XVII. LINGUISTICS*

Prof. R. Jakobson
Prof. A. N. Chomsky
Prof. M. Halle
Prof. L. M. Kampf
Prof. A. L. Lipson
Prof. H. Putnam
Dr. G. H. Matthews
Dr. Paula Menyuk
T. G. Bever
J. A. Fodor
C. Fraser
Barbara C. Hall
J. J. Katz
S. J. Keyser
D. T. Langendoen
T. M. Lightner
P. M. Postal
J. Reitzes
J. J. Viertel
D. E. Walker

A. CONTEXT-FREE GRAMMARS AND PUSHDOWN STORAGE

A context-free (CF) grammar is given by a finite vocabulary $V = V_N \cup V_T$ (V_N, V_T disjoint), a designated symbol $S \in V_N$, and a finite set of rules $A \rightarrow \theta$, where $A \in V_N$ and θ is a string in V. θ follows from π if and only if $\pi = \phi A \psi$, $\theta = \phi \sigma \psi$, and $A \rightarrow \sigma$ is a rule of G. A π-derivation of θ is a sequence of strings $\sigma_1, \ldots, \sigma_n$ such that $\pi = \sigma_1$, $\theta = \sigma_n$, and σ_{i+1} follows from σ_i, for each $i < n$. The language L(G) generated by G is the set of strings x in V_T which is such that there is an S-derivation of x. We say that L(G) is, in this case, a context-free (CF) language. CF grammars have been studied rather widely, both with respect to empirical adequacy[1, 2] and abstractly.[3-6] There is no doubt that they are not adequate for natural language, although certain important features of natural languages can be represented by systems of this kind. Ginsburg and Rice point out that certain programming languages, in particular, ALGOL, have CF grammars, as do familiar artificial languages.

Consider an automaton M with two infinite tapes blocked off into squares each of which can contain a symbol, one an input tape, the other a storage tape. The control unit consists of a finite set of states; it can read the symbols of an input alphabet A_I from the input tape and the symbols of an output alphabet $A_0 \supset A_I$ from the storage tape. It can write the symbols of A_0 on the storage tape. If M is in state S_i scanning a on the input tape and b on the storage tape, we say that it is in the situation (a, S_i, b). The identity element a_0 cannot appear in a square of the input tape, and cannot be written on a square of the output tape. In the initial tape-machine configuration, M is in a designated initial state S_0, the storage tape contains a_0 in every square, and the input tape contains the symbols $b_1, \ldots, b_n$ of A_I in successive squares, with b_1 being the scanned symbol and $\# \notin A_0$ in every other square. M computes in the manner indicated below until its first return to S_0. If, at this point, it is in the situation $(\#, S_0, a_0)$, we say that it accepts the input string $b_1 \ldots b_n$. The set of strings that can be accepted by the device we call the language that it accepts.

The computation of M is controlled by a set of instructions of the form

(1) $(a, S_i, b) \rightarrow (S_j, x, k)$,

---

*This work was supported in part by the National Science Foundation (Grant G-7364 and Grant G-13903); and in part by the National Institutes of Health (Grant MH-04737-02).

where $a \in A_I$; $b \in A_0$; x is a string in A_0; S_i and S_j are states of the control unit; if $x = a_0$, then $k = 0$ or -1; if $x \neq a_0$, then k = length of x (i.e., $x = c_1 \ldots c_k$, where $c_m \in A_0$, $c_m \neq a_0$). We can assume, with no loss of generality, that $b = a_0$ only if $i = 0$ and $k \geq 0$. The instruction (1) applies when M is in the situation (a, S_i, b) and has the following effect: the control unit switches to state S_j; the input tape is moved one square to the left and the storage tape k squares to the left; if $k > 0$, the symbols of x are printed successively on the newly exposed squares of the storage tape as this tape moves to the left. Thus after the instruction has been carried out, M will be in the situation (c, S_j, d), where c is the symbol to the right of the previously scanned symbol a on the input tape, and d is the right-most symbol of x. A shift of -1 square to the left is to be interpreted as a shift of 1 square to the right. If $a = a_0$, we can interpret (1) as the instruction to operate as above, independently of what appears on the input tape, and without moving this tape.

Notice that a symbol is written on a square of the storage tape when and only when that tape moves to the left. We can, if we like, think of all symbols to the right of the scanned symbol of the storage tape as being automatically erased (replaced by a_0) as the tape shifts to the right. Then only the right-most symbol of the storage tape is available at each stage of the computation. The symbol most recently written in storage is the earliest to be read out of storage. In any case, we define the contents of the storage tape as the string to the left of and including the scanned symbol, and we say that the storage tape contains this string. More precisely, if $a_0, b_1, \ldots, b_n$ appear in successive squares of the storage tape and b_n is the scanned symbol, then $b_1 \ldots b_n$ is the contents of the storage tape. Notice that M accepts the input x only if it terminates its computation (starting in its initial configuration with # x # on the input tape) with a_0 as the contents of the storage tape, i.e., with the storage tape blank.

The device M that behaves in the way described above will be called a pushdown storage (PDS) automaton, following Newell, Shaw and Simon. It is a special case of a linear bounded automaton in the sense of Myhill.[7] The utility of such devices for analysis of syntactic structure by computer has been noted by many authors. The theory of PDS automata with possibly nondeterministic control units (as above) is the theory of "predictive analysis" in the sense of Rhodes, Oettinger, and others (cf., e.g., Oettinger[8]). We shall show that it has essentially the same limitations as the theory of context-free grammar.

THEOREM 1: A language L is accepted by some PDS automaton if and only if it is a CF language.

Thus we have a simple characterization of CF languages in terms of restricted infinite automata. We shall also extend Theorem 1 to a simple algebraic characterization of CF languages, and to a proof that PDS automata can, without loss of generality, be

restricted to partial control from the storage tape, in a manner indicated precisely below.

In general, the next step of a PDS automaton M depends on its entire situation (a, S_i, b). Suppose, however, that M never shifts the storage tape to the right, i.e., in each instruction of the form (1), k is non-negative. It is easy to see that in this case, control from the storage tape is playing no essential role in the operation of the device. In fact, let us construct a new device T with states $[S_i, a]$, where S_i is a state of M and $a \in A_0$, and with the initial state $[S_0, a_0]$. If M has the rule (1), T will have the rule $(a, [S_i, b], b) \rightarrow ([S_j, c], x, k)$, where $c \neq a_0$ and $x = yc$, or $x = a_0$ and $b = c$. Clearly the behavior of T is in no way different from that of M, but for T, the next step of a computation depends only on the input symbol and the present state. Eliminating redundant specifications, we can give each rule of T in the form $(a, \Sigma_i) \rightarrow (\Sigma_j, x)$, which indicates that when in state Σ_i reading the symbol a on the input tape, T switches to state Σ_j, writes x to the right of the present contents of the storage tape, and moves the tapes the appropriate distance to the left. T is thus a finite transducer of a very general sort. Both T and M map input strings into output strings, and languages into languages. T and M perform the same mapping. Given M (or T), we can construct immediately the inverse transducer T' that maps the string x into the string y if and only if M maps y into x, and that maps the language L into the set of all strings x such that $M(x) \in L$.

In general, where T is a transducer and L a language, we shall designate by T(L) the set of strings y such that for some $x \in L$, T maps x into y.

PROOF OF THEOREM 1: Suppose that M is a PDS automaton with input alphabet A_I and output alphabet $A_0 = \{a_0, \ldots, a_q\}$. We construct a new device M^* with the input alphabet A_I and the output alphabet A'_0 with 2q+1 symbols, where $A'_0 = A_0 \cup \{a'_1, \ldots, a'_q\}$. We shall treat each element a'_i as, essentially, the "right inverse" of a_i. More formally, let us say that the string x <u>reduces to</u> y if and only if there is a sequence of strings $z_1, \ldots, z_m$ $(m \geq 1)$ such that $z_1 = x$, $z_m = y$, and, for each $i < m$, there are strings w_i, $\overline{w}_i$, and $a_{\beta_i} \in A_0$ such that $z_i = w_i a_{\beta_i} a'_{\beta_i} \overline{w}_i$ and $z_{i+1} = w_i \overline{w}_i$. In other words, x reduces to y if $x = y$ or if y can be formed from x by successive deletions of substrings $a_j a'_j$.

We shall say that the string x is <u>blocked</u> if $x = y a_i z a'_j w$, where $i \neq j$ and z reduces to a_0. If x is blocked, then, for all y, xy is blocked and does not reduce to a_0. We shall say that the storage tape is blocked if the string that it contains is blocked.

Suppose that K and K^* are tape-machine configurations of M and M^*, respectively, meeting the following condition. When M is in K and M^* is in K^*, then M and M^* are scanning the same square of identical input tapes and are in the same internal state, and the string w contained on the storage tape of M^* reduces to the string y contained on the storage tape of M. Furthermore, if $y \neq a_0$, then $w = z a_k$, for some k. In this case we say that K and K^* <u>match</u>. Notice that when K and K^* match, either the storage tape of M is blank, in which case the contents of the storage tape of M^* is $z a'_k$, or M and M^*

are in the same situation.

With these notions defined, we return to the construction of M^*, given M. M^* will be a device that never switches the storage tape to the right. It will be constructed in such a way that if M does not accept x, then, with x as input, M^* will terminate its computation before reading through x, or with the storage tape blocked; and if M does accept x, then M^* will be able to compute in such a way that when it has read through all of x, the storage tape will not be blocked – that is, its contents will reduce to a_0.

The states of M^* will be designated by the same symbols as those of M, and S_0 will again be the initial state. The instructions of M^* are determined by those of M by the following rule. Let (1) be an instruction of M. If $k \geq 0$, M^* will have the instruction (1). Suppose that $k = -1$. Thus $b = a_m$, for some $m > 0$. Then M^* will have the instruction (2i), and (2ii), for each n.

(2) (i) $(a, S_i, a_m) \rightarrow (S_j, a'_m, 1)$

(ii) $(a, S_i, a_m) \rightarrow (S_j, a'_m a'_n a_n, 3)$

We can now easily prove the following fact (for details, see Chomsky[9]). Suppose that K_1 and K_1^* are matching configurations of M and M^*, respectively, and that instruction I_1 carries from K_1 to K_2. Then M^* has an instruction I_1^* that carries it from K_1^* to a configuration K_2^* that matches K_2. Furthermore, if I_2^* is an instruction of M^* that carries it from K_1^* to K_3^*, then either the storage tape is blocked in K_3^* or there is an instruction I_2 of M that carries it to a configuration K_3 that matches K_3^*. But M and M^* have the same initial configuration, with identical input tapes. Thus if M accepts x, M^* will be able to compute with input x until it terminates in the situation $(\#, S_0, a_0)$ or $(\#, S_0, a'_m)$, for some m, with a string y that reduces to a_0 contained on the storage tape. And if M does not accept x, then every computation of M^* with x as input will either block before termination in the situation $(\#, S_0, a)$ (for some $a \in A_0$), or it will terminate in this situation with the storage tape blocked. Notice that once the storage tape of M^* is blocked, it will remain blocked for the remainder of the computation.

The principle of computation of M^* is, briefly, this. Suppose that M and M^* are, respectively, in the matching configurations K_1 and K_1^*. M^* makes the guess that, after erasing the symbol a_m which is being scanned on the storage tape and entering the configuration K_2, M will either: (I) terminate with a blank storage tape, or (II) be scanning a_n on the storage tape. In case I, M^* writes a'_m on its storage tape and terminates. In case II, it writes $a'_m a'_n a_n$ on its storage tape, so that it, too, is now scanning a_n, having "erased" both a_m (by a'_m) and a_n (by a'_n). In either case, if the guess was right, the new configuration of M^* matches K_2. If it was wrong, M^* is blocked or its tape is blocked, and the computation cannot terminate with the storage tape containing a string that reduces to a_0.

But M^* is a PDS device that never moves the storage tape to the right. Hence, as noted above, we can construct a finite transducer T^* that maps x into y if and only if M^* will compute with x as input (in its initial configuration) until termination with y as output. Suppose that T^* is constructed from M^* in the manner indicated above. Let us now construct T from T^* by selecting two new symbols σ, $\sigma' \notin A_0$ and replacing each instruction of T^* of the form $(a, \Sigma_0) \rightarrow (\Sigma_j, x)$ by $(a, \Sigma_0) = (\Sigma_j, \sigma x)$; and each instruction of T^* of the form $(b, \Sigma_i) \rightarrow (\Sigma_0, y)$ by $(b, \Sigma_i) \rightarrow (\Sigma_0, y\sigma')$, where Σ_0 is the initial state of T^*; and otherwise, taking all instructions of T^*, and only these, as instructions for T, with Σ_0 as initial state of T. Let us now extend the definition of reduction given above by adding that $\sigma\sigma'$ reduces to a_0. The transducer T, so constructed, maps a string x into a string y that reduces to a_0, if and only if M accepts x. We thus have the following result.

LEMMA 1: Given a PDS automaton M, we can construct a transducer T with the following property: T maps x into a string y that reduces to a_0, if and only if M accepts x.

Suppose now that L(M) is the language accepted by the PDS automaton M; that T is the corresponding transducer guaranteed by Lemma 1; that K is the set of strings in the output alphabet of T which reduce to a_0; that U_I is the set of all strings in the input alphabet of T; and that T' is the "inverse" transducer to T. Then $L(M) = T'(K \cap T(U_I))$. But $T(U_I)$ is a regular event and K is a CF language. Furthermore, it is known that the intersection of a CF language and a regular event is a CF language[4] and, by a slight extension of this proof, that a finite transducer maps a CF language into another CF language (cf. also Schützenberger,[10] and Ginsburg and Rose[11]). Consequently, L(M) is a CF language. We thus have the following result.

LEMMA 2: If M is a PDS automaton, the language that it accepts is a CF language.

Suppose now that G is a CF grammar generating the language L(G). It is known that we can construct a normal CF grammar G' generating L(G) and containing only rules of the form $A \rightarrow BC$, $A \rightarrow a$, where $A, B, C \in V_N$ and $a \in V_T$.[3] By a slight extension of this argument, we can show easily that there is a normal CF grammar G^* generating L(G) and containing, furthermore, no pair of rules $A \rightarrow BC$, $D \rightarrow CE$. In this case we can tell unambiguously, for each nonterminal, whether it appears on a left branch or a right branch of a derivation.

Let us now construct a transducer T with states Σ, input alphabet A_I, and output alphabet A_0, as follows. Σ contains two states A_ℓ and A_r corresponding to each nonterminal A of G^*, and, in addition, an initial state σ. A_I contains the terminal vocabulary of G and the identity element a_0. A_0 contains symbols σ, σ' and, for each α in the vocabulary (terminal and nonterminal) V of G^*, the symbols α, α'. Where $A \rightarrow a$ $(a \in V_T)$ is a rule of G^*, T will have the instruction

(3) $(a, A_\ell) \rightarrow (A_r, Aaa'A')$.

If $A \rightarrow BC$ $(B, C \in V_N)$ is a rule of G^*, T will contain the instructions:

(4) $(a_0, A_\ell) \rightarrow (B_\ell, A)$

$(a_0, B_r) \rightarrow (C_\ell, a_0)$

$(a_0, C_r) \rightarrow (A_r, A')$

Also, we add the instructions $(a_0, \sigma) \rightarrow (S_\ell, \sigma)$ and $(a_0, S_r) \rightarrow (\sigma, \sigma')$, where S is the initial symbol of G^*.

T, so constructed, is a finite transducer, and it is immediately obvious that if and only if G generates x, T will map x into a string y that reduces to a_0 by successive cancellation of substrings $\alpha\alpha'$, as above. In fact, the output of T with x as input is simply the bracketed "structural description" of x given by the grammar G as x is derived, where A is regarded as a left bracket labelled A, and A' as the corresponding right bracket. The output of T is given by systematically tracing through the tree diagram representing the derivation of x by G^*.

But we can now construct a PDS automaton M that will accept x if and only if T maps x into a string y that reduces to a_0. Wherever T writes a symbol α', M will erase the scanned symbol α from the storage tape. Wherever T writes a symbol α, M will also write α. More precisely, we associate with G^* the PDS automaton M with the set of instructions

(5) $(a, A_\ell, \alpha) \rightarrow (A_r, a_0, 0)$,

for each α of the output alphabet, corresponding to each rule $A \rightarrow a$ $(a \in V_T)$ of G^*; and with the instructions

(6) $(a_0, A_\ell, \alpha) \rightarrow (B_\ell, A, 1)$

$(a_0, B_r, \alpha) \rightarrow (C_\ell, a_0, 0)$

$(a_0, C_r, A) \rightarrow (A_r, a_0, -1)$,

for each α of the output alphabet, corresponding to each rule $A \rightarrow BC$ $(B, C \in V_N)$ of G^*, as well as the rules $(a_0, \sigma, \alpha) \rightarrow (S, a_0, 0)$ and $(a_0, S_r, \alpha) \rightarrow (\sigma, a_0, 0)$, for each α of the output alphabet. Clearly M accepts just those strings generated by G^*. Thus, in conjunction with Lemma 2, we have Theorem 1.

But observe that the automaton M constructed to accept the language L(G) operates independently of the scanned symbol of the storage tape, except when it is using an "erase" instruction (i.e., moving the storage tape to the right). We call such a device a PDS automaton with restricted control. We see, then, that:

THEOREM 2: If M is a PDS automaton, there is a PDS automaton with restricted control that accepts the language accepted by M.

Suppose that we define a PDS automaton without control as one that operates independently of the scanned symbol of the storage tape, no matter what instruction it is applying. In this case, it is using the storage tape simply as a counter, and it is clear that not every CF language can be accepted – in particular, the set of all strings xx^*, where x^* is the reflection of x, cannot be accepted by any device of this type. Hence we cannot extend Theorem 2 to the case for which erase instructions are also independent of the scanned symbol of the storage tape.[12]

Let us consider again the transducer T associated by the construction (3), (4) with G^*. Let K be the set of strings in the output alphabet A_0 of T that reduce to a_0 by successive cancellation of substrings $\alpha\alpha'$ or $\alpha'\alpha$. We are thus, essentially, regarding α, α' as strict inverses in the free group $\mathcal{G}$ with the generators $\alpha \in A_0$. Notice, furthermore, that the grammar G^* from which T was constructed contains no rules $A \to BC$, $D \to CE$. Consequently, the output of T can never contain a substring $\alpha' x \alpha$, where x reduces to a_0. Hence even under this extended definition of "reduction," the transducer T will still have the property that G^* generates x if and only if T maps x into a string y that reduces to a_0.

Suppose that we now assume, as is natural, that the vocabulary V from which all CF grammars are constructed is a fixed finite set of symbols, so that K is a fixed CF language in the vocabulary V' containing V, σ, σ', and α', for each $\alpha \in V$. Let ϕ be the homomorphism (i.e., the one-state transducer) such that $\phi(a) = a$ for $a \in V_T$ and $\phi(\alpha) = a_0$ (the identity element of V') for each $\alpha \in V' - V_T$. Let U be the set of all strings in V'. Observe now that where G, G^* and T are as above, we have, in particular, the result that $L(G) = \phi(K \cap T(U))$.

It is a straightforward matter to construct a PDS automaton that will accept K; consequently, K is a CF language. It is well known[10] that for each transducer τ, $\tau(U)$ is a regular event. We have just seen that corresponding to each CF grammar G generating L(G) we can construct a transducer T such that $L(G) = \phi(K \cap T(U))$. Furthermore, as noted above, the intersection of a CF language and a regular event is a CF language, and transduction carries CF languages into CF languages. Summarizing these facts, then, we have the following general observation:

THEOREM 3: Given a regular event R, let $\psi(R) = \phi(K \cap R)$. Then $\psi(R)$ is a CF language, for each regular event R, and each CF language $L = \psi(R)$, for some regular event R.

Thus a CF language is completely determined by the choice of a certain regular event, and each such choice gives a CF language, given K, ϕ, ψ, as above.

The construction given above that gave a PDS automaton with restricted control

corresponding to each CF grammar is essentially a special case of the construction in Chomsky[3] that gives an optimal strictly finite "recognition routine" for a normal CF grammar. The construction here is much simpler, because of the relaxation of the requirement of strict finiteness. What we have here, essentially, is a mechanical procedure for associating with a normal CF grammar G a "recognition routine" that assigns to input strings the structural descriptions that are provided for them by G, using a potentially infinite (pushdown) memory for storage; whereas in the other case, a mechanical procedure was presented for associating with G a strictly finite "recognition routine" that does as well as can be done, with that memory restriction, in assigning to input strings the structural descriptions that are provided for them by G.

The results reported above are the result of work done jointly with M. P. Schützenberger. See Schützenberger[13] for generalizations and related results.

A. N. Chomsky

References

1. N. Chomsky, Syntactic Structures (Mouton and Company, 's-Gravenhage, 1957).

2. P. M. Postal, On the limitations of context-free phrase-structure descriptions, Quarterly Progress Report No. 64, Research Laboratory of Electronics, M.I.T., January 15, 1962, pp. 231-238.

3. N. Chomsky, On certain informal properties of grammars, Information and Control 2, 137-167 (1959).

4. Y. Bar-Hillel, M. Perles, and E. Shamir, On Formal Properties of Simple Phrase Structure Grammars, Technical Report No. 4, Office of Naval Research, Information Systems Branch, Washington, D.C., 1960.

5. M. P. Schützenberger, Some remarks on Chomsky's context free languages, Quarterly Progress Report No. 63, Research Laboratory of Electronics, M.I.T., October 15, 1961, pp. 155-170.

6. S. Ginsburg and H. G. Rice, Two Families of Languages Related to ALGOL, Technical Memorandum, Systems Development Corporation, Santa Monica, California, January 1961.

7. J. Myhill, Linear Bounded Automata, WADD Technical Note 60-165, 1960.

8. A. Oettinger, Automatic syntactic analysis and the pushdown store, Proc. Symposia on Applied Mathematics, Vol. 12, pp. 104-129, 1961.

9. N. Chomsky, Formal properties of grammars (in preparation).

10. M. P. Schützenberger, A remark on finite transducers, Information and Control 3, 185-196 (1961).

11. S. Ginsburg and G. Rose, Operations That Preserve Definability, Technical Memorandum, Systems Development Corporation, Santa Monica, California, 1961.

12. M. P. Schützenberger, Un problème de la théorie des automates (Séminaire Dubreil-Pisot, 13ème année, no. 3, Paris, 1959-1960).

13. M. P. Schützenberger, On a family of formal power series (to be published in Proc. Am. Math. Soc.).

B. INSEPARABLE PREFIXES OF GERMAN VERBS

In investigating the structure of the German verb phrase the problem of the verb prefixes presents interesting possibilities. The separable prefixes represent an extremely complex and productive system, closely interrelated, in all probability, with the whole system of verb-phrase modification, and, therefore, with modification in general. I have begun, however, by looking at the inseparable prefixes, assuming that this was the simpler problem, but the complexity that is revealed here is also of a very high order.

This type of affixation has traditionally been treated under the heading of 'word formation', in a purely morphological manner. But word formation in general is based on syntactic relationships. In German it is an extremely productive and important part of the grammar.

The list of inseparable prefixes is a small one – seven in all. However, this device is so active in the language that it multiplies the vocabulary of verbs several times.

The simplest class of derivations can be described in terms of phrase-structure (i.e., simple expansion) rules:

(1) (a) Er baut ein Haus.
Er andert den Plan.

Structure: NP + Nom + V + Acc + NP

(i) (P-S rule) V → ER + v

(ii) V → VER + v

(b) Er erbaut ein Haus.
Er verandert den Plan.

NP + Nom + {ER / VER} + Acc + NP

The application of these expansion rules requires a class of verbs that can be expanded by rule (i) and a class that can be expanded by rule (ii). A stem can fall into several of these classes. Thus BAUEN is both in (i) and (ii):

(a) Er baut das Haus. He builds the house.

(b) Er erbaut das Haus. He builds (up) the house.

(c) Er verbaut das Haus. He builds the house {wrong. / badly.}

It would be possible then to set up classes of verbs on which both rules (i) and (ii), or other rules of the same type, work.

But no one prefix ER-, VER-, MISS-, etc., is confined to such phrase-structure expansions. This leads us to the second type – transformational rules.

The phrase-structure rules simply add a morpheme without changing the structure of the sentence in any other way. In the transformational rules, the adding of the prefix morpheme is associated with changes in the syntactic structure, but the underlying

relations, i.e., those of the underlying sentence, remain preserved in this new formal guise:

(2) (a) Er baut ein Haus auf das Grundstück. — He builds a house on (to) the land.

$\underbrace{\text{NP + Nom}}_{1} + \text{V} + \text{Acc} + \text{NP} + (\text{Prep} = \text{AUF} + \text{Acc}) + \text{NP}$
1 2 3 4 5 6

(b) Er bebaut das Grundstück (mit einem Haus). — He covers the land by building (a house).

1 + BE + 2 + 3 + 6 + ((Prep = MIT) + 4)

(3) (a) Er baut ein Haus {mit seinem Geld. / aus Holz.} — He builds a house {with his money. / of wood.}

$\text{NP}_1 + \text{Nom} + \text{V} + \text{Acc} + \text{NP}_2 + \left(\text{Prep} = \left\{\begin{matrix}\text{AUS} + \text{Dat} \\ \text{MIT}\end{matrix}\right\}\right) + \text{NP}_3$
1 2 3 4 5 6

(b) Er verbaut {sein Geld / das Holz} (an einem Haus). — He uses up {his money / the wood} on a house.

1 + VER + 2 + 3 + 6 (+ (Prep = AN + Dat) + 4

There is a severe restriction on NP_3 (element 6). Thus if, instead of MIT SEINEN GELD (in (2)(a)), we selected MIT EINEM FREUND, MIT EINER SÄGE (He builds the house with a friend, with a saw), then the transformation is not applicable. Thus for NP_3 in such sentences, the transformation specifies a semantic category 'material'.

It will be noted that the prefix VER + the verb stem BAUEN occur both in the phrase-structure rule (1) (ii) and the transformation (3). The result is, of course, two entirely different sentences with two different meanings of the word VERBAUEN. Thus a formal definition of this difference is given.

Furthermore, we have:

(4) (a) Er baut ein Haus über den Weg. — He builds a house across the way.

$\text{NP}_1 + \text{Nom} + \text{V} + \text{Acc} + \text{NP}_2 + (\text{Prep} = \text{ÜBER} + \text{Acc}) + \text{NP}_3$
1 2 3 4 5 6

(b) Er verbaut den Weg (mit einem Haus). — He obstructs the way by building (a house).

1 + ver + 2 + 3 + 6 (+ (Prep = MIT) + 4)

Here is given the formal derivation for a third meaning of VERBAUEN. Note that again there is a restriction on NP_3 which specifies the class of, say, "means of access."

Still another type of linguistic relation revealed by the system of inseparable verb prefixes is one that can not be formalized by phrase-structure rules or by transformations as defined above, i.e., for which the underlying relation remains preserved. On

the contrary, in these more complicated and problematic derivations, a shift, a transfer, takes place.

Thus:

(5) (a) Er fangt einen {Fisch. / Ball.} He catches a {fish. / ball.}

(b) Er empfangt {den Lohn. / das Geschenk.} He receives the {reward, wage. / present.}

The derivation of (b) from (a) could be represented simply by a phrase-structure rule

(iii) V → ENT + v

(and, of course, by the morphophonemic rule ENT → EMP in env. _ + f) were it not for the fact that the set from which the object NP in (5)(a) can be selected does not coincide with that from which the object NP in (5)(b) can be selected.

A simple solution would be to list two different verbs FANGEN and EMPFANGEN, and let it go at that. However, the fact remains that the two words are related in the intuition of the speaker – and that, naively stated, the meaning of the one is somehow derived from the other. A specification of the sets containing the elements with which the verb forms enter into syntactic relationships, for instance, with their objects, as shown above, seems to offer a possibility of defining such derivations, or, at least, of approaching a definition.

An additional problem is to find the regularities that make possible statements of the relations between the sets containing the elements in the underlying constructions, and the sets containing the elements in the derived constructions. The relation between two (or more) such sets would then define the "shift" in meaning. The large number of cases in which there is such a shift in one definite direction, as, for instance, from "concrete" to "abstract," encourages the belief that this extremely difficult, but interesting, problem can yield, at least, a partial solution.

But for the whole problem of derivational morphology the questions still remain: To what extent can such rules be generalized? How extensive are the classes that they define, and, as a consequence, how large a number of such rules (or rules and subrules) must be written? Can the restriction that they define be extended to yield inclusive and revealing sets?

J. J. Viertel