1. Introduction

When a language is analyzed in accordance with a phrase structure grammar, it is customary to regard a terminal string x as grammatical according to a grammar G if one can start from the initial string of the grammar and apply the rules of G, successively rewriting strings until x is obtained. With the resulting derivation of a generated string x, a structural description of x is associated consisting of a labeled bracketing which indicates the nonterminal symbol(s) rewritten to obtain substrings of x. When a phrase structure grammar contains only context-free rules, each generated string can be analyzed and its structural descriptions computed with considerable efficiency. In the event that some rules are context-sensitive, however, no general analysis procedure of comparable efficiency is known. In this paper I discuss a means for allowing the use of context-sensitive rules in the description of context-free languages to the end of providing greater economy of description and analysis. I will show that if phrase structure grammars are allowed to define languages in a different way than is usual, then certain context-free languages can be analyzed more quickly, using less storage than under the standard interpretation, although no noncontext-free languages can be so analyzed. Furthermore, the new way in which a grammar defines a language seems to be a more adequate reconstruction of the use to which context-sensitive rules were put in immediate constituent analysis.

Assume we are given a phrase structure grammar $\underline{G}$ and a string $\underline{x}$ and we ask whether it is possible to analyze $\underline{x}$ in accordance with the rules of $\underline{G}$. The answer is in the affirmative if $\underline{G}$ assigns some labeled bracketing to $\underline{x}$ as its structural description. This suggests that we think of $\underline{x}$ as being provided with an arbitrary well-formed labeled bracketing $\underline{\varphi}$ and check whether each phrase of $\underline{x}$ determined by a matched pair of labeled brackets in $\underline{\varphi}$ is divided into subphrases in accord with the rules of $\underline{G}$. For a phrase to satisfy a rule $\underline{\underline{R}}$ of $\underline{G}$, the matched pair of brackets determining that phrase must enclose the particular sequence of phrases and members of $\underline{G}$'s terminal vocabulary that $\underline{\underline{R}}$ says the phrase may imediately contain. Furthermore, if $\underline{\underline{R}}$ is context-sensitive with context $\underline{\alpha}_1 \ldots \underline{\alpha}_m$ — $\underline{\beta}_1 \ldots \underline{\beta}_n$, then immediately to the left (right) in $\underline{x}$ of the phrase in question must be a sequence $\underline{y}_1 \ldots \underline{y}_m$ $(\underline{z}_1 \ldots \underline{z}_n)$ of strings such that (a) $\underline{y}_i = \underline{\alpha}_i$ $(\underline{z}_j = \underline{\beta}_j)$ if $\underline{\alpha}_i$ $(\underline{\beta}_j)$ is in the terminal vocabulary and (b) $\underline{y}_i$ $(\underline{z}_j)$ is a phrase of type $\underline{\alpha}_i$ $(\underline{\beta}_j)$ according to the labeled bracketing $\underline{\varphi}$ of $\underline{x}$ if $\underline{\alpha}_i$ $(\underline{\beta}_j)$ is in the nonterminal vocabulary, for $\underline{1 \leq i \leq m}$ $(\underline{1 \leq j \leq n})$. If some well-formed labeled bracketing of x is analyzable by G in this fashion, we can think of it as a structural description assigned to x by G . If G is context-free, the language associated with it in this rather natural fashion is clearly the same as the language generated by G in the usual fashion and the structural descriptions assigned to strings by G are the same in the two cases. If G contains rules with nonnull context, however, it is not obvious whether the language associated in the above manner is the same as the language generated. So that we can investigate this question, let us proceed with precise definitions of the new concepts which have appeared informally.

2. Definitions

For familiar concepts I will simply refer to definitions in the literature (cf. Peters and Ritchie, 1969b). Recall that a (context-sensitive) phrase structure grammar is an ordered quadruple (V_T, V_N, S, R) such that V_T and V_N are finite, nonempty, disjoint sets (the terminal vocabulary and nonterminal vocabulary, respectively), S is a member of V_N (the initial symbol) and R is a finite set of rules of the type (1),

(1) $A \longrightarrow \gamma_1 \ldots \gamma_\ell \,/\, \alpha_1 \ldots \alpha_m \;\text{——}\; \beta_1 \ldots \beta_n$

where $\_ \geq 0$, m, $n \geq 0$, $A \in V_N$, γ_i, α_j, $\beta_k \in V_T \cup V_N$ ($1 \leq i \leq \_$, $1 \leq j \leq m$, $1 \leq k \leq n$) and $\longrightarrow$, / and — are special symbols not in $V_T \cup V_N$. The rule (1) is often written as (2).

(2) $\alpha_1 \ldots \alpha_m A \beta_1 \ldots \beta_n \longrightarrow \alpha_1 \ldots \alpha_m \gamma_1 \ldots \gamma_{\_} \beta_1 \ldots \beta_n$

The notation (1) more clearly brings out the possibilities for immediate constituency allowed by the rule and the contextual conditions imposed by the rule on those possibilities. Let $L = \{[_A \mid A \in V_N\}$ and $R = \{]_A \mid A \in V_N\}$ be sets of left and right labeled brackets.

Definition 1: A labeled bracketing (finite string over $V_T \cup L \cup R$) φ is said to be well-formed if (i) $\varphi \in V_T$, (ii) $\varphi = [_A \psi]_A$ or (iii) $\varphi = \psi\omega$, where ψ and ω are well-formed labeled bracketings and $A \in V_N$.

The language generated by G (written $L(G)$) and the set of structural descriptions generated by G (written $L(G)$) are as usual (cf. Peters and Ritchie, 1969b, Definitions in § 2). A set L of strings is called a context-sensitive language if there is a phrase structure grammar G such that $L = L(G)$. A phrase structure grammar G is context-free if every rule (1) of G has $m = n = 0$ (i.e. $\alpha_1 \ldots \alpha_m = \beta_1 \ldots \beta_n = e$, where e is the empty string). A set L of strings is a context-free language if there is a context-free grammar G such that $L = L(G)$.

Definition 2: A triple (ψ_1, ψ_2, ψ_3) is called a *node* of a well-formed labeled bracketing φ if $\varphi = \psi_1 \psi_2 \psi_3$ and there are $A \in V_N$ and a well-formed labeled bracketing ω such that $\psi_2 = [_A \omega]_A$. The node (ψ_1, ψ_2, ψ_3) *satisfies* rule (1) if there are labeled bracketings $\pi_0, \pi_1, \ldots, \pi_m, \sigma_1, \ldots, \sigma_m, \chi_0, \chi_1, \ldots, \chi,$ $\omega_1, \ldots, \omega_\ell, \rho_0, \rho_1, \ldots, \rho_n, \tau_1, \ldots, \tau_n$ such that

(i) $\psi_1 = \pi_0 \sigma_1 \pi_1 \cdots \sigma_m \pi_m$, $\psi_2 = [_A \chi_0 \omega_1 \chi_1 \cdots \omega_\ell \chi_\ell]_A$ and $\psi_3 = \rho_0 \tau_1 \rho_1 \cdots \tau_n \rho_n$.

(ii) $\pi_i \chi_j, \rho_k \in (L \cup R)^*$, $1 \leq i \leq m$, $0 \leq j \leq \ell$, $0 \leq k \leq n-1$ and

(iii) $$\sigma_i = \begin{cases} \alpha_i, \text{ if } \alpha_i \in V_T \\ [_{\alpha_i} \sigma_i']_{\alpha_i}, \text{ if } \alpha_i \in V_N \ (\sigma_i' \text{ well-formed}) \end{cases}, 1 < i < m,$$

$$\omega_j = \begin{cases} \gamma_j, \text{ if } \gamma_j \in V_T \\ [_{\gamma_j} \omega_j']_{\gamma_j}, \text{ if } \gamma_j \in V_N \ (\omega_j' \text{ well-formed}) \end{cases}, i < j < \ell \text{ and}$$

$$\tau_k = \begin{cases} \beta_k, \text{ if } \beta_k \in V_T \\ [_{\beta_k} \tau_k']_{\beta_k}, \text{ if } \beta_k \in V_N \ (\tau_k' \text{ well-formed}) \end{cases}, 1 < k < n.$$

Definition 3: The *debracketing function* d is the homomorphism from $(V_T \cup L \cup R)^*$ onto V_T^* defined by

(i) $d(\underline{\alpha}) = \begin{cases} \underline{\alpha}, & \text{if } \underline{\alpha} \in \underline{V_T} \\ e, & \text{if } \underline{\alpha} \in L \cup R \end{cases}$ and

(ii) $d(\underline{\varphi\psi}) = D(\varphi)d(\psi)$ for any labeled bracketings $\underline{\varphi}$ and $\underline{\psi}$.

A labeled bracketing $\underline{\varphi}$ is <u>analyzed</u> by $\underline{G}$ if $d(\varphi) \in \underline{V_T}^*$, if there is a well-formed labeled bracketing $\underline{\psi}$ such that $\underline{\varphi} = [_{\underline{S}}\ \underline{\psi}\ ]_{\underline{S}}$ and if every node of $\underline{\varphi}$ satisfies some member of $\underline{R}$. We say that a string $\underline{x}$ is <u>parsed</u> by $\underline{G}$ if there is a labeled bracketing $\underline{\varphi}$ such that $\underline{\varphi}$ is analyzed by $\underline{G}$ and $d(\varphi) = \underline{x}$. The set of labeled bracketings analyzed by $\underline{G}$ will be written $A(\underline{G})$ and the set of strings parsed by $\underline{G}$ will be written $P(\underline{G})$.

3. The Languages Parsed by Phrase Structure Grammars

We can think of the labeled bracketings analyzed by a phrase structure grammar $\underline{G}$ as being strings over a terminal vocabulary which is the union of $\underline{G}$'s terminal vocabulary and its set of left and right labeled brackets. We may then ask what type of language $A(\underline{G})$ is. Theorem 1 provides the answer that $A(\underline{G})$ is a context-free language and from this Theorem 3.8 of Peters and Ritchie (1969a) follows immediately as Corollary 1. We now proceed to state these results.

<u>Theorem 1</u>: If $\underline{G}$ is a phrase structure grammar, then $A(\underline{G})$

is a context-free language.

Proof: Let $\underline{G} = (\underline{V}_{\underline{T}}, \underline{V}_{\underline{N}}, \underline{S}, \underline{R})$ be any phrase structure grammar and let L and R be the corresponding set of left and right labeled brackets. To prove the theorem, it suffices to describe a pushdown-storage automaton $\underline{M}$ which accepts A($\underline{G}$) since pushdown-storage automata accept just the context-free languages (Chomsky, 1963, Theorem 6). I will describe the automaton $\underline{M}$ informally since this will provide more insight into its operation. Formal construction of $\underline{M}$ from this description is a straightforward and tedious exercise and is therefore omitted.

$\underline{M}$ can receive as input any string over $\underline{V}_{\underline{T}} \cup L \cup R$. Its pushdown-store can contain symbols from $\underline{V}_{\underline{T}} \cup \underline{V}_{\underline{N}} \cup R \cup R'$, where R' is a set of symbols each corresponding to the string resulting from inserting a single "pointer" (|) in the left-context portion of a rule (e.g. (3)) or to the string resulting from insertion of a | in any string which is the right-context of a rule of $\underline{R}$ (e.g. $\underline{\beta}_{\underline{1}} \ldots | \underline{\beta}_{\underline{i}} \ldots \underline{\beta}_{\underline{n}}$).

(3) $\underline{A} \longrightarrow \underline{\gamma}_{\underline{1}} \ldots \underline{\gamma}_{\_} \ / \ \underline{\alpha}_{\underline{1}} \ldots | \underline{\alpha}_{\underline{i}} \ldots \underline{\alpha}_{\underline{m}} — \underline{\beta}_{\underline{1}} \ldots \underline{\beta}_{\underline{n}}$

M contains a finite set of states sufficient to "remember" two tables: a rule table and a right-context table. The rule table plays a dual role; it is used to determine that a node of the input is tentatively indicated as satisfying a rule only if the left-context of that rule is indeed satisfied when the left bracket determining the node is reached in the input and it is used to store an indicator at that point which will allow M to check as the input is read further whether the immediate constituency and the right-context of the node are as required by the rule. The right-context table is used in checking whether the right-context of a rule tentatively identified as being satisfied by a node does indeed appear immediately to the right of the right bracket determining that node. For each rule (1) of R, the rule table contains $m + 1$ positions and the ith position contains an entry consisting either of the symbol (3) or the symbol (4).

(4) $A \longrightarrow \gamma_1 \ldots \gamma_\ell \; / \; | \; \gamma_1 \ldots \gamma_m \text{ — } \beta_1 \ldots \beta_n$

The rule table will be updated as the input is read so that when any position corresponding to any rule (1) of R contains the entry (3), then immediately to the left in the input of the scanned symbol is a string analyzable as $\alpha_1 \ldots \alpha_i$. Thus if a pointer appears in the entry of a position immediately to the left of the symbol —— (dash), then the left-context of the corresponding rule is satisfied at that point in the input. It is clear that the rule table can be "remembered" in a finite set of states. For each distinct string $\beta_1 \ldots \beta_n$ appearing as the right-context of a rule in R, the right-context table contains $n+1$ positions the ith one of which can contain either the entry $\beta_1 \ldots | \beta_i \ldots \beta_n$ or $\beta_1 \ldots \beta_n |$. When the right bracket determining a node is reached in the input, a position corresponding to the right-context of the rule which was tentatively identified as being satisfied at the node receives a pointer to the left of its leftmost symbol. As the input is read further, pointers are advanced to the right in this string as each successive portion of the context appears under the scanning head. This allows M to check whether the tentatively identified rule is indeed satisfied by the node. "Remembering" the right-context table also requires only a finite number of states.

When started in its initial state scanning the leftmost symbol on the input tape with an empty pushdown-store, M prints S on the store and initializes its tables as follows: for each rule (1) of R a corresponding position of the rule table receives the entry (4) and each position of the right-context table receives an entry with a pointer at its extreme right. At each successive step of its computation, M performs whichever one of the operations (5)... (8) is possible in view of the top symbol on its pushdown-store, the scanned symbol on its input tape and the contents of its tables. If none of the operations can be performed, M blocks and fails to accept the input. Since M is nondeterministic, a particular input string is accepted if some computation of M on that input terminates in the accepting state with an empty pushdown-store.

(5) If you see a nonterminal symbol A, on top of the pushdown-store if the scanned input symbol is $[_A$ and if some rule table position contains the entry (3) with A to the left of the arrow and a pointer immediately to the left of the dash, then (i) advance the input tape one square, (ii) remove the symbol A from the top of the pushdown-store, (iii) for every rule table entry $B \longrightarrow \delta_1 \ldots \delta_u / \zeta_1 \ldots | A \ldots \zeta_v - \nu_1 \ldots \nu_w$ nondeterministically decide whether to leave it unchanged or to change it to $B \longrightarrow \delta_1 \ldots \delta_u / | \zeta_1 \ldots \zeta_v - \nu_1 \ldots \nu_w$ and insert in the pushdown-store the single symbol $B \longrightarrow \delta_1 \ldots \delta_u / \zeta_1 \ldots A | \ldots \zeta_v - \nu_1 \ldots \nu_w$, (iv) for every right-context table entry $\delta_1 \ldots | A \ldots \delta_k$ nondeterministically decide whether to leave it unchanged or to change it to $\delta_1 \ldots \delta_k |$ and insert the single symbol $\delta_1 \ldots A | \ldots \delta_k$ in the pushdown-store and (v) insert in the pushdown-store the $\ell + 2$ symbols $| \beta_1 \ldots \beta_n,]_A, \gamma, \ldots, \gamma_1$ (so that γ_1 is on top).

(6) If you see a member a of V_T on top of the pushdown-store,

if the scanned input symbol is $\underline{a}$ and if every right-context table entry has a pointer either at its extreme right or immediately to the left of an $\underline{a}$, then (i) advance the input tape one square, (ii) for every rule table entry (3) change it to $\underline{A} \longrightarrow \underline{\gamma}_1 \ldots \underline{\gamma}$ / $\underline{\alpha}_1 \ldots \underline{\alpha}_i \mid \ldots \underline{\alpha}_m - \underline{\beta}_1 \ldots \underline{\beta}_n$ if $\underline{\alpha}_i = \underline{a}$ or to (4) if $\underline{\alpha}_i \neq \underline{a}$ or the $\mid$ is next to the dash, (iii) for every entry $\underline{\delta}_1 \ldots \mid \underline{a} \ldots \underline{\delta}_k$ in the right-context table change it to $\underline{\delta}_1 \ldots \underline{\delta}_k \mid$ and enter $\underline{\delta}_1 \ldots \underline{a} \mid \ldots \underline{\delta}_k$ in the appropriate table position and (iv) remove the $\underline{a}$ from the top of the pushdown-store.

(7) If a right bracket $]_{\underline{A}}$ is on top of the pushdown-store and if $]_{\underline{A}}$ is the scanned input symbol, then (i) advance the input tape one square, (ii) remove the symbol $]_{\underline{A}}$ from the top of the pushdown-store and (iii) if every right-context table entry has a pointer at its extreme right, then nondeterministically decide whether or not to enter the accepting state.

(8) If you see a member of R' on top of the pushdown-store, then enter it in the appropriate position of the rule table or the right-context table.

Let $\underline{\varphi}$ be any labeled bracketing in $(\underline{G})$. Since $\underline{\varphi}$ is analyzed by $\underline{G}$, every node $(\underline{\psi}_1, \underline{\psi}_2, \underline{\psi}_3)$ of $\underline{\varphi}$ satisfies some rule in $\underline{R}$, say (1). By Definition 2, $\underline{\varphi}$ can be factored into $\underline{\pi}$'s, $\underline{\sigma}$'s, $[_{\underline{A}}$,

χ's, ω's, $]_A$, ρ's and τ's with the appropriate properties. But then as M scans the first symbol of σ_1 it can advance a pointer past α_1 in its rule table (and store the resulting symbol if α_1 is a member of V_N). Continuing in this fashion, M can advance a pointer across the entire left-context of (1) since if any α_i is in V_N, the symbol (3) appears in the pushdown-store just below the $]_{\alpha_i}$ determining the node which satisfied this portion of the environment and thus will be reentered in the rule table for further advancement of the pointer just after the corresponding $]_{\alpha_i}$ has been scanned on the input tape and hence just in time for α_{i+1} to be spotted. So the pointer in the left-context of (1) will be immediately to the left of the dash when the first symbol of ψ_2 is scanned. At this time the A which can be on top of the pushdown-store can be removed and replaced by $\gamma_1 \ldots \gamma \;]_A \mid \beta_1 \ldots \beta_n$. Then as each ψ_i is scanned M can proceed ultimately removing the $]_A$ from the pushdown-store and entering $\mid \beta_1 \ldots \beta_n$ in the right-context table. The pointer can be advanced across the β_j's just as across the α_i's and thus the right-context table will contain no bar to acceptance of φ when the end of the input tape is reached. For this reason M accepts φ.

For the other direction, let φ be any string which is accepted by $\underline{M}$, it is clear that $\underline{\varphi}$ must be well-formed. Let $(\underline{\psi}_1, \underline{\psi}_2, \underline{\psi}_3)$ by a node of $\underline{\varphi}$. Consider a computation by which $\underline{M}$ accepts $\underline{\varphi}$ and let (1) be the rule which was utilized by operation (5) when the first symbol of $\underline{\psi}_2$ was scanned on the input tape. From the desceiption of $\underline{M}$ one can find the $\underline{\pi}$'s, $\underline{\sigma}$'s, $[_A$, $\underline{\chi}$'s $\underline{\omega}$'s, $]_A$ $\underline{\rho}$'s and $\underline{\tau}$'s of Definition 2 and thus determine that the node satisfies rule (1). But since $(\underline{\psi}_1, \underline{\psi}_2, \underline{\psi}_3)$ was any node of $\underline{\varphi}$, $\underline{\varphi}$ is analyzed by $\underline{G}$, completing the sketch of the proof of the theorem.

Corollary 1: For every phrase structure grammar $\underline{G}$, P($\underline{G}$) is a context-free language and conversely.

Proof: Let $\underline{G}$ be any phrase structure grammar. By Theorem 1, A($\underline{G}$) is a context-free language. By Definition 3, P($\underline{G}$) is the image of A($\underline{G}$) under the homomorphism d. The context-free languages are closed under homomorphism (Chomsky, 1963, Theorem 31). Therefore P($\underline{G}$) is a context-free language. For the converse, let $\underline{G}$ by any context-free grammar. Clearly L($\underline{G}$) $\subseteq$ A($\underline{G}$) since any labeled bracketing that can be obtained by rewriting the initial symbol of $\underline{G}$ is analyzed by $\underline{G}$. But A($\underline{G}$) $\subseteq$ L($\underline{G}$) also since a top to bottom, left to right derivation of any $\underline{\varphi} \in$ A($\underline{G}$) can be obtained by reading off the left labeled brackets of $\underline{\varphi}$. Thus L($\underline{G}$) = A($\underline{G}$) and so $L(\underline{G}) = d(L(\underline{G})) = d(A(\underline{G})) = P(\underline{G})$.

Remark: For any phrase structure grammar $\underline{G}$, a pushdown-storage automaton $\underline{M}'$ accepting $P(G)$ can be obtained from the automaton $\underline{M}$ described in the proof of Theorem 1 by altering operations (5) and (7) so that they apply regardless of what input symbol is scanned and do not move the input tape.

4. Applications

In a context-free grammar, the only way to express grammatical agreement between phrases which are not immediate constituents of the same phrase is by introducing additional nonterminal symbols and rules into the grammar. For example, there are good reasons to split an English declarative sentence into a subject noun phrase and a predicate verb phrase. The noun phrase will contain the subject noun as a constituent and the verb phrase will contain the main verb of the sentence. Now the noun and verb must agree in number and person and with the constituency described the only way to achieve this effect with context-free rules is by means of rules such as (9).

(9) $S \longrightarrow NP_{sg}\ VP_{sg}$

$S \longrightarrow NP_{pl}\ VP_{pl}$

$NP_{sg} \longrightarrow Det\ N_{sg}$

$NP_{pl} \longrightarrow Det\ N_{pl}$

$VP_{sg} \longrightarrow V_{sg}$

$VP_{pl} \longrightarrow V_{pl}$

$VP_{sg} \longrightarrow V_{sg}\ NP$

$VP_{pl} \longrightarrow V_{pl}\ NP$

$NP \longrightarrow NP_{sg}$

$NP \longrightarrow NP_{pl}$

It would be better to use context-sensitive rules such as in (10) to describe these constructions.

(10) $S \longrightarrow NP\ VP$

$NP \longrightarrow Det\ N$

$N \longrightarrow N_{sg}$

$N \longrightarrow N_{pl}$

$VP \longrightarrow V$

$VP \longrightarrow V\ NP$

$V \longrightarrow V_{sg}\ /\ N_{sg}$

$V \longrightarrow V_{pl}\ /\ N_{pl}$

If we are concerned only with analyzing context-free languages, we can use such rules to parse sentences rather than to generate them. Straightforward modification of existing context-free analysis computer programs such as that of Earley (1969) will permit them to handle arbitrary phrase structure grammars with the same efficiency they possess for context-free grammars. Thus for each grammar *G*, there is a constant k_G such that Earley's program can parse an input string of length *n* in an amount of time no more than $k_G n^3$. But k_G depends on the number of rules in *G*, so using fewer context-sensitive rules rather than more context-free rules can speed up parsing by a constant factor. This gain in speed could be of significance in natural language processing situations.

References

Chomsky, N. (1963) "Formal Properties of Grammar", In R. Bush, R. Luce and E. Galanter (eds.) Handbook of Mathematical Psychology, Vol. II, New York, Wiley.

Earley, J. (1969) "An Efficient Context-Free Parsing Algorithm" (to appear).

Peters, S. and R. W. Ritchie (1969a) "Context Sensitive Immediate Constituent Analysis——Context-Free Languages Revisited", (submitted to J. A. C. M.).

________ (1969b) "On the Generative Power of Transformational Grammars", (submitted to Information Sciences).