INFORMATION AND CONTROL 40, 123–143 (1979)

Parsing Macro Grammars Top Down*

KURT MEHLHORN

Fachbereich 10, Universität des Saarlandes, 6600 Saarbrücken, West Germany

A top-down parsing scheme for macro grammars is proposed. It extends the recursive descent method frequently used in context-free parsing. It is shown to be decidable if a macro grammar is top-down parsable. Nearly matching lower and upper bounds for the complexity of the decision procedure are derived. A time and space efficient recognition algorithm is described.

Recursive descent is for its ease of description and for its transparency one of the popular parsing methods (Gries, 1971; Knuth, 1978). The class of languages, for which recursive descent works as a parsing method, is known as the LL languages; their properties were studied by Lewis and Stearns (1968), Rosenkranz and Stearns (1970), and many others (see Aho and Ullman (1972) for complete references).

In the late 1960s several extensions of context-free languages were proposed to cope with the non-context-free features of programming languages (e.g., applied and defining occurrences of identifiers). Two remarkable examples are the macro languages of Fischer (1968) and the indexed languages of Aho (1968). Because of the lack of efficient parsing methods for these classes of grammars, they were never used in actual programming language design.

Weiss (1975) proposed a top-down parsing scheme for indexed languages. He introduced the notion of indexed LL grammars and showed that ϵ-free indexed LL grammars can be parsed efficiently (time $O(n^2)$). His work was the starting point for this paper.

In Section 1 we introduce macro grammars and formulate the LL property for macro grammars. In Section 2 we give first evidence for the power of MLL languages: every deterministic context-free language is generated by an MLL grammar. In Section 3 we show that transformation to standard form can be done while preserving the LL property. In Section 4 we show that it is decidable whether an arbitrary macro grammar is MLL(k) for a fixed k. We derive closely matching upper and lower bounds for the complexity of the decision problem. In Section 5 we comment on the relationship to Weiss' work on indexed grammars and in Section 6 we describe a parsing method of time complexity $O(n^2)$. Various open problems are also stated.

* A preliminary version of this paper was presented at the 6th annual conference of the Gesellschaft für Informatik (GI), Stuttgart, September 29 to October 1, 1976.

1. Macro Grammars, LL Property

A macro grammar (Fischer, 1968) is a 6-tuple $(\Sigma, \mathscr{F}, \mathscr{V}, \rho, S, P)$, where Σ is a finite set of terminal symbols; $\mathscr{F}$ is a finite set of nonterminal or function symbols; $\mathscr{V}$ is a finite set of argument or variable symbols; ρ is a function from $\mathscr{F}$ into nonnegative integers ($\rho(F)$ is the number of arguments which F takes); $S \in F$ is the start symbol, $\rho(S) = 0$; P is a finite set of productions of the form $F(x_1, \ldots, x_{\rho(F)}) \to \tau$, where $F \in \mathscr{F}$, $x_1, \ldots, x_{\rho(F)}$ are distinct members of $\mathscr{V}$, and τ is a term over Σ, $\{x_1, \ldots, x_{\rho(F)}\}$, $\mathscr{F}$, ρ.

The set of terms over Σ, $\mathscr{V}$, $\mathscr{F}$, ρ is defined inductively,

(a) ϵ is a term, a is a term for every $a \in \Sigma$, and x is a term for every $x \in \mathscr{V}$;

(b) if τ_1 and τ_2 are terms then $\tau_1 \cdot \tau_2$ is a term;

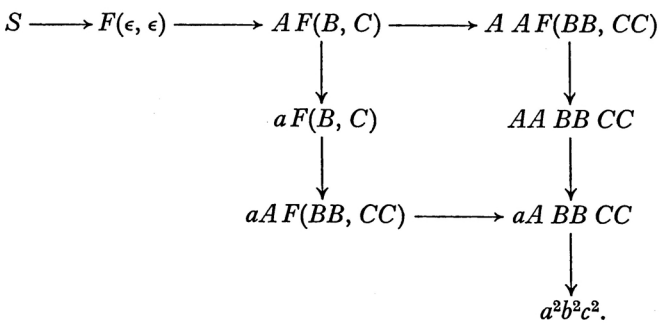
(c) if $F \in \mathscr{F}$ and $\tau_1, \ldots, \tau_{\rho(F)}$ are terms, then $F(\tau_1, \ldots, \tau_{\rho(F)})$ is a term.

We consider macro grammars with the outside-in (OI) mode of derivation (Fischer, 1968; Nivat, 1974), i.e., only top-level occurrences of function symbols can be rewritten at every step. More formally, let $z = \tau' \cdot F(\tau_1, \ldots, \tau_{\rho(F)}) \cdot \tau''$ be a term over Σ, $\mathscr{F}$, and ρ with F at the topmost level and let $F(x_1, \ldots, x_{\rho(F)}) \to \tau$ be a rule. Then z derives $\tau' \cdot \tau[\tau_1/x_1, \ldots, \tau_{\rho(F)}] \cdot \tau''$, where $\tau[\tau_1/x_1, \ldots]$ is the term obtained from τ by substituting τ_i for x_i, $1 \leqslant i \leqslant \rho(F)$.

Example. A macro grammar generating $\{a^n b^n c^n; n \geqslant 0\}$

$$
\begin{aligned}
S &\to F(\epsilon, \epsilon),\\
F(x, y) &\to AF(xB, yC),\\
F(x, y) &\to xy,\\
A &\to a,\\
B &\to b,\\
C &\to c.
\end{aligned}
$$

F is a function symbol of arity 2 and S, A, B, C are function symbols of arity 0. A sample derivation is



Note that we had the choice of rewriting either A of F in the sentential form $AF(B, C)$. We could not have rewritten B or C since they do not occur at the top level but rather within a parameter list. Rewriting A in $AF(B, C)$ corresponds to a leftmost derivation.

The macro grammar given above suggests a top-down parsing algorithm (recursive descent) for the language $\{a^n b^n c^n; n \geqslant 0\}$

```
procedure S; call F(ϵ, ϵ) end;
procedure F(x, y);
  begin case next-symbol in
    a: call A; call F(xB, yC);
      b, eof: let xy = τ1 · τ2 ··· τk
        where τi is a term starting with a function symbol;
          for i from 1 to k do call τi ;
    c: Error
  end;
procedure A;
  begin case next-symbol in
    a: advance reading head by one and read the next symbol;
    b, c: Error
  end;
⋮
```

The parse is performed in a single left-to-right scan of the input string. Next-symbol always contains the symbol of the input string which is presently scanned (end-of-file (eof) designates the end of the input string). Within each procedure we branch on the symbol under the reading head and call the appropriate production. This strategy is possible whenever the decision between the different alternatives for a function symbol can be made on the basis of knowing the next (the next k for some fixed k) input symbol. This leads to the following definition.

DEFINITION. (a) Let $r = F(x_1, \ldots, x_{\rho(f)}) \to \tau$ be a rule of a macro grammar and let k be an integer. Then

$$\text{First}_k(r) = \{u;\ S \xrightarrow[lm]{*} wF(\tau_1, \ldots, \tau_{\rho(F)})\,\tau' \xrightarrow[lm]{} w\tau[\tau_1/x_1, \ldots, \tau_{\rho(F)}/x_{\rho(F)}]\,\tau'$$

$$\xrightarrow[lm]{*} wuv;\ w, u, v \in \Sigma^*,\ \tau', \tau_1, \ldots, \tau_{\rho(F)} \text{ are terms},$$

$$|u| = k \text{ or } |u| < k \text{ and } v = \epsilon\}.$$

$\tau[\tau_1/x_1, \ldots, \tau_{\rho(F)}/x_{\rho(F)}]$ is the term obtained by replacing x_i by τ_i, $1 \leqslant i \leqslant \rho(F)$, in τ.

(b) A macro grammar has the LL(k) property if for every pair r_1, r_2 of distinct rules having the same left-hand side:

$$\text{First}_k(r_1) \cap \text{First}_k(r_2) = \varnothing.$$

In this case we say that the grammar is MLL(k) (is an MLL(k) grammar).

Our example grammar is MLL(1). Other definitions of MLL(k) grammars are possible (see Section 5). In the context-free case our definition corresponds to strong LL(k) grammars.

2. MLL Grammars and Deterministic Languages

In this section we relate MLL languages and deterministic context-free languages.

Theorem 1. *Given any deterministic pushdown automaton A, we can find an equivalent* MLL(1) *grammar G, i.e., $L(A) = L(G)$.*

Proof. Let $A = (S, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a deterministic pushdown automaton accepting by final state. $S = \{q_0, \ldots, q_n\}$ is the set of states, Σ the input alphabet, Γ the stack alphabet, δ is the transition function, q_0 the start state, Z_0 the symbol initially placed at the bottom of the pushdown store, and F is the set of final states. The transition function δ maps $S \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ into $S \times \Gamma^*$. $\delta(p, a, A) = (q, A_1 \cdots A_k)$ means that the automaton in state p reading symbol $a \in \Sigma \cup \{\epsilon\}$ and having A at the top of the pushdown store changes its state to q and replaces A by $A_1 \cdots A_k$ ($k \geqslant 0$). A_1 will be the new top of the pushdown store. We may assume w.l.o.g. that A writes at most two symbols onto the stack in a single move, never empties its pushdown store and that A enters a final state at most once during a sequence of ϵ-moves. The first two restrictions are standard, the third restriction is obtained as follows. Duplicate all states, i.e., $\{q_0, \ldots, q_n\} \cup \{\bar{q}_0, \ldots, \bar{q}_n\}$ is the new state set, and change δ as follows:

Assume $\delta(p, a, A) = (q, \gamma)$.

Case 1. $a \neq \epsilon$. Then $\delta'(p, a, A) = \delta'(\bar{p}, a, A) = (q, \gamma)$.

Case 2. $a = \epsilon$. Then

$$\begin{aligned} \delta'(\bar{p}, a, A) &= (\bar{q}, \gamma), \\ \delta'(p, a, A) &= (q, \gamma) \qquad \text{if } p \text{ is nonfinal}, \\ \delta(p, a, A) &= (\bar{q}, \gamma) \qquad \text{if } p \text{ is final}. \end{aligned}$$

With this modification A enters a barred state whenever it accepts for the first time during a sequence of ϵ-moves and stays in a barred state until it reads a

proper symbol. Until the end of the proof we assume that A obeys both restrictions.

The macro grammar G has function symbols $S \times \Gamma \cup \{\text{START}\}$; START has arity 0, all other function symbols have arity $|\ S\ |$.

The rules are:

(1) $\text{START} \to [q_0, Z_0](\epsilon, \epsilon, \ldots, \epsilon)$ $|\ S\ |$-times,

(2) for $a \in \Sigma \cup \{\epsilon\}$ and $\delta(q, a, A) = (q_i, \epsilon)$ $[q, A]\ (x_0, \ldots, x_n) \to a x_i$,

(3) for $a \in \Sigma \cup \{\epsilon\}$ and $\delta(q, a, A) = (q_i, B)$ $[q, A](x_0, \ldots, x_n) \to a[q_i, B](x_0, \ldots, x_n)$,

(4) for $a \in \Sigma \cup \{\epsilon\}$ and $\delta(q, a, A) = (q_i, BC)$ $[q, A](x_0, \ldots, x_n) \to a[q_i, B]([q_0, C](x_0, \ldots, x_n), [q_1, C](x_0, \ldots, x_n), \ldots, [q_n, C](x_0, \ldots, x_n))$.

(5) for all $q \in F$ and $A \in \Gamma$ $[q, A]\ (x_0, \ldots, x_n) \to \epsilon$.

In the second components of the function symbols we build up the pushdown store. In the first components we store the state. The only problem is: how to remember the state during a stack erasing move. The solution is to provide for all possible successor states (clause 4) and select the proper one (clause 2). A formal proof is left to the reader. It can be carried out by proving the following claim.

Claim. Let $a_1, a_2, \ldots, a_n \in \Sigma$, $Z_0, Z_1, \ldots, Z_j \in \Gamma$, and $q \in S$. Then

$$(q_0, a_1 a_2 \cdots a_n, Z_0) \vdash^* (q, \epsilon, Z_j \cdots Z_1)$$

iff

$$\text{START} \overset{*}{\to} a_1 a_2 \cdots a_n [q, Z_j]\ (\text{expansion of } Z_{j-1} \cdots Z_1),$$

where

$$(\text{expansion of } \epsilon) = \underbrace{(\epsilon, \epsilon, \ldots, \epsilon)}_{|S|\text{-times}}$$

and

$$(\text{expansion of } Z\alpha) = ([q_0, Z](\text{expansion of } \alpha), \ldots, [q_n, Z](\text{expansion of } \alpha))$$

It remains to show that G is MLL(1). Consider any function symbol $[q, Z]$ of G.

Case 1. $\delta(q, a, Z)$ is defined for some $a \in \Sigma$ and hence $\delta(q, \epsilon, Z)$ is undefined. Then for any $b \in \Sigma$ at most one rule r_b with left-hand side $[q, Z]$ is introduced by clauses 2, 3, 4 and $\text{First}_1(r_b) = \{b\}$. Since $\text{First}_1(r) = \{\epsilon\}$ for all rules r introduced by clause 5 we are done in this case.

Case 2. $\delta(q, a, Z)$ is undefined for all $a \in \Sigma$ and hence $\delta(q, \epsilon, Z)$ may be defined. Then at most two rules with left-hand side $[q, Z]$ exist, say rule r defined by clause 2, 3, or 4 and rule s defined by clause 5. If both exist then q is final. $\text{First}_1(s) = \{\epsilon\}$ since s is defined by clause 5. We have to show $\epsilon \notin \text{First}_1(r)$.

Since A never empties its pushdown store every derivation using r has the following form by the claim above:

$$\begin{aligned} \text{START} &\overset{*}{\to} w[q, Z](\tau_1, \tau_2, \ldots, \tau_{|S|}) \\ &\to w\tau[\tau_1/x_1, \ldots, \tau_{|S|}/x_S] \\ &\overset{*}{\to} wu[p, Z'](\cdots) \to wu, \end{aligned}$$

where $r \equiv [q, Z](x_1, \ldots, x_{|S|}) \to \tau$ and p is a final state. Since A enters at most one final state during a sequence of ϵ-moves $u \neq \epsilon$ by the claim above. This shows $\epsilon \notin \text{First}_1(r)$. ∎

COROLLARY. *The class of* MLL(1) *languages properly contains the deterministic context-free languages.*

Proof. By Theorem 1 and the example in Section 1. ∎

3. TRANSFORMATION TO STANDARD FORM

A macro grammar is in *standard form* if every one of its rules is in one of the following four forms

$$F(x_1, \ldots, x_n) \to G(H_1(x_1, \ldots, x_n), \ldots, H_m(x_1, \ldots, x_n)) \qquad \text{with } n, m \geqslant 0; \quad (1)$$

$$F(x_1, \ldots, x_n) \to x_1 \cdots x_n, \qquad n \geqslant 1; \quad (2)$$

$$F(x_1, \ldots, x_n) \to x_i, \qquad n \geqslant 1, 1 \leqslant i \leqslant n; \quad (3)$$

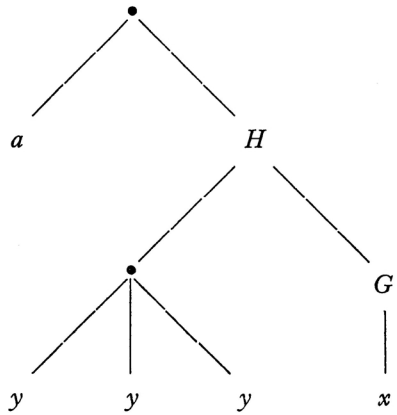
$$F(x_1, \ldots, x_n) \to a \qquad \text{for } a \in \Sigma \cup \{\epsilon\}, n \geqslant 0. \quad (4)$$

Fischer showed that every macro grammar has an equivalent standard form grammar. We observe that this transformation preserves the MLL(k) property for every k.

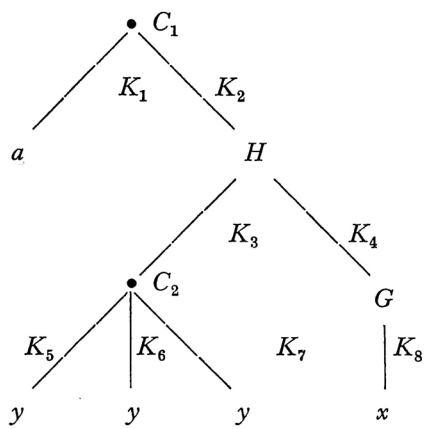
In the sequel we frequently talk about the size of a grammar. Since a listing of the productions of a grammar is sufficient to infer all information needed to define a grammar, we define the total number of symbols in the productions of G to be the size of G (notation: size(G)). The maximal rank of any function symbol in G is denoted by max-rank(G). $f(G)$ denotes the number of function symbols of G and $p(G)$ the number of productions of G.

THEOREM 2. *Given any* MLL(k) *grammar G, we can find an equivalent standard form* MLL(k) *grammar G' with* $f(G') \leqslant \text{size}(G)$, $p(G') \leqslant \text{size}(G)$, $\text{max-rank}(G') \leqslant \text{max-rank}(G)$ *and* $\text{size}(G') \leqslant 2 \cdot (2 + \text{max-rank}(G))^2 \cdot \text{size}(G)$.

Proof (sketch). Consider any rule $F(x_1, \ldots, x_n) \to \tau$ which is not in standard form (e.g., $F(x, y) \to aH(yyy, G(x))$). We may write τ as a tree, e.g., $aH(yyy, G(x))$ corresponds to the following tree.



For every edge of the tree we introduce a new function symbol of arity n (the arity of the left-hand side). In our example we introduce the symbols $K_1, \ldots, K_8$ of arity 2.



For every node labeled by $\cdot$ we introduce a function symbol whose arity is equal to the number of direct descendants of that node. In our example we introduce symbols C_1, C_2 of arity 2 and 3, respectively. For every internal node v of the tree let G be the function symbol at the node, H_0 is the function symbol at the incoming edge (H_0 is F if v is the root) and the $H_1, \ldots, H_m$ be the function symbols at the outgoing edges. Introduce the new tule (type (1))

$$H_0(x_1, \ldots, x_n) \to G(H_1(x_1, \ldots, x_n), \ldots, H_m(x_1, \ldots, x_n)).$$

For every internal node v labelled $\cdot$ in the original tree and having k descendants introduce the rule (type (2))

$$C(x_1, \ldots, x_k) \to x_1 \cdots x_k .$$

For every leaf of the tree let G be the function symbol at the incoming edge and let a be the label at the leaf. Introduce the rule (type (3) or (4))

$$G(x_1, \ldots, x_n) \to a.$$

In our example we replace the rule $F(x, y) \to aH(yyy, G(x))$ by the following set of rules:

$$\begin{aligned}
F(x, y) &\to C_1(K_1(x, y), K_2(x, y)),\\
K_1(x, y) &\to a,\\
K_2(x, y) &\to H(K_3(x, y), K_4(x, y)),\\
K_3(x, y) &\to C_2(K_5(x, y), K_6(x, y), K_7(x, y),\\
K_4(x, y) &\to G(K_8(x, y)),\\
K_5(x, y) &\to y,\\
K_6(x, y) &\to y,\\
K_7(x, y) &\to y,\\
K_8(x, y) &\to x,\\
C_1(x_1, x_2) &\to x_1 \cdot x_2 ,\\
C_2(x_1, x_2, x_3) &\to x_1 \cdot x_2 \cdot x_3 .
\end{aligned}$$

The derivations according to G and G' are in a 1–1 correspondence, only one rule is applicable to each new function symbol. Hence G' is also MLL(k). Furthermore, simple counting implies $f(G') \leqslant \text{size}(G)$, $p(G') \leqslant \text{size}(G)$, max-rank$(G') \leqslant$ max-rank(G), and size$(G') \leqslant f(G') \cdot 2 \cdot (2 +$ max-rank$(G'))^2 \cdot$ max-rank(G'). ∎

4. Testing for the MLL(k) Property

In this section we show that it is decidable whether a macro grammar is MLL(k). We describe a decision procedure and analyze its time and space requirements. Then we derive a lower bound for the space complexity of the problem.

Theorem 3. *Given a macro grammar G and an integer k it is decidable whether G is* MLL(k) *in time* $O(|\Sigma|^{k+1} \cdot \text{max-rank}(G)^{11} \cdot \text{size}(G)^5 \cdot 2^{11 \cdot k^2 \cdot \text{max-rank}(G)^2})$ *and space* $O(k^2 \cdot \text{max-rank}(G)^3 \cdot \text{size}(G) \cdot 2^{3k^2 \text{max-rank}(G)})$.

Proof. We first transform G into a standard form grammar G'. G is MLL(k) iff G' is MLL(k) (compare Theorem 2). Hence we may assume w.l.o.g. that G is in standard form. Let $r \equiv H(x_1, \ldots, x_p) \to \tau$ be a rule of G. We want to compute $\text{First}_k(r)$. We proceed in two steps:

(1) Let Σ be the terminal alphabet of G. Then the language L_r over $\Sigma \cup \{\bar{a}; a \in \Sigma\}$ is a macro language, where

$$\begin{aligned} L_r = \{ & a_1 \cdots a_m \bar{a}_{m+1} \cdots \bar{a}_n ; S \overset{*}{\to} a_1 \cdots a_m H(\tau_1, \ldots, \tau_p)\, \tau' \\ & \to a_1 \cdots a_m \tau[\tau_1/x_1, \ldots, \tau_p/x_p]\, \tau' \\ & \overset{*}{\to} a_1 \cdots a_m a_{m+1} \cdots a_n \in L(G)\}. \end{aligned}$$

(2) For any $x \in \Sigma^*$;

$$\text{if } |x| < k \text{ then } x \in \text{First}_k(r) \text{ iff } (L_r \cap \Sigma^* \bar{x}) \neq \varnothing,$$

$$\text{if } |x| = k \text{ then } x \in \text{First}_k(r) \text{ iff } (L_r \cap \Sigma^* \bar{x} \bar{\Sigma}^*) \neq \varnothing.$$

Since the class of macro languages is closed under intersection with a regular set and their emptiness problem is decidable (Fischer, 1968), this implies the decidability of the MLL(k) property. Fischer showed the decidability of the emptiness problem by reducing it to the emptiness problem for indexed languages and appealing to a result of Aho. We give a direct proof here; this provides us with a tighter time bound.

LEMMA 1. *Given a standard form grammar G and a production r, we can find a standard form grammar G_r generating L_r with* $\text{max-rank}(G_r) \leqslant 3 \cdot \text{max-rank}(G), f(G_r) \leqslant 3 \cdot f(G_r), p(G_r) \leqslant (\text{max-rank}(G) + 3) \cdot p(G)$ *and* $\text{size}(G_r) \leqslant (\text{max-rank}(G) + 3) \cdot 3 \cdot \text{size}(G)$.

Proof. For every function symbol F in G there are function symbols F, $\tilde{F}$, and $\bar{F}$ in G_r of arity $\rho(F)$, $3 \cdot \rho(F)$ and $\rho(F)$, respectively. They carry the following semantics:

$$F(\,) \overset{*}{\underset{G}{\to}} a_1 a_2 \cdots a_m \quad \text{iff} \quad F(\,) \overset{*}{\underset{G_r}{\to}} a_1 \cdots a_m \quad \text{and} \quad \bar{F}(\,) \overset{*}{\underset{G_r}{\to}} \bar{a}_1 \cdots \bar{a}_m$$

and

$$\begin{aligned} & F(\,) \overset{*}{\to} a_1 \cdots a_j H(\tau_1, \ldots, \tau_p)\, \tau' \\ & \quad \to a_1 \cdots a_j \tau[\tau_1/x_1, \ldots, \tau_p/x_p]\, \tau' \\ & \quad \overset{*}{\to} a_1 \cdots a_j a_{j+1} \cdots a_m \end{aligned}$$

iff

$$\tilde{F}(\,) \overset{*}{\to} a_1 \cdots a_j \bar{a}_{j+1} \cdots \bar{a}_m$$

The start symbol of G_r is $\tilde{S}$, where S is the start symbol of G. Let $x = (x_1,..., x_n)$. Then we use $\bar{x}$ to denote $(\bar{x}_1, \bar{x}_2,..., \bar{x}_n)$ and $\tilde{x}$ to denote $(x_1, \tilde{x}_1, \bar{x}_1, x_2, \tilde{x}_2, \bar{x}_2,...)$. For every rule in G the following rules are in G_r:

(1) If the rule is of the form

$$F(x) \to G(H_1(x),..., H_m(x))$$

then

$$F(x) \to G(H_1(x),..., H_m(x)),$$
$$\bar{F}(\bar{x}) \to \bar{G}(\bar{H}_1(\bar{x}),..., \bar{H}_m(\bar{x})),$$
$$\tilde{F}(x) \to \tilde{G}(H_1(x), \tilde{H}_1(\tilde{x}), \bar{H}_1(\bar{x}), H_2(x), \tilde{H}_2(\tilde{x}), \bar{H}_2(\bar{x}),...).$$

(2) If the rule is of the form $F(x) \to x_1 \cdots x_n$ then

$$F(x) \to x_1 \cdots x_n,$$
$$\bar{F}(\bar{x}) \to \bar{x}_1 \cdots \bar{x}_n,$$
$$\tilde{F}(\tilde{x}) \to x_1 \cdots x_{i-1}\tilde{x}_i\bar{x}_{i+1} \cdots \bar{x}_n, \qquad 1 \leqslant i \leqslant n.$$

(3) If the rule is of the form $F(x) \to x_i$ then

$$F(x) \to x_i,$$
$$\bar{F}(\bar{x}) \to \bar{x}_i,$$
$$\tilde{F}(\tilde{x}) \to \tilde{x}_i.$$

(4) If the rule is of the form $F(x) \to a$ with $a \in \Sigma \cup \{\epsilon\}$ then

$$F(x) \to a,$$
$$\bar{F}(\bar{x}) \to \bar{a}.$$

(5) If the rule is the rule $r \equiv H(x) \to \tau$ then

$$\tilde{H}(\tilde{x}) \to \bar{\tau},$$

where $\bar{\tau}$ is the term obtained from τ by barring all symbols.

Note that only function symbols F and $\bar{F}$ have terminal productions and that only clause 5 generates a production which allows us to get rid of the "twiggled" function symbol. Keeping this and the proof of Theorem 1 in mind the reader should have no difficulties in verifying the theorem. ∎

LEMMA 2. *Given any standard form grammar G and nondeterministic finite automaton A with s states, we can find a standard form grammar G' with $L(G') = L(G) \cap L(A)$ and* max-rank$(G') = S^2$ max-rank(G)

$$f(G') = s^2 \cdot f(G) + 1, \qquad p(G') \leqslant (1 + p(G)) \max(s^2, s^{\text{max-rank}(G)+1}),$$

and

$$\text{size}(G') \leqslant (p(Q) + 1) \cdot \max^2, s^{\text{max-rank}(G)+1})) \cdot \text{size}(G).$$

Proof. Let A be a finite automaton with states $S = \{t_1, \ldots, t_s\}$, start state t_1 and the set $F \subseteq S$ of final states. Let δ be the transition function of A. Let $G = (\Sigma, \mathscr{F}, \mathscr{V}, \rho, S_0, P)$ be a macro grammar. The grammar G' has function symbols $S \times \mathscr{F} \times S \cup \{\text{START}\}$. START has arity 0 and $[t_i, F, t_j]$ has arity $s^2 \cdot \rho(F)$. START is the axiom of G'. The function symbol $[t_i, F, t_j]$ generates a string w if and only if F derives w in G and $t_j \in \delta(t_i, w)$. The rules of G' are:

(1) for every $t \in F$ the rule

$$\text{START} \to [t_1, S_0, t],$$

(2) for every rule $F(x_1, \ldots, x_n) \to G(H_1(x_1, \ldots, x_n), \ldots, H_m(x_1, \ldots, x_n))$ and every $t, t' \in S$ the rule

$$[t, F, t'](\cdots) \to [t, G, t']([t_1, H_1, t_1]( ), [t_1, H_1, t_2]( ), \ldots,$$
$$[t_1, H_1, t_s]( ), [t_2, H_1, t_1]( ), \ldots),$$

(3) for every rule $F(x_1, \ldots, x_n) \to x_1 \cdots x_n$ and every sequence $i_0, \ldots, i_n$ the rule

$$[t_{i_0}, F, t_{i_n}](x_1^{11}, \ldots, x_1^{1s}, x_1^{21}, \ldots) \to x_1^{i_0 i_1} x_2^{i_1 i_2} \cdots x_n^{i_{n-1} i_n},$$

(4) for every rule $F(x_1, \ldots, x_n) \to x_i$ and every $t_j, t_k \in S$ the rule

$$[t_j, F, t_k](\cdots) \to x_j^{jk},$$

(5) for every rule $F(x_1, \ldots, x_n) \to a$, $a \in \Sigma \cup \{\epsilon\}$ and every $t_j, t_k \in S$ with $t_k \in \delta(t_j, a)$ the rule

$$[t_j, F, t_k](. \ . \ . \ .) \to a.$$

The correctness of the construction is proved as in Theorem 1 and therefore left to the reader. The claims made about the various parameter of G' are obvious. ∎

LEMMA 3. *Given any standard form grammar G, we can decide $L(G) = \varnothing$ in time*

$$O(f(G) \cdot p(G) \cdot \text{size}(G) \cdot 2^{(\text{max-rank}(G)+1)^2})$$

and space

$$O(f(G) \cdot 2^{\text{max-rank}(G)} + \text{size}(G)).$$

Proof. We proceed in two steps.

(1) Replace all rules of the form $F(x_1, \ldots, x_n) \to a$ for $a \in \Sigma$ by $F(x_1, \ldots, x_n) \to \epsilon$. Then $L(G) \neq \varnothing$ iff the new grammar generates the empty string.

Step (1) neither increases the maximal rank not the size.

(2) A sample derivation $S \overset{*}{\to} \epsilon$ might look as follows:

$$S \to H_1(F_1, F_2, F_3, F_4)$$
$$\overset{*}{\to} F_2 F_4 \overset{*}{\to} \epsilon,$$

since

$$H_1 \overset{*}{\to} x_2 x_4$$

and

$$F_2, F_4 \to \epsilon.$$

To detect derivations of this form we have to determine for every function symbol $F(x_1, \ldots, x_n)$ all subsets $J \subseteq \{1, \ldots, n\}$ with $F(x_1, \ldots, x_n) \overset{*}{\to} x_{i_1} \cdots x_{i_m}$ and $J = \bigcup \{i_l\}$. To do so we consider the pairs (F, J) for $F \in \mathcal{F}$ and $J \subseteq \{1, \ldots, \rho(F)\}$. We mark these pairs in an iterative process. The pair (F, J) will be marked if and only if $F(x_1, \ldots, x_{\rho(F)}) \overset{*}{\to} x_{i_1} \cdots x_{i_m}$ with $J = \bigcup \{i_l\}$; then $\epsilon \in L(G)$ iff $(S, \varnothing)$ is marked upon termination of the algorithm:

for all rules of the form $F(\,) \to \tau$ where τ does not contain any function symbol
do mark (F, J) where $J = \bigcup_{x_i \in \tau} \{i\}$;
comment note that a rule $F(x_1, \ldots, x_n) \to \epsilon$ causes $(F, \varnothing)$ to be marked;
while there is a production $F(x_1, \ldots, x_n) \to H_0(\tau_1, \ldots, \tau_k)$ with
(1) (H_0, J_0) is marked,
(2) $J = \bigcup_{i \in J_0} J_i$, where $\tau_i = H_i(x_1, \ldots, x_n)$ and (H_i, J_i) is marked,
(3) (F, J) is unmarked.
do mark (F, J).

CLAIM. *(F, J) is marked during this process iff $F(x_1, \ldots, x_n) \overset{*}{\to} x_{i_1} \cdots x_{i_m}$ with $J = \bigcup \{i_l\}$.*

Proof. The proof is similar to the proof of the corresponding claim of Aho (1968) and therefore left to the reader.

There are $\leqslant 2^{\text{max-rank}(G)} \cdot f(G)$ pairs (F, J). Since every execution of the body of the while-loop marks one additional pair the body is executed at most $2^{\text{max-rank}(G)} \cdot f(G)$ times. Each execution of the body requires us to look at every rule; for every rule we have to look at the $\leqslant (2^{\text{max-rank}(G)})(2^{\text{max-rank}(G)})^{\text{max-rank}(G)}$ possibilities of combining the J_i's. Each possibility may be examined in time $O(\text{size}(G))$. Hence the running time of the algorithm is bounded by

$$O(2^{\text{max-rank}(G)} \cdot f(G) \cdot 2^{\text{max-rank}(G)(\text{max-rank}(G)+1)} \cdot p(G) \cdot \text{size}(G))$$
$$= O(f(G) \cdot p(G)\, \text{size}(G) \cdot 2^{(\text{max-rank}(G)+1)^2}).$$

To Execute the algorithm in the form given above we need a bit vector of size $2^{\text{max-rank}(G)} \cdot f(G)$ to store the mark bits. Hence the space requirement is

$$O(f(G) \cdot 2^{\text{max-rank}(G)} + \text{size}(G)). \quad \blacksquare$$

Putting Lemmas 1, 2, and 3 together yields the following algorithm for MLL(k) testing

for every rule r of G *do*
beg construct a grammar for L_r ;
  for every $x \in \Sigma^*$ with $|x| \leqslant k$ *do*
    if $|x| < k$ *then* construct a macro grammar for $L_r \cap \Sigma^*\bar{x}$ and determine if this language is empty;
    if $|x| = k$ *then* construct a macro grammar for $L_r \cap \Sigma^*\bar{x}\bar{\Sigma}^*$ and determine if this language is empty;
end

A finite automaton for the language $\Sigma^*\bar{x}(\Sigma^*\bar{x}\bar{\Sigma}^*)$ has $|x|$ states. Hence we infer the following time and space bounds from our preceding lemmas

$$\text{time: } O(|\Sigma|^{k+1} \text{ max-rank}(G)^{11} \text{ size}(G)^5 \, 2^{11 \cdot k^2 \cdot \text{max-rank}(G)^2})$$

and

$$\text{space: } O(k^2 \cdot \text{max-rank}(G)^3 \cdot \text{size}(G) \cdot 2^{3k^2 \text{max-rank}(G)}). \quad \blacksquare$$

COROLLARY. *Given an arbitrary macro grammar G, we can test if G is* MLL(1) *in time*

$$O(|\Sigma|^2 \cdot \text{max-rank}(G)^{11} \text{ size}(G)^5 \cdot 2^{11 \cdot \text{max-rank}(G)^2})$$

and space

$$O(\text{max-rank}(G)^3 \cdot \text{size}(G) \cdot 2^{3 \text{ max-rank}(G)}).$$

Since max-rank(G) might be on the order of size(G) the running time of our decision procedure is exponential in size(G). However, for grammars of bounded rank, the running time is polynomial in size(G) and the space requirement is linear in size(G). Next we show that this inefficiency is inherent to the problem.

THEOREM 4. *Every algorithm which tests if an arbitrary macro grammar is* MLL(1) *takes time $c^{\text{size}(G)}$ for some constant c and space* size$(G)^{2-\epsilon}$ *for every $\epsilon > 0$ infinitely often.*

Proof. We use the following fact from Hunt and Rosenkranz.

Fact. Every algorithm which decides $L(G) = \varnothing$ for arbitrary macro grammars G takes time $c^{\text{size}(G)}$ for some constant c and space size$(G)^{2-\epsilon}$ for every $\epsilon > 0$ infinitely often.

We reduce the emptiness problem to MLL(1) testing. The following trivial MLL(1) grammar generates Σ^*

$$S_0 \to \epsilon \mid aS_0 \mid bS_0 \mid \cdots.$$

Let $G = (\Sigma, \mathscr{F}, \mathscr{V}, \rho, S, P)$ be a macro grammar with $S_0, S' \notin \mathscr{F}$. Consider

$$\begin{aligned} G' &= (\Sigma, \mathscr{F}', \mathscr{V}, \rho', S', P') && \text{with } \mathscr{F}' = \mathscr{F} \cup \{S_0, S'\} \\ \rho'(F) &= \rho(F) && \text{if } F \in \mathscr{F}, \\ &= 0 && \text{if } F = S_0 \text{ or } F = S', \\ P' &= P \cup \{S' \to S_0 \mid S\} \cup \{S_0 \to \epsilon \mid aS_0 ; a \in \Sigma\}. \end{aligned}$$

Then G' is MLL(1) if and only if $L(G) = \varnothing$. Furthermore $\text{size}(G') = \text{size}(G) + O(|\Sigma|) = O(\text{size}(G))$. Since $L(G) = \varnothing$ may be tested by constructing G' and testing it for the MLL(1) property, MLL(1) testing takes time $c^{\text{size}(G)}$ and space $\text{size}(G)^{2-\epsilon}$ for some c and every $\epsilon > 0$ infinitely often. ∎

5. Macro Grammars, Indexed Grammars, and Alternate Definitions of the LL-Property

In Section 1 we proposed a definition of macro-LL grammars. Our definition corresponds to strong LL context-free grammars. The following definition corresponds to context-free LL grammars.

Let

$$\begin{aligned} \text{First}_k(r) = \{(w, u);\ S \overset{*}{\to} wF(\tau_1, \ldots, \tau_{\rho(F)})\tau' \to w\tau[\tau_1/x_1, \ldots, \tau_{\rho(F)}/x_{\rho(F)}]\tau' \to wuv, \\ \text{where } w, u, v \in \Sigma^*, |\mathscr{U}| \leqslant k \text{ and } |\mathscr{U}| < k \text{ implies } k = \epsilon\}. \end{aligned}$$

A grammar is called "general" MLL(k) if for every pair r_1, r_2 of rules having the same left-hand sides

$$\text{First}_k(r_1) \cap \text{First}_k(r_2) = \varnothing.$$

Open Problem. Is the class of "general" MLL(k) languages strictly larger than the class of MLL(k) languages? (The answer is no in the context-free case!)

Open Problem. Prove Theorems 3 and 4 for "general" MLL(k) grammars.

Weiss (1975) studied indexed LL grammars. He introduced γ-, β-, and α-indexed LL grammars (ILL). It is easy to see that α-ILL(k) grammars are equivalent to "general" MLL(k) grammars and that β-ILL(k) grammars are equivalent to MLL(k) grammars. γ-ILL(k) grammars are a very limited subclass of the β-grammars and do not even include all context-free LL grammars. γ-grammars were intensively studied by Weiss.

6. An Efficient MLL(k) Parser

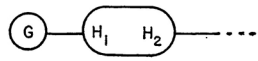
In this section we describe an efficient (time $O(n^2)$ and space $O(n^2)$) parsing method for ϵ-free MLL(k) grammars, i.e., MLL(k) grammars without ϵ-rules.

For the remainder of this section G is a standard form MLL(k) grammar without productions of the form

$$F(x_1, \ldots, x_1) \rightarrow \epsilon.$$

Open Problem. Characterize the class of languages generated by ϵ-free MLL(k) grammars.

To parse a string w, we construct a leftmost derivation in a top-down fashion. One major problem is to represent the intermediate sentential forms in a concise way. Note that application of a rule $F(x, y) \rightarrow G(H_1(x, y), H_2(x, y))$ may double the length of a sentential form, and therefore intermediate sentential forms may have length 2^n, where n is the length of the input string. Note however, that H_1 and H_2 have the *same* arguments and that it should be possible to write them down only once. We might represent $G(H_1(x, y), H_2(x, y))$ as



meaning that G has two arguments H_1, H_2 which in turn have arguments .... More generally, we represent sentential forms as ordered tree (cf. Aho (1968, 1969)). Let T be an ordered tree whose nodes are labeled by strings of function symbols. We draw trees with their root to the right and branches growing to the left. The branches are ordered from top to bottom. Let v be any node in T, let $v_0, v_1, \ldots, v_k = v$ be the path from the root of T to node v and let F be any function symbol in the label of v. Then (that occurrence of) F represents the following term with respect to T.

If $k = 0$ then F represents the term F; if $k > 0$ then let $A_1 \cdots A_m$ be the label of the father v_{k-1} of v and let $\tau_1, \ldots, \tau_m$ be the terms represented by $A_1, \ldots, A_m$ respectively. Then F represents $F(\tau_1, \ldots, \tau_m)$.

We are now ready to define the sentential form represented by a tree T together with a pointer p (the active function symbol pointer). p always points to some function symbol in some node v to T. If p points to the rightmost function symbol in the uppermost leaf of T then (T, p) represents whatever that function symbol represents. If p points to a function symbol A_i in some leaf v of T labeled $A_1 \cdots A_i A_{i+1} \cdots A_k$ with $i < k$, then (T, p) represents the term represented by A_i followed by the term (T, p'), where p' points to A_{i+1}. If p points to the rightmost function symbol F in some leaf node v of F, let v' be the leaf immediately above v and let p' point to the first function symbol in v'. Then (T, p) represents the term represented by F followed by the term represented by (T, p'). Finally if p points to function symbol F in a nonleaf node v then let p' point to the left-most function symbol in the downmost leaf of T. Then (T, p) represents the term represented by F followed by the term (T, p').

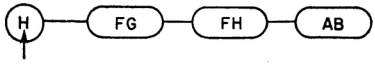
EXAMPLE. Consider the following grammar

$$F(x, y) \to y, \tag{1}$$
$$H(x, y) \to H(F(x, y), G(x, y)), \tag{2}$$
$$H(x, y) \to H(F(x, y), G(x, y)), \tag{3}$$
$$B \to b, \tag{4}$$

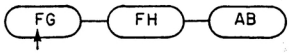
and the sentential form

$$H(F(F(A, B), H(A, B)), G(F(A, B), H(A, B)))$$

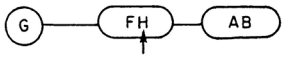
which we write as



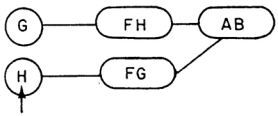
The arrow ↑ indicates the active function symbol pointer. Applying rule (2) yields the term $F(F(A, B), H(A, B)) \cdot G(F(A, B), H(A, B))$ which we write as



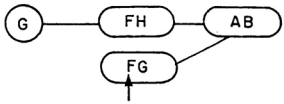
Applying (1) yields $H(A, B) \cdot G(F(A, B), H(A, B))$ which we draw as



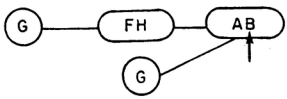
Applying rule (3) now yields $H(F(A, B), G(A, B)) \cdot G(\cdots)$ which we draw as



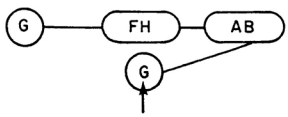
Applying (2) again yields $F(A, B) \cdot G(A, B) \cdot G(\cdots)$,



then (1) yields $B \cdot G(A, B) \cdot G(\cdots)$

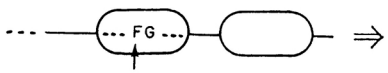


and finally (4) yields $G(A, B) \cdot G(\cdots)$.

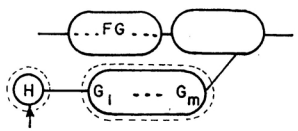


It should be clear from this example that the application of standard form rules corresponds to simple changes of the tree structure. More precisely:

(1) $F(\,) \to H(G_1(\,), \ldots, G_m(\,))$ corresponds to

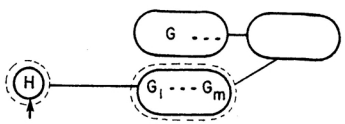


(a)



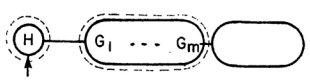
if F is in a nonleaf node,

(b)



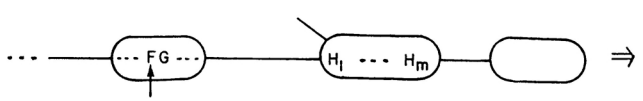
if F is the leftmost function symbol in a leaf and G exists,

(c)

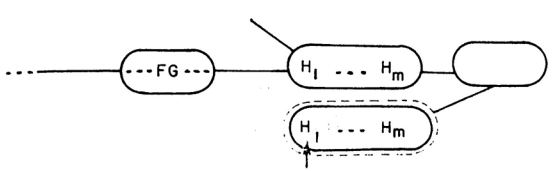


if F is the only function symbol in a leaf.

(2) $F(x_1 \cdots x_m) \to x_1 \cdots x_m$ corresponds to

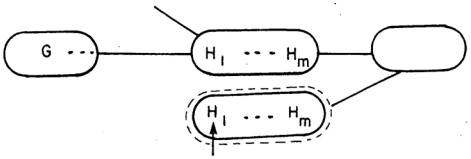


(a)



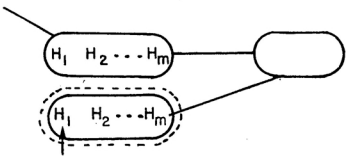
if F is in a nonleaf node,

(b)



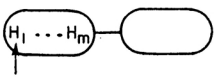
if F is the leftmost function symbol in a leaf and G exists,

(c1)



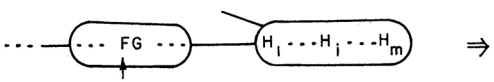
if F is the only function in a leaf which is not the single son of its father,

(c2)

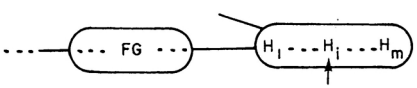


if F is the only function symbol in a leaf which is the single son of its father.

(3) $F(x_1, \ldots, x_m) \to x_i$ corresponds to

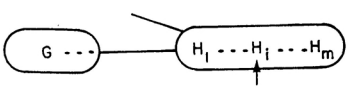


(a)



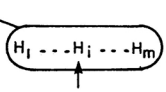
if F is in a nonleaf node,

(b)



if F is the leftmost function symbol in a leaf and G exists,

(c1)



if F is the only function symbol in a leaf which is not the single son of its father,

(c2)

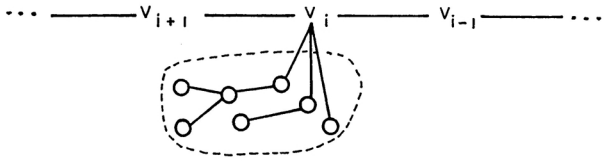
if F is the only function symbol in a leaf which is the single son of its father.

(4) $F(x_1 \cdots x_m) \to a$ corresponds to:

If F is the leftmost function symbol in a leaf then erase F. If the leaf is labeled by the empty string after erasing F then delete the leaf. In any case position the active pointer at the leftmost function symbol of the downmost leaf.

Rule (4) is applied exactly n times, where n is the length of the input. Whenever rule (4) is applied the active pointer is positioned at the leftmost function symbol of the downmost leaf. Between application of rule (4) the active pointer moves right on the downmost branch (rules (2c2), (3a), (3b), (3c1)). From time to time new subtrees are grown (rules (1a), (ab), (1c), (2a), (2b), (2c2), (3c2), i.e., new nodes are constructed. The newley constructed nodes are doubly circled in the diagrams above. Also some old nodes may be discarded (rules (1c), (2c), (3c2)). Suppose the active pointer points at node v and we construct new nodes. Then the new nodes will be descendants of v's father but they are not descendants of v. So if a node becomes active then it will not receive new descendants until the next application of rule (4).

Consider any application of rule (4). Let T be the tree after application of rule (4) and let v_k, v_{k-1},..., v_0 be the downmost branch in T; v_k is the leaf and v_0 is the root. Then the active pointer points at v_k. Let $T = T_0, T_1, T_2, T_3, \ldots$ be the sequence of trees constructed by the parsing algorithm until the next application of rule (4). We refer to all nodes of T_0 as "old" nodes and to every node of T_i, $i \geqslant 1$, which was not a node of T_0 as a "new" node. For every new node v there is some i, $k - 1 \geqslant i \geqslant 0$, such that v is a descendant of v_i but not of v_{i+1}. We refer to v as a new node in the forest grown at v_i.

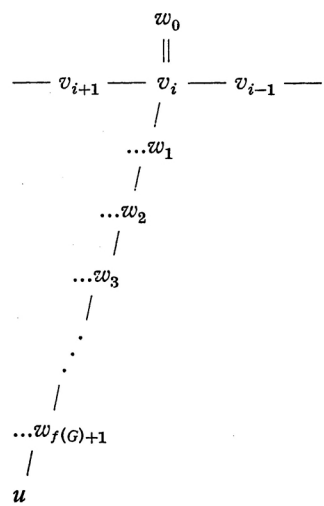


We want to show that for every j, $j \geqslant 0$, and i, $0 \leqslant i \leqslant k - 1$, the forest grown at v_i in T_j contains at most

$$d := [3f(G)^{(\text{max-rank}(G)+1)} \text{ max-rank}(G)]^{f(G)+1} \text{ nodes.}$$

Assume otherwise. Then let j be minimal such that the forest grown at v_i in T_j contains $>d$ nodes. If a forest contains many nodes then it either contains a long path or two paths are identical.

Case 1. There is a path of length $f(G) + 2$ in the forest grown at v_i in T_j. Since there was no such path in T_{j-1} the last node of that path was constructed by the last application of a rule, the active pointer points at the leaf u of that path and the path is downmost.



Let $u, w_{f(G)+1}, \ldots, w_1, w_0 = v_i$ be that path, u is the active node. Let F_l, $1 \leqslant l \leqslant f(G) + 1$ be the active function symbol (the function symbol pointed at by the active pointer) immediately before the construction of w_l. F_l is part of the label of a son of w_{l-1}. (This son may be deleted in the process of constructing w_l.) All w_l's, $l \geqslant 1$, are constructed by applications of rules of type (1). Otherwise, w_l would be active after its creation and therefore not receive additional descendants. There are l, h such that $l < h$ and $F_l = F_h$. Since the parsing algorithm is deterministic the same rule of type (1) is applied to F_l and F_h. It is now easy to see that the parsing algorithm cycles and constructs an infinite path.

Case 2. All paths in the forest grown at v_i in T_j have length $\leqslant f(G) + 1$. Since all node labels are strings of function symbols of length $\leqslant$max-rank(G), there are $\leqslant h := f(G)^{\text{max-rank}(G)+1}$ different node labels. Since the forest grown at v_i in T_j has more than d nodes there exists a node v (either v_i or a new node in the forest grown at v_i) which has $>3 \cdot h \cdot$ max-rank(G) sons. Let $w_1, w_2, w_3, \ldots$ be the sons of v ordered from top to bottom. When w_l, $l \geqslant 2$, was constructed either a function symbol in w_{l-1} was active (rule of type (1)) or a function symbol in the downmost son of w_{l-1} was active (rule of type (2)) and

then w_l has the same label as w_{l-1} or a son of a lateron discarded son of v was active (rule (3c2)). Since v has $>3 \cdot h \cdot \text{max-rank}(G)$ sons there exists $l < g$ such that w_l has the same label as w_g and is constructed by the application of the same rule. Hence the parsing algorithm cycles and grows an infinite forest at v_i.

This shows that the size of the forest grown at v_i is bounded by the constant d; otherwise the parsing algorithm loops. Hence there is a constant d' ($\geqslant d$) such that either the algorithm loops indefinitely on the forest grown at v_i or $\leqslant d'$ tree manipulations are used to grow the forest at v_i. Hence the depth of the tree can only grow by a constant between scanning proper inputs and the depth of the tree after reading the ith input symbol is $O(i)$. Furthermore, only $O(i)$ time units are spent between the $(i-1)$st and ith application of a rule of type (4). From this we conclude that the parser has time and space complexity $O(n^2)$.

THEOREM 5. *There is a deterministic parser for ϵ-free* MLL(k) *grammars which works in time and space* $O(n^2)$.

Open Problem. How about MLL(k) grammars with ϵ-productions.

In Section 2 we described the construction of a MLL(1) grammar for each deterministic context-free language. The parser constructed above works in linear time for these MLL(1) grammars and so it does for the grammar for $\{a^n b^n c^n;\ n \geqslant 1\}$ given in the Introduction.

Open Problem. Characterize the linear time parsable MLL(1) grammars.

RECEIVED: July 8, 1977; REVISED: June 1, 1978

REFERENCES

AHO, A. V. (1968), Indexed grammrs—an extension of context-free grammars, *J. Assoc. Comput. Mach.* **15**, 647–671.

AHO, A. V. (1969), Nested stack automata, *J. Assoc. Comput. Mach.* **16**, 383–406.

AHO, A. V., AND ULLMAN, J. D. (1972), "The Theory of Parsing, Translation and Compiling," Series in Automatic Computation, Prentice–Hall, Englewood Cliffs, N. J.

FISCHER, M. (Ed.), Grammars with macro-like instructions, 9th SWAT conference, 1968.

GRIES, D. (1971), "Compiler Construction for Digital Computers," Addison–Wesley, Reading, Mass.

KNUTH, D. E. (1967), Top-down syntax analysis, Lecture Notes, International Summer School on Computer Programming, Copenhagen, Denmark.

LEWIS, P. M. II, AND STEARNS, R. E. (1968), Syntax directed transduction. *J. Assoc. Comput. Mach.* **15**, 464–488.

NIVAT, M., "On the Interpretation of Recursive Program Schemes," IRIA Rapport Laboria 84, 1974.

ROSENKRANTZ, D. J., AND STEARNS, R. E. (1970), Properties of deterministic top-down grammars, *Inform. Contr.* **17**, 226–256.

WEISS, K. (1975), "Deterministische indizierte Grammatiken, 2te Kolloquium über Automatentheorie und formale Sprachen, Kaiserslautern, 1975," Lecture Notes in Computer Science, Vol. 33; also Ph. D. thesis, University of Karlsruhe, 1976.