A New Normal-Form Theorem for Context-Free Phrase Structure Grammars

SHEILA A. GREIBACH

Harvard University, Cambridge, Massachusetts

Abstract. A context-free phrase structure generator is in *standard form* if and only if all of its rules are of the form: $Z \rightarrow aY_1, \cdots, Y_m$ where Z and Y_i are intermediate symbols and a is a terminal symbol, so that one input symbol is processed at each step. Standard form is convenient for computer manipulation of context-free languages. A proof is given that every context-free phrase structure generator is strongly equivalent to one in standard form; it is in the form of an algorithm now being programmed, and offers an independent proof of a variant of the Chomsky-Schützenberger normal form theorem.

1. *Introduction*

The connection between context-free languages and pushdown-store automata has been established independently by Chomsky [1] and Every [2]. Recently several different algorithms, utilizing forms of pushdown-store machines, have been proposed for the syntactic analysis of natural or artificial languages which place the same restriction on context-free grammars [3, 4, 5]. In order to work efficiently and in some cases to work at all, these algorithms forbid infinite left-going structures resulting from generations: $Z \overset{*}{\rightarrow} Z\gamma$. It has been hypothesized [6] that natural languages do not permit infinite loops of this kind; the authors of these algorithms feel that one can avoid such structures in designing the syntax of programming languages [3, 4]. In fact, as shown below, every context-free grammar is strongly equivalent to one that does not contain such structures.

In the multiple-path predictive analysis of English, Kuno and Oettinger [5] have employed a special kind of nondeterministic pushdown-store automaton. In a given analysis of a given input string, the machine scans the input strictly from left-to-right. At each operation it (1) scans a new input symbol, (2) scans and erases the top symbol on the pushdown store, (3) adds a (possibly null) string of symbols to the pushdown store, (4) prints an output.

In the general model for pushdown-store automata, the machine is allowed to perform an indefinite series of actions without advancing the input tape [19]. Rules allowing such behavior appear inelegant and counter-intuitive; it is shown later in this paper that they are not needed. A pushdown-store machine corresponding to multiple-path predictive analysis would have one state and advance the input tape after every action. When such a machine handles a deterministic grammar, it operates in real time in the strictest sense, scanning a new input symbol and printing a new output at every step. These machines correspond naturally to phrase structure generators in standard form, which is defined below. Also proved is that every context-free language can be generated by a phrase structure generator in standard form.

Definition 1.1. By a psg $(I, T, X, \mathcal{P})$ we mean a context-free phrase structure grammar where

(1) I is a finite vocabulary of intermediate symbols,

(2) T is a finite vocabulary of terminal symbols and $I \cap T = \phi$,

(3) X is the designated initial symbol and $X \in I$,

(4) The rules of $\mathcal{P}$ are of the forms, $Z \rightarrow AY_1 \cdots Y_n \quad n \geqq 1$, $Z \in I$, $A, Y_i \in I \cup T$ and $Z \rightarrow a, \quad Z \in I, \quad a \in T$.

Definition 1.2. If $\alpha = \beta Z \gamma$, β is a string in T, and $Z \rightarrow \delta$ is a rule of $\mathcal{P}$, then we write $\alpha \rightarrow \beta\delta\gamma$. If there are strings $\alpha_1, \cdots, \alpha_n$ such that $\alpha \rightarrow \alpha_1, \alpha_1 \rightarrow \alpha_2, \cdots, \alpha_n \rightarrow \beta$, then we write $\alpha \overset{*}{\rightarrow} \beta$.

All generations proceed from left to right, expanding the left-most member of I first. Members of T are denoted by lower-case letters; upper-case letters are used for members of I or $I \cup T$.

Definition 1.3. A psg $(I, T, X, \mathcal{P})$ is in *standard form* (is an s-psg) if and only if all of the rules of $\mathcal{P}$ are of the forms

$$Z \rightarrow a$$

$$Z \rightarrow aY_1 \cdots Y_n \qquad n \geqq 1, \quad a \in T, \quad Y_i \in I.$$

If a psg is in standard form then *a fortiori* it can have no infinite left-going structures. Using standard form we can give particularly elegant proofs of the connection between context-free languages, pushdown-store machines and deterministic linear-bounded automata. One might suspect the universality of standard form follows from the following theorem due to Chomsky and Schutzenberger [7].

THEOREM 1.1. *Every context-free language L can be expressed in the form $L = \phi(R \cap K)$ where R is a standard regular event, K is a Dyck language and ϕ is a homomorphism.*

Since K can be generated by a standard form psg and the intersection of context-free and finite state grammars can be made to preserve standard form (by a modified version of the proof due to Bar-Hillel et al [8]) it is clear that any language when expressed as in Theorem 1.1 can be placed in standard form under one condition. If AA' is a cancelling pair, we must have $\phi(A)$ terminal, (and in particular $\phi(A) \neq \wedge$, where $\wedge$ is the empty symbol) or else applying ϕ will not preserve standard form. This condition can be met, though it is not satisfied in earlier proofs [19] of Theorem 1.1.

Most of this paper is devoted to an entirely independent proof, which has the following advantages: (1) preserves ambiguities in an obvious way, (2) is practically constructive (i.e., in algorithmic form easy to program) and in fact is being programmed at Harvard [9, 10], (3) is independent of Theorem 1.1, (4) yields an elegant, practically constructive proof of Theorem 1.1.

The rules in Definition 1.3 differ from the general form of psg rules in two respects: (a) Y_i must be a member of I, and (b) a must be a member of T.

The first restriction, $Y_i \in I$, clearly has no effect on generative power. The second restriction is the crucial one. The left-most symbol must be a terminal symbol; therefore, a new terminal symbol will appear at every step of generation. Most of this paper will be devoted to proving that the second restriction does not limit generative power.

To facilitate discussion we introduce the following definitions.

Definition 1.4. If $G = (I, T, X, \mathcal{P})$ is a psg then

$$L(G) = \{\alpha \mid X \xrightarrow{*} \alpha \text{ and } \alpha \text{ is a string in } T\}.$$

Definition 1.5. A psg $(I, T, X, \mathcal{P})$ is *admissible* if and only if either it is null ($\mathcal{P} = \phi$), or else

(1) Each symbol of T appears in some terminal string.

(2) Each symbol of I is used in a generation of some terminal string.

(3) Each rule is used in a generation of some terminal string.

(4) Any string in $I \cup T$ generated by the rules of $\mathcal{P}$ can be completed, that is, can be expanded by rules of $\mathcal{P}$ to a terminal string.

It is well known that no generality is lost by restricting attention to admissible psg's generating strictly from left to right [2, 8].

Definition 1.6. In a rule $Z \rightarrow AY_1 \cdots Y_m$, Z is the *generatrix* and A the *handle*.

Definition 1.7. In a psg the set of all rules with generatrix Z are the Z*-rules* for that psg.

Definition 1.8. For a psg $G = (I, T, X, \mathcal{P})$

$$H(G) = \{A \mid A \text{ is a handle of some rule of } \mathcal{P}\}$$

$$M(G) = \{Z \mid \mathcal{P} \text{ contains a } Z\text{-rule with handle in } I\}.$$

If G is an admissible psg in standard form we have

$$T = H(G)$$

$$M(G) = \phi.$$

Employing an iterative procedure we can construct psg's in standard form. Using a technique developed in Lemmas 2.1 and 2.2, we treat one by one all members of I appearing as handles. At every step we eliminate some member of I from the set of handles. New symbols are created, but the new rules added are such that no new symbol ever appears as a handle. Eventually, all handles will be members of T.

In the proof we assume the relevant facts concerning regular expressions, finite-state graphs and finite-state grammars [11, 1, 6]. Essentially, the proof below depends on the well-known fact that a given finite-state language L can be generated either by a psg containing only left-linear rules: $Z \rightarrow aY$, $Z \rightarrow a$, or by a psg containing only right-linear rules: $Z \rightarrow Ya$, $Z \rightarrow a$, and a psg containing either only left-linear rules or only right-linear rules will generate a finite-state language.

2. *Main Construction*

The lemmas which follow are illustrated by a running example.

Example 2.1.

$$T = \{a, b, d, e\} \qquad I = \{X, Y\}$$

$$\mathcal{P}: \; X \to a; \;\; X \to Xb; \;\; X \to Ya; \;\; Y \to e; \;\; Y \to YYd.$$

LEMMA 2.1. *For an admissible* psg $(I, T, X, \mathcal{P})$ *and* $Z \in I$, *one can effectively find a regular expression* R *over* $I \cup T$, *such that:*

(a) *If* α *is denoted by* R, *then* $Z \overset{*}{\to} \alpha$.

(b) *If* $Z \overset{*}{\to} \alpha$ *and* α *is a string in* T, *either* α *is denoted by* R, *or there is a* β *such that* $Z \overset{*}{\to} \beta \overset{*}{\to} \alpha$ *and* β *is denoted by* R.

(c) *R is of the form:* $(a_1\alpha_1 + \cdots + a_m\alpha_m + b_1\beta_1 + \cdots + b_r\beta_r)\, cl(Q)$ *where* a_i, $b_i \in T$, α_i *are (possibly null) strings in* $I \cup T$, Q, β_i *are regular expressions over* $I \cup T$ *(and we may have* $Q = \wedge$*),* $m \geqq 0$, $r \geqq 0$, *and* $m + r > 0$.

PROOF. First construct a finite-state graph for Z. Z is initial state; all the states are either members of I or else certain numbered states, the final states. A state Y has as many branches as there are Y-rules. For a rule $Y \to aY_1 \cdots Y_n$ with $a \in T$, we have $\lambda(Y, t) = aY_1 \cdots Y_n$ for a new numbered state t; i.e., Y is connected to a numbered state t, and $aY_1 \cdots Y_n$ appears along the arrow. For a rule: $Y \to AY_1 \cdots Y_n$ with $A \in I$, $\lambda(Y, A) = Y_1 \cdots Y_n$, i.e., the state Y is connected to the state A and $Y_1 \cdots Y_n$ appears along the arrow. Numbered states are terminal states and are not connected to any other state. Each handle not in T is the name of a state. Strings along the arrows from the letter states represent strings appearing to the right of the handle labeling the state at the head of the arrow, contrary to the more common finite-state diagram conventions.

Example 2.1 Taking $Z = X$, we construct the graph in Figure 1.

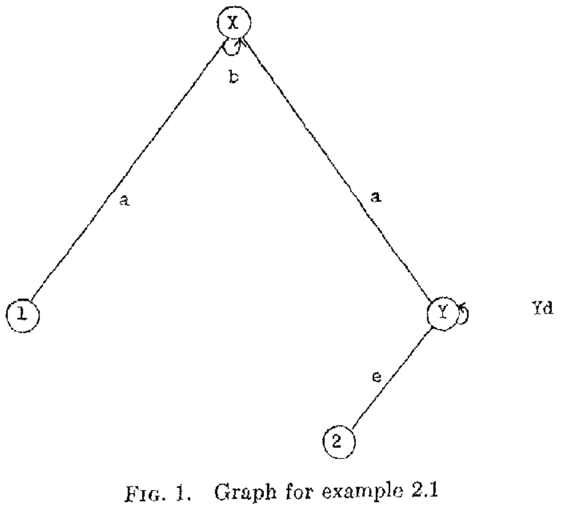


FIG. 1. Graph for example 2.1

Case (a): *Z-loop.* Consider any path from Z in the finite-state graph for Z which either returns to Z or ends at a terminal state. If it returns to Z, we have the first case, case (a) *Z-loop.*

Taking Z as initial state and as final state, we can find a regular expression Q over $I \cup T$ denoting all paths on the graph (of length 1 or longer) which start at Z and end at Z but do not pass through Z enroute. We have: $Z \xrightarrow{*} ZQ$, and $Z \xrightarrow{*} Zcl(Q)$.

Example 2.1 We have only one X-loop: $X \to Xb$, Q is b. For all $n \geqq 1$, we have $X \to Xb^n$, which we can write, $X \to Xcl(b)$.

Case (b): *no loops at all.* In this case the path leads from Z to terminal states and there are no possible loops on the way. For such paths we can write directly $Z \xrightarrow{*} a_i\alpha_i$ where α_i is a string in $I \cup T$, i.e., a regular expression without closure (cl)) or union (+), and $a_i \in T$.

Example 2.1 The branch in Figure 1 connecting X and 1 falls under case (b).

We may have a very complicated scheme of interlocking loops.

Case (c): *Loops, but no Z-loops.* In Figure 1 we have the subgraph containing X, Y and 2. Much more complicated graphs are possible. Let $m + 1, \cdots, m + r$ denote all terminal states reachable through such paths. We can find for each i, $1 \leqq i \leqq r$, a regular expression β_i in $I \cup T$ such that $Z \xrightarrow{*} b_i\beta_i$ and $b_i\beta_i$ denotes all paths with initial state Z and terminal state $m + i$ and not passing through Z. Since $m + i$ is a terminal state, $b_i \in T$.

Example 2.1 In this example we have $X \xrightarrow{*} ecl(Yd)a$ where β is $cl(Yd)a$.

Combining cases (a), (b), (c), we let:

$$R = (a_1\alpha_1 + \cdots + a_m\alpha_m + b_1\beta_1 + \cdots + b_r\beta_r)cl(Q).$$

We show below that R has the desired properties. Since the psg is admissible, either case (b) or case (c) must occur. Hence $m + r > 0$. If $Z \xrightarrow{*} \alpha$, we have first $Z \xrightarrow{*} Z\gamma$ for some γ, which may be null. Eventually, Z must cease being the handle and a terminal state be reached. γ is denoted by Q (or is Λ), since Q denotes all Z-loops. We must get $Z \xrightarrow{*} d\delta\gamma \xrightarrow{*} \alpha$ where $d \in T$. By construction, d must be either an a_i or a b_i, and so either $\delta = \alpha_i$, or else δ is denoted by β_i. In either case $d\delta\gamma$ is denoted by R. On the other hand, let α be denoted by R. Then either $\alpha = a_i\alpha_i\gamma$ or $\alpha = b_i\delta_i\gamma$ where γ is denoted by Q (or is Λ) and δ_i is denoted by β_i. Then clearly $Z \xrightarrow{*} Z\gamma$ and either $Z \xrightarrow{*} Z\gamma \xrightarrow{*} a_i\alpha_i\gamma$ or $Z \xrightarrow{*} Z\gamma \xrightarrow{*} b_i\delta_i\gamma$.

Example 2.1 The regular expression R for this example is

$$R = (a + ecl(Yd)a)cl(b).$$

To ensure strong equivalence in the following lemma one should use the construction in the Appendix to go from finite-state graph to regular expression. One must take care not to use any identities for regular expressions, unless, indeed, one wants to eliminate some ambiguities from the resulting psg in this fashion.

LEMMA 2.2. *For an admissible* psg $G = (I, T, X, \mathcal{P})$ *and* $Z \in I$, $Z \in M(G)$, *there is an admissible* psg $K = (I*, T, X, \mathcal{P}*)$ *such that*

(a) *K is strongly equivalent to G.*

(b) *$Z \notin H(K)$ and $Z \not\subset M(K)$, i.e., Z is not a handle and all Z-rules have handles in T.*

(c) *$I* \cap H(K) \subset I$, i.e., no new symbol is a handle.*

(d) *$I* \cap H(K) \subseteq M(K)$, i.e., a handle is in T or else generatrix of a rule not in standard form.*

(e) *$M(K) \cap I \subset M(G)$, i.e., decrease members of I which are generatrices of rules not in standard form.*

PROOF. By Lemma 2.1 we get the regular expression

$$R = (a_1\alpha_1 + \cdots + a_m\alpha_m + b_1\beta_1 + \cdots + b_r\beta_r)cl(Q)$$

$a_i, b_i \in T$, $\alpha_1, \cdots, \alpha_m$ strings in $I \cup T$, $\beta_1, \cdots, \beta_r$, Q regular. This gives us the rules

$$\begin{aligned} \mathcal{P}_Z: \quad & Z \to a_i\alpha_i & i &= 1, \cdots, m \\ & Z \to b_iD_i & i &= 1, \cdots, r \\ & Z \to a_i\alpha_iY & i &= 1, \cdots, m \\ & Z \to b_iD_iY & i &= 1, \cdots, m \end{aligned}$$

where $D_1, \cdots, D_r, Y$ are new symbols.

We construct an s-psg $(I_Y, T_Y, Y, \mathcal{P}_Y)$ generating Q from $Y (Y \in I_Y$, of course) and s-psg's $(I_i, T_i, D_i, \mathcal{P}_i)$ generating β_i from D_i, $D_i \in I_i$. These psg's are in standard form, so that no symbol of $I_Y, I_1, \cdots, I_r$ appears as a handle anywhere. We can assume that $I_Y \cap I_i = \phi$, $I_i \cap I_j = \phi$ for $i \neq j$, $I_Y \cap I = \phi$, $I_i \cap I = \phi$, for $i = 1, \cdots, r$; in other words, all symbols are new and distinct. Let

$$\begin{aligned} \mathcal{P}_- &= \mathcal{P} - Z\text{-rules of } \mathcal{P}. \\ \bar{\mathcal{P}} &= \mathcal{P}_Z \cup \mathcal{P}_- \cup \mathcal{P}_Y \cup \mathcal{P}_1 \cup \cdots \cup \mathcal{P}_r. \\ \bar{I} &= I \cup I_Y \cup I_1 \cup \cdots \cup I_r. \end{aligned}$$

$\bar{G} = (\bar{I}, T, X, \bar{\mathcal{P}})$ differs from G only in Z-rules and new rules; the generatrix of a new rule is either Z or a new symbol. The same set of strings is still generated from Z and other members of I are not affected. Hence, $L(\bar{G}) = L(G)$. Z-rules have handles in T and new symbols never appear as handles. Suppose $U \in H(\bar{G}) \cap \bar{I}$. Then, $U \in I$. If $U \in M(\bar{G})$, we do nothing. Otherwise, all U-rules have handles in T.

If we have $W \to U\gamma$ and $U \to \delta_i$, $i = 1, \cdots, t$, then we can plug the U-rules in for U to get $W \to \delta_i\gamma$, $i = 1, \cdots, t$.

These W-rules now have handles in T. U can be eliminated from the set of handles in this manner. Treating all members of $\bar{I} \cap H(\bar{G})$ in this manner, we get a new psg $G' = (\bar{I}, T, X, \mathcal{P}')$ such that G' satisfies (a)–(e) of this lemma. G' may not be admissible, so we construct an admissible psg $K = (I*, T, X, \mathcal{P}*)$ entirely equivalent to G'. Since the rules of K are a subset of the rules of G', K will satisfy (a)–(e) as required.

Example 2.1 From the regular expression

$$X \xrightarrow{*} (a + ecl(Yd)a)cl(b)$$

we can write at once the rules:

$$\begin{array}{lll} X \to a; & X \to ea; & X \to eDa; \\ X \to aB; & X \to eaB; & X \to eDaB; \\ B \to b; & B \to bB; & \\ D \to Yd; & D \to YdD. & \end{array}$$

B and D are new symbols. B is used to generate $cl(b)$; D is used to generate $cl(Yd)$. The two Y-rules are retained intact at this step.

LEMMA 2.3. *Given an admissible psg $G = (I, T, X, \mathcal{P})$ we can effectively find a strongly equivalent psg $K = (I*, T, X, \mathcal{P}*)$ such that $M(K) = \phi$ and $H(K) \subseteq T$.*

PROOF. Let $G_1 = G$. Consider G_i. If $I \cap M(G_i) = \phi$, we stop. If not, we can use the proof of Lemma 2.2 to find an admissible psg $G_{i+1} = (I_{i+1}, T, X, \mathcal{P}_{i+1})$ such that $L(G_i) = L(G_{i+1})$, $M(G_{i+1}) \cap I \subseteq M(G_i)$, $I_{i+1} \cap H(G_{i+1}) \subset I$, $I_{i+1} \cap H(G_{i+1}) \subseteq M(G_{i+1})$. At each step we eliminate a member of $M(G_{i+1}) \cap I$, and never add one. No new symbol ever becomes a handle. Ultimately, since I is finite we reach an n such that $M(G_n) \cap I = \phi$.

Consider $M(G_n)$. If $M(G_n) = \phi$ we are finished. Suppose $W \in M(G_n)$. Then we must have a rule: $W \to AY_1 \cdots Y_m$ for $m > 1$, and $A \in I_n$. By construction, A cannot be a new symbol; i.e., A must belong to I. Then $A \notin M(G_n)$, since $M(G_n) \cap I = \phi$. This gives, $A \notin M(G_n)$, $A \in I_n$ and $A \in H(G_n)$, but by construction $H(G_n) \cap I_n \subseteq M(G_n)$ so we have a contradiction; therefore, $W \notin M(G_n)$. Hence, $M(G_n) = \phi$ and $H(G_n) \subseteq T$. Let $K = G_n$ and we are done.

Example 2.1 The X-rules and B-rules now all have handles in T. B and D do not appear as handles. Y is the only handle not in T. We iterate the procedure. A regular expression for Y is clearly $Y \xrightarrow{*} ecl(Yd)$.
Observing that D has already been used to generate $cl(Yd)$, we need only substitute the Y-rules:

$$Y \to e; \qquad Y \to eD$$

for the Y-rules:

$$Y \to e; \qquad Y \to YYd$$

and use as D-rules:

$$\begin{array}{l} D \to ed, \\ D \to edD, \\ D \to eYd, \\ D \to eYdD. \end{array}$$

3. *Main Theorem and Consequences*

THEOREM 3.1. *Given an admissible psg G, we can effectively construct an admissible psg K in standard form strongly equivalent to G.*

PROOF. From Lemma 2.3 we can construct G' strongly equivalent to G with $H(G') \subseteq T$. G' is almost in standard form. It is trivial to obtain a psg K in standard form strongly equivalent to G'.

Theorem 3.1 is a new normal-form theorem for context-free phrase structure generators. As we have remarked, a psg in standard form is in a particularly convenient form for handling on a computer, whether one wishes to analyze or to generate sentences.

Definition 3.1. A psg $(I, T, X, \mathcal{P})$ is in *reverse standard form* if and only if all the rules of $\mathcal{P}$ are of the forms: $Z \rightarrow a, \quad Z \in I, \quad a \in T$ and $Z \rightarrow Y_1 \cdots Y_m a$, $Z, Y_i \in I, \quad a \in T, \quad m \geqq 1$.

A proof entirely similar to that for Theorem 4.1 yields:

COROLLARY 3.1. *Given an admissible psg G, we can effectively construct a strongly equivalent psg K in reverse standard form.*

The techniques of predictive analysis can be used to analyze sentences of a context-free language with a grammar in standard form. (Cf. Kuno and Oettinger [5].) Dependency analysis [12, 13] can be used for either standard or reverse standard form.

To obtain Theorem 1.1 we quickly develop another normal form.

DEFINITION 3.2. A psg $(I, T, X, \mathcal{P})$ is in standard n-form if all of the rules of $\mathcal{P}$ are of the forms: $Z \rightarrow a, \quad Z \in I, \quad a \in T$ or $Z \rightarrow aY_1 \cdots Y_k, \quad Z, Y_i \in I$, $a \in T, \quad k \leqq n$.

COROLLARY 3.2. *Given an admissible psg G, we can effectively construct an admissible psg K in standard 2-form strongly equivalent to G.*

PROOF. It suffices to give an algorithm for obtaining standard $(n - 1)$-form from standard n-form, for $n \geqq 3$.

Let $G = (I, T, X, \mathcal{P})$ be in standard n-form, for $n \geqq 3$. For each couple (A, B) where $A, B \in I$, create a new symbol $S(A, B)$. For each rule: $Z \rightarrow aY_1 \cdots Y_{n-1}Y_n$ place in $\mathcal{P}'$ the rule: $Z \rightarrow aY_1 \cdots Y_{n-2}S(Y_{n-1}, Y_n)$. These rules are all in standard $(n - 1)$-form.

Now, for each new symbol $S(A, B)$, scan all the A-rules of $\mathcal{P}$. For each rule of the form: $A \rightarrow aY_1 \cdots Y_k$ with $k \leqq n - 2$, place in $\mathcal{P}'$ the rule: $S(A, B) \rightarrow aY_1 \cdots Y_kB$. These rules, too, are all in standard $(n - 1)$-form. For each rule of the form: $A \rightarrow aY_1 \cdots Y_{n-1}$ place in $\mathcal{P}'$ the rule: $S(A, B) \rightarrow aY_1 \cdots Y_{n-2}S(Y_{n-1}, B)$. Again, these rules are all in standard $(n - 1)$-form. For each rule of the form $A \rightarrow aY_1 \cdots Y_n$ place in $\mathcal{P}'$ the rule: $S(A, B) \rightarrow aY_1 \cdots Y_{n-3}S(Y_{n-2}, Y_{n-1})S(Y_n, B)$.

All the rules of $\mathcal{P}'$ are in standard $(n - 1)$-form. Now eliminate from $\mathcal{P}$ all rules not in standard $(n - 1)$-form, and call the result $\mathcal{P}_-$. Let $\bar{\mathcal{P}} = \mathcal{P}_- \cup \mathcal{P}'$, and let $\bar{I} = I \cup \{S(A, B) \mid A, B \in I\}$. Let $\bar{G} = (\bar{I}, T, X, \bar{\mathcal{P}})$. $\bar{G}$ may not be admissible, so we can find an admissible psg K equivalent to $\bar{G}$. K is strongly equivalent to G, and is in standard $n - 1$ form. Q.E.D.

For our proof of Theorem 1.1 we use a version of Floyd's operator form [14].

Definition 3.3. A psg $(I, T, X, \mathcal{P})$ is in *standard operator form* if all of the rules of $\mathcal{P}$ are of the forms:

$$\begin{aligned} &Z \rightarrow a \\ &Z \rightarrow aY \\ &Z \rightarrow aYb \\ &Z \rightarrow aYbV \end{aligned}$$

We have at once the following corollary.

COROLLARY 3.3.[1] *Given any admissible psg we can effectively construct a strongly equivalent psg in standard operator form.*

PROOF. Using standard 2-form, we can obtain standard operator form using the same device found in the proof of Corollary 3.2.

Finally we have:[2]

COROLLARY 3.4 (Theorem 1.1). *Given any admissible psg G we can effectively construct a homomorphism ϕ, a standard regular event R and a Dyck language K such that $L(G) = \phi(R \cap K)$, ambiguities are preserved, $\phi\ (aa') \neq \Lambda$ and, if G is in standard operator form, the resulting grammars are isomorphic.*

PROOF. Starting from standard operator form one can use the construction indicated by Chomsky and Schutzenberger for linear languages [7], thus obtaining directly ϕ and a grammar for R.

We see that we can obtain Theorem 1.1 from Theorem 3.1 just as we can obtain Theorem 3.1 from Theorem 1.1. There are advantages to both methods of proof. The construction of Theorem 3.1 is being programmed and it should be possible before long to take a subset of, say, ALGOL, turn it into standard form and then try out analysis routines, such as the Kuno-Oettinger English analyzer, on sentences in the language.

REFERENCES

1. CHOMSKY, N. Formal properties of grammars. In *Handbook of Mathematical Psychology, Vol. II*. Luce, R. D., Bush, R. R., and Galanter, E. (Eds.), Wiley, New York, 1963.
2. EVEY, J. The Theory and Applications of Pushdown Store Machines. Doct. Thesis, Harvard U., Cambridge, Mass., 1963.
3. CONWAY, M. E. Design of a separable transition-diagram compiler. *Comm. ACM 6*, 7 (July 1963), 396–408.
4. IRONS, E. T. An Error-Correcting Parse Algorithm. *Comm. ACM 6*, 11 (Nov. 1963), 669–673.
5. KUNO, S., AND OETTINGER, A. G. Multiple-path syntactic analyzer. In *Information Processing 62*, C. M. Popplewell (Ed.), North Holland, Amsterdam, 1962–1963.
6. YNGVE, V. A model and hypothesis for language structure. *Proc. Amer. Phil. Soc. 107*, 5 (Oct. 1960).
7. CHOMSKY, N., AND SCHUTZENBERGER, M. P. The algebraic theory of context-free languages. In *Computer Programming and Formal Systems*, Braffort, P., and Hirschberg, D. (Eds.), North Holland, Amsterdam, 1963.
8. BAR-HILLEL, Y., PERLES, M., AND SHAMIR, E. On formal properties of simple phrase structure grammars. *Zeit. Phonet. Sprachwiss. Kommunik. Forsch. 14*, 2 (1961).
9. KUNO, S. Automatic transformation of an admissible PSG into a standard form PSG. Unpublished paper.
10. FENICHEL, R. Private communication.
11. RABIN, M. O., AND SCOTT, D. Finite automata and their decision problems. *IBM J. Res. Develop. 3* (1959), 114–125.
12. HAYS, D. G. Grouping and dependency theories. Paper P-1010, RAND Corp., Santa Monica, Calif., 1960.
13. GAIFMAN, M. Dependency systems and phrase structure systems. Paper P-2315, RAND Corp., Santa Monica, Calif., 1961.

[1] Suggested by J. Spielman [15].

[2] Suggested by C. Ziegler [16].

14. FLOYD, R. W. Syntactic analysis and operator precedence. *J. ACM 10*, 3 (July 1963), 316–333.
15. SPIELMAN, J. Operator and precedence grammars. Unpublished paper.
16. ZIEGLER, C. A theorem of N. Chomsky and M. P. Schützenberger. Unpublished paper.
17. GREIBACH, S. Inverses of phrase structure generators. Doct. Thesis, Harvard U., Cambridge, Mass., 1963.
18. GINSBURG, S., AND RICE, H. G. Two families of languages related to ALGOL. *J. ACM 9*, 3 (July 1962), 350–371.
19. CHOMSKY, N. Context-free grammars and pushdown storage. RLE Quart. Prog. Rep. No. 65, M.I.T., Cambridge, Mass., Mar. 1962.
20. MCNAUGHTON, R., AND YAMADA, H. Regular expressions and state graphs for automata. *IRE Trans. EC-9*, (1960), 39–47.

APPENDIX

We are dealing with finite state graphs $(I, T, I_0, I_f, \mathcal{P})$, where I is a finite set of states, T is a finite vocabulary, $W(T)$ is the universal language and $\mathcal{P}$ is a map $\mathcal{P}$: $I \times I \rightarrow W(T)$. $\mathcal{P}$ associates with each ordered pair of states a finite, possibly empty, set of strings in T. I_0 is a subset of initial states of I, while I_f is the subset of final sets of I. We can state the relevant half of the Kleene-Myhill theorem as:

THEOREM A.1. *Given a finite state graph $(I, T, I_0, I_f, \mathcal{P})$ we can effectively construct a regular expression denoting all and only paths from members of I_0 to members of I_f.*

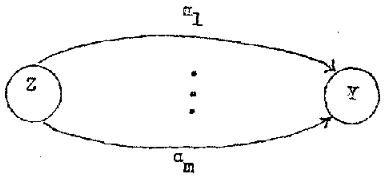
PROOF. For the purposes of this paper, we need only consider finite-state graphs where I_0 consists of only one state X, and the states of I_f are not connected to any other states; i.e., $\mathcal{P}(Z, Y)$ is undefined for any $Z \in I_f$ and any $Y \in I$.

Moreover, we can find a regular expression R_Y denoting all paths from X to Y for each $Y \in I_f$. Then the desired regular expression is

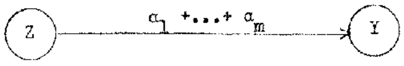
$$R = R_{Y_1} + \cdots + R_{Y_m}$$

where $I_f = \{Y_1, \cdots, Y_m\}$. Here as always in this algorithm we must be careful *not* to make any simplifications in the regular expression, but to carry the entire expression along intact.

Finally, let $R(T)$ be the set of all regular expressions over T, and note that since a regular expression of regular expressions is regular, if we let $\mathcal{P}$: $I \times I \rightarrow R(T)$ instead of $\mathcal{P}$: $I \times I \rightarrow W(T)$ the theorem will go through unchanged. Hence, if there is more than one transition from Z to Y so that we have:



we can replace this part of the graph by



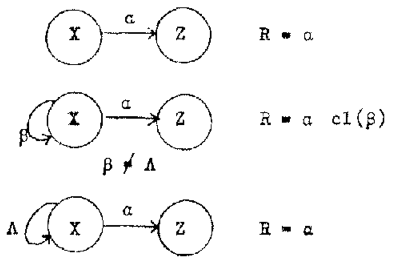
and thus proceed as if $\mathcal{P}$ were single-valued, where defined.

We need to consider only two cases.

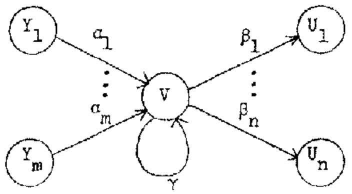
(1) To denote paths from X to Z not passing through X, where $X \neq Z$ $Z \in I_f$, and there is at least one such path.

By assumption, Z is a final state and no branches leave Z. First we erase al parts of the graph that cannot be involved in a path from X to Z. We procee by induction on $N(I)$, the number of states in I. The convention regardin arrows and symbols is the same as in Lemma 2.1, the reverse of the more usua notation.

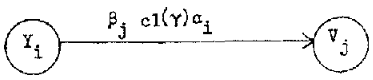
If we have $N(I) = 2$, we have three cases, which can be represented as follow (where α and β are appropriate regular expressions):



If $N(I) \geqq 3$, let $X \neq V \neq Z$, $V \in I$. The subgraph containing V and all branches to and from V can be represented as:



for appropriate states and expressions. V is distinct from the Y_i and U_j, but the Y_i may or may not be distinct from the U_j. We erase V and all arrows to and from V. We substitute, for all pairs Y_i, V_j, $1 \leqq i \leqq m$, $1 \leqq j \leqq n$:



(If $\gamma = \Lambda$, we omit $cl(\gamma)$). Now $N(I') = N(I) - 1$, and we proceed as before.

(2) To denote paths from X to X, not passing through X.

Split X into two states, X_I and X_F. X_I has those branches of X leading out, while X_F has those branches leading in to X. A branch from X to X becomes a branch from X_I to X_F. Now consider paths from X_I to X_F as in case (1).

RECEIVED MARCH, 1964.