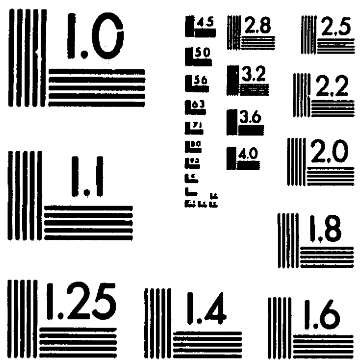
1995



24:1

GENERATION AND RECOGNITION OF FORMAL LANGUAGES

by

LEONARD HAROLD HAINES

A.B., Harvard College
(1957)
M.A., University of California, Berkeley
(1960)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June, 1965

Signature of Author ....
Department of Mathematics
February 12, 1965

Certified by ....
Thesis Supervisor

Accepted by ....
Chairman, Departmental Committee on
Graduate Students

GENERATION AND RECOGNITION OF FORMAL LANGUAGES

by

LEONARD HAROLD HAINES

Submitted to the Department of Mathematics on February 12, 1965, in partial fulfillment of the requirement for the degree of doctor of philosophy.

Abstract

This thesis continues the investigation of formal languages initiated by Chomsky. Some representative results are indicated below.

As a by-product of a new proof of the Chomsky-Schützenberger theorem that CF (the class of context-free languages) = PDS (the class of languages accepted by nondeterministic, push-down store automata), we show that unambiguous CF = unambiguous PDS, and that PDS = real time PDS. A normal form theorem, somewhat stronger than Greibach standard form, also follows. (Call a PDS <u>unambiguous</u> if it accepts each input with a unique computation; call it <u>real time</u> if each instruction reads a non-null input character and if the storage tape is empty on acceptance.)

There are minimal linear languages with noncontext-free complement which are also inherently ambiguous over the CF grammars. This result gave rise to the (false) conjecture that the unambiguous CF languages are just those with CF complement. The conjecture does hold for the family of languages accepted by deterministic PDS which we prove closed under complementation; the analogous conjecture holds for deterministic linear bounded automata and unambiguous CS (context-sensitive) languages.

Left context-sensitive grammars are defined (in the obvious way) and shown to be no more powerful than context-free grammars.

Thesis Supervisor: Noam Chomsky
Title: Professor of Linguistics

ACKNOWLEDGEMENT

I would like to thank Professor Noam Chomsky for his generous assistance and encouragement.

I am also indebted to Paul S. Peters, S. Y. Kuroda, and Dr. Peter Anderson for several helpful discussions.

TABLE OF CONTENTS

INTRODUCTION

A language is any set of finite strings on a finite alphabet. As originally defined, a grammar is any device which generates (enumerates) or recognizes (accepts) a language. The formal languages (i.e., those having grammars) are just the recursively enumerable sets. Chomsky (1959) initiated a classification of these formal languages into families. Each family consists of all languages having grammars meeting certain criteria. Notions of unambiguity of a grammar and ambiguity of a language (over a family of grammars) have been introduced and intensively investigated for certain grammars. Boolean properties of languages, equivalence theorems for grammars and devices which map languages have been useful in other parts of the theory in addition to being of some intrinsic interest. This thesis continues the investigation of formal languages in much the same spirit.

The regular languages (i.e., those accepted by finite automata) are closed under all Boolean operations and have simple unambiguous CF (context-free) grammars. The minimal linear languages consist of regular languages and the simplest instances of nonregular languages. In Section I we prove that there is a minimal linear language L with noncontext-free complement. This settles a problem posed by Chomsky. We also show that L is inherently ambiguous over the CF grammars. All other sections of the thesis are motivated, in part, by these results.

In Section II PDS (nondeterministic push down storage automata) are defined and used to prove that the nondoubling language is CF; this solves another problem posed by Chomsky. An immediate generalization leads to an interesting necessary condition that a language be regular; we conjecture that the condition is also sufficient. The main result of Section II is Theorem 5--a real time result for PDS which also serves as an important lemma for Section IV.

The languages exhibited in Section I, numerous other examples and a suggestive result of Chomsky and Schützenberger (1962) about minimal linear languages all support the conjecture that the unambiguous CF languages are just those with CF complement. In general, this conjecture fails in both directions (personal communication, T. Hibbard, and J. Ullian). The conjecture does hold for the family of languages accepted by deterministic PDS which we prove closed under complementation in Section III. In Section V we show that the analogous conjecture holds for deterministic linear bounded automata and unambiguous CS (context-sensitive) languages.

Sections II and IV lead to a new proof of the Chomsky-Schützenberger theorem that CF (the class of context-free languages) = PDS (the class of languages accepted by nondeterministic push down storage automata).

As a by-product of the new proof we show that PDS = real time PDS and unambiguous CF = unambiguous PDS. A normal form theorem, somewhat stronger than Greibach standard form, also follows. (Call a PDS _unambiguous_ if it accepts each input with a unique computation; call it _real time_ if each instruction reads a non-null input character and the storage tape is empty on acceptance.)

Section IV concludes by pointing out an essential difference between CF and CS languages. Left context-sensitive grammars are defined (in the obvious way) and shown to be no more powerful than CF grammars.

PRELIMINARIES

Let W be a finite set of symbols. If $\alpha_i \in W$, $1 \leq i \leq n$, then $\phi = \alpha_1\alpha_2 \cdots \alpha_n$ is called a string on W of length $\lambda(\phi) = n$. W^* denotes the set of all strings on W, including the null string e of length zero. If $\psi = \beta_1\beta_2 \cdots \beta_m$, $\beta_j \in W$, $1 \leq j \leq m$, then

$$\phi\psi = \alpha_1\alpha_2 \cdots \alpha_n \beta_1\beta_2 \cdots \beta_m,$$

$$\phi^2 = \phi\,\phi, \quad \phi^{k+1} = \phi^k\,\phi .$$

If $W_1, W_2 \subset W^*$, then $W_1W_2 = \{\phi\psi : \phi \in W_1, \psi \in W_2\}$. Under the operation of string concatenation, W^* is a semi-group, the free semi-group generated by W.

A formal grammar G is a quadruple (V_T, V_N, S, R) where

(1) V_T and V_N are disjoint non-empty finite sets of symbols called, respectively, the terminal and non-terminal vocabularies of G. $V = V_T \cup V_N$ is the vocabulary of G.

(2) S is a distinguished symbol of V_N, called the initial symbol of G.

(3) R is a finite set of substitution rules of the form $\phi \rightarrow \psi$ where $\phi \in V^* V_N V^*$ and $\psi \in V^*$.

<u>Warning</u>: Throughout the sequel we observe the following notational conventions when naming strings, except for $S \in V_N$.

Small Greek letters denote members of V^*.

Small Latin letters denote members of V_T^*.

Capital Latin letters denote members of V_N^*.

Early letters of all alphabets denote single symbols.

Late letters of all alphabets denote strings of arbitrary length.

R induces a binary relation $\overset{*}{\rightarrow}$ on V^*: $\sigma \overset{*}{\rightarrow} \tau$ iff there are strings ω_1, ω_2, ϕ and ψ such that $\sigma = \omega_1 \phi \omega_2$, $\tau = \omega_1 \psi \omega_2$ and $\phi \rightarrow \psi$ is a rule of R. We say that σ dominates τ and write $\sigma \Rightarrow \tau$ iff there are strings $\tau_0, \tau_1, \cdots \tau_n$ such that $\sigma = \tau_0$, $\tau = \tau_n$ and $\tau_i \overset{*}{\rightarrow} \tau_{i+1}$ for $i = 0, 1, \cdots, n-1$. ($\tau_0 \overset{*}{\rightarrow} \tau_1 \overset{*}{\rightarrow} \cdots \overset{*}{\rightarrow} \tau_{n-1} \overset{*}{\rightarrow} \tau_n$ is called a σ-derivation for τ. Each τ_i is called a line of the derivation.)

The formal language L(G) generated by G is the set of all $x \in V_T^*$ such that $S \Rightarrow x$. The complement of L(G) is $\overline{L(G)} = V_T^* - L(G)$.

G is type 1 if for each rule $\phi \rightarrow \psi$ we have $\lambda(\phi) \leq \lambda(\psi)$.

G is context-sensitive (CS) if each rule of G has the form $\phi_1 A \phi_2 \rightarrow \phi_1 \omega \phi_2$, $\omega \neq e$.

G is context-free (CF) if each rule of G has the form $A \rightarrow \omega$, $\omega \neq e$.

A language L is any set of strings on a finite set of symbols. L is a formal language if L = L(G) for some formal grammar G. In this case we simply say that L has the grammar G or G is a grammar for L. A language is context-sensitive or context-free if it has, respectively, a CS or CF grammar.

L(G) is sometimes called the weak generative capacity of the grammar G. If G is CF, we will define the strong generative capacity, $\Sigma(G)$, as follows:

$\Sigma(G) = L(G')$ where $G' = (V_T', V_N, S, R')$,

$$V_T' = V_T \cup \{[_A : A \in V_N\} \cup \{]\} ,$$

and $A \to [_A \omega]$ is a rule of G' iff $A \to \omega$ is a rule of G.

Define d to be the unique homomorphism which maps the free semi-group $(V_T')^*$ onto the free semi-group V_T^* so that

$$d(a) = \begin{cases} a \text{ if } a \in V_T \\ \\ e \text{ if } a \in V_T' - V_T . \end{cases}$$

If d is 1-1 on $\Sigma(G)$ we say that G is an unambiguous CF grammar. Unambiguous CS grammars may be similarly (but less elegantly) defined as in Parikh (1961). The details are omitted.

A language is unambiguous CS(CF) if it has an unambiguous CS(CF) grammar. A CS or CF language which has no unambiguous CS or CF grammar is termed inherently ambiguous over the CS or CF grammars, respectively.

A left to right derivation in a CS grammar is a derivation in which each line is derived from its predecessor by a rule which replaces the leftmost non-terminal. Right to left derivations are similarly defined.

On occasion we will identify G with R. While not strictly correct, no confusion can arise from this practice.

Formal grammars are generators of languages. We are also interested in devices which recognize languages. One of the simplest such devices is a finite automaton. We recall the definition.

A finite automaton M is a quintuple $(A_I, \Sigma, S_o, \Sigma_f, \Phi)$ where A_I is a finite input alphabet, Σ is a finite set of states, $S_o \epsilon \Sigma$ is a distinguished initial state and $\Phi \subset A_I \times \Sigma \times \Sigma$ is a (possibly multi-valued) transition function which maps $A_I \times \Sigma$ into Σ.

M accepts (recognizes) a string $a_1 a_2 \cdots a_n \epsilon A_I^*$ iff $\exists$ a sequence $\{T_i\}$ of states of M $\ni$:

(1) $(a_i, T_i, T_{i+1}) \epsilon \Phi, \quad 1 \leq i \leq n$

(2) $T_1 = S_o$ and $T_{n+1} \epsilon \Sigma_f$.

L(M) will denote the language accepted by M; i.e., the set of all strings on A_I accepted by M. A language is regular if it is accepted by some finite automaton.

M can be realized by a computing machine having a set of internal states Σ and a read head which scans squares of a one-channel input tape. A finite transducer T is a mechanized finite automaton equipped with an output tape on which symbols may be written. Both tapes move in only one direction. T maps an input string into an output string when the read head runs off the input tape with T in a final state. T(L) is the set of all output strings arising from a language L of input strings. We omit the formal details.

(I) MINIMAL LINEAR LANGUAGES

In this section we exhibit a pathological minimal linear language and prove that its complement is not context-free. This settles a problem posed by Chomsky. The language is also inherently ambiguous over the CF grammars.

<u>Definition</u>: A rule is terminal iff it is of the form $A \to z$.

<u>Definition</u>: A linear grammar is a CF grammar with all non-terminal rules of form $A \to xBy$.

<u>Definition</u>: A minimal linear grammar G is a linear grammar with a single non-terminal symbol (namely S) and a single terminal rule $S \to c$ where c does not appear in any other rule. c is called the designated center of the strings of L(G).

<u>Definition</u>: A language L is linear or minimal linear if there is, respectively, a linear or minimal linear grammar which generates L.

There are several reasons for investigating properties of minimal linear languages. First of all, Chomsky proved that the regular languages are precisely those linear languages which have grammars whose non-terminal rules are either all of the form $A \to Bx$ or all of the form $A \to xB$. Structurally, then, the minimal linear languages are almost regular (i.e., they consist of regular languages and the simplest instances of non-regular languages).

Secondly, most of the unsolvability results in the theory follow by reduction to the Post Correspondence problem. Towards this end the minimal linear languages are a natural object for study since the correspondence problem can be reformulated as a simple set-theoretic problem involving such languages; Schützenberger (1961).

By making reductions to this simple form of the correspondence problem Chomsky (1963) simplified the proofs of several unsolvability results. However, a proposed simple proof of one result hinged upon proving that the complement of any minimal linear language is context-free.

The linear languages are not closed under complementation. Moreover, there are linear languages whose intersection is not context free. These results, contained (implicitly) in Scheinberg (1960) suggested the existence of linear languages with non-context free complement. It is not hard to find such languages.

For example, let $L = \{a^p b^q c^r: \ p \neq q \text{ or } q \neq r, \ pqr > 0\}$. Then $L = L_1 \cup L_2 \cup L_3 \cup L_4$ where

$$L_1 = \{a^p b^q c^r: \ p > q, \ pqr > 0\}$$

$$L_2 = \{a^p b^q c^r: \ p < q, \ pqr > 0\}$$

$$L_3 = \{a^p b^q c^r: \ q > r, \ pqr > 0\}$$

$$L_4 = \{a^p b^q c^r: \ q < r, \ pqr > 0\}$$

The following set of rules is a linear grammar for L_1:

$S \rightarrow Ac$

$A \rightarrow Ac$

$A \rightarrow aBb$

$B \rightarrow aBb$

$B \rightarrow D$

$D \rightarrow aD$

$D \rightarrow a$

Similar linear grammars generate L_2, L_3, and L_4 and therefore, since linear languages are closed under union, L is linear.

To prove $\bar{L}$ non-context-free, we intersect $\bar{L}$ with the regular language

$$R = \{a^p b^q c^r: \quad pqr > 0\}$$

which yields

$$\bar{L} \cap R = \{a^p b^q c^r: \quad p = q = r\}$$

which Scheinberg (1960) proved non-context-free.

Since the intersection of a context-free and regular language is context free (Corollary 16a), $\bar{L}$ cannot be context-free.

Schützenberger and Chomsky (1962) proved that for a certain subfamily of the minimal linear languages the complement is not only context-free but is itself linear. This subfamily consists of those minimal linear languages for which the substring left of the designated center uniquely determines the substring on the right.

Since the regular languages are closed under all Boolean operations and since the minimal linear languages are almost regular, it was hoped that at least the weak form of the result (i.e., that the complement is context-free) would extend to the full family of minimal linear languages. However, we have the following theorem.

<u>Notation</u>: If $\phi = \alpha_1\alpha_2 \cdots \alpha_n$, $|\phi| = \lambda(\phi) = n$ and $\hat{\phi} = \alpha_n\alpha_{n-1} \cdots \alpha_2\alpha_1$.

<u>Theorem 1</u>: There are minimal linear languages with non-context-free complement.

<u>Proof</u>: Let G be the following minimal linear grammar:

$S \rightarrow aSa$ $\qquad V_N = \{S\}$, $V_T = \{a,b,c,d\}$

$S \rightarrow bSb$ $\qquad W = \{a,b\}$.

$S \rightarrow dSa$

$S \rightarrow dSb$

$S \rightarrow dS$

$S \rightarrow c$

Then

$$L = L(G) = \left\{ z: z = d^{n_{k+1}} x_k d^{n_k} \cdots d^{n_2} x_1 d^{n_1} c y_1 \hat{x}_1 y_2 \cdots y_k \hat{x}_k y_{k+1} \;\; \begin{array}{l} \text{for some } k.\ x_i, y_j \in W^* \text{ and} \\ n_j \geq |y_j'|, \quad 1 \leq i \leq k, \ 1 \leq j \leq k+1 \end{array} \right\}$$

Let

$$R = \{d^p b d^q c b a^r b: \ pqr > 0\},$$

a regular language.

$$L \cap R = \left\{ d^p b d^q c b a^r b: \ pqr > 0 \text{ and for some } y_1, y_2 \in W^*, \ b a^r b = y_1 \hat{b} y_2, \ p \geq |y_2|, \ q \geq |y_1| \right\}$$

$$= \{d^p b d^q c b a^r b: \ pqr > 0 \text{ and } (p \geq r+1 \text{ or } q \geq r+1)\}$$

since $\hat{b} = b$.

Hence

$$\bar{L} \cap R = \{d^p b d^q c b a^r b: \ pqr > 0 \text{ and } r+1 > \max(p,q)\}$$

which transduces into

$$L' = \{a^p b^q c^r : \ r > \max(p,q), \ pq > 0\}.$$

By the following lemma, L' is not context-free. Therefore, since the property of being context-free is invariant under both intersections by regular language (Corollary 16a) and transduction (Theorem 16), $\bar{L}$ is not context-free.

Lemma 1a:

$$L = \{a^p b^q c^r : \ r > \max(p,q); \ pq > 0\}$$

is not context-free.

Proof: Suppose that G is a CF grammar for L. Without loss of generality we may assume that for each $A \in V_N$, $A \Rightarrow z$ for infinitely many $z \in V_T^*$.

For if not, we can construct G' from G by discarding all rules of the form $A \to \omega$ and replacing each rule $B \to \phi A \psi$, $B \neq A$, by the finite set of rules

$$\{B \to \phi \ z \ \psi : \ A \Rightarrow z\}.$$

Clearly $L(G') = L(G)$.

Choose $m = n_1 \cdot n_2 + n_1 + 2$ where $n_1 = \max\{|\omega| : A \to \omega$ is a rule of $G\}$ n_2 = the number of symbols of V_N. Let

$$S \overset{*}{\to} x_1 A_1 \phi_1 \overset{*}{\to} x_1 x_2 A_2 \phi_2 \phi_1 \overset{*}{\to} \cdots \overset{*}{\to}$$

$$x_1 x_2 \cdots x_M A_M \phi_M \cdots \phi_2 \phi_1 \overset{*}{\to} x_1 x_2 \cdots x_M z \phi_M \cdots \phi_2 \phi_1 \overset{*}{\to} \cdots \overset{*}{\to}$$

$$a^m b^m a^{m+1}$$

be a <u>fixed</u> left-to-right S-derivation for $a^m b^m a^{m+1}$. The nonterminal rules

$$S \to x_1 A_1 \phi_1,\ A_1 \to x_2 A_2 \phi_2,\ \cdots,\ A_{M-1} \to x_M A_M \phi_M,$$

are applied in succession until the first instance of a terminal rule, $A_M \to z$.

Let

$$\phi_j \Rightarrow y_j \quad (1 \le j \le M)$$

in the <u>fixed</u> derivation so that

$$x_1x_2 \cdots x_M z\phi_M \cdots \phi_2\phi_1 \Rightarrow x_1x_2 \cdots$$

$$x_M\, zy_M \cdots y_2y_1 = a^m b^m a^{m+1} .$$

Suppose that

$$y_M \cdots y_2y_1 = yba^{m+1}$$

for some y. Then

$$x_1x_2 \cdots x_M\, wy_M \cdots y_2y_1 = x_1x_2 \cdots x_M\, wyba^{m+1} \in L$$

for all w such that $A_M \Rightarrow w$. But there are infinitely many such w and therefore arbitrarily large such w. Since m is fixed, for sufficiently large w we would have

$$x_1x_2 \cdots x_M\, wyba^{m+1} \notin L,$$

a contradiction.

Hence

Hence $y_M \cdots y_2 y_1 \neq yba^{m+1}$ for any y or equivalently, $x_1 x_2 \cdots x_M z = a^m b^m a^{m'}$ for some $m' \geq 0$, so that $|x_1 x_2 \cdots x_M z| = 2m + m' \geq 2m$.

The following inequalities are now immediate because of the choice of n_1.

$$|x_1 x_2 \cdots x_M| \geq 2m - n_1 \text{ since } |z| \leq n_1$$

$$2m > |x_1 x_2 \cdots x_N| \geq 2m - n_1 \text{ for some } N \leq M$$

since $|x_i| \leq n_1$, $1 \leq i \leq M$. Therefore $x_1 x_2 \cdots x_N = a^m b^n$ where

$$m - n_1 \leq n < m. \qquad (1)$$

To facilitate the discussion, the following definitions are introduced:

<u>Definition</u>: A pair of integers (k,ℓ) is a <u>cycle</u> if

(1) $1 \leq k \leq \ell \leq N$

(2) $A_{k-1} = A_\ell$ (where $A_0 = S$).

Definition: An integer i belongs to a cycle (k,ℓ) if $k \leq i \leq \ell$.

Definition: An integer is cyclic if it belongs to at least one cycle.

Definition: $i \equiv j$ if there exists a cycle (k,ℓ) such that both i and j belong to (k,ℓ).

Obviously "$\equiv$" is an equivalence relation which decomposes the set of all cyclic integers into disjoint subsets of integers which belong to maximal cycles

$$(k_1, \ell_1), (k_2, \ell_2), \cdots , (k_s, \ell_s).$$

Hence

$$\sum_{i \text{ cyclic}} |x_i| = \sum_{i=k_1}^{\ell_1} |x_i| + \sum_{i=k_2}^{\ell_2} |x_i| + \cdots + \sum_{i=k_s}^{\ell_s} |x_i| \quad (2)$$

while

$$\sum_{i \text{ cyclic}} |y_i| = \sum_{i=k_1}^{\ell_1} |y_i| + \sum_{i=k_2}^{\ell_2} |y_i| + \cdots + \sum_{i=k_s}^{\ell_s} |y_i| \qquad (3)$$

Because of the choice of n_2, among any set of n_2+1 distinct integers, $1 \le i_0 < i_1 < \cdots < i_{n_2} \le N$, there must be at least two, say $i < j$, such that $A_i = A_j$. Therefore j is cyclic so that no more than n_2 of the integers are not cyclic.

Now we assert that

$$\sum_{i \text{ cyclic}} |x_i| > \sum_{i \text{ cyclic}} |y_i| \qquad (4)$$

Clearly

$$\sum_{i \text{ cyclic}} |x_i| = \sum_{i=1}^{N} |x_i| - \sum_{\substack{i \text{ not} \\ \text{cyclic}}} |x_i| \ge \sum_{i=1}^{N} |x_i| - \sum_{\substack{i \text{ not} \\ \text{cyclic}}} n_1$$

$$\ge \sum_{i=1}^{N} |x_i| - n_1 \cdot n_2 = m + n - n_1 \cdot n_2 \ge m + (m - n_1) - n_1 \cdot n_2$$

$$= 2m - n_1 - n_1 \cdot n_2 = m + 2 > m + 1 \ge \sum_{i=1}^{N} |y_i| \ge \sum_{i \text{ cyclic}} |y_i|$$

and therefore (4) is true.

In terms of the identities (2) and (3) this means

$$\sum_{i=k_1}^{\ell_1} |x_i| + \sum_{i=k_2}^{\ell_2} |x_i| + \cdots + \sum_{i=k_s}^{\ell_s} |x_i| > \sum_{i=k_1}^{\ell_1} |y_i| +$$

$$\sum_{i=k_2}^{\ell_2} |y_i| + \cdots + \sum_{i=k_s}^{\ell_s} |y_i|$$

and implies that for at least one maximal cycle, say (k', ℓ'), we must have

$$\sum_{i=k'}^{\ell'} |x_i| > \sum_{i=k'}^{\ell'} |y_i| \,.$$

We know that $x_1x_2 \cdots x_M\ z\ y_M \cdots y_2y_1 = a^m b^m a^{m+1} \in L$. Together with (1) this implies

$$x_1x_2 \cdots x_N\ b\ z'z\ y_M \cdots y_2y_1 = a^m b^m a^{m+1} \in L$$

for some $z' \in V_T^*$.

But then

$$z_1 x_{k'} x_{k'+1} \cdots x_{\ell'} z_2 \; b \; z_3 \; y_{\ell'} \cdots y_{k'+1} y_{k'} z_4 \in L$$

where

$$z_1 = x_1 x_2 \cdots x_{k'-1}, \; z_2 = x_{\ell'+1} \; x_{\ell'+2} \cdots x_N,$$

$$z_3 = z'z \; y_M y_{M-1} \cdots y_{\ell'+1}$$

and

$$z_4 = y_{k'-1} \cdots y_2 y_1 \; .$$

Since (k', ℓ') is a cycle, $A_{k'-1} = A_{\ell'}$ and since

$$A_{k'-1} \Rightarrow x_{k'} x_{k'+1} \cdots x_{\ell'} A_{\ell'} y_{\ell'} \cdots y_{k'+1} y_k$$

it follows (manifestly) that

$$A_{\ell'} \Rightarrow x_{k'} x_{k'+1} \cdots x_{\ell'} \; A_{\ell'} \; y_{\ell'} \cdots y_{k'+1} y_{k'} \; .$$

Therefore

$$A_{k'-1} \Rightarrow (x_{k'}x_{k'+1} \cdots x_{\ell'})^K A_{\ell'}(y_{\ell'} \cdots y_{k'+1}y_{k'})^K$$

for arbitrarily large K.

Hence by altering the original fixed S-derivation for $a^m b^m a^{m+1}$ along the above lines we can generate

$$z_1(x_{k'}x_{k'+1} \cdots x_{\ell'})^K z_2 \, b \, z_3 \, (y_{\ell'} \cdots y_{k'+1}y_{k'})^K z_4 \in L$$

for all K.

But for sufficiently large K we must have

$$|x_{k'}x_{k'+1} \cdots x_{\ell'}|^K > (|z_3| + |y_{\ell'} \cdots y_{k'+1}y_{k'}|^K + |z_4|) \cdot 2$$

since z_3 and z_4 are fixed strings and

$$\sum_{i=k'}^{\ell'} |x_i| > \sum_{i=k'}^{\ell'} |y_i| \; .$$

Or equivalently $a^p b^q a^r \in L$ and $p + q > 2r$, a contradiction which suffices to prove the lemma.

Theorem 2: There exists a minimal linear language inherently ambiguous over the CF grammars.

<u>Proof</u>: Let L be the minimal linear language and R the regular language of Theorem 1 so that

$$L \cap R = \{d^p bd^q cba^r b: \ pqr > 0 \text{ and } (p \geq r+1 \text{ or } q \geq r+1)\}$$

Suppose that L is unambiguous CF. Then Corollary 16a implies that $L \cap R$ has an unambiguous CF grammar.

Let T be the transducer such that

$$T: \ d^p bd^q cba^r b \rightarrow a^p b^q a^r .$$

$$T(L \cap R) = \{a^p b^q a^r: \ r < p \ \text{ or } \ r < q\}.$$

Since T maps distinct strings of $L \cap R$ into distinct strings, $T(L \cap R)$ is unambiguous CF by Theorem 16.

However, $T(L \cap R)$ has no unambiguous CF grammar. To prove this result the reader is referred to Ginsburg and Ullian (1964).

(II) REAL TIME PUSH DOWN STORAGE AUTOMATA

In this section we prove that any language accepted by a Push Down Storage Automaton is also accepted by a real time Push Down Storage Automaton (Theorem 5).

Chomsky and Schützenberger (1962) defined a family of recognizers called Push Down Storage Automata (PDS) and proved that a language is CF iff it is accepted by a PDS (we will usually write CF = PDS for this result).

PDS are quite useful for proving complicated languages context free. On the other hand, they are of little help in proving a language non-context-free. For example, although it is fairly obvious that no PDS can accept the language, $\{a^p b^q a^r : r > \max(p,q),\ pq > 0\}$, treated in Lemma 1a, a formal proof within the framework of PDS seems out of reach. This situation obtains, in part, because PDS are not constrained to operate in real time. We hope that real time PDS can serve as a vehicle for formal proof.

Chomsky (personal communication) has pointed out that Theorem 5 follows from the CF = PDS result and a normal form theorem due to Greibach (1963). However, we independently prove Theorem 5 since we will subsequently use it to prove stronger forms of both the CF = PDS and Greibach results.

In order to establish notation we will recall the definition of PDS. Our definition is mechanistic.

A Push Down Storage Automaton (PDS) is a machine M uniquely determined by a quintuple $(A_I, A_O, \Sigma, S_O, \Phi)$ where A_I is a finite input alphabet, A_O is a finite output alphabet, Σ is a finite set of states, $S_O \in \Sigma$ is a distinguished initial and final state and Φ is a finite set of instructions.

M has a read head which scans symbols written on squares of two infinite tapes connected to M, an input tape and a storage tape. A situation of M is a triple (a,S,b) where S is the current state of M and a and b are the scanned symbols on the input and storage tapes, respectively.

All instructions have the form $(\alpha,S,\beta) \rightarrow (S',\gamma)$ where $S, S' \in \Sigma$, $\alpha \in A_I \cup \{e, \#\}$, $\beta \in A_O \cup \{\#\}$ and $\gamma \in A_O \cup \{e, \sigma\}$. In the situation (a,S,b) M may only execute an instruction of the form

$$I = (a,S,b) \rightarrow (S',c) \text{ or } J = (e,S,b) \rightarrow (S',c).$$

I has the following effect on the machine tape configuration of M; J has the same effect as I except that (1) does not occur.

(1) M reads the input tape; i.e., the input tape moves one square left.

(2) M enters state S'.

(3) (i) If c = e, the storage tape is neither altered nor moved.

(ii) If $c \in A_0$, the storage tape is moved one square left and c is written on the new scanned square of the storage tape.

(iii) If $c = \sigma$, b is erased and the storage tape moves one square right.

Depending on whether $c = e$, $c \in A_0$ or $c = \sigma$, I is called a read-only, read-write or read-erase instruction, respectively: J is called a dormant, write-only or erase-only instruction, respectively.

An input $x = a_1 a_2 \cdots a_n$, $a_i \in A_I$, is written on n consecutive squares of the input tape, # is written on the n+1 st square and on the storage tape and all other squares on both tapes contain boundary symbols $\Delta \notin A_I \cup A_0 \cup \{\#\}$. M is started in situation $(a_1, S_0, \#)$ and accepts x if the situation $(\Delta, S_0, \#)$ subsequently occurs. M halts if no instruction applies in a given situation.

$L(M) = \{x \in A_I^*: \text{ M accepts } x\}$. M is non-deterministic in the sense that more than one instruction may apply in a given situation. $x \in L(M)$ iff <u>some</u> computation with input x leads to situation $(\Delta, S_0, \#)$.

At any step in a computation the contents of the (input) storage tape is the string to the (left) right of # up to and including the scanned symbol. (The contents of either tape may be e). The (machine-tape) configuration of M is the triple (u, S, v) where S is the state of M and u and v are the contents of the storage and input tapes, respectively.

The following identity simplifies notation in the sequel.

$$I = (a,S,e) \rightarrow (S',c) = \{(a,S,b) \rightarrow (S',c) \colon b \in A_O \cup \{\#\} \}.$$

I is usually considered a single instruction which acts independently of the scanned symbol of the storage tape.

We will occasionally identify M with its instructions Φ. When more than one PDS is under consideration, the notation $A_I(M)$, $A_O(M)$, $\Sigma(M)$, $\cdots$ will sometimes be used.

There are many other equivalent formalisms for PDS. We will illustrate our definition with an example which solves another problem posed by Chomsky.

Let $W = \{a,b\}$. One of the first CS languages proved non-context-free was $L_O = \{z\colon\ z = xcx,\ x \in W^*\}$.

Consider

$$L = \{z\colon\ z = x\ c\ y;\ x,\ y \in W^*,\ x \neq y\}.$$

L is essentially the complement of L_O, i.e., $L = \bar{L}_O \cap R$ where $R = \{x\ c\ y\colon\ x,\ y \in W^*\}$ is regular.

One might conjecture that any PDS which accepts L could readily be converted into a PDS which accepts L_o. Because of this, several attempts were made to prove L non-context-free. However, L is in fact context-free.

The technique employed in the following proof can be used to construct PDS for other seemingly non-context-free languages.

Theorem 3:

$$L = \{x\ c\ y:\ \ x, y \in W^* \text{ and } x \neq y\} \text{ is CF.}$$

Proof: L splits into three disjoint languages,

$$L_1 = \{x\ c\ y:\ \ x, y \in W^*,\ \lambda(x) < \lambda(y)\}$$

$$L_2 = \{x\ c\ y:\ \ x, y \in W^*,\ \lambda(x) > \lambda(y)\}$$

$$L_3 = \{x\ c\ y:\ \ x, y \in W^*,\ \lambda(x) = \lambda(y),\ x \neq y\}$$

It is a simple exercise to construct a PDS which accepts L_1 or L_2 and therefore they are both CF. By intersecting L_3 with a suitable regular language, e.g., $\{a^p b^q c a^r b^s:\ \ pqrs > 0\}$, it is not hard to see, at least informally, that L_3 is not context-free.

However, note that for any L_3' such that $L_3 \subset L_3' \subset L$, we have $L = L_1 \cup L_2 \cup L_3'$. Since CF languages are closed under union, it is enough to prove that some L_3' is CF.

In particular choose

$$L_3' = \left\{ \begin{array}{c} x\ c\ y:\ x, y \in W^* \text{ and } \exists\, u, u', v, v' \\ x - u\ \alpha\ u',\ y = v\ \beta\ v',\ \lambda(u) = \lambda(v) \\ \text{and } \alpha \neq \beta \end{array} \right\}$$

We can construct a PDS M for L_3' as follows:

Take $\Sigma = \{S_0, S_1, S_2, S_3.\ S^a, S^b\}$

Φ consists of the following instructions for all $\alpha, \beta, \gamma \in W$.

(1) $(e, S_0, e) \rightarrow (S_1, e)$

$(\alpha, S_1, e) \rightarrow (S_1, \alpha)$

$(\alpha, S_1, e) \rightarrow (S_2, \alpha)$

(2) $(\alpha, S_2, e) \rightarrow (S_2, e)$

$(c, S_2, \alpha) \rightarrow (S^{\alpha}, \sigma)$

(3) $(\beta, S^{\alpha}, \gamma) \rightarrow (S^{\alpha}, \sigma)$ (3.1)

$(\beta, S^{\alpha}, \#) \rightarrow (S_3, e), \quad \alpha \neq \beta$ (3.2)

(4) $(\beta, S_3, e) \rightarrow (S_3, e)$

$(\#, S_3, e) \rightarrow (S_C, e)$

Given an input string x c y, by means of the instructions (1), M copies the symbols of x onto its storage tape until it exercises a non-deterministic option and enters state S_2. At this point M has stored a candidate string $u\alpha$.

The instructions (2) feed the remainder of x past the read head until c is scanned--whereupon M erases α and enters state S^{α}.

The instructions (3.1) merely count off the first $\lambda(u)$ symbols of y; i.e., $\lambda(u)$ symbols of y have been read when u is completely erased and M is scanning #. Note that (3.1) is executed even when $\beta \neq \gamma$.

The decisive instruction is (3.2); if $\alpha \neq \beta$, M accepts x c y via (4), otherwise M blocks.

We can immediately generalize Theorem 3. For any language L define $V(L) = \{a: x\ a\ y \in L, \text{ some } x, y\}$ and

$$N(L,c) = \{x\ c\ y: x, y \in L,\ c \notin V(L),\ x \neq y\}.$$

<u>Theorem 4</u>: L regular $\Rightarrow N(L,c)$ context free.

<u>Proof</u>: If $c \in V(L)$, $N(L,c) = \emptyset$ and the theorem is trivial. If $c \notin V(L)$ define

$$L' = \{x\ c\ y: x, y \in (V(L))^*,\ x \neq y\}.$$

By Theorem 3, L' is CF since the proof of that theorem did not depend on the cardinality of W. Since L regular implies $L\ c\ L$ regular, $N(L,c) = L' \cap (L\ c\ L)$ is context-free.

We conjecture that this result also holds in the reverse direction, i.e.:

Conjecture: L regular $\Leftrightarrow N(L,c)$ context-free.

At this time we have no formal proof.

<u>Definition</u>: A PDS is a real time PDS if each of its instructions is a read-only, read-write or read-erase instruction.

A real time PDS terminates each computation with input x within $\lambda(x)+1$ steps; acceptable computations require precisely $\lambda(x)+1$ steps.

The proof of Theorem 5 revolves about the following definitions.

__Definition__: A string $x = b_1 b_2 \cdots b_n$ is eliminable by a PDS M iff there is a sequence of states $\{T_k\}$ such that $(e, T_k, b_k) \rightarrow (T_{k-1}, \sigma) = I_k$ is an instruction of M for $1 \leq k \leq n$. $[T_n, T_o]$ is called an elimination pair for x. M eliminates x from state T_n if M executes $I_n, I_{n-1}, \cdots, I_1$.

__Definition__: A string $x = b_1 b_2 \cdots b_n$ is generable by a PDS M iff there is a sequence $\{T_k\}$ of states such that $I_k = (e, T_{k-1}, e) \rightarrow (T_k, b_k)$ is an instruction of M for $1 \leq k \leq n$. $[T_n, T_o]$ is called a generation pair for x. M generates x from state T_o if M executes $I_1, I_2, \cdots, I_n$.

The equivalent real time PDS computes and stores an elimination pair for a string x instead of writing x--guessing that x will subsequently be eliminated. Instead of generating a string y, a generation pair for y is stored.

By using a large enough output alphabet, elimination pairs can be stored in symbols ultimately erased by read-erase instructions, generation pairs can be stored in symbols written by read-write instructions. Therefore, erase-only and write-only instructions are unnecessary. This is the main idea behind the proof.

The proof follows by successive application of a sequence of lemmas. Each lemma brings the PDS closer to the real time machine while preserving generative capacity. The proof of each lemma is only concerned with weak generative capacity; for PDS no notions of strong generative capacity have been defined. We now introduce the following definition.

Definition: A PDS M is unambiguous if it accepts each string of L(M) with a unique computation.

(Note that an unambiguous PDS can reject a string in more than one way.) Almost all constructions preserve unambiguity, but we do not choose to spell this out at each point. Alternative constructions are sketched whenever ambiguity is introduced.

Theorem 5: Given a PDS M one can construct an equivalent real time PDS M'. Moreover if M is unambiguous, so is M'.

Proof: The following definition simplifies the proof of the theorem.

Definition: A PDS M is normal if each instruction of M takes one of the following forms:

(1) $(a, S, b) \rightarrow (S', \sigma) \quad b \neq e$

(2) $(a, S, e) \rightarrow (S', c) \quad c \neq \sigma$ and not $a = c = e$.

A normal PDS references the output tape iff an erase instruction is executed; it has nc dormant instructions.

<u>Lemma 5a</u>: Given a PDS, one can construct an equivalent normal PDS.

<u>Proof</u>: Let M_o be a PDS. By a series of instruction replacements which do not affect generative capacity, we construct a new PDS M_{i+1} from M_i, $0 \leq i \leq 3$, until M_4 is normal.

(1) M_1 from M_o: Choose a new initial and final state $\hat{S}_o$ and a new output symbol $\hat{\#}$.

$$\text{Add the rules} \begin{cases} (e, \hat{S}_o, e\ ) \rightarrow (S_o, \hat{\#}) \\ (e, S_o, \hat{\#}) \rightarrow (\hat{S}_o, \sigma) \end{cases}$$

and replace $(a, S, \#) \rightarrow (S', c)$ by $(a, S, \hat{\#}) \rightarrow (S', c)$.

$\hat{\#}$ serves as a pseudo boundary symbol. All computation relative to $\#$ in M_o is relative to $\hat{\#}$ in M_1, but note that $\hat{\#}$ may be erased without terminating a computation.

(2) M_2 from M_1: replace $(a, S, b) \rightarrow (S',c)$ $c \neq \sigma$, $b \neq e$

$$\text{by} \begin{cases} (a, S, b) \rightarrow (\overset{S'}{bc}, \sigma) \\ (e, \overset{S'}{bc}, e) \rightarrow (\overset{S'}{c}, b) \\ (e, \overset{S'}{c}, e) \rightarrow (S', c) \end{cases}$$

where $bc^{S'}$ and $c^{S'}$ are new states.

(M_2 is a PDS with restricted control as defined by Chomsky (1963)).

(3) M_3 from M_2: replace $(a, S, e) \rightarrow (S', \sigma)$ by

$\{(a, S, b) \rightarrow (S', \sigma): b \in A_0\}$.

(4) M_4 from M_3: replace I_1 by I_{n+1} whenever $I_1, I_2, \cdots, I_{n+1}$ is an executable sequence of instructions of M, $I_1, I_2, \cdots, I_n$ are dormant and I_{n+1} is normal.

<u>Definition</u>: $x \in L(M,k)$ iff on some computation M accepts x and executes at most k consecutive erase-only instructions.

<u>Lemma 5b</u>: Given a normal PDS M one can construct an equivalent normal PDS M' such that $L(M') = L(M', 2)$.

<u>Proof</u>: Let $M = (A_I, A_0, \Sigma, S_0, \Phi)$

$M' = (A_I, A_0', \Sigma', S_0, \Phi')$

where $A_0' = A_0 \cup (\Sigma \times \Sigma)$ and $\Sigma' = \Sigma \cup \Sigma^3$.

In (1) through (6), below, we construct Φ' from Φ. To improve readability we write column vectors read from top to bottom for row vectors read from left to right. Any state occurring on the right, but not on the left of an implication sign, =>, ranges over all of Σ. For example, (2.3) is in Φ' for all $u \in \Sigma$.

(1) Φ' contains all of Φ, except for erase-only instructions.

(2)

$$\left.\begin{array}{c}(a,S,e)\to(S',a')\\[2em] (e,T',a')\to(T,\sigma)\\[2em] \text{in } \Phi\end{array}\right\} \Rightarrow \left\{\begin{array}{ll}(a,S,e)\to(S',\overset{T'}{T}) & (2.1)\\[2em] (a,S,e)\to\left(\begin{matrix}S' & \\ T' & \\ T, & e\end{matrix}\right) & (2.2)\\[2em] \left(\begin{matrix} & S & \\ & T & \\ a, & u, & e\end{matrix}\right)\to\left(\begin{matrix}S' & \\ T' & \\ u, & e\end{matrix}\right) & (2.3)\\[2em] \left(\begin{matrix} & S & \\ & u' & \\ a, & u, & e\end{matrix}\right)\to\left(\begin{matrix}S' & \\ T' & u' \\ T, & u\end{matrix}\right) & (2.4)\\[2em] \left(\begin{matrix} & S & \\ & T & \\ a, & u, & e\end{matrix}\right)\to\left(\begin{matrix} & T' \\ S', & u\end{matrix}\right) & (2.5)\\[2em] \text{in } \Phi'. & \end{array}\right.$$

(3)

$$\left.\begin{array}{c}(a,S,e) \rightarrow (S',e) \\ \\ \text{in } \Phi\end{array}\right\} \Rightarrow \left\{\begin{array}{c}\begin{pmatrix} S \\ T \\ a,u,e \end{pmatrix} \rightarrow \begin{pmatrix} S' \\ T \\ u,e \end{pmatrix} \quad (3.1) \\ \\ \text{in } \Phi'\end{array}\right.$$

(4)

$$\begin{pmatrix} & u' \\ e,T,u \end{pmatrix} \rightarrow \begin{pmatrix} T \\ u' \\ u,\sigma \end{pmatrix} \qquad \text{in } \Phi' \text{ for all } u, u', T \in \Sigma$$

(5)

$$\begin{pmatrix} u' \\ u'V' \\ e,W,V \end{pmatrix} \rightarrow \begin{pmatrix} W \\ V' \\ V,\sigma \end{pmatrix} \qquad \text{in } \Phi' \text{ for all } u', V, V', W \in \Sigma$$

(6)

$$\left.\begin{array}{l}(a,S,b) \rightarrow (S',c) \\ \text{in } \Phi \text{ and not erase-only}\end{array}\right\} \Rightarrow \left\{\begin{array}{l}\begin{pmatrix} u' \\ u' \\ a,S,b \end{pmatrix} \rightarrow (S',c) \\ \text{in } \Phi'\end{array}\right. \qquad (6.1)$$

M may have infinitely many eliminable strings, but only finitely many elimination pairs. Instead of writing an eliminable string x, M' has the option of computing (via 2.2, 2.3, 3.1) and writing (via 2.1, 2.4, 2.5) an elimination pair, say [u',u], for x.

In other words, M' may make a non-deterministic guess that M will subsequently enter state u' and eliminate x. To simulate M, M' need only erase one symbol, namely [u',u] (as in 4). If T = u', M' may follow (4) by (6.1). This corresponds to M eliminating x.

The preceding paragraph is an oversimplification. Let [V',V] and [V'',V] be elimination pairs for y and y y', respectively. Note that M may first write y x, subsequently eliminate x, and then write y'--extending y to an eliminable string y y'. To simulate this situation, M' must execute two consecutive erase-only instructions. M' must follow (4) with (5) and then, via (2.3) or (3.1), update [V',V] (the analogue of extending y), eventually writing [V'',V], via (2.4) or (2.5).

On the other hand, if M' correctly guesses the extent of all eliminable strings subsequently eliminated by M, it will never execute more than two consecutive erase-only instructions.

M' as constructed here will not, in general, be normal since (2.2) or (2.3) with a = e would introduce instructions of the form $(e,V,e) \rightarrow (V',e)$. However, these dormant instructions may be replaced as in Step (4) of Lemma 5a without affecting L(M',2).

To preserve unambiguity, certain combinations of instructions must be avoided. For example, any three instruction subcomputation of types (4), (5) and (6.1), in order, may be replaced by a two instruction subcomputation of types (4) and (6.1). We may suppose that M' immediately blocks on executing such combinations since there are only finitely many. This remark also applies to Lemma 5e.

Definition: A PDS M is of erase order k if M executes at most k consecutive erase-only instructions.

Lemma 5c: Given a normal PDS M and an integer $k \geq 0$, one can construct a normal PDS M' of erase order k such that L(M') = L(M,k).

Proof: The proof is informal. M' is the same as M except that M' has a counter θ, initially zero. θ is reset to zero whenever M' executes an instruction other than an erase-only instruction.

If M' executes an erase-only instruction and $\theta = k$, M' blocks; otherwise θ is increased by one. Since k is fixed, the counter can be stored within the state descriptions of M'.

Lemma 5d: Given a normal PDS M of erase order $k \leq 2$, one can construct an equivalent normal PDS M' of erase order k-1.

Proof: Let

$$M = (A_I, A_0, \Sigma, S_0, \Phi)$$

$$M' = (A_I, A_0', \Sigma', S_0, \Phi')$$

where

$$A_0' = A_0 \cup A_0^2$$

and

$$\Sigma' = \Sigma \cup (\Sigma \times A_0).$$

Φ': (1) Φ' contains all of Φ except for erase-only instructions.

(2) $$\left.\begin{array}{l}(a,S,e) \to (S',a') \\ (e,T',a') \to (T,\sigma) \\ \text{in } \Phi\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(a,S,e) \to ([S',a'],e) \\ \text{in } \Phi'\end{array}\right.$$

(3) $$\left.\begin{array}{l}(b,S,e) \to (S',b') \\ \text{in } \Phi,\ b' \neq e\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(b,[S,a'],e) \to (S',[a',b']) \\ \text{in } \Phi' \text{ for all } a' \in A_0\end{array}\right.$$

(4) $$\left.\begin{array}{l}(c,S,b') \to (S',\sigma) \\ \text{in } \Phi\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(c,S,[a',b']) \to ([S',a'],\sigma) \\ \text{in } \Phi' \text{ for all } a' \in A_0\end{array}\right.$$

(5) $$\left.\begin{array}{l}(e,S,a') \to (S',\sigma) \\ \text{in } \Phi\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(e,[S,a'],e) \to (S',e) \\ \text{in } \Phi'\end{array}\right.$$

The construction is fairly straightforward. Instead of writing two consecutive symbols a' and b', M' has the option (as in (2) and (3)) of incorporating both in a single symbol [a',b']--providing a' is eliminable by M. The elimination of a' is simulated by an identity transition (as in (5)).

The proof of the following lemma is similar to the proof of Lemma 5b, except that generation pairs play the role of elimination pairs. The proof is easier since M has no erase-only instructions.

<u>Lemma 5e</u>: If M is a normal PDS with no erase-only instructions, one may construct an equivalent real time normal PDS M'.

<u>Proof</u>: Let $M = (A_I, A_O, \Sigma, S_o, \Phi)$

$M' = (A_I, A_O', \Sigma', S_o, \Phi')$

where $A_O' = A_O \cup \Sigma^2 \cup (A_O \times \Sigma^2)$

$\Sigma' = \Sigma \cup \Sigma^3$

Once again to improve readability column vectors, read from top to bottom, replace the row vectors, read from left to right in (1) through (6) below.

Instead of generating strings, M' stores generation pairs. If M executes a read-write or read-only instruction after generating a string xb from state S, M' stores a generation pair [T,S] for xb by executing an instruction of form 2.1 or 3.1, respectively. If M executes a read-erase instruction after generating xb, M' stores a generation pair [u,S] for x via (4.1). Because of the conditions on M, these are the only possibilities.

A stored generation pair for a string y induces a non-deterministic computation when erased via (5.1) or (6.1). M' essentially simulates the simultaneous generation and erasure of y via (6.2) or (6.3). At any point M' may terminate the simulation by writing a generation pair, via (6.4), for that part of y not yet "erased".

(1) Φ' contains all of Φ, except for write-only instructions.

(2) $$\left.\begin{array}{l} (a,T,e) \rightarrow (T',a') \\ \\ a' \neq e,\ \sigma \text{ in } \Phi \\ \\ \end{array}\right\} \Rightarrow \left\{\begin{array}{l} (a,S,e) \rightarrow \left(\begin{array}{c} a' \\ T \\ T',S \end{array}\right) \\ \\ [T,S] \text{ generation pair} \quad (2.1) \\ \\ \quad \text{in } \Phi' \end{array}\right.$$

$$\left.\begin{array}{l}(3)\quad (a,T,e) \rightarrow (T',e) \\ \\ \qquad\qquad \text{in } \Phi \\ \end{array}\right\} \Rightarrow \left\{\begin{array}{l} (a,S,e) \rightarrow (T',\overset{T}{S}) \\ \\ [T,S] \text{ generation pair} \quad (3.1) \\ \\ \qquad\qquad \text{in } \Phi' \end{array}\right.$$

$$\left.\begin{array}{l}(4)\quad (a,T,b) \rightarrow (T',\sigma) \\ \\ \qquad (e,u,e) \rightarrow (T,b) \\ \\ \qquad\qquad \text{in } \Phi \end{array}\right\} \Rightarrow \left\{\begin{array}{l} (a,S,e) \rightarrow (T',\overset{u}{S}) \\ \\ [u,S] \text{ generation pair} \quad (4.1) \\ \\ \qquad\qquad \text{in } \Phi' \end{array}\right.$$

$$\left.\begin{array}{l}(5)\quad (a,S,b) \rightarrow (S',\sigma) \\ \\ \qquad\qquad \text{in } \Phi \end{array}\right\} \Rightarrow \left\{\begin{array}{l} \begin{pmatrix} & & b \\ & & T \\ a,&S,&u \end{pmatrix} \rightarrow \begin{pmatrix} S' \\ T \\ u,\sigma \end{pmatrix} \qquad\qquad (5.1) \\ \\ \qquad \text{in } \Phi' \end{array}\right.$$

$$(6)\qquad \left.\begin{array}{l}(a,S,b)\to(S',\sigma)\\[6em](e,T,e)\to(T',b)\end{array}\right\} \Rightarrow \left\{\begin{array}{lcll} \begin{pmatrix} & & T' \\ a, & S, & u\end{pmatrix} & \to & \begin{pmatrix} S' \\ T \\ u,\sigma\end{pmatrix} & (6.1) \\[3em] \begin{pmatrix} S \\ T' \\ a,u,e\end{pmatrix} & \to & \begin{pmatrix} S' \\ T \\ u,e\end{pmatrix} & (6.2) \\[3em] \begin{pmatrix} S \\ T' \\ a,T,e\end{pmatrix} & \to & \begin{pmatrix} S',e\end{pmatrix} & (6.3) \\[3em] \begin{pmatrix} S \\ T' \\ a,u,e\end{pmatrix} & \to & \begin{pmatrix} & T \\ S', & u\end{pmatrix} & (6.4) \end{array}\right.$$

(III) DETERMINISTIC PUSH DOWN STORAGE AUTOMATA[1]

In Section II we indicated that there were several ways to define PDS. For non-deterministic PDS most formalisms are equivalent, i.e., each formalism gives rise to the same family of acceptable languages--the context-free languages. However, consider the following definition.

Definition: A PDS M is deterministic if each situation of M determines at most one executable instruction.

Because of Theorem 5 we may consider rt PDS a new formalism for PDS. But although rt PDS = PDS, we will show below that deterministic rt PDS $\subsetneq$ determistic PDS. Therefore, the generative capacity of the automata defined by the above definition is not independent of the original formalism chosen for PDS. Several families of automata have been studied intermediate in power between deterministic rt PDS and the family of deterministic PDS considered here.

By definition, a deterministic PDS M accepts each string of L(M) with a unique computation and rejects each string of $\overline{L(M)}$ with a unique computation. The converse is not strictly true, but the following theorem is almost a tautology.

Theorem 6: If a PDS M accepts each string of L(M) uniquely and rejects each string of $\overline{L(M)}$ uniquely, there is an equivalent deterministic PDS M'.

[1]Several results similar to those proved here have been independently discovered by Dr. Seymour Ginsberg and Dr. Sheila Greibach.

Proof: Suppose two (call them ambiguous) instructions I_1 and I_2 of M are both executable in the situation (a,S,b). If situation (a,S,b) occurs during the course of an M computation with input x, then $x \in L(M)$ because of the conditions on M. Therefore, without affecting generative capacity, the pair of instructions $[I_1, I_2]$ may be replaced by a deterministic submachine which transforms any machine-tape configuration (xb, S, ay) into (e,S_0,e). Note that if one, but not both, of the instructions I_1, I_2 does not read the input tape, then it is essential that the first instruction executed by the submachine also not read the input tape. By replacing all ambiguous pairs in this way, we construct deterministic M' equivalent to M.

There are, of course, languages acceptable by unambiguous PDS which cannot be accepted by any deterministic PDS. For example, it is a simple exercise to construct an unambiguous PDS which accepts $L = \{z: \ z = x\hat{x}, \ x \in \{a,b\}^*\}$. Once Theorem 8 is established, it is fairly easy to prove that no deterministic PDS can accept L.

Theorem 7: Given a deterministic PDS M there is an equivalent unambiguous real time PDS M'.

Proof: Immediate corollary of Theorem 5.

The constructions carried out in proving Theorem 5 preserve unambiguity; they do not preserve determinacy. In general, one cannot remove erase-only instructions without simultaneously introducing non-determinacy. However, we have the following theorem.

Theorem 8: If M is a deterministic PDS, there is an equivalent normal deterministic PDS M' without write-only instructions.

Proof: Our first proof is non-constructive. We may assume M normal since the proof of Lemma 5a preserves determinacy.

Suppose that the configuration $K = (y\ b,\ S,\ z)$, $z = az'$ or $z = e$, occurs during the course of an M computation. Unless M blocks or endlessly executes write-only and erase-only instructions, finitely many steps after K one of the following configurations must occur first, for some state T.

(1) $(y\ b,\ T,\ z')$

(2) $(y,\ T,\ z)$

(3) $(y,\ T,\ z')$

(4) $(y\ b\ x,\ T,\ z')$, $\quad x = a_1a_2 \cdots a_n$

If (4) occurs first, we say that M ultimately produces x. Since M is deterministic, the number of strings ultimately produced is less than the number of situations of M and is therefore finite. Let m be the length of the longest string ultimately produced by M.

If $M = (A_I, A_O, \Sigma, S_o, \Phi)$ we will construct M' with

$$A_O' = \bigcup_{k=1}^{m} A_O^i$$

and

$$\sum{}' = \sum \times \left(\bigcup_{k=1}^{m-1} A_O^k \right).$$

If $w = b_1 b_2 \cdots b_k \in A_O^*$ define

$$< w > = [b_1, b_2, \cdots, b_k] \in A_O^k$$

and

$$Sw = [S, b_1, b_2, \cdots, b_k] \in \sum \times A_O^k \quad .$$

$$Se = S \in \sum .$$

Φ' consists of the following instructions.

(I) Φ' contains Φ except for write-only instructions.

(II) If a,S,b and T are as above, $(e,S,e) \to (S',a')$ is in Φ and (1), (2), (3), or (4) occurs then:

(1') $(a,S,e) \to (T,e)$ in Φ' if (1) occurs first

(2') $(e,S,b) \to (T,\sigma)$ in Φ' if (2) occurs first

(3') $(a,S,b) \to (T,\sigma)$ in Φ' if (3) occurs first

$$\left.\begin{array}{ll}(4')\ (i) & (a,S,e) \to (T,<x>) \\ \quad (ii) & (a,Sw,e) \to (Tx, <w>)\end{array}\right\} \begin{array}{l}\text{are in } \Phi' \text{ if (4)} \\ \text{occurs first for } w \ni : \\ 1 \leq \lambda(w) < m\end{array}$$

$$\text{(III)}\quad \left.\begin{array}{l}(c,u,d) \to (u',\sigma) \\ \text{in } \Phi\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(c,u,<wd>) \to (u'w,\sigma) \\ (c,uwd,e) \to (u'w,e) \\ \text{in } \Phi' \text{ for } w \ni :\ 0 \leq \lambda(w) < m-1\end{array}\right.$$

$$\text{(IV)}\quad \left.\begin{array}{l}(c,u,e) \to (u',c') \\ \text{in } \Phi,\ c \neq e,\ c' \neq e,\sigma\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(c,uw,e) \to (u',<wc'>) \\ \text{in } \Phi' \text{ for } w \ni :\ 1 \leq \lambda(w) < m\end{array}\right.$$

$$\text{(V)}\quad \left.\begin{array}{l}(c,u,e) \to (u',e) \\ \text{in } \Phi,\ c \neq e\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(c,uw,e) \to (u',<w>) \\ \text{in } \Phi',\ \text{for } w \ni :\ 1 \leq \lambda(w) < m\end{array}\right.$$

The crucial instructions above are introduced in II (4')(i). Instead of proceeding from K with a computation that begins with $(e,S,e) \rightarrow (S',a')$, possibly executing other write-only and erase-only instructions before reading a, M' immediately reads a and writes one symbol--the vector which corresponds to the string x ultimately produced by M.

Instructions introduced by III, IV, and V merely enable M' to reference components of the vector. Instructions introduced by II (4')(ii) simulate the case where M ultimately produces w', subsequently erases w', and then ultimately produces x.

Note that the instructions II are independent of y and z' and depend only on the situation (a,S,b), or (#,S,b) if a = e.

To convert the above proof into a constructive proof, we need a rule that guarantees that one of (1), (2), (3), or (4) occurs within a computable number of steps after K or else never occurs. The following definition plays a key role in determining such a rule.

Definition: M produces ub' with production quadruple [b,S,b',S'] if there is an M computation C such that:

(1) C takes M from configuration (yb,S,z) to (ybub',S',z).

(2) The length of the contents of the storage tape is never less than $\lambda(yb)$ during C.

Lemma 8a: Let M have p states and q output symbols. Then if M is started in the configuration $K = (yb,S,z)$, $z = az'$ or $z = e$, then (1), (2), (3), or (4) occurs within pq^{pq} steps after K or not at all.

Proof: If M produces a string u, it produces all of its substrings. Hence for u sufficiently long, e.g., $\lambda(u) > pq$, M produces a substring vd of u with production quadruple $[d,V,d,V]$ for some d and V. In this case, since M is deterministic, it produces $(vd)^k$ for all k so that none of (1), (2), (3), or (4) occurs.

On the other hand, if within pq^{pq} steps after K none of (1), (2), (3), or (4) occurs and M fails to produce a string of length $pq+1$, then a machine-tape configuration K' must recur. This follows since for pq^{pq} steps after K the length of the contents of the storage tape would be bounded by $\lambda(yb)$ and $\lambda(yb) + pq$ while the input tape remains fixed. Since M is deterministic, the recurrence of K' before (1), (2), (3), or (4) occurs, implies that they never occur.

Theorem 9: If M is a deterministic PDS, there is an equivalent normal deterministic PDS M' which terminates each computation with input x within $2 \cdot \lambda(x) + 1$ steps.

Proof: By Theorem 8 there is a normal deterministic PDS M' equivalent to M but without write-only instructions. If

n_1, n_2, n_3 and n_4 are, respectively, the number of occurrences of read-write, read-erase, read-only and erase-only instructions in an M' computation with input x, then

$$n_1 + n_2 + n_3 + n_4 \leq \lambda(x) + 1 + n_4 \leq$$

$$\lambda(x) + 1 + n_1 \leq 2 \cdot \lambda(x) + 1 .$$

This is a best-possible real time result for deterministic PDS. No matter what formalism is initially chosen for PDS, it is clear that no deterministic PDS can accept

$$L = \{z: \ z = a^n b^p a^q b^n;\ n > 0,\ p > q > 0\}$$

in real time.

Theorem 10: If M is a deterministic PDS, there is a deterministic PDS M' which accepts $\overline{L(M)}$.

Proof: By Theorem 9 we may assume that each M computation terminates and by Step (1) of Lemma 5a we may assume that M employs a pseudo-boundary symbol on its storage tape. It is therefore a simple exercise to construct a deterministic PDS M' which accepts x iff M rejects x. The details are omitted.

An immediate corollary of Theorem 10 is a new formal proof that nondeterministic PDS $\supsetneq$ deterministic PDS; i.e., one need only exhibit a CF L such that $\bar{L}$ is not CF.

(IV) STRONG EQUIVALENCE

In this section we give a new proof of the Chomsky-Schützenberger result that the context free languages are just those languages accepted by PDS automata. The proof establishes a strong equivalence in the sense that unambiguity is preserved. Thus, the unambiguous CF languages are just those accepted by unambiguous PDS. A byproduct of the construction is a new normal form for context-free grammars, stronger than the Greibach-normal form, and essentially a dual of the real time normal form for PDS. Left context sensitive grammars are defined and shown to be no more powerful than context-free grammars.

Theorem 5 plays a key role in proving Theorem 11.

<u>Theorem 11</u>: If a PDS M accepts L, then L is CF.

<u>Proof</u>: By Theorem 5 we may assume that M is a real time normal PDS. Furthermore, we may assume that M has no instructions of the form $(\#,S,b) \rightarrow (S_o,\sigma)$. Simply replace each such instruction by $(\#,S,e) \rightarrow (S_o',e)$ and $(e,S_o',b) \rightarrow (S_o,\sigma)$, where S_o' is a new state, and again apply the construction of Theorem 5. Let $M = (A_I, A_O, \Sigma, S_o, \Phi)$.

We will construct a CF grammar $G = (V_T, V_N, S,R)$ for L so that $V_T = A_I$ and $V_N = \{S\} \cup (\Phi_1 \times \Phi_2)$ where Φ_1 consists of the read-only and read-write instructions of M, and Φ_2 consists of the read-only and read-erase instructions of M.

In order to motivate the direct construction of R from Φ itself, we will first construct a subset R(C) of R from an acceptable M computation $C = (I_1, I_2, \cdots, I_{n+1})$ for an input string $x = b_1 b_2 \cdots b_n$.

The following definitions simplify the discussion.

Definition: If I_k is a read-only instruction, the occurrence of b_k in x is called C-free.

Definition: If I_k is a read-write instruction $(b_k, S, e) \rightarrow (S', b_k')$ then, since C is acceptable, for some j $(n \geq j > k)$ a read-erase instruction I_j must erase b_k' written by I_k. In this case we say that the occurrence of b_k in x is C-matched by the occurrence of b_j in x.

For simplicity we will drop "the occurrence of" and say b_k is C-free or b_k is C-matched by b_j.

If $n = 1$, take $R(C) = \{S \rightarrow b_1\}$

If $n = 2$, take $R(C) = \{S \rightarrow b_1 b_2\}$

For $n \geq 3$, R(C) consists of the following rules, provided they are well formed. If a non-terminal $[I_j, I_k]$ appears in a rule, the rule is well formed only if $I_j \in \Phi_1$, $I_k \in \Phi_2$ and $1 \leq j \leq k \leq n$.

(1) $S \rightarrow b_1 \; [I_2, I_n]$	if b_1 is C-free
$S \rightarrow b_1 b_2 \; [I_3, I_n]$	if b_1 is C-matched by b_2
$S \rightarrow b_1 \; [I_2, I_{n-1}] \; b_n$	if b_1 is C-matched by b_n
$S \rightarrow b_1 \; [I_2, I_{m-1}] \; b_m \; [I_{m+1}, I_n]$	if b_1 is C-matched by b_m, $2 < m < n$.
(2) $[I_p, I_q] \rightarrow b_p \; [I_{p+1}, I_q]$	if b_p is C-free
$[I_p, I_q] \rightarrow b_p b_{p+1} [I_{p+2}, I_q]$	if b_p C-matched by b_{p+1}
$[I_p, I_q] \rightarrow b_p \; [I_{p+1}, I_{q-1}] b_q$	if b_p C-matched by b_q
$[I_p, I_q] \rightarrow b_p [I_{p+1}, I_{r-1}] b_r [I_{r+1}, I_q]$	if b_p C-matched by b_r, $p + 1 < r < q$.
(3) $[I_p, I_p] \rightarrow b_p$	if b_p is C-free
$[I_p, I_{p+1}] \rightarrow b_p b_{p+1}$	if b_p is C-matched by b_{p+1}

The conditions on b_1 are mutually exclusive and exhaustive so that precisely one rule from (1) belongs to R(C). Similarly, for fixed p and q, precisely one of the rules (2) and one of the rules (3) belong to R(C).

Clearly S => x by means of an R(C) left to right S derivation which is a generative analogue of the machine computation C. Conversely, each R(C) left to right S derivation of a string y gives rise to an acceptable M computation for y. Furthermore, if C' is another acceptable M computation, the previous sentence holds with R(C) U R(C') in place of R(C).

At this point we could stop if we were content with a non-constructive proof by simply taking $R = R_1 = \cup \{R(C_z): C_z$ is an acceptable M computation for $z\}$. However, we can now readily describe the construction of R directly from M itself.

Again some definitions are useful.

Definition: a is free in I if I has the form $(a,S,e) \rightarrow (S',e)$.

Definition: The non-terminal [I,J] matches a and b if

$$I = (a, S, e) \rightarrow (S', a')$$

and

$$J = (b, T, a') \rightarrow (T', \sigma).$$

$[I,J]$ links a and b if, in addition, $T = S'$.

<u>Definition</u>: If I in Φ and $I = (a,S,b) \to (S',c)$, a successor I^+ of I is an instruction of Φ_1 of the form $(d,S',e) \to (S'',f)$ a predecessor I^- of I is an instruction of Φ_2 of the form $(d,S'',f) \to (S,g)$. I is pre-terminal if $(\#,S',e) \to (S_0,e)$ is in Φ and $I \in \Phi_2$. $I = (a,S_0,e) \to (S',c)$ is initial.

Providing all instruction pairs are in $\Phi_1 \times \Phi_2$, R consists of the following instructions for all definable I^+, J^-, J^+, K^-.

(1) If I is initial and K is pre-terminal:

$S \to a\ [I^+,K]$	if a is free in I
$S \to ab[J^+,K]$	if $[I,J]$ links a and b
$S \to a[I^+,K^-]b$	if $[I,K]$ matches a and b
$S \to a[I^+,J^-]\ b\ [J^+,K]$	if $[I,J]$ matches a and b.

(2)

$[I,J] \to a[I^+,J]$	if a is free in I
$[I,K] \to ab[J^+,K]$	if $[I,J]$ links a and b
$[I,J] \to a[I^+,J^-]b$	if $[I,J]$ matches a and b
$[I,K] \to a[I^+,J^-]\ b\ [J^+,K]$	if $[I,J]$ matches a and b.

(3) $[I,I] \rightarrow a$ if a is free in I

$[I,J] \rightarrow ab$ if $[I,J]$ links a and b.

Clearly $R_1 \subset R$ and any rule in $R - R_1$ can never be used in a terminal derivation. Therefore

$$L(G(V_T, V_N, S, R)) = L(G(V_T, V_N, S, R_1)) = L(M).$$

Theorem 12: If M is an unambiguous PDS, L(M) has an unambiguous CF grammar.

Proof: Corollary of the proof of Theorem 11 (and Theorem 5).

Corollary 12a: If M is a deterministic PDS, L(M) has an unambiguous CF grammar.

Proof: By definition deterministic PDS $\subset$ unambiguous PDS.

For real time normal PDS the proof of Theorem 11 is an efficient algorithm for constructing a simple strongly equivalent CF grammar. Since a specific PDS can usually be brought into real time normal form without applying the complicated constructions of Theorem 5, the proof of Theorem 11 is a useful research tool. The Chomsky-Schützenberger construction introduces ambiguity and an enormous number of non-terminals, regardless of the initial form of the PDS.

For example, by the method of Theorem 11 we can construct a simple unambiguous CF grammar for the Dyck languages D_{2n} on 2n symbols. Let M be the following PDS:

$$(\alpha, S_o, e) \rightarrow (S_1, \alpha)$$

$$(\alpha, S_1, \beta) \rightarrow (S_1, \alpha), \qquad \alpha \neq \beta^{-1}$$

$$(\alpha, S_1, \beta) \rightarrow (S_1, \sigma), \qquad \alpha = \beta^{-1}$$

$$(\#, S_1, e) \rightarrow (S_o, e)$$

Then $L(M) = D_{2n}$ if α and β range over an alphabet of n symbols and their unique inverses. It is a simple exercise to bring M into real time normal form without destroying its determinacy. Therefore Theorem 11 yields the desired grammar.

Definition: If G is any CS grammar, the left language $L_\ell(G)$ generated by G is the set of all $x \in L(G)$ such that $S \Rightarrow x$ by means of a left to right derivation. The right language $L_r(G)$ is defined similarly.

Mathews (1963) has shown that $L_r(G)$ and $L_\ell(G)$ are CF for any CS grammar G. In particular, the left language generated by any CF grammar is CF. A related but different notion is given in the following definition.

<u>Definition</u>: A left context sensitive (LCS) grammar G is a CS grammar with all rules of the form $\phi A \rightarrow \phi\omega$.

It is not hard to exhibit LCS grammars G for which $L_\ell(G) \neq L(G)$ and we give a simple example below. Furthermore, there are LCS grammars G such that $L_\ell(G) \neq L(G)$ no matter how the definition of left to right derivation is generalized (e.g., see Mathews, (1963)).

We will prove that L(G) is CF if G is a LCS grammar (Theorem 15). For expository purposes we will first prove Theorem 13 which is actually a corollary of Theorem 15.

<u>Lemma 13a</u>: Given a LCS grammar there is an equivalent LCS with all rules of the form $A \rightarrow a$, $A \rightarrow BD$ or $AC \rightarrow AD$.

<u>Proof</u>: Let $G = (V_N, V_T, S, R)$ be a LCS grammar.

(1) Take $G' = (V'_N, V'_T, S, R')$ where $V'_N = V_N \cup V_T$, $V'_T = V_T \times \{1\}$ and

$$R' = R \cup \{a \rightarrow [a,1]: \ a \in V_T\}.$$

Clearly G' is equivalent to G and each rule of G' has the form $XA \rightarrow XY$ $(Y \neq e)$ or $A \rightarrow a$ where $X, Y \in (V'_N)^*$, $A \in V'_N$, $a \in V'_T$.

(2) Replace each rule of G' of the form XGBA $\rightarrow$ XGBY by the set of rules

$$\{XGB \rightarrow XGB^{AY},\ B^{AY} A \rightarrow BA_Y,\ A_Y \rightarrow Y\}$$

where B^{AY} and A_Y are (possibly) new non-terminals, until all rules of G' have the form AC $\rightarrow$ AD, A $\rightarrow$ Y ($Y \neq e$) or A $\rightarrow$ a.

(3) Replace each rule of G' of the form A $\rightarrow$ BCDZ by the pair of rules A $\rightarrow$ BC_{DZ} and $C_{DZ} \rightarrow$ DZ, where C_{DZ} is a (possibly) new non-terminal, until each rule of G' has the form A $\rightarrow$ a, A $\rightarrow$ BD or AC $\rightarrow$ AD.

Corollary 13a: Given a CF grammar there is an equivalent CF grammar with all rules of the form A $\rightarrow$ d or A $\rightarrow$ BD.

Proof: Follows from proof of the lemma.

Theorem 13: If G is CF, one can construct a PDS M which accepts L(G). Moreover if G is unambiguous, so is M.

Proof: Let G = (V_N, V_T, S, R) be a CF grammar for L. By Corollary 13a we may assume that all rules of G have the form A $\rightarrow$ BD or A $\rightarrow$ a.

Take $A_0(M) = V_N$, $\Sigma(M) = \{S_o, S_1, S_2\} \cup V_N \cup V_N^2$ and $\delta(M)$ as follows, where we write S^D and S^{BD} for D and $[B,D]$ of $\Sigma(M)$ to improve readability.

$$\left.\begin{array}{ll}(1) & (a,S_o e) \to (S_1,A), \\ & (a,S_1,e) \to (S_1,A) \\ & \quad \text{in } M\end{array}\right\} \qquad \text{if } A \to a \text{ in } R$$

$$\left.\begin{array}{ll}(2) & (e,S_1,D) \to (S^D, \sigma) \\ & (e,S^D,B) \to (S^{BD},\sigma) \\ & \quad \text{in } M\end{array}\right\} \qquad \text{for all } B,\ D \in V_N$$

$$\left.\begin{array}{ll}(3) & (e,S^{BD},e) \to (S_1,A) \\ & \quad \text{in } M\end{array}\right\} \qquad \text{if } A \to BD \text{ in } R$$

$$\left.\begin{array}{ll}(4) & (e,S_1,S) \to (S_2,\sigma) \\ & (\#,S_2,e) \to (S_o,e) \\ & \quad \text{in } M\end{array}\right\}$$

If G is CF, then $L_r(G) = L(G)$. Given an input string x, M tries all possible right to left derivations for x. At any step in an acceptable M computation, the concatenation of the contents of the storage and input tapes is a line of a right to left derivation.

The rules (2) and (3) permit M to alter the contents of the storage tape from XBD to XA, providing $A \to BD$ is a rule of G. Hence M works backward from the terminal string. If, on some computation, M can reduce x to S, then M accepts x via (4).

Theorem 14: If G is a CF grammar, there is an equivalent CF grammar G' with all rules of the form:

$A \to a$

$A \to aB$

$A \to aBb$

$A \to aBbC$

Proof: By applying Theorems 13 and 11 in succession, we can construct G' with all rules as above, except for rules of the form $A \to abB$ and $A \to ab$, which are replaceable by rules of the form $A \to a$ and $A \to aB$.

This normal form, dual to real time PDS normal form, may lead to an interesting characterization of CF languages.

The proof of Theorem 13 rests upon an unproved but obvious lemma, namely: $L_r(G) = L(G)$ for CF G. For LCS G this result does not hold. For example, let G_o be the following LCS grammar:

$$\begin{array}{ll}
S \to GH & A \to a \\
G \to AB & B \to b \\
H \to CD & C' \to c' \\
BC \to BC' & D \to d
\end{array}$$

Then $L_r(G_o) = L_\ell(G_c) = \not{\phi}$, but $L(G_o) = \{a\ b\ c'\ d\}$.

Because of this, Theorem 15 cannot be proved by merely appending the instructions (5) below to the PDS M of Theorem 13.

$$(5)\quad \left.\begin{array}{l}
(e, S^{BD}, e) \to (T^C, B) \\
(e, T^C, e) \to (S_1, C) \\
\qquad \text{in } M
\end{array}\right\} \qquad \text{if } BC \to BD \text{ in } R$$

M so constructed would only accept $L_r(G)$.

The LCS node tree of Figure 1 below captures the essential structure of the derivation of a b c' d in G_o.

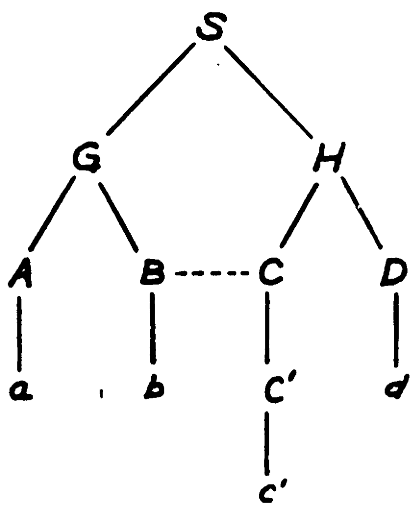


Figure 1

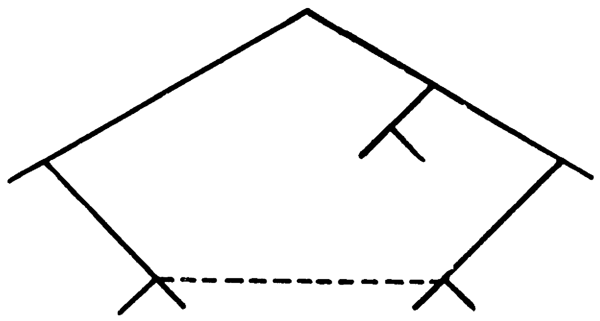


Figure 2

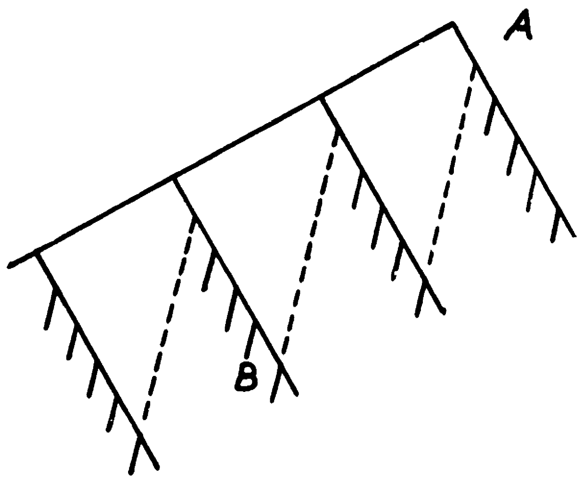


Figure 3

In a similar manner any derivation of any left sensitive grammar (in the normal form of Lemma 13) may be represented. Evidently left sensitive node trees are context-free node trees except that some nodes are connected by dotted lines to indicate the use of a rule of form $BC \to BC'$.

Clearly only certain nodes may be connected by dotted lines. The rules are simple.

(1) If $A \Rightarrow \phi B \psi$ in the derivation, no dotted line may join A and B.

(2) After joining all terminal symbols with a line, no dotted line may intersect any other line--dotted or undotted.

Conversely, any node tree which satisfies (1) and (2) represents a derivation in some left sensitive grammar. An immediate consequence of (2) is the following rule.

Rule (*) Any closed polygon in a node tree which does not properly contain another closed polygon has empty interior. In other words, no left sensitive node tree can have a subtree of the form indicated in Figure 2.

Let us write $B \ll A$ if A and B are nodes in a fixed tree and $A \Rightarrow \phi B \psi$. With respect to $\ll$ any node tree is an upper semi-lattice. Let $[A,B] = \text{lub}(A, B, \ll)$. Let A_ℓ denote the immediate lower left node of a binary node A. Finally write $A \sim B$ if A and B are connected by a dotted line and $A \ll [A,B]_\ell$.

The PDS M of Theorem 13 essentially traces the nodes of a possible CF node tree for a given input in the following order: A before B, i.e., $A < B$ iff ($A \ll B$ or $A \ll [A,B]_\ell$).

One might hope to prove Theorem 14 by constructing a PDS M' the same as M except that M' occasionally guesses $A \sim B$ and therefore ignores A and all $C > A$ until B is traced. However, it is easy to see that no such simple scheme can handle a node tree of the sort indicated in Figure 3.

Furthermore, a structure exactly like A could be attached at node B. This rules out even extensive modifications of M. To prove Theorem 15 we use a PDS automaton as a device which maps one language into another.

Definition: A PDS $M = (A_I, A_O, \Sigma, S_o, \mathfrak{A})$ maps x into y if some computation takes M from configuration (e, S_o, x) into (y, S_o, e). For $L \subset A_I^*$, M(L) is the set of all y such that M maps $x \in L$ into y.

Warning: M(L) is not the language transduced by M. Push down storage transducers employ three tapes, input, storage, and output.

<u>Theorem 15</u>: If G is a LCS grammar, L(G) is CF.

<u>Proof</u>: Let $G = (V_N, V_T, S, R)$. By Lemma 13a we may assume that all rules of G have the form $A \to BC$, $BC \to BD$ or $A \to a$.

Consider the CF grammar $G' = (V_N, V_T', S, R')$ where

$$V_T' = V_T \cup (V_N \times \{1,2\})$$

and R' consists of the following rules.

(1) All CF rules of R are in R'.

(2) $A \to A\ [A,1]$ in R' for all $A \in V_N$.

(3) $$\left.\begin{array}{c} AB \to AD \\ \text{in } R \end{array}\right\} \Rightarrow \left\{\begin{array}{c} B \to [A,2]\ D \\ \text{in } R' \end{array}\right.$$

Let θ be the homomorphism on $(V_N \times \{1\})^* \ni \theta([A,1]) = [A,2]$ for all $A \in V_N$.

Suppose that a fixed node, say A, of a G derivation tree is such that $A \sim B_1, A \sim B_2, \ldots, A \sim B_n$. That is, A figures in n rules of the form $AB_i \to AD_i$, $1 \leq i \leq n$. By means of the rules (2) and (3) there is a corresponding G' derivation in which $A \Rightarrow A\,[A,1]^n$ and $B_i \Rightarrow [A,2]D_i$, $1 \leq i \leq n$.

Hence, it follows from rule (*) above that

$$(4)\quad z \in L(G) \text{ iff } \begin{cases} \exists\, w \in L(G') \ni \text{ for some } k \\ w = x_o y_1 y_1' x_1 y_2 y_2' \cdots y_k y_k' x_k \text{ where} \\ z = x_o x_1 \cdots x_k \text{ and } y_i' \\ \quad = \theta(\hat{y}_i)\ 1 \leq i \leq k. \end{cases}$$

Let M be the following PDS:

$(e,S_0,e) \rightarrow (S_1,e)$

$(a,S_1,e) \rightarrow (S_1,a)$ for all $a \in V_T \cup (V_N \times \{1\})$

$(\theta(a),S_1,a) \rightarrow (S_1,\sigma)$ for all $a \in V_N \times \{1\}$

$(\#,S_1,e) \rightarrow (S_0,e)$

Because of (6) we have $L(G) = M(L(G'))$. Since $L(G')$ is context free, Theorem 17 implies that $L(G)$ is context free.

Bar-Hillel, Perles and Shamir proved the weak form of the following theorem, and Chomsky and Schützenberger used it to prove CF = PDS. We deduce the strong form as a fairly easy corollary of our independently proved CF = PDS result.

Theorem 16: If L is CF and T is a transducer, then T(L) is CF. Moreover, if T maps distinct strings of L into distinct strings and L is unambiguous then T(L) is unambiguous.

Proof: Let M be a real time normal PDS which accepts L. By using the identity $T(L) = \{x: T^{-1}(x) \cap L \neq \emptyset\}$ we will construct a PDS M' which accepts T(L). M' dovetails the computation of T^{-1} and M so that M' accepts y iff T^{-1} maps y into x on some computation and M accepts x on some computation.

Take

$$\Sigma(M') = \Sigma(T^{-1}) \times \Sigma(M). \quad S_o(M') = [S_o(T^{-1}), S_o(M)]$$

$$\left.\begin{array}{c}(a,S) \rightarrow (S',e) \\ \text{in } T^{-1}\end{array}\right\} \Rightarrow \left\{\begin{array}{c}(a,[S,V],e) \rightarrow ([S',V],e) \\ \text{in } M' \text{ for all } V \in \Sigma(M)\end{array}\right.$$

$$\left.\begin{array}{c}(A,S) \rightarrow (S',b) \text{ in } T^{-1} \\ (b,V,c) \rightarrow (V',d) \text{ in } M \\ b \neq e\end{array}\right\} \Rightarrow \left\{\begin{array}{c}(a,[S,V],c) \rightarrow ([S',V'],d) \\ \text{in } M'\end{array}\right.$$

By Theorem 13 if L has an unambiguous grammar, we may assume that M is unambiguous. Suppose that T maps distinct strings of L into distinct strings. This simply means that T^{-1} cannot map a string into distinct strings of L and therefore M' is unambiguous. Therefore by Theorem 12 T(L) has an unambiguous CF grammar.

<u>Corollary 16a</u>: If L is CF and R is regular, then $L \cap R$ is CF. If L has an unambiguous CF grammar, so does $L \cap R$.

<u>Proof</u>: We can choose T so that $T(L) = L \cap R$.

<u>Theorem 17</u>: If L is CF and M is a PDS, then M(L) is CF.

<u>Proof</u>: The proof is a generalization of the argument used in proving Theorem 16.

(V) LINEAR BOUNDED AUTOMATA

Kuroda defined nondeterministic linear bounded automata (LBA) and proved the following two results.

(1) The CS languages are just those languages accepted by LBA.

(2) If M is a deterministic LBA, then $\overline{L(M)}$ is CS.

In this section we sharpen both results and give easier proofs. In Sections III and IV we proved that the family of languages accepted by deterministic PDS are closed under complement and have unambiguous CF grammars. The sharper results proved here establish the analogous theorem for deterministic LBA and unambiguous CS languages.

A linear bounded automaton (LBA) is a nondeterministic Turing Machine $(A_I, A_O, \Sigma, S_o, \Sigma_f, \Phi)$ which can only read and write symbols on those squares of its tape which contain the input string. The output alphabet A_O is, in general, larger than the input alphabet A_I. In another formalism the alphabets are the same but the LBA is able to use a scratch tape K times the length of its input for arbitrary fixed K (i.e., each machine is assigned one integer K for all inputs). For details the reader is referred to Kuroda (1964).

A deterministic LBA is an LBA with a single valued transition function Φ.

<u>Theorem 18</u>: If an LBA M accepts L, then L is context sensitive. Moreover, if M is deterministic, then L has an unambiguous CS grammar.

<u>Proof</u>: We begin by constructing a Type 1 grammar G for $L\# = \{x\#: \ x \in L\}$.

Take

$$V_T = A_I \cup \{\#\}$$

$$V_N = \{A_1\} \cup \{A_2\} \cup (A_0 \times A_I) \cup (\Sigma' \times A_0 \times A_I)$$

where

$$\Sigma' = \Sigma \cup \{T\}, \ T \notin \Sigma .$$

A_1 is the initial symbol of G.

To improve readability and also to help motivate the construction of G from M, column vectors read from top to bottom are used instead of row vectors read from left to right.

(1) The following rules are in G for all $a \in A_I$.

$$A_1 \rightarrow \begin{pmatrix} S_o \\ a \\ a \end{pmatrix} A_2$$

$$A_2 \rightarrow \begin{pmatrix} a \\ a \end{pmatrix} A_2$$

$$A_2 \rightarrow \#$$

(2) If $(S,a) \rightarrow (a',S',\ell)$ is in M, then

(i) $\ell = 0 \} \Rightarrow \begin{pmatrix} S \\ a \\ b \end{pmatrix} \rightarrow \begin{pmatrix} S' \\ a' \\ b \end{pmatrix}$ in G for all $b \in A_I$.

(ii) $\ell = -1 \} \Rightarrow \begin{pmatrix} c \\ d \end{pmatrix} \begin{pmatrix} S \\ a \\ b \end{pmatrix} \rightarrow \begin{pmatrix} S' \\ c \\ d \end{pmatrix} \begin{pmatrix} a' \\ b \end{pmatrix}$ in G for all

$b, d \in A_I$, $c \in A_0$.

(iii) $\ell = +1 \} \Rightarrow \begin{pmatrix} S \\ a \\ b \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} \rightarrow \begin{pmatrix} a' \\ b \end{pmatrix} \begin{pmatrix} S' \\ c \\ d \end{pmatrix}$

in G for all b, d ϵ A_I, c ϵ A_O.

(iv) $\ell = +1$ and S' ϵ Σ_f $\} \Rightarrow \begin{pmatrix} S \\ a \\ b \end{pmatrix} \# \rightarrow \begin{pmatrix} T \\ a' \\ b \end{pmatrix} \#$

in G for all b ϵ A_I .

(3) The following rules are in G for all b, d ϵ A_I, a, c ϵ A_O

$$\begin{pmatrix} c \\ d \end{pmatrix} \begin{pmatrix} T \\ a \\ b \end{pmatrix} \rightarrow \begin{pmatrix} T \\ c \\ d \end{pmatrix} \begin{pmatrix} T \\ a \\ b \end{pmatrix}$$

$$\begin{pmatrix} T \\ a \\ b \end{pmatrix} \rightarrow b$$

By means of the rules (1)

$$A_1 \Rightarrow \begin{pmatrix} S_o \\ b_1 \\ b_1 \end{pmatrix} \begin{pmatrix} b_2 \\ b_2 \end{pmatrix} \cdots \begin{pmatrix} b_k \\ b_k \end{pmatrix} \#$$

for all $b_i \in A_I$, $1 \leq i \leq k$. Thus A_1 essentially dominates an initial configuration of M with arbitrary input $x = b_1 b_2 \cdots b_k$ written twice.

The rules 2 (i), (ii), and (iii) are exactly the transitions of M except that dummy symbols (representing the input string) are carried along and never altered during any derivation employing these rules.

A rule of the form 2 (iv) can only be applied if M accepts x, but once applied, G can generate x # by the rules (3). These simply strip away the upper components of each vector symbol, leaving the original input.

It is a simple matter to construct a Type 1 grammar G' for L from G. G' contains all rules of G except those involving #; these are replaced by introducing new non-terminals $B^{\#}$ as follows:

$$\left.\begin{array}{l} A \rightarrow BD \\ \\ D \rightarrow \# \\ \\ \text{in } G \end{array}\right\} \Rightarrow \left\{\begin{array}{l} A \rightarrow B^{\#} \\ \\ \text{in } G' \\ \\ \\ \end{array}\right.$$

$$\left.\begin{array}{c} A\# \rightarrow B\# \\ \\ \text{in } G \end{array}\right\} \Rightarrow \left\{\begin{array}{c} A^{\#} \rightarrow B \\ \\ \text{in } G' \end{array}\right.$$

Finally, note that all rules of G' are CF except for those of the form $AB \rightarrow CD$. Each such rule may be replaced by the three CS rules $AB \rightarrow A'B$, $A'B \rightarrow A'D$ and $A'D \rightarrow CD$ where A' is a new non-terminal.

Corollary: If L is a rudimentary event, then L has an unambiguous CS grammar.

Proof: Myhill (1960) showed every rudimentary event is accepted by a deterministic LBA.

Theorem 19: Given a deterministic LBA M which accepts L, one can construct a deterministic LBA M' which accepts $\overline{L}$.

Proof: The proof is informal. Choose an integer n so that

$$n^{\lambda(y)} \geq |\Sigma| \; \lambda(y) \; |A_0|^{\lambda(y)}$$

for all $y \in A_0^*$.

For a given input x, M has at most $n^{\lambda(x)}$ distinct machine-tape configurations. Therefore, since M is deterministic, either M terminates its computation with $n^{\lambda(x)}$ steps or else a configuration recurs and M never terminates its computation on x.

By providing M' with enough output symbols, e.g.,

$$|A_0'| = (n+1)\ (|\sum| + 1)\ \ |A_0|,$$

M' can be a deterministic LBA which functions as follows.

M' simulates M with input x. Whenever M executes an instruction, M' simulates the execution of the same instruction and also adds one to a counter, initially zero. If M accepts x, M' blocks. If M blocks or if the counter exceeds $n^{\lambda(x)}$, then M' accepts x.

Evidently $L(M') = \overline{L}$.

Theorem 20: If L is in the Boolean algebra generated by the CF languages, then L has an unambiguous CS grammar.

Proof: The deterministic LBA are obviously closed under union and intersection and therefore by Theorem 19 they form a Boolean algebra. It is easy to prove that any CF language can be accepted by a deterministic LBA. Theorem 18 implies the result.

BIBLIOGRAPHY

Bar-Hillel, Y., Perles, M., and Shamir, E. (1961), On formal properties of simple phrase structure grammars. Phonetik, Sprachwiss. u. Kommunikationsforsch. Band 14, 143-72.

Chomsky, N. (1959), On certain formal properties of grammars. Inform. Control 2, 137-167.

Chomsky, N. (1963), Formal properties of grammars. In R. D. Luce, R. Bush, and E. Galanter, eds., "Handbook of Mathematical Psychology." Wiley, New York.

Chomsky, N., and Schützenberger, M. P., (1962), The algebraic theory of context-free languages. P. Braffort and D. Hirschberg, eds., "Computer Programming and Formal Systems," pp. 118-161. North Holland, Amsterdam.

Ginsburg S., and Ullian, J. (1964) Ambiguity in context-free languages, SDC Document TM-738/005/00.

Greibach, S. (1963), Inverses of phrase structure generators, Mathematical Linguistics and Automatic Translation, Report No. NSF-11, Computation Laboratory, Harvard University.

Kuroda, S. Y., (1964), Classes of languages and linear-bounded automata, Inform. Control 7, 207-223.

Matthews, G. H., (1963), Discontinuity and asymmetry in phrase structure grammars. Inform. Control 6, 137-146.

Myhill, J. (1960), Linear bounded automata, WADD Technical Note 60-165. Wright Air Development Division, Wright-Patterson Air Force Base, Ohio.

Parikh, R. (1961), Language-generating devices, RLE Quart. Prog. Rept., No. 60, pp. 199-212, MIT, Cambridge, Massachusetts.

Scheinberg, S. (1960), Note on the Boolean properties of context-free languages. Inform. Control 3, 372-375.

Schützenberger, M. P. (1961), Some remarks on Chomsky's context-free languages. RLE Quart. Prog. Rept. No. 63, 155-170. MIT, Cambridge, Massachusetts.

BIOGRAPHY

In June, 1953, Leonard Harold Haines graduated from Thomas Jefferson High School in Brooklyn, New York, and was awarded a scholarship to Harvard College by the Harvard Club of New York. At Harvard he majored in mathematics, graduating magna cum laude in June, 1957.

Mr. Haines joined IBM applied programming in New York City and worked on various problems in numerical analysis. Taking educational leave from IBM he accepted an appointment as University Fellow in mathematics at the University of California at Berkeley; the M.A. degree was awarded September, 1960.

Returning to IBM, Mr. Haines designed and partly wrote the IBM 1401 FORTRAN compiler. The design scheme is described in "Serial Compilation and the 1401 FORTRAN Compiler" appearing in the February, 1965, IBM Systems Research Journal.

In September, 1961, Mr. Haines accepted a three-year IBM doctoral fellowship in mathematics at MIT. Since September, 1964, he has held the position of Advisory Mathematician, IBM Systems Research and Development, Corporate Division.

Mr. Haines is married and has three children.