computation* of M on string α is a sequence $S_0X_1S_1X_2\cdots S_{m-1}X_mS_m$ such that (1) each $S_i \in Q$; (2) each $X_i \in V \cup \{\lambda\}$ and $\alpha = X_1X_2\cdots X_m$; (3) $S_0 = q_0$; and (4) $S_{i+1} \in \delta(S_i, X_i)$.

peated pairs of corresponding states" argument suffices to show that $| \gamma | \leq | Q |^2 = O(k^2 n^2)$.

The sharper bound mentioned in the statement of this corollary may be achieved by noting that the length of the lookahead component of the states in any computation must be monotonically nondecreasing. Since the construction of $M(G, u)$ implies there is at most one lookahead string of any given length, γ can be partitioned into at most $2(k + 1)$ segments in which the lookahead component of the states of both computations remains constant. In any computation, at least one state of Q_2 must appear between any two grammar symbols. Thus no segment (as defined above) of γ can be as long as $| Q_2 |^2 = n^2$ or else there would exist a member of $CONF(G, u)$ which is shorter than γ. Hence $| \gamma | < 2(k + 1) \cdot n^2$. $\square$

Before presenting the fastest known algorithm for $LR(k)$ testing we must first consider the complexity of testing $CONF(G, u)$ for emptiness.

Lemma 2.8. *Let G be a cfg of size n and k be an integer. Let $u \in FIRST_k(\Sigma^*\$)$. Then $CONF(G, u)$ may be tested for emptiness in*
(a) *deterministic time $O(k + 1)^3 \cdot n^2)$,*
(b) *nondeterministic time $O((k + 1) \cdot n^2)$.*

Proof of (a). To perform the test deterministically, we construct $M(G, u)$ and then determine all pairs of states which are "mutually accessible," i.e. both members of $\delta(q_0, \gamma)$ for some γ.

It should be obvious that Q may be constructed in time $O((k + 1) \cdot n)$ and that all transitions of types (a), (b), and (d) may be computed in $O((k + 1) \cdot n)$ steps. Determining the set of type (c) transitions and calculating which members of Q are final states is a bit more complex. In order to do these we need to know for each suffix β of a right side of a production, exactly which subwords of u are derivable from β.

To do this, a grammar $\bar{G} = (V, \Sigma, \bar{P}, \bar{S})$ is constructed in which

$$\bar{G} = V \cup \{[A \to \beta_1.\beta_2] \mid A \to \beta_1\beta_2 \in P\}$$

and

$$\bar{P} = P \cup \{[A \to \beta_1.X\beta_2] \to X[A \to \beta_1X.\beta_2] \mid A \to \beta_1 X\beta_2 \in P\} \cup \{[A \to \beta.] \to \lambda \mid A \to \beta \in P\}.$$

Notice that $| \bar{G} | = O(| G |)$ and that $[A \to \beta_1.\beta_2] \Rightarrow_{\bar{G}}^* w$ iff $\beta_2 \Rightarrow_G^* w$.

G can be converted to 2-normal form (i.e. the right side of any production is of length 0, 1, or 2) without altering its size or deleting any nonterminals. If we now apply Younger's recognition algorithm [7] to $\bar{G}$ and $u = a_1 \cdots a_k$, we can compute a table T such that

$$T(i, j) = \{[A \to \beta_1.\beta_2] \mid \beta_2 \Rightarrow_G^* a_i \cdots a_j\}.$$

Construction of this table takes at most $O(| u |^3 \cdot | \bar{G} |) = O(k^3 n)$ steps. This table then makes it trivial to compute which type (c) transitions are to be included in δ.

Notice that a simpler procedure may be used in the special case $k = 0$. In any case, $O((k + 1)^3 \cdot n)$ steps suffice to compute the set of allowed type (c) transitions.

A similar table and construction may be used to calculate F. Thus $M(G, u)$ may be constructed deterministically in time $O((k + 1)^3 \cdot n)$.

We shall test $CONF(G, u)$ for emptiness by constructing a set

$$S = \{(s, t) \in Q \times Q \mid \exists \alpha \text{ such that } s, t \in \delta(q_0, \alpha)\}.$$

S represents the set of pairs of states which are mutually reachable by some input string. $CONF(G, u)$ is empty iff no such distinct pair is a member of $F_1 \times F_1$ or $F_1 \times F_2$.

The set S will be stored as a $| Q | \times | Q |$ bit matrix. We will also maintain a stack $STACK$ of backlogged members of S. The algorithm for computing S appears in Figure 2.

It should be clear that the algorithm computes S. Let us consider the amount of work performed by each of the numbered statements.

Since no pair of states is ever stacked twice, statements 1, 2, and 5 require a maximum of $O(| Q |^2) = O((k + 1)^2 \cdot n^2)$ work. In the worst case statements 3 and 4 will be executed for every pair (p, q) of $Q \times Q$. The amount of work done in statement 3 will then be $(\sum_{p \in Q} | \delta(p, \lambda) |) \cdot | Q |$ representing one traversal of the $\delta(p, \lambda)$ list for each p and q. However,

$$\sum_{p \in Q} | \delta(p, \lambda) | = \sum_{p \in Q_1} | \delta(p, \lambda) | + \sum_{p \in Q_2} | \delta(p, \lambda) | = O((k + 1) \cdot n) + O((k + 1)^2 \cdot n) = O((k + 1)^2 \cdot n).$$

as explained in the proof of Lemma 2.4. Hence the work charged to statement 3 (and similarly to statement 4) is $O((k + 1)^3 \cdot n^2)$.

Fig. 2.

```
procedure INSERT(p, q);
  if (p, q) ∉ S then
    begin
      S ← S ∪ {(p, q)};
      add (p, q) to STACK
    end
end INSERT;
begin
  S ← ∅;
  INSERT(q_0, q_0);
1. while STACK not empty do
     begin
2.     pop (p, q) from STACK;
3.     ∀q' ∈ δ(p, λ), INSERT(p, q');
4.     ∀p' ∈ δ(q, λ), INSERT(p', q);
5.     if p = [A → α_1.Xα_2, v] and q = [B → β_1.Yβ_2, w] and X = Y
         then INSERT(δ(p, X), δ(q, X));
     end
end
```

PROOF OF (b). In order to test $CONF(G, u) \neq \varnothing$ nondeterministically, we simply guess a string γ and test whether $\gamma \in CONF(G, u)$. By Corollary 2.7, we may assume that $|\gamma| \leq 2(k + 1)n^2$. We must next verify that γ supports two accepting computations with distinct final states. In order to check this condition, we need only construct those portions of M which are actually used in either accepting computation. Observe that any computation of M on α contains at most k transitions of type (c), $|\alpha|$ transitions of type (b), and as many as $n \cdot |\alpha|$ transitions of types (a) and (d).

Recall that it is possible to test nondeterministically whether $X \Rightarrow_G^* z$ in time $O(|z| \cdot |G|) = O(|z| \cdot n)$. Since the sum of the lengths of the lookahead components contributed by each type (c) transition in an accepting computation is exactly k, we may guess which type (c) transitions we shall need, using a total time of $O(kn)$.

Finally, we may nondeterministically bypass any portion of the computation which consists only of type (a) and (d) transitions. Each such bypass requires unit time if we have precomputed the set $\{(X, Y) \mid X \Rightarrow^* Y\}$ (which may be constructed in time $O(n^2)$).

Thus a representation of an accepting computation of α can be constructed in time

$$O(|\alpha|) = O((k + 1) \cdot n^2). \qquad \square$$

We now present the first major result of this section.

THEOREM 2.9. *An arbitrary cfg of size n may be tested for the LR(k) property in*
(a) *nondeterministic time* $O((k + 1) \cdot n^2)$,
(b) *deterministic time* $O((k + 1)^3 \cdot n^{k+2})$.

PROOF. In the nondeterministic case, we actually accept those grammars G which are *not* $LR(k)$. We do this by guessing an element u of $FIRST_k(\Sigma^*\$)$ and then testing $CONF(G, u)$ for emptiness, as explained in Lemma 2.8.

For the deterministic test, we test $CONF(G, u)$ for emptiness for each string u in $FIRST_k(\Sigma^*\$)$. Since there are at most n^k such strings and each test of $CONF(G, u)$ for emptiness can be performed deterministically in time $O((k + 1)^3 \cdot n^2)$ we can perform the entire $LR(k)$ test in time $O((k + 1)^3 \cdot n^{k+2})$. $\square$

A special case of interest arises when we choose k to be a fixed integer.

COROLLARY 2.10. *For every k there exists a deterministic algorithm for testing the LR(k) property whose running time is a polynomial function of the size of the grammar being tested.*

Thus $LR(0)$ testing can be done in time $O(n^2)$, $LR(1)$ testing in time $O(n^3)$ and so forth. Moreover, these same techniques can be applied to most other parameterized classes of grammars.

Recall that a grammar is $SLR(k)$ if and only if for every pair of $LR(0)$ items, $[A \to \alpha., \lambda]$ and $[B \to \beta_1.\beta_2, \lambda]$ which are both valid for some viable prefix γ, $FOLLOW_k(A)$ and $EFF_k(\beta_2 \circ FOLLOW_k(B))$ are disjoint. $FOLLOW_k(A)$ is $FIRST_k$ of the set of terminal strings which can follow A in a derived string of the grammar at hand. (The reader is referred to [3] for a more detailed definition and development.)

We may therefore proceed to test the $SLR(k)$ property in the following fashion. For every $u \in FIRST_k(\Sigma^*\$)$ construct $M(G, u)$ and determine which final states are accessible from the start state. This takes a total time of $O(n^{k+1})$ because there are $O(n^k)$ such strings u, the construction of machine $M(G, u)$ takes time $O((k + 1)^3 n)$ as shown in the proof of part (a) of Lemma 2.8, and finding the accessible states of each machine takes time $O((k + 1)n)$ by the standard graph exploration techniques.

Observe that $[A \to \alpha., u]$ is an accessible final state of $M(G, u)$ if and only if $u \in FOLLOW_k(A)$. Moreover, there exists a v such that $[B \to \beta_1.\beta_2, v]$ is an accessable final state of $M(G, u)$ if and only if $u \in EFF_k(\beta_2 \circ FOLLOW_k(B))$.

So now to complete the $SLR(k)$ test, we compute all item pairs $[A \to \alpha., \lambda]$ and $[B \to \beta_1.\beta_2, \lambda]$ which are both valid for some γ and test whether for some $u \in FIRST_k(\Sigma^*\$)$ and v a suffix of u, both $[A \to \alpha., u]$ and $[B \to \beta_1.\beta_2, v]$ are accessible final states of $M(G, u)$. This takes $O(n^2 \cdot n^k \cdot (k + 1))$ time. Thus the entire SLR (k)test requires $O(n^{k+2})$ time.

The methods of Brosgol [6] may be used to reduce $LL(k)$ and $LC(k)$ testing to $LR(k)$ testing and strong $LL(k)$ and strong $LC(k)$ testing to $SLR(k)$ testing. For example, a grammar of size n is $LL(k)$ if and only if an easily constructed grammar of size $2n$ is $LR(k)$. Thus all of $LL(k)$, $LC(k)$, strong $LL(k)$, and strong $LC(k)$ testing can be performed in $O(n^{k+2})$ time.

The results of the preceeding paragraphs may be stated generically.

THEOREM 2.11. *For every C in $\mathfrak{C}$ and integer k there exists a deterministic algorithm for testing the C(k) property. Moreover, the running time of the algorithm is polynomial in the size of the grammar being tested.*

It is important to note that this theorem fixes k *before* selecting the decision procedure. That is, it establishes the existence of separate (and distinct) polynomial time algorithms for $C(0)$ testing, $C(1)$ testing, etc.

The reader should be aware that fast algorithms have been previously noted for certain special cases of practical interest. For instance, Johnson and Sethi [9] gives an $O(n^2)$ algorithm for $LL(1)$ testing and the methods of Hunt, Szymanski, and Ullman [1] may easily be adapted to yield $O(n^2)$ algorithms for $LL(1)$ and $SLR(1)$ testing.

We can also apply Theorem 2.9 to the case in which we have a single decision procedure which takes both k and a grammar as inputs.

THEOREM 2.12. *For each class of grammars C in $\mathcal{C}$, there exists a nondeterministic algorithm A such that* (1) *input to A consists of a context-free grammar G and an integer k,* (2) *A accepts only those inputs G and k such that G is not $C(k)$, and* (3) *A's running time is polynomial in both $|G|$ and k.*

3. Lower Bounds for Free k

Hartmanis [2] has shown that arbitrary Turing machine computations may be embedded in context-free languages. We shall exploit this idea to embed nondeterministic time bounded computations in context-free languages. The grammars we produce will turn out to be $C(k)$ if and only if the embedded Turing machine does not accept its input. The parameter k will roughly correspond to the square of the time bound on the Turing machine in question. Thus a general algorithm for performing $C(k)$ testing with both k and the subject grammar G as parameters must be at least as powerful as any nondeterministic Turing machine which performs at most $k^{1/2}$ steps. More specifically, let us select some "natural" encoding of context free grammars as strings of zeros and ones.

Definition 3.1. Let C be any class in $\mathcal{C}$.
(a) $L_{c,u} = \{g \# 1^k \mid g \in \{0, 1\}^+, k \geq 0$, and g is the encoding of a grammar which is not $C(k)\}$.
(b) $L_{c,b} = \{g \# w \mid g, w \in \{0, 1\}^+$ and g is the encoding of a non-$C(k)$ grammar where k is now that integer whose binary representation is $w\}$.

The subscripts u and b of course stand for "unary" and "binary" respectively. We shall show that $L_{c,u}$ is NP-complete and $L_{c,b}$ is NE-complete.

Our chief tool in establishing these results is provided by the following technical lemma. The reader should note that the lemma also applies to the $LALR$ grammars which are *not* a member of our collection $\mathcal{C}$.

LEMMA 3.2. *Let $T(n)$ be any time-constructable function. Let M be a nondeterministic Turing machine subject to the following constraints:*
(a) *M has one semi-infinite tape and only one accepting state.*
(b) *M never "falls off" the left end of its tape.*
(c) *M never performs more than $T(n)$ steps on input of length n.*
(d) *M only enters an accepting state when at the extreme right end of the utilized portion of its tape.*
(e) *M can make no move out of an accepting state.*

Let Σ be the input alphabet of M. Let $x \in \Sigma^*$ and $k \geq 8 \cdot [T(|x|)]^2$. Then in time $O(|x|)$ we can find a grammar G of size $c + |x|$ where c depends only on M, such that the following statements are equivalent.

(1) *M accepts x.*
(2) *G is ambiguous.*
(3) *G is not $LR(k)$.*
(4) *G is not $LC(k)$.*
(5) *G is not $LL(k)$.*
(6) *G is not $LALR(k)$.*
(7) *G is not $SLR(k)$.*
(8) *G is not strong $LC(k)$.*
(9) *G is not strong $LL(k)$.*

PROOF. Let Γ be the tape alphabet of M, Q be the state set of M, q_f be M's sole accepting final state, and q_0 be M's initial state. Let ƀ be the blank tape symbol for M. We shall first establish the equivalence of clauses (1)–(9) by showing how G is constructed.

The details of this construction are important because we shall latter claim that under certain conditions, G belongs to many different subclasses of the context-free grammars. Accordingly we explicitly list all productions of G.

Let $G = (N \cup \Sigma, \Sigma, P, S)$ where

$$N = \{S, A, B, C, D, E, \bar{A}, \bar{B}, \bar{C}, \bar{D}, \bar{E}, \bar{F}\} \cup \Gamma \times Q \times \Gamma \cup Q \times \Gamma,$$

$$\Sigma = \Gamma \cup Q \cup \{\#, \$\}.$$

P contains the productions shown in Figure 3.

The two S-productions divide the grammar into two "halves" which will turn out to be easily distinguishable iff M rejects x. The nonterminal C produces all strings of the form $y \# z^{\text{rev}}$ in which y and z are configurations of M and $y \vdash_M z$. Since B produces all strings in $(C\#)^* \$$ we see that

$$L(B) = \{y \# z^{\text{rev}} \# \mid y \text{ and } z \text{ are configurations of } M \text{ and } y \vdash_M z\}^* \circ \{\$\}.$$

On the other hand, since $\bar{D}$ produces strings $y \# y^{\text{rev}}$, where y is a configuration of M, and $\bar{C}$ pro-

Fig. 3.

$S \to AB \mid \bar{A}\bar{B}$		
$A \to \lambda$		
$\bar{A} \to \lambda$		
$B \to C \# B \mid \$$		
$C \to q_i bE \langle q_i b \rangle$		$\forall q_i \in Q, b \in \Gamma$
$C \to D$		
$D \to aDa$		$\forall a \in \Gamma$
$D \to aq_i bE \langle aq_i b \rangle$		$\forall a, b \in \Gamma, q_i \in Q$
$D \to aq_i \# \langle aq_i \text{ƀ} \rangle$		$\forall a \in \Gamma, q_i \in Q$
$E \to aEa \mid \#$		$\forall a \in \Gamma$
$\langle aq_i b \rangle \to q_j ac$	whenever	$aq_i b \vdash_M q_j ac$
$\langle aq_i b \rangle \to aq_j c$	whenever	$aq_i b \vdash_M aq_j c$
$\langle aq_i b \rangle \to acq_j$	whenever	$aq_i b \vdash_M acq_j$
$\langle q_i b \rangle \to q_j c$	whenever	$q_i b \vdash_M q_j c$
$\langle q_i b \rangle \to cq_j$	whenever	$q_i b \vdash_M cq_j$
$\bar{B} \to q_0 x \# \bar{C}$		
$\bar{C} \to \bar{D} \# \bar{C} \mid q_f \bar{F}$		
$\bar{D} \to a\bar{D}a$		$\forall a \in \Gamma$
$\bar{D} \to q_i \bar{E} q_i$		$\forall q_i \in Q - \{q_f\}$
$\bar{E} \to a\bar{E}a \mid \#$		$\forall a \in \Gamma$
$\bar{F} \to a\bar{F} \mid \#\$$		$\forall a \in \Gamma$

duces $(\bar{D}\ \#)^* z^{\text{rev}}$, where z is an accepting configuration of M, we see that

$$L(\bar{B}) = \{\bar{x} \text{ is the initial i.d. of } M \text{ on } x\} \circ \{y^{\text{rev}} \# y\# \mid y \text{ is a nonaccepting configuration of } M\}^* \circ \{z^{\text{rev}}\# \$ \mid z \text{ is an accepting configuration of } M\}.$$

Clearly then,

$$L(B) \cap L(\bar{B}) = \{w_1 \# w_2 \# \cdots \# w_{2n} \# \$ \mid n \leq T(|x|),\ w_1 = q_0 x \vdash w_3 \vdash w_5 \vdash \cdots \vdash w_{2n-1} \vdash w_{2n}^{\text{rev}},\ w_{2n} \text{ is an accepting configuration, and } w_{2i-1} = w_{2i}^{\text{rev}} \text{ for } 1 \leq i < n\}.$$

Thus M accepts x if and only if $L(B) \cap L(\bar{B}) \neq \varnothing$. Clearly then, if M accepts x, G is ambiguous and certainly not a member of any of the grammar classes listed in the statement of the lemma. Thus (1) implies (2) through (9).

By similar reasoning, we note that $L(B)$ and $L(\bar{B})$ can have no common prefix longer than $2 \cdot T(|x|) \cdot (T|x| + 2) + 1 < 8 \cdot [T(|x|)]^2$

Thus M accepts x iff

(*) $FIRST_k(B) \cap FIRST_k(\bar{B}) \neq \varnothing$.

Observe that for any $Y \neq S$, if $Y \to \alpha$ and $Y \to \beta$ are distinct productions then

$$FIRST_k(\alpha\ FOLLOW_k(Y)) \cap FIRST_k(\beta\ FOLLOW_k(Y)) = \varnothing.$$

Thus G is strong $LL(k)$ if and only if

$$FIRST_k(AB\ FOLLOW_k(S)) \cap FIRST_k(\bar{A}\bar{B}\ FOLLOW_k(S)) = \varnothing$$

if and only if

$$FIRST_k(B) \cap FIRST_k(\bar{B}) = \varnothing.$$

Inspection of the canonical collection of sets of $LR(0)$ items for G reveals that the only possible parsing action conflict which can arise occurs when attempting to reduce both $A \to \lambda$ and $\bar{A} \to \lambda$. An SLR parser can resolve this conflict if and only if $FOLLOW_k(A) \cap FOLLOW_k(\bar{A}) = \varnothing$, that is, if and only if $FIRST_k(B) \cap FIRST_k(\bar{B}) = \varnothing$.

Finally, we note that if an attempt is made to construct the strong $LC(k)$ parsing machine [8] for G, the only possible source of nondeterminism involves machine rules of the form

$(S, w) \to (S, A)$ for $w \in FIRST_k(B)$

and

$(S, w) \to (S, \bar{A})$ for $w \in FIRST_k(\bar{B})$.

Thus G is strong $LC(k)$ if and only if

$$FIRST_k(B) \cap FIRST_k(\bar{B}) = \varnothing.$$

Appealing to the standard inclusion properties for classes of grammars, we have established that $FIRST_k(B) \cap FIRST_k(\bar{B}) = \varnothing$ implies that G is $SLL(k)$, $SLC(k)$, $SLR(k)$, $LALR(k)$, $LL(k)$, $LC(k)$, $LR(k)$ and unambiguous. By (*), we thus have (2) through (9) imply (1). □

We may now state the main result of this section.

THEOREM 3.3. *For each class of grammars C in $\mathcal{C}$, (a) $L_{c,b}$ is NE-complete, (b) $L_{c,u}$ is NP-complete.*

PROOF of (a). We shall first show that $L_{c,b} \in NE$. Recall that Theorem 2.12 has established the existence of an algorithm A which accepts those pairs of grammars and integers (G, k) such that G is not $C(k)$. Moreover, there exists some polynomial p such that A uses nondeterministic time $p(|G| + k)$.

Any string $w \in L_{c,b}$ must be of the form $g \# \bar{k}$ where g is the encoding of some cfg G and $\bar{k}$ is the binary representation of some integer k. Checking that w does indeed have the proper form is an easy task. We may then use A to test whether G is not $C(k)$ (and hence whether $w \in L_{c,b}$) in time

$$p(|G| + k) \leq p(|w| + 2^{|\bar{k}|}) \leq p(|w| + 2^{|w|}) \leq 2^{d|w|}$$

for some constant d. Thus $L_{c,b}$ is accepted by a nondeterministic exponentially time-bounded Turing machine and hence is a member of NE.

Let M be any nondeterministic $2^{p(n)}$ time bounded Turing machine where $p(n)$ is a polynomial. Let L be the language accepted by M. Let $x \in \Sigma^+$ and $\hat{x}$ be the string $g(M, x) \# 10^k$ where $g(M, x)$ is the encoding of the grammar $G(M, x)$ described in Lemma 3.2 and $k = 2 \cdot p(|x|) + 3$. Certainly $\hat{x}$ may be deterministically produced from x in time polynomial in the length of x. Moreover, Lemma 3.2 and Definition 3.1 guarantee that $x \in L$ if and only if $\hat{x} \in L_{c,b}$. Thus L is polynomially transformable to $L_{c,b}$ and we conclude that $L_{c,b}$ is NE-complete.

PROOF OF (b). The details of the proof of part (b) of the theorem are analogous to those of part (a) and are left to the reader. □

A curious anomaly exists in the case of $LALR$ testing. The previous theorem certainly suffices to show that $L_{LALR,u}$ and $L_{LALR,b}$ are hard to recognize. However, Theorem 2.12 does not yield corresponding upper bounds for recognizing these languages. Accordingly, we note

COROLLARY 3.4. (a) *All languages in NE are polynomially transformable to $L_{LALR,b}$.* (b) *All languages in NP are polynomially transformable to $L_{LALR,u}$.*

The best-known algorithms for testing the $LALR(k)$ property essentially involve constructing the canonical collection of $LR(0)$ item sets for the grammar in question. This method gives an exponential time algorithm for recognizing $L_{LALR,u}$ and a two-level exponential (i.e. 2^{2^n}) time algorithm for recognizing $L_{LALR,b}$.

We leave open the problem of tightening these bounds.

4. Conclusion

We have investigated the complexity of testing context-free grammars for the $LR(k)$, $LC(k)$, $LL(k)$, $SLR(k)$, strong $LC(k)$, and strong $LL(k)$ properties.

Let us refer generically to all of the classes by using the notation $C(k)$.

We have shown that the running time of any uniform algorithm for testing the $C(k)$ property (i.e. any algorithm taking both the grammar and k as parameters) depends very strongly on the representation chosen for k. Even in the case where k is expressed in the most space consuming natural manner (i.e. in unary), the problem of $C(k)$ testing is NP-complete. We thus have provided strong evidence that no deterministic polynomially time-bounded algorithm exists for uniform $C(k)$ testing.

We have shown how to construct, for each fixed value of k, an efficient algorithm for $C(k)$ testing. In those cases which are of practical importance, for instance the $SLR(1)$ grammars, our methods yield low degree polynomial time algorithms.

We leave open the problems of
(1) determining lower bounds on $C(k)$ testing for fixed k with $C \in \mathcal{C}$,
(2) determining the exact complexity of $LALR(k)$ testing with fixed values of k, and
(3) determining bounds on the running time of uniform algorithms for $LALR$ testing.

Appendix. Proof of Lemma 2.3

PROOF OF $\subseteq$.

Recall that the states of Q_2 are LR items. We shall show that every item in $Q_2 \cap \delta(q_0, \gamma)$ is valid for γ. We proceed by induction on the number of states of Q_2 occurring in a computation of $M(G, u)$.
Accordingly, let $q \in Q_2 \cap \delta(q_0, \gamma)$.

Basis. If q is the only Q_2 state in the computation, then the computation must be q_0q. Thus $\gamma = \lambda$ and q is of the form $[S \rightarrow .\alpha, \lambda]$ which clearly is valid for γ.

Step. Assume that if $q \in \delta(q_0, \gamma)$ by a computation involving $n \geq 1$ states of Q_2, then q is valid for γ. Suppose that $q \in \delta(q_0, \gamma)$ by a computation involving $n + 1$ states of Q_2. Let p be the predecessor state of q in the computation.

Case 1. $p \in Q_2$. Then p has the form $[A \rightarrow \beta_1.X\beta_2, v]$, q has the form $[A \rightarrow \beta_1X.\beta_2, v]$, and the computation is $\ldots pXq$. Thus $\gamma = \alpha\beta_1X$ for some α and by the inductive hypothesis, p is valid for $\alpha\beta_1$. Therefore there exists a derivation $S \Rightarrow^*_{\rm rm} \alpha Ax \Rightarrow^*_{\rm rm} \alpha\beta_1X\beta_2x$ with v a prefix of x. Clearly then this same derivation implies that q is valid for $\alpha\beta_1X = \gamma$.

Case 2. $p \in Q_1$. Then p', the predecessor of p in the computation, must be a member of Q_2. Write p' as $[A \rightarrow \beta_1.X\beta_2, v]$. The transitions from p' to p to q must be λ transitions so by the inductive hypothesis, p' is valid for γ, which can be written $\alpha\beta_1$ for some α. Thus there exists a derivation

$$S \Rightarrow^*_{\rm rm} \alpha AX \Rightarrow_{\rm rm} \alpha\beta_1X\beta_2x = \gamma X\beta_2x$$

in which v is a prefix of x. Several subcases now arise.

Subcase 2a. $v = \lambda$ and p is $[X, \lambda]$. Then q can be written $[X \rightarrow .\theta, \lambda]$. Let z be any string derivable from β_2. Then

$$S \Rightarrow^*_{\rm rm} \gamma X\beta_2x \Rightarrow^*_{\rm rm} \gamma Xzx \Rightarrow^*_{\rm rm} \gamma\theta zx.$$

Thus $[X \rightarrow .\theta, \lambda]$ is valid for γ.

Subcase 2b. $v = \lambda$ and p is $[X, w]$ with $w \neq \lambda$. Then $w \in FIRST_{|w|}(\beta_2)$ and so we can write $\beta_2 \Rightarrow^* wz$ for some z. Moreover, q is of the form $[X \rightarrow .\theta, w]$. Consider the derivation

$$S \Rightarrow^*_{\rm rm} \gamma X\beta_2x \Rightarrow^*_{\rm rm} \gamma Xwzx \Rightarrow_{\rm rm} \gamma\theta wzx.$$

Clearly $[X \rightarrow .\theta, w]$ is valid for γ.

Subcase 2c. $v \neq \lambda$ and p is $[X, wv]$. Then $\beta_2 \Rightarrow^* w$ and q is of the form $[X \rightarrow .\theta, wv]$. Consider the derivation

$$S \Rightarrow^*_{\rm rm} \gamma X\beta_2x \Rightarrow^*_{\rm rm} \gamma Xwzx \Rightarrow_{\rm rm} \gamma\theta wzx.$$

Since v is a prefix of x, we can conclude that $[X \rightarrow .\theta, wv]$ is valid for γ. Q.E.D. $\subseteq$.

PROOF of $\supseteq$.

We shall show that if $q \in Q_2$ is a valid item for γ, then $q \in \delta(q_0, \gamma)$. We proceed by induction on the length of γ. The basis of the induction is similar to the step which is given below and is left to the reader. Accordingly, suppose that whenever $|\gamma| < n$, $\delta(q_0, \gamma)$ includes all elements of Q_2 which are valid for γ.

Now suppose that $|\gamma| = n$. We may thus write γ as $X_1 \cdots X_n$. Suppose that $q = [B \rightarrow \beta_1.\beta_2, v]$ is valid for γ. Therefore there exists a w and a derivation

$$S \Rightarrow^*_{\rm rm} \alpha\beta vw \Rightarrow \alpha\beta_1\beta_2vw \text{ with } \alpha\beta_1 = \gamma.$$

Two cases now arise.

Case 1. $\beta_1 \neq \lambda$. Certainly then $\beta_1 = \beta_1'X_n$ and $p = [B \rightarrow \beta_1'.X_n\beta_2, v]$ is valid for $X_1 \cdots X_{n-1}$. By the inductive hypothesis, $p \in \delta(q_0, X_1 \cdots X_{n-1})$. Inspection of the move rules of $M(G, u)$ then reveals that $[B \rightarrow B'_1X_n.\beta_2, v] = q$ is a member of $\delta(q_0, X_1 \cdots X_n) = \delta(q_0, \gamma)$.

Case 2. $\beta_1 = \lambda$. Clearly it suffices to show that $[B, v] \in \delta(q_0, \gamma)$.

By definition of valid item, there exists a derivation

$$S \Rightarrow^*_{\rm rm} X_1 \cdots X_nBvx \Rightarrow_{\rm rm} X_1 \cdots X_n\beta_1\beta_2vx$$

for some x. Let us consider the last instant in this derivation when a production is applied to some position $j \leq n$ of a sentential form, and let us consider all subsequent applications of productions to position $n + 1$. We may thus write the derivation as

$$\begin{aligned} S &\Rightarrow^*_{\text{rm}} X_1 \cdots X_{j-1}A_0w_0 \Rightarrow_{\text{rm}} X_1 \cdots X_nA_1\theta_1w_0 = \gamma A_1\theta_1w_0 \\ &\Rightarrow^*_{\text{rm}} \gamma A_1w_1w_0 \Rightarrow_{\text{rm}} \gamma A_2\theta_2w_1w_0 \\ &\qquad\vdots \qquad\qquad \vdots \\ &\Rightarrow^*_{\text{rm}} \gamma A_{m-1}w_{m-1} \cdots w_0 \Rightarrow_{\text{rm}} \gamma A_m\theta_mw_{m-1} \cdots w_0 \\ &\Rightarrow^*_{\text{rm}} \gamma A_mw_m \cdots w_0 = \gamma Bvx. \end{aligned}$$

Notice that $\theta_i \Rightarrow^* w_i$ for $1 \leq i \leq m$. Moreover, there exists some l such that $v = w_m \cdots w_{l+1}w_l'$ where w_l' is a prefix of w_l. Several subcases arise depending on the value of l.

Subcase 2a. $l = 0$. Then $[A_0 \rightarrow X_j \cdots X_nA_1\theta_1, w_0]$ is valid for $X_1 \cdots X_{n-1}$ and by the inductive hypothesis is a member of $\delta(q_0, X_1 \cdots X_{n-1})$. By inspection of the move rules of $M(G, u)$ it can be seen that $[A_0 \rightarrow X_j \cdots X_n.A_1\theta_1, w_0']$ is valid for $\delta(q_0, X_1 \cdots X_n) = \delta(q_0, \gamma)$ and so $[A_1, w_1w_0] \in \delta(q_0, \gamma)$. We may next show that for $1 \leq i < m$, $[A_i \rightarrow .A_{i+1}\theta_{i+1}, w_i \cdots w_1w_0']$ and $[A_{i+1}, w_{i+1} \cdots w_1w_0']$ are elements of $\delta(q_0, \gamma)$. But this means that $[A_m, w_m \cdots w_1w_0'] = [B, v]$ is in $\delta(q_0, \gamma)$.

Subcase 2b. $l = 1$. In a fashion similar to subcase 2a, we can show that $[A_0 \rightarrow X_j \cdots X_n.A_1\theta_1, \lambda] \in \delta(q_0, \gamma)$ and so $[A_1, w_1'] \in \delta(q_0, \gamma)$. By a straightforward induction, $[A_m, w_m \cdots w_2w_1'] = [B, v]$ is in $\delta(q_0, \gamma)$.

Subcase 2c. $l > 1$. After showing that $[A_1, \lambda] \in \delta(q_0, \gamma)$, we show that $[A_l, w_l'] \in \delta(q_0, \gamma)$ and then establish that $[A_m, w_m \cdots w_{l+1}w_l']$ is in $\delta(q_0, \gamma)$. Thus $[B, v] \in \delta(q_0, \gamma)$.

References

1. Hunt, H.B. III, Szymanski, T.G. and Ullman, J.D. Operations on sparse relations and efficient algorithms for grammar problems. IEEE Conf. Rec., 15th Ann. Symp. on Switching and Automata Theory, 1974, pp. 127–132.
2. Hartmanis, J. Context-free languages and Turing machine computations. Proc. Symp. Appl. Math. Vol. 19, Amer. Math. Soc., Providence R.I., 1967, pp. 45–51.
3. Aho, A.V. and Ullman, J.D. *The Theory of Parsing, Translation, and Compiling, Vols. 1 and 2.* Prentice-Hall, Englewood Cliffs, N.J., 1972 and 1973.
4. Knuth, D.E. On the translation of languages from left to right. *Info. Contr., 8,* 6 (1965), 607–639.
5. Korenjak, A.J., and Hopcroft. J.E. Simple deterministic languages. IEEE Conf. Rec., 7th Ann. Symp. on Switching and Automata Theory, 1966, pp. 36–46.
6. Brosgol, B.M. Deterministic translation grammars. TR-3-74, Center for Research in Computing Technology, Harvard U., 1974 (also Proc. 8th Ann. Princeton Conf. on Inform. Sci. and Systems, 1974, pp. 300–306.
7. Younger, D. H. Recognition and parsing of context-free languages in time n^3. *Inform. Contr., 10,* 2 (1967), 189–208.
8. Rosenkrantz, D.J. and Lewis P.M. II Deterministic left corner parsing. IEEE Conf. Rec., 11th Ann. Symp. on Switching and Automata Theory, 1970, pp. 139–152.
9. Johnson, D.B. and Sethi, R. Efficient construction of *LL*(1) Parsers. TR-164, Dep. of Computer Sci., Penn State U., State College, Pa., March 1975.

Programming Languages

B. Wegbreit Editor

A Fast and Usually Linear Algorithm for Global Flow Analysis (Abstract Only)

Susan L. Graham and Mark Wegman
University of California, Berkeley

A new algorithm for global flow analysis on reducible graphs is presented. The algorithm is shown to treat a very general class of function spaces. For a graph of e edges, the algorithm has a worst case time bound of O(e log e) function operations. It is also shown that in programming terms, the number of operations is proportional to e plus the number of exits from program loops. Consequently a restriction to one-entry one-exit control structures guarantees linearity. The algorithm can be extended to yet larger classes of function spaces and graphs by relaxing the time bound. Examples are given of code improvement problems which can be solved using the algorithm.

Key Words and Phrases: global flow analysis, data flow, code optimization, common subexpression elimination, live-dead analysis, information propagation, flow graph, reducibility, go-to-less programming, depth-first search, path compression

CR Categories: 4.12, 5.24, 5.25, 5.32

Complete Paper: J. ACM 23, 1 (Jan. 1976)

An extended abstract of this paper was presented at the Second ACM Symposium on Principles of Programming Languages, Palo Alto, Calif., Jan. 20–,22 1975.

This research was supported by the National Science Foundation under grant DCR74-07644-A01. Authors' present addresses: S.L. Graham, Computer Science Division, 577 Evans Hall, University of California, Berkeley, CA 94720; M. Wegman, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.