TOP DOWN PARSING OF MACRO GRAMMARS

(Preliminary Report)

by

Manfred Heydthausen und Kurt Mehlhorn

Fachbereich 10 -
Angewandte Mathematik
und Informatik der
Universität des Saarlandes
D-6600 Saarbrücken

März 1976

A 76/3

Recursive descent is for its ease of description and for its transparency one of the popular parsing methods [Gries, Knuth]. 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 & Stearns, Rosenkranz & Stearns and many others (see [Aho & Ullman] for complete references).

In the late 60's several extensions of context-free languages were proposed in order to cope with the non context-free features of programming languages (e.g. applied and defining occurences of identifiers). Two remarkable examples are the macro languages of Fischer [Fischer] and the indexed languages of Aho [Aho 68]. Because of the lack of efficient parsing methods for these classes of grammars, they were never used in actual programming language design.

Weiß [Weiß] 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 I we introduce macro grammars and formulate the LL property for macro grammars. In section II we give first evidence for the power of MLL languages: every deterministic context-free language is generated by an MLL grammar. In section III we show that transformation to standard form can be done whilst preserving the LL property. In section IV we show that it is decidable whether an arbitrary macro-grammar is MLL(k) for a fixed k. Our decision procedure has time complexity $O(2^{(1+\varepsilon)n^2})$ and space complexity $O(2^{(1+\varepsilon)n})$ for some $\varepsilon > 0$ where n is the size of the grammar. We also show that c^n $(n^{2-\varepsilon})$ for some constant $c > 1$ is a lower bound for the time (space) complexity of MLL(1) testing. In section V we review Weiß's definition of indexed LL grammars, show that it is far too restrictive and then give a new (more general) definition for ILL grammars. Then we show the equivalence of MLL and ILL languages. In section VI we give an automata-theoretic characterization of the class of MLL languages and show that MLL languages can be parsed in time $O(n^2)$ and space $O(n)$ where n is the length of the input. Finally we give some examples of MLL grammars.

I. Macro grammars, LL property

A macro grammar [Fischer] is a 6-tuple $(\Sigma, \mathcal{F}, \mathcal{V}, \rho, S, P)$ where:
Σ is a finite set of terminal symbols;
$\mathcal{F}$ is a finite set of non-terminal or function symbols;
$\mathcal{V}$ is a finite set of argument or variable symbols,
ρ is a function from $\mathcal{F}$ 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
$F(x_1,\dots,x_{\rho(F)}) \to \tau$ where $F \in \mathcal{F}$, $x_1,\dots,x_{\rho(F)}$ are distinct members of $\mathcal{V}$, and τ is a term over Σ , $\{x_1,\dots,x_{\rho(F)}\}$, $\mathcal{F}$, ρ.

The set of terms over $\Sigma, \mathcal{V}, \mathcal{F}, \rho$ is defined inductively,

a) ε is a term,
   a is a term for every $a \in \Sigma$.
   x is a term for every $x \in \mathcal{V}$.

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

c) if $F \in \mathcal{F}$ and $\tau_1,\dots,\tau_{\rho(F)}$ are terms, then $F(\sigma_1,\dots,\sigma_{\rho(F)})$ is a term.

We consider macro grammars with the outside-in (OI) mode of derivation [Fischer, Nivat], i.e. only top-level occurrences of function symbols can be rewritten at every step. Instead of giving a formal definition of this mode of derivation, we give an example.

<u>Example:</u> A macro grammar generating $\{a^n b^n c^n; n \geq 0\}$

$$
\begin{array}{lll}
S & \to & F(\varepsilon,\varepsilon) \\
F(x,y) & \to & A\ F(xB,yC) \\
F(x,y) & \to & xy \\
A & \to & a \\
B & \to & b \\
C & \to & c
\end{array}
$$

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

$$
\begin{array}{ccccc}
S \to F(\varepsilon,\varepsilon) \to & A\ F(B,C) & \to & A\ A\ F(BB,CC) & \\
& \downarrow & & \downarrow & \\
& a\ F(B,C) & & AA\ BB\ CC & \\
& \downarrow & & \downarrow & \\
& aA\ F(BB,CC) & \to & aA\ BB\ CC & \\
& & & \downarrow^{*} & \\
& & & a^2\ b^2\ c^2 &
\end{array}
$$

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

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

```
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: write xy as τ_1·τ_2 ...τ_k
                  where τ_i is a term starting with a function
                  symbol;
                  for i from 1 to k do call τ_i;
          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 appropiate production. This strategy is possible whenever the decision between the different alternatives for a function symbol can be made on the bases of knowing the next (the next k for some fixed k) input symbol. This leads to the following definition.

Definition:

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

$$\mathrm{First}_k(r) = \{ u;\ S \overset{*}{\underset{lm}{\to}} w\, F(\tau_1,\dots,\tau_{\varrho(F)})\, \tau' \underset{lm}{\to} w\, \tau[{}^{\tau_1}/_{x_1},\dots,{}^{\tau_{\varrho(F)}}/_{x_{\varrho(F)}}]\, \tau' \overset{*}{\underset{lm}{\to}} w\,u\,v\ ;\ w,u,v \in \Sigma^*, \tau',\tau_1,\dots,\tau_{\varrho(F)} \text{ are terms, } |u| = k \text{ or } |u| < k \text{ and } v = \varepsilon \}$$

$\tau[{}^{\tau_1}/_{x_1},\dots,{}^{\tau_{\varrho(F)}}/_{x_{\varrho(F)}}]$ is the term obtained by replacing x_i by τ_i, $1 \le i \le \varrho(F)$, 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:

$$\mathrm{First}_k\,(r_1) \cap \mathrm{First}_k\,(r_2) = \emptyset.$$

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

Our example grammar is MLL(1).

II. The power of MLL grammars

In this section we show that MLL-grammars generate a proper superset of the deterministic context-free languages.

Thm. 1: Given any deterministic pushdown automaton A, we can find an equivalent MLL grammar G, i.e. $L(A)\dashv = L(G)$, where $\dashv$ is the end marker.

Proof: Let $A = (S, \Sigma, \Gamma, q_o, Z_o, F)$ be a deterministic pushdown automaton (accepting by final state). $S = \{q_o, \ldots, q_n\}$ is the set of states, Σ the input alphabet, Γ the stack alphabet, q_o the start state, Z_o the symbol initially placed at the bottom of the pushdown store and F is the set of final states. We may assume w.l.o.g. that A writes at most two symbols onto the stack in a single move.

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

(1) $START \rightarrow [q_o, Z_o] \underbrace{(\varepsilon, \varepsilon, \ldots, \varepsilon)}_{|S|\text{-times}}$

(2) for $a \in \Sigma \cup \{\varepsilon\}$ and $\delta(q,a,A) = (q_i, \varepsilon)$

$[q,A]\ (x_o, \ldots, x_n) \rightarrow a x_i$

(3) for $a \in \Sigma \cup \{\varepsilon\}$ and $\delta(q,a,A) = (q_i, B)$

$[q,A]\ (x_o, \ldots, x_n) \rightarrow a[q_i, B]\ (x_o, \ldots, x_n)$

(4) for $a \in \Sigma \cup \{\varepsilon\}$ and $\delta(q,a,A) = (q_i, BC)$

$[q,A](x_o, \ldots, x_n) \rightarrow$

$a[q_i,B]([q_o,C](x_o,\ldots,x_n),[q_1,C](x_o,\ldots,x_n),\ldots,[q_n,C]\ (x_o,\ldots,x_n))$

(5) for all $q \in F$ and $A \in \Gamma$

$$[q,A](x_o,\ldots,x_n) \rightarrow \dashv$$

The correctness of the construction follows from the following claim which is proved by induction on the length of the computation (derivation).

<u>Claim:</u> Let $x_1,\ldots,x_n \in \Sigma$, Z_o, $Z_1,\ldots,Z_j \in \Gamma$ and $q \in S$.

Then

$(q_o, x_1 \ldots x_n, Z_o) \vdash^{*} (q, \varepsilon, Z_j \ldots Z_1)$

iff

$$\text{START} \overset{*}{\rightarrow} x_1 \ldots x_n [q,Z_j] (\text{expansion of } Z_{j-1} \ldots Z_1)$$

where

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

and

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

The macro grammar G is MLL(1) since A is deterministic.

<u>Corollary:</u> The class of MLL(1) languages properly contains the deterministic context-free languages.

III. 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

(1) $F(x_1,\ldots,x_n) \to G(H_1(x_1,\ldots,x_n),\ldots,H_m(x_1,\ldots,x_n))$ with $n, m \geq 0$

(2) $F(x_1,\ldots,x_n) \to x_1 \ldots x_n$, $n \geq 0$;

(3) $F(x_1,\ldots,x_n) \to x_i$, $n \geq 0$, $1 \leq i \leq n$;

(4) $F(x_1,\ldots,x_n) \to a$ for $a \in \Sigma \cup \{\varepsilon\}$, $n \geq 0$

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 will 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 a 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.

Thm.: Given any MLL(k) grammar G, we can find an equivalent standard form MLL(k) grammar G' with f(G') $\leq$ size(G), p(G') $\leq$ size(G), size(G') $\leq$ O(max-rank(G)$\cdot$size(G)) and max-rank(G') $\leq$ max-rank(G).

Proof: (sketch)
The transformation is done in two steps. In step 1 we add a new 0-ary function symbol A for every $a \in \Sigma \cup \{\varepsilon\}$, add the rules $A \to a$ and replace all occurences of terminal symbols in the RHS of productions by their respective nonterminal.

This leaves us with rules

$$F(x_1,\dots,x_n) \rightarrow \tau \quad , n \geq 0$$

and

$$F(x_1,\dots,x_n) \rightarrow a$$

where τ is a term over $\mathcal{F}$, $x_1,\dots,x_n$ and $a \in \Sigma \cup \{\varepsilon\}$. In step 2 we break up the terms on the right hand sides by the following process.

Let $F(x_1,\dots,x_n) \rightarrow \tau$ be a production being not in standard form. Then $\tau = \tau_1 \dots \tau_k$ for some $k > 0$, where $\tau_i = F_i(\quad)$ or $\tau_i = x_j$ for some j.

Case 1: $k = 1$: Then $\tau = G(\tau'_1,\dots,\tau'_m)$ for some m. We delete $F(\;) \rightarrow \tau$ from the set of productions and add $F(x_1,\dots,x_n) \rightarrow G(H_1(x_1,\dots,x_n),\dots,H_m(x_1,\dots,x_n))$ and $H_i(x_1,\dots,x_n) \rightarrow \tau'_i$ where the H_i's are new function symbols.

Case 2: $k > 1$: Then $\tau = \tau_1 \dots \tau_k$. We remove $F(x_1,\dots,x_n) \rightarrow \tau$ from the set of productions and add $F(x_1,\dots,x_n) \rightarrow G(H_1(x_1,\dots,x_n),\dots,H_k(x_1,\dots,x_n))$ and $G(x_1,\dots,x_n) \rightarrow x_1 \dots x_n$ and $H_i(x_1,\dots,x_n) \rightarrow \tau_i$ where G and H_i are new function symbols $(1 \leq i \leq k)$.

We iterate the process described above until all rules are in standard form. Single rules of G correspond to packages of rules of G'. The derivations according to G and G' are in a 1-1 correspondence. Hence the MLL(k) property is preserved.

Example: We transform our example grammar from section 1 into standard form. The rules $S \rightarrow F(\varepsilon,\varepsilon)$ and $F(x,y) \rightarrow AF(xB,yC)$ are not in standard form. The first rule is transformed into $S \rightarrow F(E,E)$ and $E \rightarrow \varepsilon$ in step (1) and the second rule is transformed in step (2) into

(1) $F(x,y) \rightarrow Conc(H_1(x,y),H_2(x,y))$

(2) $H_1(x,y) \rightarrow A$

(3) $H_2(x,y) \rightarrow F(xB,yC)$

(4) $Conc(x,y) \rightarrow xy$

Rules (1), (2) and (4) are in standard form, rule (3) is transformed into

(5) $H_2(x,y) \rightarrow F(H_4(x,y), H_5(x,y))$

(6) $H_4(x,y) \rightarrow xB$

(7) $H_5(x,y) \rightarrow yC$

Then (6) is transformed into

(8) $H_4(x,y) \rightarrow Conc(H_6(x,y),H_7(x,y))$

(9) $H_6(x,y) \rightarrow x$

(10) $H_7(x,y) \rightarrow B$

and (7) is transformed analogously. We end up with the following standard form grammar:

$S \rightarrow F(E,E)$

$E \rightarrow \varepsilon$

$F(x,y) \rightarrow Conc(H_1(x,y), H_2(x,y)) \quad | \; xy$

$H_1(x,y) \rightarrow A$

$H_2(x,y) \rightarrow F(H_4(x,y),H_5(x,y))$

$H_4(x,y) \rightarrow Conc(H_6(x,y), H_7(x,y))$

$H_6(x,y) \rightarrow x$

$H_7(x,y) \rightarrow B$

$H_5(x,y) \rightarrow Conc(H_8(x,y), H_9(x,y))$

$H_8(x,y) \rightarrow y$

$H_9(x,y) \rightarrow C$

$Conc(x,y) \rightarrow x \cdot y$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$

IV. Testing for the LL(k) property

In this section we will show that it is decidable if an arbitrary macro grammar is MLL(k). We assume w.l.o.g. that all macro grammars are in standard form.

Given a macro grammar G (in standard form) and a rule $r = F(x_1,\dots,x_n) \to \tau$ of this grammar we want to compute $First_k(r)$. 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

$$L_r = \{ a_1 \dots a_m \bar{a}_{m+1} \dots \bar{a}_n ; S \overset{*}{\to} a_1 \dots a_m \; F(\tau_1,\dots,\tau_p)\tau'$$

$$\to a_1 \dots a_m \tau[{}^{\tau_1}/_{x_1},\dots,{}^{\tau_p}/_{x_p}]\; \tau'$$

$$\overset{*}{\to} a_1 \dots a_m \; a_{m+1} \dots a_n \in L(G)\}$$

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

if $|x| < k$ then $x \in First_k(r)$ iff $(L_r \cap \Sigma^* \;\; \bar{x}) \neq \emptyset$

if $|x| = k$ then $x \in First_k(r)$ iff $(L_r \cap \Sigma^* \; \bar{x} \; \bar{\Sigma}^*) \neq \emptyset$

Since the class of macro languages is closed under intersection with a regular set and their emptiness problem is decidable [Fischer] 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 will provide us with a tighter time/bound.

<u>Lemma 1:</u> Given a macro grammar G and a production r, we can find a macro grammar G_r generating L_r with max-rank$(G_r) \leq 3\cdot$max-rank(G), size$(G_r) \leq (3+$max-rank(G)$) \cdot$ size(G), $f(G_r) \leq 3\cdot f(G)$ and $p(G_r) \leq (3+$max-rank(G)$) \cdot p(G)$.

<u>Proof:</u> For every function symbol F in G there are function symbols F^{Σ}, $F^{\bar{\Sigma}}$ and F^{mixed} in G_r having arity $\varrho(F)$, $\varrho(F)$ and $3\cdot\varrho(F)$ respectively. For every rule in G the following rules are in G_r:

(1) if the rule is of the form $F(x_1,\ldots,x_n) \to H(H_1(x_1,\ldots,x_n),\ldots,H_k(x_1,\ldots,x_n))$ then

$$F^{\Sigma}(\ ) \to H^{\Sigma}(H_1^{\Sigma}(\ ),\ldots,H_k^{\Sigma}(\ ))$$

$$F^{\bar{\Sigma}}(\ ) \to H^{\bar{\Sigma}}(H_1^{\bar{\Sigma}}(\ ),\ldots,H_k^{\bar{\Sigma}}(\ ))$$

$$F^{mixed}(x_1^{\Sigma},\ x_1^{\bar{\Sigma}},\ x_1^{mixed},\ldots,x_n^{\Sigma}\ ,\ x_n^{\bar{\Sigma}},\ x_n^{mixed})$$

$$\to H^{mixed}(H_1^{\Sigma}(x_1^{\Sigma},\ldots,x_n^{\Sigma}),\ H_1^{\bar{\Sigma}}(x_1^{\bar{\Sigma}},\ldots,x_n^{\bar{\Sigma}}),H_1^{mixed}(x_1^{\Sigma},x_1^{\bar{\Sigma}},x_1^{mixed}\ ,\ldots),\ldots)$$

(2) if the rule is of the form $F(x_1,\ldots,x_n) \to x_1 \ldots x_n$ then

$$F^{\Sigma}(x_1,\ldots,x_n) \to x_1 \ldots x_n$$
$$F^{\bar{\Sigma}}(x_1,\ldots,x_n) \to x_1 \ldots x_n$$
$$F^{mixed}(x_1^{\Sigma},x_1^{\bar{\Sigma}},x_1^{mixed},\ldots) \to x_1^{\Sigma} \ldots x_{i-1}^{\Sigma}\ x_i^{mixed}\ x_{i+1}^{\bar{\Sigma}} \ldots x_n^{\bar{\Sigma}}$$

for every i with $1 \leq i \leq n$

(3) if the rule is of the form $F(x_1,\ldots,x_n) \to x_i$ then

$$F^{\Sigma}(x_1,\ldots,x_n) \to x_i$$

$$F^{\bar{\Sigma}}(x_1,\ldots,x_n) \to x_i$$

$$F^{mixed}(x_1^{\Sigma},x_1^{\bar{\Sigma}},x_1^{mixed},\ldots\ ) \to x_i^{mixed}$$

(4) if the rule is of the form $F(x_1,\dots,x_n) \to a$ for $a \in \Sigma \cup \{\varepsilon\}$ then

$F^{\Sigma}(x_1,\dots,x_n) \to a$

$F^{\bar{\Sigma}}(x_1,\dots,x_n) \to \bar{a}$

(5) and finally if $F(x_1,\dots,x_n) \to \tau$ is the rule r we add

$F^{mixed}(x_1^{\Sigma},x_1^{\bar{\Sigma}},x_1^{mixed},\dots) \to \bar{\tau}$

where $\bar{\tau}$ is obtained from τ by adding the superscript $\bar{\Sigma}$ to all symbols. The start symbol of G' is S^{mixed}.

Note that only function symbols of the form F^{Σ} and $F^{\bar{\Sigma}}$ have terminal rules and that rule (5) is the only rule with a function symbol F^{mixed} on the left hand side and no "mixed" symbol on the right hand side. Therefore rule (5) has to be used to get rid off the mixed function symbols. Keeping this in mind the reader should have no difficulties in verifying the assertions made in the theorem.

Lemma 2: Given any macro grammar G and a deterministic finite automaton A with s states, we can find a macro grammar G' with $L(G') = L(G) \cap L(A)$ and $\text{max-rank}(G') = s^2\ \text{max-rank}(G)$,

$\text{size}(G') \le s^{\text{max-rank}(G)} \cdot \text{size}(G), f(G') \le s^2 f(G)$ and

$p(G) \le s^{\text{max-rank}(G)} \cdot p(G)$

Proof: Similar to the proof ot theorem 1.

Lemma 3: Given any macro grammar G, we can decide $L(G) \neq \emptyset$ in time

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

Proof: We proceed in three steps

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

Step (1) does neither increase the maximal rank nor the size.

(2) Eliminiate ε-rules by the following process:

while there is a rule of the form $F(x_1,\dots,x_n) \to \varepsilon$ with F not being the start symbol

do apply the rule $F(x_1,\dots,x_n) \to \varepsilon$ to the right-hand sides of all productions in G (even if the occurrence of F is not at the top-level) and delete all rules having F( ) as their left-hand side.

We are now left with a grammar G' all of whose rules are of the form:

$S \to \varepsilon$, where S is the start symbol;

$F(x_1,\dots,x_n) \to H(\tau_1,\dots,\tau_k)$ with $\tau_i = \varepsilon$ or $\tau_i = H_i(x_1,\dots,x_n)$

$F(x_1,\dots,x_n) \to x_1 \dots x_n$

$F(x_1,\dots,x_n) \to x_i$

Apparently $\varepsilon \in L(G')$ iff $\varepsilon \in L(G)$. Step (2) does neither increase maximal rank nor size. If $S \to \varepsilon$ is a rule of G' then $L(G') \neq \emptyset$. Otherwise we go to step (3).

(3) At this point an example might be useful. We apply step (1) and (2) to the standard form grammar of section 3. In step (1) we replace the rules $A \to a$, $B \to b$ and $C \to c$ by $A \to \varepsilon$, $B \to \varepsilon$, $C \to \varepsilon$, and in step (2) we get the rules

$S \to F(\varepsilon,\varepsilon)$

$F(x,y) \to Conc(\varepsilon,H_2(x,y))$

$F(x,y) \to xy$

$H_2(x,y) \to F(H_4(x,y),H_5(x,y))$

$H_4(x,y) \to Conc(H_6(x,y),\varepsilon)$

$H_6(x,y) \to x$

$H_5(x,y) \to Conc(H_8(x,y),\varepsilon)$

$H_8(x,y) \to y$

$Conc(x,y) \to x \cdot y$

A sample derivation $S \overset{*}{\to} \varepsilon$ is:

$S \to F(\varepsilon,\varepsilon) \to \varepsilon \cdot \varepsilon$

In order to detect derivations of this form we have to determine for every function symbol $F(x_1,\dots,x_n)$ all subsets $J \subseteq \{1,\dots,n\}$ with $F(x_1,\dots,x_n) \overset{*}{\to} x_{i_1} \dots x_{i_m}$ and $J = \bigcup \{ i_l \}$. To do so we consider the pairs

(F,J) for $F \in \mathcal{F}$ and $J \subseteq \{1,\dots,\rho(F)\}$. We mark these pairs in an iterative process.: The pair (F,J) will be marked if and only if $F(x_1,\dots,x_{\rho(F)}) \overset{*}{\rightarrow} x_{i_1} \dots x_{i_m}$ with $J = \cup\{i_l\}$; then $\varepsilon \in L(G)$ iff $(S,\emptyset)$ is marked upon termination of the algorithm.

__for__ all rules of the form $F(\,) \rightarrow \tau$ where τ does not contain any function symbol

__do__ mark (F,J) where $J = \bigcup_{x_i \in \tau} \{i\}$;

__while__ there is a production $F(x_1,\dots,x_n) \rightarrow H_o(\tau_1,\dots,\tau_k)$ with

(1) (H_o,J_o) is marked,

(2) $J = \bigcup_{i \in J_o} J_i$ where

either $\tau_i = H_i(x_1,\dots,x_n)$ and (H_i,J_i) is marked

or $\tau_i = \varepsilon$ and $J_i = \emptyset$

(3) (F,J) is unmarked

__do__ mark (F,J).

__Claim:__ (F,J) is marked during this process iff $F(x_1,\dots,x_n) \overset{*}{\rightarrow} x_{i_1} \dots x_{i_m}$ with $J = \cup\{i_l\}$.

__Proof:__ the proof is similar to the proof of the corresponding claim in [Aho 68] and therefore left to the reader.

There are $\leq 2^{\text{max-rank}(G)} \cdot f(G)$ 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 $\leq (2^{\text{max-rank}(G)})(2^{\text{max-rank}(G)})^{\text{max-rank}(G)}$ possibilities of combining the J_i's, $1 \le i \le \rho(F)$. Each possibility may be examined in time O(size(G)). Hence the running time of the algorithm is bounded by

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

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

We execute the algorithm on our example grammar. In the initialisation phase the pairs $(F,\{1,2\})$, $(H_6,\{1\})$ $(H_8\{2\})$ and $(Conc,\{1,2\})$ are marked. During execution of the while-loop the following pairs are labelled in some order: $(S,\emptyset)$, $(H_4,\{1\})$, $(H_5,\{2\})$, $(H_2,\{1,2\})$.

Thm.: Given any macro grammar G and an integer k, we can test if G is MLL(k) in time $O(|\Sigma|^k \cdot 2^{(1+\varepsilon)k^4 \cdot \text{size}^2(G)})$ and space $O(2^{(1+\varepsilon)k^2 \cdot \text{size}(G)})$ for some $\varepsilon > 0$.

Proof: We compute $\text{First}_k(r)$ for every rule r of G using the strategy described at the beginning of the section.

for every rule r of G do
beg construct a grammar for L_r;
  for every $x \in \Sigma^*$ with $|x| \le k$ 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.

time: $O(k^{2+6\text{max-rank}(G)} \cdot |\Sigma|^k \cdot 2^{k^4 \cdot \text{max-rank}^2(G)} \cdot f(G) \cdot p(G)^2 \cdot \text{size}(G) \cdot \text{max-rank}(G)^2)$

space: $O(k^2 \cdot f(G) \cdot 2^{k^2 \cdot \text{max-rank}(G)})$

or easier to remember

time: $O(|\Sigma|^k \cdot 2^{(1+\varepsilon)\, k^4 \cdot \text{size}^2(G)})$

space: $O(2^{(1+\varepsilon) \cdot k^2 \cdot \text{size}\,(G)})$

for some $\varepsilon > 0$.

<u>Corollary:</u> Given an arbitrary macro grammar G , we can test if G is MLL(1) in time $O(2^{(1+\varepsilon)\ \text{size}^2(G)})$ for some $\varepsilon > 0$.

The running time of our decision procedure is exponential. We will show next that this inefficiency is inherent to our problem.

<u>Thm.:</u> Every algorithm which tests if an arbitrary macro grammar is MLL(1) takes time $c^{\text{size}(G)}$ for some constant c and space $\text{size}(G)^{2-\varepsilon}$ for every $\varepsilon > 0$ infinitely often.

<u>Proof:</u> We use the following fact from [Hunt & Rosenkrantz].

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

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

$$S_o \to \varepsilon \mid a\ S_o \mid b\ S_o \mid \ldots$$

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

$G' = (\Sigma, \mathfrak{F}', \mathcal{V}, \rho', S', P')$ with $\mathfrak{F}' = \mathfrak{F} \cup \{S_o, S'\}$

$$\rho'(F) = \begin{cases} \rho(F) & \text{if } F \in \mathfrak{F} \\ o & \text{if } F = S_o \text{ or } F = S' \end{cases}$$

$$P' = P \cup \{S' \to S_o | S\} \cup$$

$$\{S_o \to \varepsilon \; | aS_o \; ; \; a \in \Sigma\}$$

Then G' is MLL(1) if and only if $L(G) = \emptyset$. Furthermore size $(G') = \text{size}(G) + O(|\Sigma|) = O(\text{size}(G))$. Since $L(G) = \emptyset$ 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 $\text{size}(G)^{2-\varepsilon}$ for some c and every $\varepsilon > o$ infinitely often.

V. Macro Grammars and Indexed Grammars

Weiß [Weiß] introduced the notion of indexed LL(k) grammars. We give his definition for k = 1.

An indexed grammar [Aho 68] $G = (V,F,\Sigma,S,P)$ is ILL(1) if for every pair of distinct rules r_1, r_2 having the same left hand side the sets First(r) are disjoint, where

1) if $A \to \alpha \in f$ is an index production

$$\text{First } (A \to \alpha) = \{u;\ u \in \Sigma^*,\ |u| \leq 1:$$

$$\exists \delta,\ \gamma' \in (V \cup F \cup \Sigma)^*:$$

$$\left.\begin{array}{l}(a)\ l(u) < 1 \\ (b)\ Af\delta \Rightarrow \alpha\delta \overset{*}{\Rightarrow} u\gamma'\}\end{array}\right\} \gamma' = \varepsilon$$

and

2) if $A \to \alpha \in P$ then

$$\text{First } (A \to \alpha) = \{u; u \in \Sigma^*,\ |u| \leq 1:$$

$$\exists \delta,\ \gamma' \in (V \cup F \cup \Sigma)^*$$

$$\left.\begin{array}{l}(a)\ l(u) < 1 \\ (b)\ A\delta \Rightarrow \alpha\delta \overset{*}{\Rightarrow} u\gamma'\}\end{array}\right\} \gamma' = \varepsilon$$

The following context-free grammar is LL(1), cf.[Aho & Ullman], Thm 5.2 .

$S \to AB$
$B \to b$
$A \to \varepsilon | a$

However, if viewed as an indexed grammar, this grammar is not ILL(1). Taking $\delta = A$ we get $AA \to \varepsilon A \to a$ and hence $a \in \text{First}(A \to \varepsilon)$. Obviously $a \in \text{First}(A \to a)$ and therefore $\text{First}(A \to \varepsilon) \cap \text{First}(A \to a) \neq \emptyset$.

<u>Observation:</u> Weiß's definition of ILL(k) grammars is <u>not</u> a generalization of context-free LL(k) grammars.

The ε-rule $A \rightarrow \varepsilon$ was essential for our example; indeed, LL(k) grammars without ε-rules are ILL(k)-grammars in the sense of Weiß. A more serious flaw of the definition is exposed by the following ε-free indexed grammars which is apparently top down parsable with look-ahead 1. $G = (\{A,S\},\{f,g\}, \{a\}, S, P)$ where P contains the following rules

$$S \rightarrow Af \mid A$$
$$A \rightarrow a \in f$$
$$A \rightarrow a \in g$$

This grammar allows exactly one derivation:

$$S \rightarrow Af \rightarrow a,$$

the production $S \rightarrow A$ is useless. However, taking $\delta = g$ we obtain $Sg \rightarrow Ag \rightarrow a$ and therefore $a \in \text{First}(S \longrightarrow A)$. Thus $a \in \text{First}(S \longrightarrow A) \cap \text{First}(S \rightarrow Af)$ and our grammar is not ILL(1) in the sense of Weiß.

<u>Conclusion:</u> Weiß's definition of indexed LL(k) grammars does not capture the essence of top-down parsing (without back-up).

Comparing his definition with the definition given in [Aho & Ullman] for the context-free case we see what went wrong. The sentential form Sg should be derivable from the start symbol. This leads to the following definition which is a proper generalization of the context-free case.

<u>Definition:</u> Let $G = (V,F,\Sigma,S,P)$ be an indexed grammar. Let r be any rule in P.

a) if $r = [A \to \alpha] \in f$ is an index rule then

$\text{First}_k(r) = \{ x \in \Sigma^*;\ \exists u \in \Sigma^*, \delta \in (V \cup F)^*,\ w \in \Sigma^*$ with

$$(1)\quad S \overset{*}{\to} uAf\delta \to u\alpha\delta \overset{*}{\to} uxw$$

$$(2)\quad |x| < k \Rightarrow w = \varepsilon$$

$$(3)\quad |x| \le k$$

if $r = A \to \alpha \in P$ then

$\text{First}_k(r) = \{ x \in \Sigma^*;\ \exists u \in \Sigma^*, \delta \in (V \cup F)^*,\ w \in \Sigma^*$ with

$$(1)\quad S \overset{*}{\to} uA\delta \to u\alpha\delta \overset{*}{\to} uxw$$

$$(2)\quad |x| < k \Rightarrow w = \varepsilon$$

$$(3)\quad |x| \le k \qquad \}$$

b) An indexed grammar is ILL(k) if for every pair r,r' of distinct rules having the same left hand side:

$\text{First}_k(r) \cap \text{First}_k(r') = \emptyset$.

In [Fischer] the (effective) equivalence of macro and indexed grammars was stated. We describe transformations which preserve the LL(k) property.

Thm.: Given any MLL(k) grammar G, we can effectively find an equivalent ILL(k) grammar G' and vice versa.

Proof:

$\Rightarrow$: We may assume w.l.o.g. that $G = (\Sigma, \mathfrak{F}, \mathcal{V}, \wp, S, P)$ is in Standard Form. The indexed grammar G' has nonterminals $\mathfrak{F}$ and indices $F = \{ < X_1, \ldots, X_k >;\ X_i \in \mathfrak{F}$ and $k \le \text{max-rank}(G') \}$
and rules

(1) $F \to G\ <H_1, \ldots, H_k>$ for $F(x_1, \ldots, x_n) \to G(H_1(x_1, \ldots, x_n), \ldots, H_k(x_1, \ldots, x_n)) \in P$

(2) $F \langle X_1,\ldots,X_n \rangle \to X_1 \ldots X_n$ for all $X_i \in \mathfrak{F}$, $1 \le i \le n$,
and $F(x_1,\ldots,x_n) \to x_1 \ldots x_n \in P$

(3) $F \langle X_1,\ldots,X_n \rangle \to X_i$ for all $X_j \in \mathfrak{F}$, $1 \le j \le n$, and
$F(x_1,\ldots,x_n) \to x_i \in P$

(4) $F \langle X_1,\ldots,X_n \rangle \to a$ for all $X_i \in \mathfrak{F}$ and
$F(x_1,\ldots,x_n) \to a \in P$ where $a \in \Sigma \cup \{\varepsilon\}$

The proof of equivalence is straightforward. Because of the 1-1 correspondence of the derivations according to G and G', the LL(k) property carries over.

$\Leftarrow$: We may assume w.l.o.g. that $G' = (\Sigma,V,F,S,P)$ is in reduced form, i.e. productions are of the form $A \to BC$, $A \to a$ for $a \in \Sigma \cup \{\varepsilon\}$, $A \to Bf$ and $Af \to B$. Let $V = \{A_1,\ldots,A_n\}$. The macro grammar G has function symbols $V \times (F \cup \{\text{dummy}\}) \cup \{\text{START}\}$ with $\rho(\text{START}) = 0$ and $\rho(F) = |V|$ for all other function symbols. The rules are

(1) $\text{START} \to [S,\text{dummy}]\ \underbrace{(\varepsilon,\varepsilon,\ldots,\varepsilon)}_{|V|\text{-times}}$

(2) $[A,f]\ (x_1,\ldots\ ) \to [B,f]\ (x_1,\ldots\ )\ [C,f](x_1,\ldots\ )$
for $A \to BC \in P$ and every $f \in F \cup \{\text{dummy}\}$

(3) $[A,f]\ (x_1,\ldots\ ) \to a$ for $A \to a \in P$ with $a \in \Sigma \cup \{\varepsilon\}$
and every $f \in F \cup \{\text{dummy}\}$.

(4) $[A,g]\ (x_1,\ldots\ ) \to [B,f]([A_1,g]\ (x_1,\ldots\ ),\ldots,[A_n,g](x_1,\ldots\ ))$
for every $g \in F \cup \{\text{dummy}\}$ and $A \to Bf \in P$.

(5) $[A,f]\ (x_1,\ldots\ ) \to x_i$
for every $Af \to B \in P$ with $B = A_i$

Note the similarity of this construction and the construction in section 2. The correctness proof goes along the same lines. Because of the 1-1 correspondence between derivations according to G and G' the LL(k) property carries over.

VI. MLL languages and restricted nested stack automata

In this section we give an "automata-theoretic" definition of ILL (and hence MLL) languages. In [Aho 69] Aho introduced one-way deterministic nested stack automata. The storage structure of such an automata is a nested stack. It operates in one of four modes:

pushdowon mode: read and write at the top of one of the nested stacks
stack reading mode: read and move up and down within a stack
stack creating mode: create a new stack
stack destruction mode: destroy an empty stack

A downward reading nested stack automaton is a nesa with the following restriction placed on the behaviour in the stack reading mode. In the stack reading mode a downward reading nesa can only move down. If it hits the bottom of the outermost stack in this mode then the machine is put in a special state and the storage tape head is placed at the top of the right-most stack.

Thm.: A language L is ILL (and hence MLL) if and only if there is a 1-way deterministic downward reading nesa accepting L.

Proof: Inspect the equivalence proof of nesa and indexed grammars in [Aho 69] closely.

We are now able to describe a parsing algorithm for MLL languages. Weiß showed that (his version of) ε-free ILL grammars can be parsed in time $O(n^2)$. He constructs an equivalent nesa and computes its running time. This construction also works for our version of ILL grammars (not necessarily ε-free). We obtain

Thm.: Let L be a MLL language. Then there is a recognizer for L working in time $O(n^2)$ and space $C(n)$.

In section 2 we constructed an MLL grammar for every deterministic context-free language. The nesa corresponding to these grammars is essentially a deterministic pushdown automata and works in linear time.

Open Problem: Find a class of MLL languages which properly includes the deterministic context-free languages but can still be parsed in linear time.

VII. An Example

The following rules are part of the ALGOL 60 syntax for assignment statements.

<assignment> → <Var> ← <Expression>

<Expression> → <Var> | (<Var> + <Expression>)

<Var> → A|B|C|...

Some of the strings which can be derived from this grammar are not legal ALGOL 68 assignment statements: if the variable on the left hand side is of type integer then the expression on the right hand side should better yield a value of type integer. This restriction is part of the semantics of the assignment statement.

In ALGOL W this restriction is made part of the syntax

<assignment> → <integer assignment> |

<real assignment>

<integer assignment> → <integer var> ← <integer expr>

⋮

<integer var> ← A|B|C

<real var> ← X|Y|Z

by explicitely listing a set of the ALGOL 60 rules for every type. In the presence of infinitely many modes the explicite listing does not suffice. In ALGOL 68 an implicite listing is achieved by means of two level grammars. The goal can also be reached using macro grammars.

Suppose that we have the modes int, long int , long long int,...; int means single precision, long int means double precision ,... We also want to include a simple form of coercion: widening. A value of type long ^{i} int is also a value of type long^{i+k}int for all $k \geq 0$. The following macro grammar generates the set of legal assignment statements.

<assignment> → F(int <var> , int)

F(x,y) → F(long x, Ay) | x ← Expression(y)

Expression(y) ← y<Var> | (y<Var> + Expression(y))

A → long | ε

This grammar is not MLL(k) for any k due to the "left recursion" in the rules for F( ). However a trick similar to the one used in the context-free case will remove left recursion:

<assignment> → G(ε)

G(y) → long G(Ay) | int< var> ← Exression(y)

Expression(y) → y int< Var> | (y int< Var> + Expression(y))

A → long | ε

Bibliography

Aho, A.V.[1968]. Indexed grammars - an extension of context-free grammars. J. ACM 15:4, 647-671.

Aho, A.V.[1969]. Nested Stack Automata, JACM, 16:3, 383 - 406

Aho, A.V. & Ullman, J.D.[1972]. The Theory of Parsing, Translation and Compiling, Prentice Hall, Series in Automatic Computation.

Fischer, M., Grammars with Macro-Like Instructions, 9th SWAT conference, 1968.

Gries, D., Compiler Construction for Digital Computers, Addison Wesley

Knuth, D.E.[1967]. Top-down syntax analysis. Lecture Notes. International Summer School on Computer Programming Copenhagen, Denmark.

Lewis, P.M.II, & Stearns, R.E.[1968]. Syntax directed transduction. J. ACM 15:3, 464-488.

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

Rosenkrantz, D.J. & Stearns, R.E.[1970]. Properties of deterministic top-down grammars. Information and Control 17:3, 226-256.

Weiß, K., Deterministische indizierte Grammatiken, 2te Kolloquium über Automatentheorie und formale Sprachen, Kaiserslautern, 1975