[*Mechanical Translation and Computational Linguistics*, vol.11, nos.1 and 2, March and June 1968]

Some Mathematical Aspects of Syntactic Description

by Itiroo Sakai,* First Research Center, Defense Agency, 2-2-1 Nakameguro, Meguro, Toyko, Japan

The purpose of this paper is to help linguists construct a consistent, sufficient, and less redundant syntax of language. A sharp distinction is made between the syntactic function, which is an attribute of strings, and the distribution class, which is a set of strings. The syntactic function of a continuous or discontinuous string is defined as the set of all the acceptable contexts of the string and is called a complete neighborhood. Four different definitions of distribution classes are proposed, and their properties are discussed. Cross-classification of strings is clearly formulated. Concatenation rules of a language can be described in terms of complete neighborhoods or distribution classes. Some possible representations and their consequences are discussed. Transformation rules are also described in a similar way. However, there is the additional problem of correspondence between original strings and their transforms. Finally, some trivial but practically useful conventions are described.

1. Introduction

The grammar of a language should be consistent throughout its entire system. No features should remain unformulated in order for the grammar to be complete. At the same time, it is desirable that the grammar be as compact as possible. These are important requirements, particularly when the grammar is a machine-oriented one. A knowledge of the formal properties of syntax will help us to construct an objective system of grammar. Every term used in a description should be rigorously defined. If the grammar rules produce strings which deviate from the proper usage of the language, we should be able to trace back the definitions and locate the source of trouble. If a set of grammar rules describes the nature of a phrase marker, the label given to each node must have an unambiguous definition which associates the node label with the strings supplied as texts.

We need at least an objective criterion by which we can specify a language. This criterion is the dichotomous decision whether or not a given symbol string belongs to the language in question. We leave the decision to native speakers and consider the "acceptable strings" undefined.

A substring of an acceptable string is said to have a syntactic function. The syntactic function of a symbol string is regarded as the set of all acceptable utterances in which the string occurs [1]. We eliminate the string in question and define its syntactic function as the set of all acceptable contexts of the string. The set of all acceptable contexts of a string is called the complete neighborhood. A distribution class can be defined as the set of all strings whose complete neighborhoods are related to a given set of contexts in a specified way. With these fundamental concepts of syntactic functions and distribution classes, we can proceed to a more formal system of syntactic description.

A few questions may be immediately raised. Is it really possible to construct a grammar in such an elementary fashion? How can we list the elements of a set, extracting them, as we are, from a practically infinite number of strings, even though each string is assumed to be of finite length? Is it not useless to establish such sets for a natural language, most of which are likely to have only one element?

We would be in a better position were we creating a new language by preparing a grammar and a lexicon. Unfortunately, the situation is quite contrary when we are handling a natural language. The language exists, and we wish to find a grammar that accounts for all and only the acceptable strings of the language. We regard a language L as the set of strings that can be generated by a mechanism M whose internal structure is not known to us. We can observe only a part of the set of these strings in a limited length of time. We want to construct a hypothetical mechanism M' that generates all and only the strings in L. The internal structure of M and M' may not be the same. The output of M' is checked as to whether it belongs to L, and strings are supplied to M' to test whether M' accepts a string if and only if it is an element of L. To do this, we must have the set L or a mechanism which tells us whether or not the given string belongs to L. We call this mechanism a normative device, and it is a native speaker if a natural language is to be discussed. We simplify the situation by assuming a few separate strata in the mechanism. A string generated is supposed to have been transferred from one stratum to another before it becomes a string of natural language. An utterance has several different forms which correspond to the strata. Each form has its own grammar. The normative device will be a linguist in this case.

Since the number of strings is practically infinite, a linguist trying to construct a grammar will find it advantageous to establish rules that hold not for individual strings or facts but for a set of strings or a set of relevant facts. A linguistic phenomenon will be analyzed from vari-

* Deceased.

ous points of view, and it will help him to avoid listing a very large number of phenomena and rules. He will attach certain markers to the strings according to the way he considers consistent with his usage of language. He will then write down the rules in terms of the markers. He may also establish his rules in terms of sets of strings which share some features in their markers. The procedure of using these rules consists of two parts: the routine that compares each rule with the text and decides whether or not the rule is to be applied, and the transfer routine by which the relevant information is read out of the applicable rules and transferred to the text. In these procedures, both comparison and transfer are carried out with the coded markers. It is important that the meaning of the codes be unambiguously defined so that the code obtained in the text is exactly what the linguist desires.

Some of his rules may account for the majority of texts he has examined but may fail to account for some others or to rule out similar but inconsistent facts. He will test his rules by applying them to natural texts or by generating strings. The normative device will tell him whether or not a string is acceptable but may not tell him why. It is obvious that these procedures cannot practically be carried out on every string that may be supplied to a machine in the future, and that no one is able to predict what can occur when an arbitrary string is supplied to the machine. Nevertheless, it is required that the grammar be able to deal with most of the texts supplied in the future.

The grammar is inevitably affected by the nature of the normative device. If the normative device is so strict that it rejects every string which fails to meet such requirements as (1) the style of the string must be an ordinary one, (2) the statement must be logically correct, (3) the lexical usage must conform to the regular usage of the language, etc., then the linguist must prepare a separate rule for almost every string. He can break down the decision procedure into a few separate steps. The first device will accept a string if the internal relationships of the string are acceptable regardless of the truth-value of what the string designates. If the grammar is to be applied to input texts which are always grammatically correct and unambiguous, then a grammar which satisfies the requirement of this device will be sufficient. However, many unusual strings will be produced if the grammar is used in random generation and many ambiguous alternatives if it is used for analysis. The second device may reject those strings whose structure has an unallowable combination of lexical elements, thus eliminating some of the ambiguous alternatives in analysis and suppressing the output with improper usage of lexical elements in synthesis. The third device may reject as unacceptable those strings which are not logically consistent. If the linguist wants to have a more rigorous grammar that may be used for the random generation of nonsurprising sentences only, he may add more devices to the preceding ones, so that the grammar may be tested from such points of view. He will prepare his grammar keeping the characteristics of his normative device in mind. A series of digits will be assigned to the coded form of markers corresponding to each decision step. The procedure will be programmed to handle these digits independently, thus allowing a number of rules to be applied to the same string. If certain digits are related to each other and if a particular combination of codes is to obey a particular rule, the rule will be prepared independently and the general procedure will be prohibited.

As we shall see in the following pages, a number of similar but different representations are possible. If we do not understand the exact meaning of codes and rules or do not prepare the right program for the representation chosen, the rules established on the basis of ad hoc definitions will result in chaos. The formal property of grammar is not confined to a particular language but is common to many, probably to all languages. A grammar will not deviate greatly from its proper construction if its formal property is carefully examined.

2. Symbols, Strings, and Languages

"Symbol" is an undefined term. Morphemes, lexes, lexemes, or other units can be regarded as symbols. A linear arrangement of symbols is called a "string." All strings are possible strings. If a string is considered meaningful, it is an "acceptable string"; "acceptable string" is an undefined term.

These definitions are quite formal. If we confine ourselves to problems in morphotactics, the symbols are morphs and the acceptable strings are known as expressions or utterances. A symbol can be a morpheme, and a linear arrangement of morphemes is an acceptable string if it is recognized as a morphemic representation of an utterance. A string need not always be a linear arrangement of items. We can regard a labeled tree (called a phrase marker) as a string and a labeled node as the representation of the substring dominated by the node, although the term "string" seems inadequate in this case. A node represents a phrase marker consisting of all terminal and nonterminal nodes it dominates. Another kind of branch can be added to the syntactic tree to indicate the logical or semantic relationship among the constituents. We call this representation a net, provisionally. We regard a net as a string consisting of a number of labeled nodes whose arrangement is represented by two kinds of branches.

We define a language as the set of all acceptable strings. An acceptable string in a natural language is considered to have as many versions as the number of strata established by linguists. Each version of an acceptable string is an element of the language defined on the stratum in question. A transfer from one version to another is essentially a translation.

3. Contexts and Neighborhoods

3.1. Segments and Fragments

Suppose we have a linear string. We interrupt the string by deleting some of the symbols therein and insert a symbol ( ) of absence at each point of deletion. If a symbol of

absence is followed immediately by another, the two are contracted into one. A linear string is continuous if it is not interrupted.

All nodes in a syntactic tree are partially ordered. A node includes another if the linear string covered by the latter is a part of the linear string covered by the former. A treelike string is continuous if and only if all its nodes are included in one node D.

A substring of a string is called a "segment" and may be continuous or discontinuous. A discontinuous segment consists of several parts separated from each other. Each part of a segment is called a "fragment," which is necessarily continuous [1].

3.2. Contexts and Acceptable Contexts

Let r be a string, and let s be a segment of r. The string r may be continuous or discontinuous. The remaining part c of r is called the "context" of s in r. If r is acceptable, then we say that c is an acceptable context of s, c is acceptable to s, or s is acceptable to c.

3.3. Neighborhoods

A context is an interrupted string which becomes a continuous string if an appropriate segment is supplied at the points of interruption. Let $y = \text{set } (c_1, c_2, \ldots, c_n)$ be a set of contexts. If every context in y becomes an acceptable string when a string s is supplied to it, the set y defines a property of s. We call the set y a neighborhood of s. If y is a neighborhood of the strings s_1, s_2, s_3, for instance, then we say y is a neighborhood of the set $S = \text{set } (s_1, s_2, s_3)$, and we say that the set y represents a syntactic property common to all strings in S. Note that the "neighborhood" and the "okrjestnostj" are not the same [2]. A set of acceptable strings which contains the segment s is called a paradigm of s [1]. Our neighborhood is a paradigm in which the segment s is lacking.

4. Equivalence of Contexts

Let c_i and c_j be two contexts. Suppose a string s is acceptable to both c_i and c_j, and another string t is acceptable neither to c_i nor to c_j. In this case, we cannot tell the difference between c_i and c_j as far as the acceptability of the strings s and t is concerned. We say these contexts are equivalent to each other and write $c_i \sim c_j$ if the condition "c_i is acceptable to a string s_h, if and only if c_j is acceptable to s_h" is satisfied for every possible string s_h, and we write $c_i \nsim c_j$ otherwise. The relation of equivalence is reflexive, symmetric, and transitive:

$$c_i \sim c_i ; \tag{1}$$

$$\text{if } c_i \sim c_j , \quad \text{then} \quad c_j \sim c_i ; \tag{2}$$

$$\text{if } c_i \sim c_j \quad \text{and} \quad c_j \sim c_k , \quad \text{then} \quad c_i \sim c_k . \tag{3}$$

5. Complete Neighborhoods

5.1

Let y be an arbitrary set of contexts. It may include contexts which are not equivalent to each other and may not include all contexts which are equivalent to some context in it. The complete neighborhood $N(y)$ of y is the set of all contexts equivalent to some c' in y:

$$N(y) = \text{set } (c: c \sim c' \text{ for some } c' \text{ in } y) .$$

A set of contexts is complete or is a complete neighborhood if and only if it is the complete neighborhood of itself. Take a string s and let $C(s)$ be the set of all contexts acceptable to s. We show that $C(s)$ is complete, for: (1) if $c \in C(s)$, then $c \in N(C(s))$; and (2) if $c \in N(C(s))$, then $c \sim c'$ for some c' in $C(s)$; then $c \sim c'$ and c' is acceptable to s; then c is acceptable to s; then $c \in C(s)$. From (1) and (2), we have $N(C(s)) = C(s)$. Therefore, $C(s)$ is complete. We call $C(s)$ the complete neighborhood of the string s.

One may pick up an arbitrary segment of an acceptable string, call the remaining part the context of the segment, and establish the complete neighborhood of the segment. This kind of complete neighborhood contributes nothing to a grammar but some redundant rules. These nonsensical complete neighborhoods give rise to no trouble, because they never appear in any rules of the language.

The complete neighborhood $C(s)$ is considered to correspond to the syntactic function of the string s. The elements of $C(s)$ share the common property that every one of them can be an acceptable context of s, while no other contexts which do not belong to $C(s)$ are acceptable to s.

Let S be an arbitrary set of contexts. Some contexts in S may be acceptable to s and some others may not. The contexts acceptable to s must, at the same time, belong to $C(s)$, that is, to $C(s) \cap S$. If $C(s) \cap S = 0$, the string s cannot occur under the contextual condition defined by S, and vice versa. If $C(s) \cap S = C(t) \cap S$, we have no means for distinguishing the syntactic functions of s and t with respect to the given S. If S is the set of all possible contexts of the language, then $C(s) \cap S = C(s)$ for all strings s. If $C(s) = C(t)$, then we have no means of distinguishing the syntactic functions of s and t as far as acceptability is concerned.

5.2

It often occurs that a string r behaves like a string s under a certain condition and like t under another condition. This phenomenon will be restated as follows: for some set S' of contexts

$$C(r) \cap S' = C(s) \cap S' ,$$

and for another set S'' of contexts

$$C(r) \cap S'' = C(t) \cap S''$$

Setting

$$x = C(r), \quad y = C(s), \quad z = C(t),$$

we have

$$x \cap S' \cap S'' = y \cap S' \cap S'',$$

and

$$x \cap S' \cap S'' = z \cap S' \cap S''.$$

Taking the union of these two, we obtain

$$x \cap S' \cap S'' = (y \cup z) \cap S' \cap S''.$$

This means that r accepts every context in $S' \cap S''$ if it is acceptable to s or t. Now, we shall see the behavior of r with respect to the context set $S = S' \cup S''$.

$$\begin{aligned} x \cap S &= x \cap (S' \cup S''), \\ &= (x \cap S') \cup (x \cap S''), \\ &= (y \cap S') \cup (z \cap S''), \\ &\subseteq (y \cap S) \cup (z \cap S), \\ &= (y \cup z) \cap S. \end{aligned}$$

This result suggests that the behavior of r can be interpreted in terms of $y = C(s)$ and $z = C(t)$ and, moreover, that y and z may account for something lacking in $x = C(r)$ with respect to $S = S' \cup S''$:

$$\begin{aligned} (y \cup z) \cap S &= (y \cup z) \cap (S' \cup S''), \\ &= ((y \cup z) \cap S') \cup ((y \cup z) \cap S''), \\ &= (x \cap S') \cup (z \cap S') \cup (y \cap S'') \cup \\ &\quad (x \cap S''), \\ &= (x \cap (S' \cup S'')) \cup (y \cap S'') \cup \\ &\quad (z \cap S'), \\ &= (x \cap S) \cup (y \cap S'') \cup (z \cap S'). \end{aligned}$$

6. Elementary Neighborhoods

We have seen above that a complete neighborhood $x = C(r)$ is interpreted in terms of $y = C(s)$ and $z = C(t)$. We expect y and z to represent a simpler and more specific syntactic function. If $x = y \cup z$, $y \neq z$, $y \neq 0$, $z \neq 0$, then, for some c_i in y and c_j in z, we have $c_i \not\sim c_j$.

A set of mutually equivalent contexts leads us to the concept of the ultimate unit of syntactic function. Given a context c_i, the elementary neighborhood $e(i)$ with c_i as an element is defined as $e(i) = \text{set } (c\colon c \sim c_i)$. Since the equivalence is reflexive, symmetric, and transitive, no two distinct elementary neighborhoods have any elements in common.

Let x be a complete neighborhood and $e(i)$ an elementary neighborhood. If an element in x is a member of $e(i)$, then $e(i)$ is a subset of x because x is complete. Take an arbitrary element c_i in x. There is an $e(i)$ such that c_i is in $e(i)$. Therefore, we have $x = \bigcup e(i)$ for every complete neighborhood x and for all $e(i)$'s having at least one element in x. Every elementary neighborhood is complete. The intersection of complete neighborhoods can be complete, because every complete neighborhood is a union of elementary neighborhoods. Every union of elementary neighborhoods is also complete.

7. Distribution Classes

We have thus far discussed the syntactic function of symbol strings in terms of their acceptable contexts. A context is an environmental condition in which the strings occur. Given a context, we can classify the strings into two categories: the strings that can occur in the given environment, and the strings that cannot occur there.

7.1

7.1.1.—That the strings s and t are always mutually replaceable means that s can occur wherever t can occur and conversely t can occur wherever s can occur. In other words, every context c is in $C(t)$ if and only if c is in $C(s)$:

$$C(t) = C(s).$$

We represent the set of such strings t by

$$J(C(s)) = \text{set } (t\colon C(t) = C(s)).$$

7.1.2.—The distribution class $I(C(s))$ is the set of all strings t that can be always replaced by s:

$$I(C(s)) = \text{set } (t\colon C(t) \subseteq C(s)).$$

7.1.3.—Suppose a string t can occur wherever s can occur but s cannot necessarily occur in all contexts acceptable to t. In this case, $C(t)$ includes $C(s)$ as a subset. We define

$$H(C(s)) = \text{set } (t\colon C(t) \supseteq C(s)).$$

7.1.4.—If there exists at least one context c in which both s and t can occur, then c is in both $C(s)$ and $C(t)$:

$$c \in C(s) \cap C(t) \neq 0.$$

We introduce the convention $A\ (=)\ B$, which denotes that the intersection of the two sets A and B is not empty, and we define the distribution class

$$\begin{aligned} G(C(s)) &= \text{set } (t\colon C(t) \cap C(s) \neq 0), \\ &= \text{set } (t\colon C(t)\ (=)\ C(s)). \end{aligned}$$

7.2

We set $x = C(s)$ so that distribution classes can be defined without referring to a particular string s. A distribution class is defined as the set of strings whose complete neighborhood is related to a certain complete neighborhood x in a specified way. The distribution classes of the types mentioned above are written as:

$$J(x) = \text{set } (t\colon C(t) = x)\,,$$
$$I(x) = \text{set } (t\colon C(t) \subseteq x)\,,$$
$$H(x) = \text{set } (t\colon C(t) \supseteq x)\,,$$
$$G(x) = \text{set } (t\colon C(t)\ (=)\ x)\,.$$

A distribution class is said to be real if it is not empty, and imaginary if it is empty.

7.3

Let us examine a simple illustration. Suppose that a language consists of the acceptable strings:

they are (flying/red/making) planes ,

a (flying/red) saucer is an object ,

(flying/making) planes is an industry ,

and only these. We observe the strings

$$s_1 = \text{flying}\,,$$
$$s_2 = \text{red}\,,$$
$$s_3 = \text{making}\,,$$

and their contexts

$$c_1 = \text{they are ( ) planes}\,,$$
$$c_2 = \text{a ( ) saucer is an object}\,,$$
$$c_3 = \text{( ) planes is an industry}\,.$$

The complete neighborhoods of the strings are

$$C(s_1) = C(\text{flying}) = \text{set } (c_1, c_2, c_3)\,,$$
$$C(s_2) = C(\text{red}) = \text{set } (c_1, c_2)\,,$$
$$C(s_3) = C(\text{making}) = \text{set } (c_1, c_3)\,.$$

The distribution classes are determined by these complete neighborhoods as shown in table 1.

TABLE 1

ILLUSTRATION OF THE DISTRIBUTION CLASSES J, I, H, AND G, AS DETERMINED BY THE COMPLETE NEIGHBORHOODS C OF THE STRINGS s

j	s_j	$C(s_j)$	$J(C(s_j))$	$I(C(s_j))$	$H(C(s_j))$	$G(C(s_j))$
1. . .	flying	(c_1, c_2, c_3)	(s_1)	(s_1, s_2, s_3)	(s_1)	(s_1, s_2, s_3)
2. . . .	red	(c_1, c_2)	(s_2)	(s_2)	(s_1, s_2)	(s_1, s_2, s_3)
3. . . .	making	(c_1, c_3)	(s_3)	(s_3)	(s_1, s_3)	(s_1, s_2, s_3)

The elementary neighborhoods are found immediately. The context c_1 is the only context acceptable to all three strings—"flying," "red," and "making." Therefore, $e(1) = \text{set } (c\colon c \sim c_1) = \text{set } (c_1)$. The context c_2 is acceptable to "flying" and "red," and neither c_1 nor c_3 is acceptable to both these two and only these two strings: $e(2) = \text{set } (c\colon c \sim c_2) = \text{set } (c_2)$. Similarly, $e(3) = \text{set } (c_3)$. Therefore,

$$C(s_1) = e(1) \cup e(2) \cup e(3)\,,$$
$$C(s_2) = e(1) \cup e(2)\,,$$
$$C(s_3) = e(1) \cup e(3)\,.$$

7.4

7.4.1.—$J(x) = I(x) \cap H(x)$, because $t \in J(x)$ if and only if $C(t) = x$, if and only if $C(t) \subseteq x$ and $C(t) \supseteq x$, if and only if $t \in I(x) \cap H(x)$.

7.4.2.—$I(x) \cup H(x) \subseteq G(x)$ for $x \neq 0$, because $t \in I(x) \cup H(x)$ if and only if $t \in I(x)$ or $t \in H(x)$, if and only if $C(t) \subseteq x$ or $C(t) \supseteq x$; then $C(t)\ (=)\ x$ for $x \neq 0$ if and only if $t \in G(x)$.

7.5

The equality $x = y$ of two sets is reflexive, symmetric, and transitive. Therefore, $J(x) = J(y)$ if and only if $J(x)\ (=)\ J(y)$. This means that any two different sets can have no elements in common and, consequently, that every element belongs to one and only one set of the type $J(x)$.

7.6

7.6.1.—Let x be an elementary neighborhood:

$$I(x) = \text{set } (t\colon C(t) \subseteq x)\,,$$
$$= \text{set } (t\colon C(t) = x)\,,$$
$$= J(x)\,,$$

so that $C(t)$ is also elementary. If $x \neq 0$, we have

$$G(x) = \text{set } (t\colon C(t)\ (=)\ x)\,,$$
$$= \text{set } (t\colon C(t) \supseteq x)\,,$$
$$= H(x)\,.$$

7.6.2.—Let x be a complete neighborhood, and let $C(t)$ be elementary for all t:

$$G(x) = \text{set } (t\colon C(t)\ (=)\ x)\,,$$
$$= \text{set } (t\colon C(t) \subseteq x)\,,$$
$$= I(x)\,.$$

If $x \neq 0$,

$$H(x) = \text{set } (t\colon C(t) \supseteq x)\,,$$
$$= \text{set } (t\colon C(t) = x)\,,$$
$$= J(x)\,,$$

so that x is also elementary.

7.6.3.—If $C(t)$ is elementary for all t, and x is also elementary and nonempty, then

$$G(x) = H(x) = I(x) = J(x)\,.$$

7.7

If $x = y \cup z$, then

$$G(x) = G(y) \cup G(z) , \tag{1}$$

$$H(x) = H(y) \cap H(z) , \tag{2}$$

$$I(x) \supseteq I(y) \cup I(z) . \tag{3}$$

Proof (1): $t \in G(x) = G(y \cup z)$ if and only if $C(t) (=) x = y \cup z$, if and only if $C(t) (=) y$ or $C(t) (=) z$, if and only if $t \in G(y)$ or $t \in G(z)$, if and only if $t \in G(y) \cup G(z)$.

Proof (2): $t \in H(x)$ if and only if $C(t) \supseteq x = y \cup z$, if and only if $C(t) \supseteq y$ and $C(t) \supseteq z$, if and only if $t \in H(y)$ and $t \in H(z)$, if and only if $t \in H(y) \cap H(z)$.

Proof (3): $t \in I(y) \cup I(z)$ if and only if $C(t) \subseteq y$ or $C(t) \subseteq z$; then $C(t) \subseteq y \cup z = x$ if and only if $t \in I(x)$.

7.8

If $x = y \cap z$, then

$$G(x) \subseteq G(y) \cap G(z) , \tag{1}$$

$$I(x) = I(y) \cap I(z) . \tag{2}$$

Proof (1): $t \in G(x)$ if and only if $C(t) \cap x \neq 0$, if and only if $C(t) \cap y \cap z \neq 0$; then $C(t) \cap y \neq 0$ and $C(t) \cap z \neq 0$, if and only if $t \in G(y) \cap G(z)$.

Proof (2): $t \in I(x)$ if and only if $C(t) \subseteq x = y \cap z$, if and only if $C(t) \subseteq y$ and $C(t) \subseteq z$, if and only if $t \in I(y)$ and $t \in I(z)$, if and only if $t \in I(y) \cap I(z)$.

8. Concatenation

8.1. Concatenation of Sets

Let a, b, c, etc., be elements of sets. We call an ordered arrangement of these elements a concatenation. Let A, B, C, etc., be sets. We define the concatenation of sets as

$$AB \ldots D = \text{set } (ab \ldots d;\ a \text{ in } A,\ \text{and } b \text{ in } B, \ldots, \text{and } d \text{ in } D) .$$

In our present discussion, the elements are all strings or all contexts.

The following formulas are frequently used:

$$AB = CD \text{ if and only if } A = C \text{ and } B = D , \tag{1}$$

because $AB = CD$ if and only if ($ab \in AB$ if and only if $ab \in CD$), if and only if (($a \in A$ and $b \in B$) if and only if ($a \in C$ and $b \in D$)), if and only if (($a \in A$ if and only if $a \in C$) and ($b \in B$ if and only if $b \in D$)), if and only if $A = C$ and $B = D$.

$$A(B \cup C) = AB \cup AC , \tag{2}$$

because $ab \in A(B \cup C)$ if and only if $a \in A$ and $b \in B \cup C$, if and only if ($a \in A$ and $b \in B$) or ($a \in A$ and $b \in C$), if and only if $ab \in AB$ or $ab \in AC$, if and only if $ab \in AB \cup AC$.

Similarly,

$$(A \cup B)C = AC \cup BC . \tag{3}$$

$$AB \cap CD = (A \cap C)(B \cap D) , \tag{4}$$

because $ab \in AB \cap CD$ if and only if $ab \in AB$ and $ab \in CD$, if and only if $a \in A$ and $b \in B$ and $a \in C$ and $b \in D$, if and only if $a \in A \cap C$ and $b \in B \cap D$, if and only if $ab \in (A \cap C)(B \cap D)$.

8.2. Concatenation of Sets of Strings

Let p be a string and let $r_1, r_2, \ldots, r_n$ be segments of p which do not mutually overlap. A segment t consisting of $r_1, r_2, \ldots, r_n$ is the concatenation of these segments. The segment t is also a segment of p, consisting of the fragments of $r_1, r_2, \ldots, r_n$ arranged in their relative order in the original string p. It is convenient to assign a definite notational form to a concatenation in order to specify the arrangement of fragments.

Let $R_1, R_2, \ldots, R_i, \ldots, R_n$ be sets of strings, and let $P = R_1R_2 \ldots R_i \ldots R_n$ be their concatenation. Let $p = r_1r_2 \ldots r_i \ldots r_n$ be a concatenation of strings which is in P. If p is an acceptable string, it is at the same time in the language L:

$$p = r_1r_2 \ldots r_i \ldots r_n \in R_1R_2 \ldots R_i \ldots R_n \cap L .$$

If $P = R_1R_2 \ldots R_i \ldots R_n \subseteq L$, then this is a grammar rule which generates the subset P of L. Assume we have such a rule and that we add another string r'_i to R_i. If the concatenation yields unacceptable strings, then do not add r'_i to R_i. If no more strings can be added to any sets R_i, the rule defines a particular type of strings in L. It should be noted, however, that this is by no means a systematic description of the grammar of L. If every p in P is acceptable, the set P is a context-free grammar rule.

8.3. Concatenation of Sets of Contexts

8.3.1.—Let $p = r_1r_2 \ldots r_i \ldots r_j \ldots r_n$ be a string of symbols, and let $c(r_i/p) = r_1r_2 \ldots r_{i-1}( )r_{i+1} \ldots r_n$ and $c(r_j/p) = r_1r_2 \ldots r_{j-1}( )r_{j+1} \ldots r_n$ be the contexts of r_i and r_j in p, respectively. By $c(r_i/p)c(r_j/p)$, we mean the concatenation of contexts, formally stated as follows:

$$c(r_i/p)c(r_j/p) = r_1 \ldots r_{i-1}( )r_{i+1} \ldots r_{j-1}( )r_{j+1} \ldots r_n$$
$$= c(r_ir_j/p) .$$

8.3.2.—Let $P = R_1R_2 \ldots R_i \ldots R_n$ be the concatenation of sets R_i's of strings. The set of all possible contexts $c(r_{ik}/p)$ of r_{ik} in R_i is written as

$$\text{set } (r_1 \ldots r_{i-1}( )r_{i+1} \ldots r_n : r_j \in R_j) ,$$

where $j = 1, 2, \ldots, i-1, i+1, \ldots, n$. This is the concatenation P in which the set R_i is lacking. So, we write

$$C(R_i/P) = R_1R_2 \ldots R_{i-1}( )R_{i+1} \ldots R_n .$$

If another set R_j is lacking in P, we have

$$C(R_iR_j/P) = R_1 \ldots R_{i-1}( )R_{i+1} \ldots R_{j-1}( )R_{j+1} \ldots R_n$$

and consider this the concatenation of $C(R_i/P)$ and $C(R_j/P)$. This means, for instance, that the concatenation of $A( )CDE$ and $ABC( )E$ is put equal to $A( )C( )E$, and that $A( )CD$ and $AB( )D$ yield $A( )( )D = A( )D$ when they are concatenated.

8.4. Linear Strings and Phrase Markers

We make a distinction between linear concatenations and tree structures. If *abcd* is a linear concatenation, we have

$$abcd = a(bcd) = (ab)(cd) = (abc)d .$$

If the concatenation is used to represent a tree structure, the forms *abc*, *(ab)c*, and *a(bc)*, for instance, are not the same. Therefore, for

$$ABC = \text{set } (abc\colon a \text{ in } A,\ b \text{ in } B, \text{ and } c \text{ in } C) ,$$

$$(AB)C = \text{set } ((ab)c\colon a \text{ in } A,\ b \text{ in } B, \text{ and } c \text{ in } C) ,$$

$$A(BC) = \text{set } (a(bc)\colon a \text{ in } A,\ b \text{ in } B, \text{ and } c \text{ in } C) ,$$

we have

$$ABC \cap (AB)C = 0 ,$$

$$(AB)C \cap A(BC) = 0 ,$$

$$A(BC) \cap ABC = 0 .$$

By an acceptable phrase marker, we mean a phrase marker which appears during the process of generating an acceptable string. Therefore, a phrase marker assigned to a potentially acceptable string introduced for the plausible derivation of an acceptable string is also an acceptable phrase marker.

We confine ourselves to binary concatenations for simplicity. The following discussions can be easily generalized to longer concatenations. An unambiguous concatenation, $ABCD$ for instance, is considered to be one of the three binary concatenations $A(BCD)$, $(AB)(CD)$, or $(ABC)D$, when the discussion is strictly binary. In a linear string description, however, this is not very important. One can assume one of these three to be acceptable and discard the other two as unacceptable. In a tree-structure description, some one of these three will be chosen so as to make the whole description of the grammar simpler.

We assume that the binary concatenations required by the grammar are $(AB)(CD)$, $A(BC)$, $(BC)D$, and only these. The possible binary tree structures of $ABCD$ are: $ABCD = A(BCD) \cup (AB)(CD) \cup (ABC)D$. Since we are to handle binary concatenations only, we regard two concatenations as not the same if their structures are not the same:

$$(AB)(CD) \cap A(BCD) = 0 ,$$

$$(AB)(CD) \cap (ABC)D = 0 .$$

Therefore, $ABCD = (AB)(CD)$ if and only if

$$(AB)(CD) \neq 0 , \tag{1}$$

$$A(BCD) = 0 , \tag{2}$$

$$(ABC)D = 0 . \tag{3}$$

By assumption, $ABC = A(BC) \cup (AB)C = A(BC)$. Therefore,

$$A(BC) \neq 0 , \tag{4}$$

$$(AB)C = 0 , \tag{5}$$

because $A(BC) \cap (AB)C = 0$. Similarly, $BCD = B(CD) \cup (BC)D = (BC)D$ if and only if

$$(BC)D \neq 0 , \tag{6}$$

$$B(CD) = 0 . \tag{7}$$

From (2) we have

$$A(BCD) = A(B(CD) \cup (BC)D) = 0 .$$

By (7) and (6),

$$A(BCD) = 0 \cup A((BC)D) = 0 ,$$

or

$$A \neq 0 ,\ (BC)D \neq 0 ,\ A((BC)D) = 0 . \tag{8}$$

From (3) we have

$$(ABC)D = (A(BC) \cup (AB)C)D = 0 .$$

By (4) and (5),

$$(ABC)D = (A(BC) \cup 0)D = (A(BC))D = 0 ,$$

$$A(BC) \neq 0 ,\ D \neq 0 ,\ (A(BC))D = 0 . \tag{9}$$

Now, we can describe the syntax of these strings in terms of binary concatenations only, if we establish the rules (1) $\sim$ (9).

9. Concatenation of Complete Neighborhoods

9.1

If the distribution classes $J(x)$ and $J(y)$ are real, then there exist strings r and s, such that $C(r) = x$ and $C(s) = y$. By definition, $C(r_i) = x$ for all r_i in $J(x)$, and $C(s_j) = y$ for all s_j in $J(y)$. Any string which contains the segment r_i—$p(r_i) = \ldots r_i \ldots$—is acceptable if and only if $p(r) = \ldots r \ldots$ is acceptable, the string $p(s_j) = \ldots s_j \ldots$ is acceptable if and only if $p(s) = \ldots s \ldots$ is acceptable. Suppose $p(r_is_j) = \ldots r_i \ldots s_j \ldots$ is a string with both r_i and s_j in it. Any such string is acceptable if and only if $p(r_is) = \ldots r_i \ldots s \ldots$ is acceptable, and $p(r_is)$ is acceptable if and only if $p(rs) = \ldots r \ldots s \ldots$ is acceptable. Therefore, $p(r_is_j)$ is acceptable if and only if $p(rs)$ is acceptable. That is,

$$C(r_is_j) = C(rs) .$$

We define the concatenation $C(r)C(s)$ of complete neighborhoods as the complete neighborhood $C(rs)$ of the

concatenated strings. Generally, we set $xy = C(rs)$, $r \in J(x)$, $s \in J(y)$ for all complete neighborhoods x and y, where $J(x)$ and $J(y)$ may be real or imaginary.

9.1.1.—We make a distinction between the concatenation of complete neighborhoods $xy = C(r)C(s)$ and the complete neighborhood $z = C(rs)$ of concatenated strings. The property of the language is introduced when they are written $xy = z$ or $C(r)C(s) = C(rs)$, where the property z of rs is derived from the properties x and y of the constituent strings r and s. Note, however, that, if $x = C(r)$ and $y = C(s)$, then $xy = C(rs)$, while $xy = C(rs)$ does not always result in $x = C(r)$ or $y = C(s)$. This is seen when one assumes that $C(rs) = C(r's')$, and hence both $qrst$ and $qr's't$ are acceptable but neither $qrs't$ nor $qr'st$ is acceptable. Set $x = C(r)$, $y = C(s)$, $x' = C(r')$, $y' = C(s')$. We see that $q(\ )st$ is in x but not in x', $q(\ )s't$ is in x' but not in x, or $x \neq x'$; and that $qr(\ )t$ is in y but not in y', $qr'(\ )t$ is in y' but not in y, or $y \neq y'$. That is,

$$x \neq x' \quad \text{and} \quad y \neq y' ,$$

even if

$$xy = x'y' .$$

9.1.2.—One may want to regard the strings r and r' as having the same syntactic function, even if their constituent segments are syntactically quite different. Formally speaking, one wants to assign the same complete neighborhood x to $r = st$ and $r' = s't'$, even if s and s' and/or t and t' are not syntactically the same. That is, setting $x = C(r) = C(r')$, $y = C(s)$, $y' = C(s')$, $z = C(t)$, $z' = C(t')$, we obtain $x = yz$ and $x = y'z'$, even if $y \neq y'$ and/or $z \neq z'$; yz' and $y'z$ may or may not be equal to x. In this case, we write

$$x = yz \ (+) \ y'z'$$

and consider this an abbreviated form of two equations.

9.2

We have generalized and transferred the concatenated strings to concatenated sets of strings and then to concatenated complete neighborhoods. The complete-neighborhood representation provides us with a less complicated approach, especially when the string is syntactically ambiguous. The distribution class $J(x)$ represents the narrowest classification of strings and no further subclassification is possible, whereas its complete neighborhood x can be subclassified if x is not an elementary neighborhood. If $r \in J(x)$ and $x = y \cup z$, then we can talk about imaginary strings r' and r'', such that $C(r') = y$ and $C(r'') = z$. These imaginary strings, always referred to implicitly in terms of distribution classes, can be discussed explicitly in terms of complete neighborhoods.

9.3

We introduce coefficients which indicate the presence or absence of sets. Let a, b, etc., be the coefficients, and x, y, etc., sets. The value of a coefficient is either 0 or 1:

$$\begin{aligned} ax = 0 &= \text{empty set}, && \text{if } a = 0 , \\ &= x , && \text{if } a = 1 . \end{aligned}$$

The sum $a + b$ of coefficients a and b is given by

$$\begin{aligned} ax \cup bx = (a + b)x &= x , && \text{if } a = 1 \text{ or } b = 1 , \\ &= 0 , && \text{if } a = b = 0 . \end{aligned}$$

The product $ab = a \times b$ of coefficients a and b is given by

$$\begin{aligned} ax \cap by = ab(x \cap y) &= (a \times b)(x \cap y) \\ &= x \cap y , && \text{if } a = b = 1 \\ &= 0 , && \text{if } a = 0 \text{ or } b = 0 . \end{aligned}$$

Therefore, the coefficients are Boolean:

$$0 + 0 = 0 , \quad 0 + 1 = 1 + 0 = 1 + 1 = 1 ,$$
$$0 \times 0 = 0 \times 1 = 1 \times 0 = 0 , \quad 1 \times 1 = 1 .$$

Consequently, for concatenation, we have

$$(ax)(by) = abxy .$$

We saw that a complete neighborhood x can be represented by the union of elementary neighborhoods $e(i)$:

$$x = \bigcup e(i) \text{ for } e(i) \ (=) \ x .$$

Let us introduce coefficients $x(i)$, such that

$$\begin{aligned} x(i) &= 0 \text{ if } e(i) \cap x = 0 , \\ &= 1 \text{ if } e(i) \subseteq x ; \end{aligned}$$

and no other cases can possibly occur. In virtue of the coefficients, we write $x = \bigcup x(i)e(i)$, $y = \bigcup y(j)e(j)$, $z = \bigcup z(k)e(k)$, where the indices i, j, k are to cover all possible elementary neighborhoods of the language.

(1): If $z = x \cup y$, then

$$\begin{aligned} x \cup y &= (\bigcup x(i)e(i)) \cup (\bigcup y(j)e(j)) , \\ &= \bigcup (x(k) + y(k))e(k) , \\ &= \bigcup z(k)e(k) . \end{aligned}$$

If $e(k) \subseteq x$ or $e(k) \subseteq y$, then $e(k) \subseteq z$. Therefore, for $x(k) + y(k) = z(k)$, we have

$$0 + 0 = 0 ,$$
$$1 + 0 = 0 + 1 = 1 + 1 = 1 .$$

(2): If $z = xy$, then

$$\begin{aligned} z &= (\bigcup x(i)e(i))(\bigcup y(j)e(j)) , \\ &= \bigcup\bigcup x(i)y(i)e(i)e(j) , \\ &= \bigcup\bigcup z(i,j)e(i)e(j) . \end{aligned}$$

By the definition of concatenation, $e(i)e(j) \subseteq xy$ if and only if $e(i) \subseteq x$ and $e(j) \subseteq y$. That is, $z(i,j) = 1$ if and

only if $x(i) = y(j) = 1$. Therefore, for $x(i)y(j) = z(i,j)$, we have

$$1 \times 1 = 1 ,$$
$$0 \times 0 = 0 \times 1 = 1 \times 0 = 0 .$$

(3): The concatenation of two elementary neighborhoods is complete, because it is the complete neighborhood of the concatenated strings r and s such that $C(r) = e(i)$, $C(s) = e(j)$. Therefore, we set $e(i)e(j) = \bigcup a(i,j,k)e(k)$, and we have

$$\begin{aligned} z &= xy , \\ &= \bigcup\bigcup z(i,j)e(i)e(j) , \\ &= \bigcup\bigcup\bigcup z(i,j)a(i,j,k)e(k) , \\ &= \bigcup z(k)e(k) . \end{aligned}$$

We have also $e(k) \subseteq z$ if and only if $e(i)e(j) \subseteq xy$ and $e(k) \subseteq e(i)e(j)$. Therefore, for the expression $z(i,j)a(i,j,k) = z(k)$, we have

$$0 \times 0 = 0 \times 1 = 1 \times 0 = 0 ,$$
$$1 \times 1 = 1 .$$

10. Concatenation of Distribution Classes

10.1

$J(u)J(v) \subseteq J(uv)$, because $rs \in J(u)J(v)$ if and only if $r \in J(u)$ and $s \in J(v)$, if and only if $C(r) = u$ and $C(s) = v$, if and only if $C(r)C(s) = uv$; then $C(rs) = uv$ if and only if $rs \in J(uv)$.

10.2

$I(u)I(v) \subseteq I(uv)$, because $rs \in I(u)I(v)$ if and only if $r \in I(u)$ and $s \in I(v)$, if and only if $C(r) \subseteq u$ and $C(s) \subseteq v$, if and only if $C(r) \cap u = C(r)$ and $C(s) \cap v = C(s)$, if and only if $C(r)C(s) \cap uv = (C(r) \cap u)(C(s) \cap v) = C(r)C(s)$, if and only if $C(r)C(s) \subseteq uv$; then $C(rs) \subseteq uv$ if and only if $rs \in I(uv)$.

10.3

$H(u)H(v) \subseteq H(uv)$, because $rs \in H(u)H(v)$ if and only if $r \in H(u)$ and $s \in H(v)$, if and only if $C(r) \supseteq u$ and $C(s) \supseteq v$, if and only if $C(r) \cap u = u$ and $C(s) \cap v = v$, if and only if $C(r)C(s) \cap uv = (C(r) \cap u)(C(s) \cap v) = uv$, if and only if $C(r)C(s) \supseteq uv$; then $C(rs) \supseteq uv$ if and only if $rs \in H(uv)$.

10.4

$G(u)G(v) \subseteq G(uv)$, because $rs \in G(u)G(v)$ if and only if $r \in G(u)$ and $s \in G(v)$, if and only if $C(r) \cap u \neq 0$ and $C(s) \cap v \neq 0$, if and only if $C(r)C(s) \cap uv = (C(r) \cap u)(C(s) \cap v) \neq 0$; then $C(rs) \cap uv \neq 0$ if and only if $rs \in G(uv)$.

11. Concatenation Rules for Generation and Recognition

The generation of acceptable strings usually begins with an axiom which is expanded repeatedly into a string of constituents. After generating one or more strings by this procedure, we transform them to yield a new string. Each rule of a grammar indicates the arrangement of a few items to be concatenated, accompanied by some other necessary information. We assume the items arranged in a rule are either complete neighborhoods or distribution classes. Let us see what happens during the generation and recognition of acceptable strings.

The axiom indicates that the string to be generated has to be acceptable. This axiom node is expanded by phrase-structure rules as a concatenation of constituents arranged in the form of a tree. Each node has a label representing its nature. It may dominate another node or a concatenation of nodes. This is terminated when the lexical items replace the nodes. In doing this, no two or more nodes are contracted to a single node or rewritten as another concatenation of nodes, and no nodes are erased.

The construction of a context-sensitive rewrite system depends on the order of applying the rules to the terminal nodes. If a node is rewritten, the context of the other nodes changes, and this change has to be taken into consideration when the rules are prepared. The situation becomes simpler if we observe the following scheme. Let A be the axiom node or the root of a phrase marker. We apply a rule to A and expand it into a few nodes, and the nodes are equally distant from A, say, one unit of distance. We rewrite these terminal nodes, and, if there are no more rules applicable or if we no longer want to apply the optional rules, we make use of the carry-down rules to rewrite the terminal nodes, that have not been rewritten, with the identical node symbol [3]. Now, all the terminal nodes are one unit farther from the root, and they are again equally distant from the root. We specify the necessary contexts of the rules not in terms of the terminal string being generated but in terms of the string that was generated. The machine makes use of two stores: the one for storing the string that was generated, and the other for writing the string being generated. Let us examine a simple example. Let A be a node and consider a derivation by the context-sensitive rules

$$A/BC ,$$
$$B/F//\_\_C ,$$
$$C/G//B\_\_ ,$$

where $X/Y//U\_\_V$ means "X is rewritten as Y in the environment $U\_\_V$." If B is replaced by F to yield FC, the third rule can no longer be applied because of the lack of necessary context $B\_\_$; if C is replaced by G first, the second rule cannot be applied. In our system, the second rule is applicable to BC because B is followed by C, and the third rule is also applicable because C is preceded by B. We call this type of rule a realization rule [5, pp. 105, 122].

One can regard each step of the above procedure as a type of transformation: Transformations and realizations are the same in that they are both mappings.

We assign lexical items to the terminal nodes before the phrase marker undergoes any transformation. This is to ensure the right choice of transformation rules in accordance with the lexical items chosen and to ensure the correspondence among the base phrase marker and its transforms without keeping a separate note of the node correspondence. "John," "doctor," and "examine" must be introduced before the sentence takes the feature of active or passive voice, so that "John was examined by the doctor" will never be paired with "the man built a house."

A symbol string is recognized as a sentence when its derivational history is accounted for by virtue of transformation and phrase-structure rules. In this case, the phrase marker is not known. Every combination of terminal and nonterminal nodes should be tested against the concatenation rules before the possible phrase markers are obtained, which are to be transformed to base phrase markers by the use of inverse transformations.

11.1

Let us consider that the node labels are complete neighborhoods.

The label of the axiom node which dominates the whole sentence to be generated is then the set of sequences of sentences in which the sentence is to be generated. We consider only the syntactic mechanism of sentence generation and do not take the environment into consideration. The normative device is then generous enough to admit sentences without logical or stylistic consideration, and the node label of the axiom is the complete neighborhood whose members are all possible sequences of all possible sentences. The axiom node is expanded as a concatenation of a few nodes labeled by their respective complete neighborhoods. Each node generated may or may not be further expanded. The expansion of a node A labeled by the complete neighborhood x into the concatenation of the two nodes B and C labeled by y and z, respectively, implies that yz be a subset of x, because every context acceptable to the concatenation BC must be acceptable to A. The expansion is terminated when every terminal node is given a lexical item whose complete neighborhood shares at least one elementary neighborhood with the node.

Every symbol in the string to be recognized as an acceptable sentence is first given its complete neighborhood. Concatenation rules are then applied to the string of complete neighborhoods, and the possible phrase markers are to be found. Let $x = C(r)$ and $y = C(s)$ be the complete neighborhoods of the segments r and s, respectively, and let a number of rules describe the relation $uv = w$ between the concatenated complete neighborhoods uv and the complete neighborhood w. A subset of $z = xy$ is obtained by these rules in the form $z \cap w = xy \cap uv$, which is not empty if $xy \cap uv = (x \cap u)(y \cap v) \neq 0$. We expect the complete neighborhood z to be determined when all the applicable rules of the grammar are applied to xy. If z is empty, the strings r and s are not to be concatenated. Otherwise, rs is a possible concatenation of the phrase marker. After repeating this procedure over every possible combination of nodes, we can find the phrase marker of the given string: It is the phrase marker which is dominated by a single node and covers the whole string. If the string is ambiguous, we have the same number of possible alternatives. These phrase markers are then to be changed to their corresponding base phrase markers by means of inverse transformations, which are also described in terms of complete neighborhoods.

11.2

Acceptable strings are also generated by means of phrase markers whose nodes are labeled by sets of strings. A set of strings is defined as a set of tree structures dominated by the node. Generation begins when the axiom is given as the set of all phrase markers that can be generated by phrase structure rules. The axiom node $P(0)$ is expanded as the concatenation of a few sets of strings $P(1)P(2) \ldots P(i) \ldots P(m)$ which is a subset of $P(0)$. Each node $P(i)$ is also a set of strings, and it may or may not be expanded as a concatenation of sets of strings $P(il) \ldots P(ij) \ldots P(in)$ which is again a subset of $P(i)$. Every time a node is replaced by a concatenation of one or more sets, it is replaced by its subset. The choice becomes narrower and narrower. When the expansion process is terminated, each terminal node is assigned a lexical item which is a member of the set on the node.

Transformations take place on this phrase marker and on the transforms generated by the preceding transformations. The phrase marker is checked against the structural description of rules which are given in terms of sets of strings. A rule is applicable to a phrase marker only if the node labels of the phrase marker are the subsets of the corresponding sets in the rule and the lexical items belong to the corresponding sets in the rule.

The phrase marker of a given string can be obtained by the use of concatenation rules prepared in terms of sets of strings. A rule is applicable to a segment of the string if the symbols of the segment are the members of the corresponding sets in the rule. The inverse-transformation rules are also prepared in terms of sets of strings.

12. Complete Neighborhood Representation of Concatenation Rules

We say a set of concatenation rules is complete if it gives the concatenation $z = xy$ of all complete neighborhoods x and y of the language. It is not necessary, however, to list all possible x's or y's. If their use is properly programmed, a small number of rules can be sufficient to cover all possible complete neighborhoods.

We assume that a rule $f(uv;w)$ represents a relation between the concatenated complete neighborhoods uv and another complete neighborhood w. Each rule gives infor-

mation to xy if $x\ (=)\ u$ and $y\ (=)\ v$: $(x \cap u)(y \cap v) = xy \cap uv$, which is a part of $z = xy$.

In order to obtain the given concatenation xy, we determine the set $R(xy)$ of rules applicable to xy. We determine whether or not each rule is applicable to xy by the condition g, so that $f(uv;w) \in R(xy)$ if and only if $g(x;u)$ and $g(y;v)$. The terms w are read out of the rules in $R(xy)$ so that $z = xy$ may be determined. It is obvious that there exist certain restrictions in choosing the type f of rules, the condition g for determining $R(xy)$, and the procedure for finding z. We must specify these three for the grammar to be written.

When the complete neighborhood z is given and its expansion xy is to be found, the set $R(z)$ of applicable rules is determined by the condition $h(z;w)$: $R(z) = \text{set}\ (f(uv;w)\colon h(z;w))$. The situation is a little complicated in this case. We can expect cases in which both $z = x_1y_1$ and $z = x_2y_2$ are true under the condition $x_1 \cap x_2 = 0$ and/or $y_1 \cap y_2 = 0$. Note that this is not the case of formal concatenation of sets $AB \cap CD = (A \cap C)(B \cap D)$: x_1y_1 and x_2y_2 happened to be z because of the syntax of the language being studied. In this case, we write

$$z = x_1y_1\ (+)\ x_2y_2$$

and mean that z is the concatenation of either x_1y_1 or x_2y_2. Finally, we have a set of possible expansions x_iy_i, each such x_iy_i being accompanied by the subset $R(z;i)$ of $R(z)$, such that x_i and y_i are determined by the rules in $R(z;i)$.

In order to observe the property of rules, we assume four simple forms of $f(uv;w)$: $uv = w$, $uv \subseteq w$, $uv \supseteq w$, and $uv\ (=)\ w$.

12.1

Let us see how the rules $f(uv;w)$ are used to obtain the complete neighborhood $z = xy$ which is the concatenation of the given x and y. The applicability condition $g(x;u)$ is assumed to be $x = u$, $x \subseteq u$, $x \supseteq u$, or $x\ (=)\ u$, and, similarly, $g(y;v)$ is also assumed. Then, the condition $g(x;u)$ *and* $g(y;v)$ imposed on each constituent separately can be replaced by the same condition imposed on the whole concatenation:

$$xy = uv \tag{1}$$

if and only if $x = u$ and $y = v$;

$$xy \subseteq uv \tag{2}$$

if and only if $xy \cap uv = (x \cap u)(y \cap v) = xy$, if and only if $x \cap u = x$ and $y \cap v = y$, if and only if $x \subseteq u$ and $y \subseteq v$;

$$xy \supseteq uv \tag{3}$$

if and only if $xy \cap uv = (x \cap u)(y \cap v) = uv$, if and only if $u = x \cap u$ and $v = y \cap v$, if and only if $x \supseteq u$ and $y \supseteq v$;

$$xy\ (=)\ uv \tag{4}$$

if and only if $xy \cap uv = (x \cap u)(y \cap v) \neq 0$, if and only if $x\ (=)\ u$ and $y\ (=)\ v$.

The rule $f(uv;w)$ is applicable to xy if $g(xy;uv)$ holds. Therefore, the concatenation $z = xy$ has to satisfy $(z = xy)$ *and* $g(xy;uv)$ *and* $f(uv;w)$. There are sixteen combinations of $g(xy;uv)$ and $f(uv;w)$ as shown in table 2.

TABLE 2

THE SIXTEEN COMBINATIONS OF $g(xy;uv)$ AND $f(uv;w)$ CLASSIFIED ACCORDING TO THE RELATION $z\ R\ w$

$(z = xy)$ and $g(xy;uv)$ and $f(uv;w)$	$z\ R\ w$
$z = xy = uv = w$	$z = w$
$z = xy = uv \subseteq w$	$z \subseteq w$
$z = xy = uv \supseteq w$	$z \supseteq w$
$z = xy = uv\ (=)\ w$	$z\ (=)\ w$
$z = xy \subseteq uv = w$	$z \subseteq w$
$z = xy \subseteq uv \subseteq w$	$z \subseteq w$
$z = xy \subseteq uv \supseteq w$	..
$z = xy \subseteq uv\ (=)\ w$	...
$z = xy \supseteq uv = w$	$z \supseteq w$
$z = xy \supseteq uv \subseteq w$	$z\ (=)\ w$
$z = xy \supseteq uv \supseteq w$	$z \supseteq w$
$z = xy \supseteq uv\ (=)\ w$	$z\ (=)\ w$
$z = xy\ (=)\ uv = w$	$z\ (=)\ w$
$z = xy\ (=)\ uv \subseteq w$	$z\ (=)\ w$
$z = xy\ (=)\ uv \supseteq w$	.
$z = xy\ (=)\ uv\ (=)\ w$	...

Let us classify the cases according to the relation $z\ R\ w$, and see how the desired complete neighborhood z can be found.

12.1.1.—We have $z = w$ only if $xy = uv$ and $uv = w$, and we see that z is found by virtue of only one rule. However, the condition $xy = uv$ is too strict to be practical: The number of rules to be prepared for all combinations of x and y is astronomical, and it is very hard to assure the equality $uv = w$.

12.1.2.—We have $z \subseteq w$ in the following cases:

$$z = xy = uv \subseteq w\,,$$

$$z = xy \subseteq uv = w\,,$$

or

$$z = xy \subseteq uv \subseteq w\,.$$

In short, these are the cases in which we have $xy \subseteq uv$ for $g(xy;uv)$ and $uv \subseteq w$ for $f(uv;w)$. For all rules applicable to xy, we have

$$z = xy \subseteq u_kv_k \subseteq w_k\,,$$

$$z = xy \subseteq u_iv_i \subseteq w_i\,,$$

$$\ldots\,,$$

$$z = xy \subseteq u_kv_k \subseteq w_k\,,$$

and hence $z \subseteq w_h \cap w_i \cap \ldots \cap w_k$. If we have enough rules applicable to xy, we can expect to have $z = w_h \cap w_i \cap \ldots \cap w_k$.

12.1.3.—The relation $z\ R\ w$ is $z \supseteq w$, if

$$z = xy = uv \supseteq w\,,$$

$$z = xy \supseteq uv = w\,,$$

or

$$z = xy \supseteq uv \supseteq w\,.$$

That is, $z \supseteq w$ if we have $xy \supseteq uv$ for $g(xy;uv)$ and $uv \supseteq w$ for $f(uv;w)$. For all rules applicable to xy, we have

$$z \supseteq w_h\,,$$

$$z \supseteq w_i\,,$$

$$\ldots\,,$$

$$z \supseteq w_k\,.$$

Then

$$z \supseteq w_h \cup w_i \cup \ldots \cup w_k\,.$$

We can expect to have $z = w_h \cup w_i \cup \ldots \cup w_k$ if we have enough rules applicable to xy.

We know that the concatenation xy of complete neighborhoods is broken down into the concatenations $e(i)e(j)$ of elementary neighborhoods and that each $e(i)e(j)$ is represented as a union of elementary neighborhoods. If

$$x \supseteq e(1)$$

$$y \supseteq e(2) \cup e(3) \cup e(4)\,,$$

for instance, and if we have the rules

$$e(1)e(2) \supseteq e(5) \cup e(6)\,,$$

$$e(1)e(3) \supseteq e(5)\,,$$

and

$$e(1)e(4) \supseteq e(6)\,,$$

then

$$xy \supseteq e(5) \cup e(6)\,.$$

These rules can be broken down as

$$e(1)e(2) \supseteq e(5)\,,$$

$$e(1)e(2) \supseteq e(6)\,,$$

$$e(1)e(3) \supseteq e(5)\,,$$

$$e(1)e(4) \supseteq e(6)\,,$$

and then contracted as

$$e(1)(e(2)\ (+)\ e(3)) \supseteq e(5)\,,$$

$$e(1)(e(2)\ (+)\ e(4)) \supseteq e(6)\,,$$

where the symbol (+) means an alternative choice.

The number of elementary neighborhoods increases rapidly as the linguistic analysis becomes more precise, and hence a grammar prepared in terms of elementary neighborhoods comprises a great number of rules. However, this kind of representation is preferred when a particular technique is available [6].

If the rules are given in terms of elementary neighborhoods in the form $e(i)e(j) = w(i,j)$, then, by virtue of the expansion coefficients $x(i)$ and $y(j)$, we have

$$x \cap u = x \cap e(i) = x(i)e(i)\,,$$

$$y \cap v = y \cap e(j) = y(j)e(j)\,,$$

$$(x \cap u)(y \cap v) = x(i)y(j)e(i)e(j)\,.$$

Therefore, the rule $e(i)e(j) = w(i,j)$ is applicable to xy if $x(i) = y(j) = 1$. The result $z = xy$ is obtained as the union of all $w(i,j)$'s of the applicable rules:

$$z = \bigcup w = \bigcup x(i)y(j)w(i,j)\,.$$

12.1.4.—The relation $z\ R\ w$ takes the form $z\ (=)\ w$, if

$$z = xy = uv\ (=)\ w\,,$$

$$z = xy \supseteq uv \subseteq w\,,$$

$$z = xy \supseteq uv\ (=)\ w\,,$$

$$z = xy\ (=)\ uv = w\,,$$

or

$$z = xy\ (=)\ uv \subseteq w\,.$$

The rules say only that the intersection of z and w is not empty. We cannot obtain the complete neighborhood z unless some other information is available.

Rules can be prepared and used more freely if we have some means of finding the intersection and the union of given complete neighborhoods. This operation is possible if we have a coding system by which every possible complete neighborhood is given a name or a code and the intersection and the union of two complete neighborhoods are obtained by definite operation on the coded form, as we do with the numbers: The numbers whose names are encoded as a string of a few digits of figures or as a combination of a few such codes can undergo an arithmetic operation, and the name of the resultant number is generated in the coded form. In the following scheme [7] a complete neighborhood is represented by a code consisting of a number of digits, and each digit is checked, modified, and transferred independently of others. Suppose x and y are given, and their concatenation $z = xy$ is required. Both x and y can be syntactically ambiguous, and their ambiguity is to be reduced in the course of finding z. Initially, z is assumed to be the set of all possible contexts. Then x, y, and z are transferred to a temporary storage space (x_1,y_1,z_1). A rule is applicable if $x\ (=)\ u, y\ (=)\ v$ and $z\ (=)\ w$, and the set (x_1,y_1,z_1) is modified every time a rule is applied. If $x_1\ (=)\ u$, $y_1\ (=)\ v$, $z_1 \cap w = 0$, then the rule is not applied to this set, and another set (x_2,y_2,z_2) is stored in another storage space as another possible result. All the applicable rules are applied one after another to all the possible sets of (x_i,y_i,z_i).

12.2

We want to find possible expansions of a complete neighborhood z as a concatenation of the complete neighborhoods x and y. The rules are applied to z if they satisfy

the condition $h(z;w)$, and the rules describe the relation $f(w;uv)$ between the complete neighborhoods w and uv: For the rules applicable to z, we have $(xy = z)$ *and* $h(z;w)$ *and* $f(w;uv)$. In order to see the property of the rules, we assume $h(z;w)$ and $f(w;uv)$ as shown in table 3.

TABLE 3

THE PROPERTY OF THE RULES APPLICABLE TO z UNDER THE ASSUMPTION OF $h(z;w)$ AND $f(w;uv)$

$(xy = z)$ and $h(z;w)$ and $f(w;uv)$	$xy\ R\ uv$
$xy = z = w = uv$	$xy = uv$
$xy = z = w \subseteq uv$	$xy \subseteq uv$
$xy = z = w \supseteq uv$	$xy \supseteq uv$
$xy = z = w\ (=)\ uv$	$xy\ (=)\ uv$
$xy = z \subseteq w = uv$	$xy \subseteq uv$
$xy = z \subseteq w \subseteq uv$	$xy \subseteq uv$
$xy = z \subseteq w \supseteq uv$	...
$xy = z \subseteq w\ (=)\ uv$	...
$xy = z \supseteq w = uv$	$xy \supseteq uv$
$xy = z \supseteq w \subseteq uv$	$xy\ (=)\ uv$
$xy = z \supseteq w \supseteq uv$	$xy \supseteq uv$
$xy = z \supseteq w\ (=)\ uv$	$xy\ (=)\ uv$
$xy = z\ (=)\ w = uv$	$xy\ (=)\ uv$
$xy = z\ (=)\ w \subseteq uv$	$xy\ (=)\ uv$
$xy = z\ (=)\ w \supseteq uv$	...
$xy = z\ (=)\ w\ (=)\ uv$	...

12.2.1.—$xy = uv$ if and only if $xy = z = w = uv$. If $z = w$ for more than one rule, we have

$$xy = z = w = u_h v_h\,,$$
$$xy = z = w = u_i v_i\,,$$
$$\ldots\,,$$
$$xy = z = w = u_k v_k\,,$$

and hence

$$xy = u_h v_h\ (+)\ u_i v_i\ (+)\ \ldots\ (+)\ u_k v_k\,.$$

That is, $x = u_h$, $y = v_h$, or

$$x = u_i\,, \quad y = v_i\,,$$
$$\ldots\,,$$

or

$$x = u_k\,, \quad y = v_k\,.$$

12.2.2.—$xy \subseteq uv$ if

$$xy = z = w \subseteq uv\,,$$
$$xy = z \subseteq w = uv$$

or

$$xy = z \subseteq w \subseteq uv\,.$$

In short, $xy \subseteq uv$ if and only if $z \subseteq w$ and $w \subseteq uv$. Let the rules $w_i \subseteq u_i v_i$, $i = 1, 2, \ldots, n$ satisfy the condition $z \subseteq w_i$. Then, we have

$$xy = z \subseteq \bigcap w_i \subseteq \bigcap u_i v_i\,.$$

We cannot conclude, however, $x \subseteq \bigcap u_i$ or $y \subseteq \bigcap v_i$. The following procedure consists of two different operations. Take the rules $w_i \subseteq u_i v_i$ and $w_j \subseteq u_j v_j$, and obtain $(u_i \cap u_j)\ (v_i \cap v_j)$, which requires an operational device for finding the intersection of given complete neighborhoods.

Operation 1.—If $z \subseteq w_i \subseteq u_i v_i$, $z \subseteq w_j \subseteq u_j v_j$, and $z \subseteq (u_i \cap u_j)(v_i \cap v_j)$, then $(x \subseteq u_i \cap u_j,\ y \subseteq v_i \cap v_j)$ is a better pair of candidates than $(x \subseteq u_i,\ y \subseteq v_i)$ and $(x \subseteq u_j,\ y \subseteq v_j)$. If we have another rule $w_k \subseteq u_k v_k$, such that $z \subseteq (u_i \cap u_j \cap u_k)(v_i \cap v_j \cap v_k)$, then this gives a still better pair of candidates: $x \subseteq u_i \cap u_j \cap u_k$, $y \subseteq v_i \cap v_j \cap v_k$. By repeating the same operation with other rules, we obtain

$$x = \bigcap u_i\,, \quad y = \bigcap v_i\,, \quad xy = z\,.$$

Operation 2.—If $z \subseteq w_h \subseteq u_h v_h$, $z \subseteq w_k \subseteq u_k v_k$, but not $z \subseteq (u_h \cap u_k)(v_h \cap v_k)$, then we have $xy = z \subseteq u_h v_h\ (+)\ u_k v_k$. That is, $(x \subseteq u_h,\ y \subseteq v_h)$ or $(x \subseteq u_k,\ y \subseteq v_k)$. Applying Operation 1 on these pairs separately, we obtain the pairs $(x = \bigcap u_h, y = \bigcap v_h)$ and $(x = \bigcap u_k,\ y = \bigcap v_k)$. Assume

$$z \subseteq (\bigcap u_{h(1)})(\bigcap v_{h(1)}) \text{ for rules in } R(z;1)\,,$$
$$z \subseteq (\bigcap u_{h(2)})(\bigcap v_{h(2)}) \text{ for rules in } R(z;2)\,,$$
$$\ldots\,,$$
$$z \subseteq (\bigcap u_{h(m)})(\bigcap v_{h(m)}) \text{ for rules in } R(z;m)\,,$$

and we have

$$z \subseteq (\bigcap u_{h(1)})(\bigcap v_{h(1)})\,,$$
$$(+)\ (\bigcap u_{h(2)})(\bigcap v_{h(2)})\,,$$
$$\ldots\,,$$
$$(+)(\bigcap u_{h(m)})(\bigcap v_{h(m)})\,.$$

That is,

$$x \subseteq \bigcap u_{h(1)}\,, \quad y \subseteq \bigcap v_{h(1)}\,,$$

or

$$x \subseteq \bigcap u_{h(2)}\,, \quad y \subseteq \bigcap v_{h(2)}\,,$$
$$\ldots\,,$$

or

$$x \subseteq \bigcap u_{h(m)}\,, \quad y \subseteq \bigcap v_{h(m)}\,.$$

Let the rules be prepared in terms of elementary neighborhoods in the form $w \subseteq e(i)e(j)$, and let xy be a possible expansion of z: $xy = z \subseteq w \subseteq e(i)e(j)$, or $xy \subseteq e(i)e(j)$. That is, $x \subseteq e(i)$ and $y \subseteq e(j)$. If $xy \neq 0$, then $e(i) \subseteq x$ and $e(j) \subseteq y$. Therefore, $x = e(i)$ and $y = e(j)$ is a possible expansion of z:

$$z = e(i(l))e(j(l))\ (+)\ \ldots\ (+)\ e(i(m))e(j(m))\,.$$

12.2.3.—$xy \supseteq uv$

if

$$xy = z = w \supseteq uv\,,$$
$$xy = z \supseteq w = uv\,,$$

or

$$xy = z \supseteq w \supseteq uv .$$

In short, $xy \supseteq uv$ if $h(z;w)$ and $f(w;uv)$ are such that $z \supseteq w \supseteq uv$, or $R(z) = \text{set } (w \supseteq uv: z \supseteq w)$. For all rules in $R(z)$, we have

$$z \supseteq w_i \supseteq u_i v_i , \quad i = 1, 2, \ldots, n ,$$

and hence

$$z \supseteq \bigcup w_i \supseteq u_1v_1 \cup u_2v_2 \cup \ldots \cup u_nv_n$$
$$= \bigcup u_iv_i , \quad i = 1, 2, \ldots, n .$$

Note, however, that $\bigcup u_iv_i$ is not always rewritten as $(\bigcup u_i)(\bigcup v_i)$, and we cannot always say $x \supseteq \bigcup u_i$ or $y \supseteq \bigcup v_j$. Instead, we have

$$xy \supseteq u_1v_1 \,(+)\, u_2v_2 \,(+)\, \ldots \,(+)\, u_nv_n .$$

If $z \supseteq (u_i \cup u_j)(v_i \cup v_j)$, then the terms $u_iv_i \,(+)\, u_jv_j$ are rewritten as

$$(u_i \cup u_j)(v_i \cup v_j) ,$$

or

$$x \supseteq u_i \cup u_j , \quad y \supseteq v_i \cup v_j .$$

If the set $R(z)$ has enough rules, we can expect to have x and y.

12.2.4.—$xy \,(=)\, uv$ if $xy = z = w \,(=)\, uv$, or $xy = z \supseteq w \subseteq uv$, or $xy = z \supseteq w \,(=)\, uv$, or $xy = z \,(=)\, w = uv$, or $xy = z \,(=)\, w \subseteq uv$.

Generally speaking, such x and y can be found by successive approximation: We take x' and y' as a pair of candidates and obtain $z' = x'y'$. The difference between z and z' is further and further diminished by adjusting x' and y':

$$(x' \cap x)(y' \cap y) = x'y' \cap xy = x' \cap z .$$

To do this, we have to know how to obtain the intersection and the union of given sets.

13. Distribution Class Representation of Concatenation Rules

Possible concatenations of a language can be formulated as concatenated sets of strings. Let $R = \text{set } (r: g(r))$ and $S = \text{set } (s: g(s))$ be sets of strings satisfying the conditions $g(r)$ and $g(s)$, respectively, and let their concatenation have the property $h(rs)$, so that $rs \in T = \text{set } (t: h(t))$. We consider the concatenation rules of the form $RS \subseteq T$, which reads: if $r \in R$ and $s \in S$, then $rs \in T$. The point of this representation is that if $r \in R_h \cap R_i \cap \ldots \cap R_k$ and $s \in S_h \cap S_i \ldots \cap S_k$, then the same number of rules are applicable to rs, and they give $rs \in T_h \cap T_i \cap \ldots \cap T_k = T'$. The intersection T' has fewer elements, and, if the rules are precise enough, the character of the strings in it is determined as precisely as required. Of course, this is not to be done by listing all members of the sets. Each set in the rules is represented by a code. Every entry of the lexicon has a code, and it can be determined whether or not this code belongs to a certain set. A similar code is to be generated and attached to rs to indicate that it belongs to the set T.

Practically, it is convenient to classify the strings according to their complete neighborhoods:

$$R = \text{set } (r: g(C(r);u)) = R(u) ,$$
$$S = \text{set } (s: g(C(s);v)) = S(v) ,$$
$$T = \text{set } (t: h(C(t);w)) = T(w) .$$

A grammar of concatenation will be a set of rules of the form $R(u)S(v) \subseteq T(w)$ with a relation $f(uv;w)$, and they can be described in a number of different ways according to the choice of $R(u)S(v)$, (Tw), and $f(uv;w)$. In order to observe the principle, we simplify the situation by making use of the distribution classes J, I, H, and G, and by assuming the relation $f(uv;w)$ as $uv = w$, $uv \subseteq w$, $uv \supseteq w$, or $uv \,(=)\, w$. The type of $T(w)$ is chosen so as to describe the language adequately.

13.1. J *Representation*

Set $R(u) = J(u)$, $S(v) = J(v)$. This type of grammar is not practical, because the rules must cover all real distribution classes J of the language. This condition corresponds to the complete-neighborhood representation of rules $f(uv;w)$ applicable to xy only if $u = x$ and $v = y$.

13.2. I *Representation*

Set $R(u) = I(u)$, $S(v) = I(v)$. If $r \in I(u)$ and $s \in I(v)$, then $rs \in I(u)I(v) \subseteq I(uv)$. Assume $uv \subseteq w$, and we have $rs \in I(uv) \subseteq I(w)$, because $t \in I(uv)$ if and only if $C(t) \subseteq uv$, where $uv \subseteq w$; then $C(t) \subseteq w$ if and only if $t \in I(w)$. If a number of rules are applicable to rs, then

$$rs \in I(u_h)I(v_h) \subseteq I(u_hv_h) \subseteq I(w_h) ,$$
$$rs \in I(u_i)I(v_i) \subseteq I(u_iv_i) \subseteq I(w_i) ,$$
$$\ldots ,$$
$$rs \in I(u_k)I(v_k) \subseteq I(u_kv_k) \subseteq I(w_k) ,$$

then

$$rs \in I(w_h) \cap I(w_i) \cap \ldots \cap I(w_k)$$
$$= I(w_h \cap w_i \cap \ldots \cap w_k) .$$

Or,

$$C(rs) \subseteq w_h \cap w_i \cap \ldots \cap w_k .$$

The situation is similar to the case of the complete-neighborhood representation $C(r)C(s) = xy \subseteq uv \subseteq w$.

13.3. H *Representation*

Set $R(u) = H(u)$ and $S(v) = H(v)$. If $r \in H(u)$ and $s \in H(v)$, then $rs \in H(u)H(v) \subseteq H(uv)$. If $uv \supseteq w$, then $rs \in H(u)H(v) \subseteq H(uv) \subseteq H(w)$, because $t \in H(uv)$ if and only if $C(t) \supseteq uv$, where $uv \supseteq w$; then $C(t) \supseteq w$ if and only if $t \in H(w)$. We set $T(w) = H(w)$ to have the

rules of the form $H(u)H(v) \subseteq H(w)$. If a number of rules are applicable and

$$rs \in H(u_h)H(v_h) \subseteq H(w_h)\,,$$
$$rs \in H(u_i)H(v_i) \subseteq H(w_i)\,,$$
$$\ldots\,,$$
$$rs \in H(u_k)H(v_k) \subseteq H(w_k)\,,$$

then

$$rs \in H(w_h) \cap H(w_i) \cap \ldots \cap H(w_k) = H(w_h \cup w_i \cup \ldots \cup w_k)\,,$$

then

$$C(rs) \supseteq w_h \cup w_i \cup \ldots \cup w_k\,.$$

The rules of this type are equivalent to the rules of complete neighborhoods of the type $xy \supseteq uv \supseteq w$, although they are coded as sets of strings.

13.4. G Representation

Put $R(u) = G(u)$ and $S(v) = G(v)$, and let r and s be in $G(u)$ and $G(v)$, respectively. If $uv = w$, then $rs \in G(u)G(v) \subseteq G(uv)$ and $C(rs)\ (=)\ uv = w$. If $uv \subseteq w$, then $rs \in G(u)G(v) \subseteq G(uv)$ and $C(rs)\ (=)\ uv \subseteq w$. If $uv \supseteq w$, then $rs \in G(u)G(v) \subseteq G(uv)$ and $C(rs)\ (=)\ uv \supseteq w$. If $uv\ (=)\ w$, then $rs \in G(u)G(v) \subseteq G(uv)$, and $C(rs)\ (=)\ uv\ (=)\ w$. Even if a few rules are applicable to rs in these cases, we have no simple means of finding $C(rs)$ from w's. We cannot specify a smaller set which adequately indicates the property of rs, unless more specific information is available.

Suppose, however, u and v are elementary. If $C(r)\ (=)\ u$ and $C(s)\ (=)\ v$, then $C(r) \supseteq u$ and $C(s) \supseteq v$. That is, $r \in G(u) = H(u)$ and $s \in G(v) = H(v)$.

Assume $C(r)$ and $C(s)$ are elementary. If $C(r)\ (=)\ u$ and $C(s)\ (=)\ v$, then $C(r) \subseteq u$ and $C(s) \subseteq v$. That is, $r \in I(u)$ and $s \in I(v)$.

14. Some Remarks on Transformations

14.1

Let us assume another function of our normative device. We give it a pair $r = (r', r'')$ of acceptable strings $r' = r'(1)r'(2) \ldots r'(i') \ldots r'(m')$ and $r'' = r''(1)r''(2) \ldots r''(i'') \ldots r''(m'')$. We set $m'' = 0$ if r'' is absent. We then give it another acceptable string $s = s(1)s(2) \ldots s(j) \ldots s(n)$, and ask it whether or not the string s as an expression is true if both r' and r'' are true. If the device says "yes," we say that the string s is generated from r by a transformation. We call r the original string and s its transform. If the device says "no," no such transformation exists. Conversely, we ask the device whether or not r' and r'' are true if s is true. If it says "yes," we say that an inverse transformation exists.

A transformation or an inverse transformation is called singulary if r'' in r is absent, and generalized if both r' and r'' are present. If this is an embedding transformation, r' and r'' are called matrix and constituent strings, respectively. We can find many cases in which the device would say "yes" for transformation but "no" for inverse transformation. Some information is supposed to have been lost in generating the string s, which cannot be retrieved unless appropriate, possibly nonlinguistic, information is supplied. This situation is beyond the scope of syntactics. If we find r and s such that r is true if and only if s is true, then we say r and s are equivalent to each other and write

$$r \text{ eqv } s\,.$$

Obviously, this equivalence is reflexive, symmetric, and transitive. Let us call a transformation that changes a string into an equivalent string an equivalence transformation. If we have a grammar consisting of equivalence transformations only, it provides us with a straightforward means for both analysis and synthesis of texts. Let us confine ourselves for a moment to equivalence transformations in order to simplify the discussion. A generalized transformation transforms a pair $r = (r', r'')$ of strings into one string s. The inverse transformation by the same rule dissolves a string s into a pair of strings (r', r''). Then, r' or r'' is regarded as an s, and, if we find an appropriate rule, it is again dissolved into two acceptable strings. By repeating the same, we have a number of equivalence relations which can be arranged as a tree:

$$s \text{ eqv } (r(1), r(2))\,,$$
$$r(1) \text{ eqv } (r(11), r(12))\,,$$
$$r(2) \text{ eqv } (r(21), r(22))\,,$$
$$r(12) \text{ eqv } (r(121), r(122))\,,$$
$$\ldots\,.$$

Throughout this process, the strings are expected to become shorter and simpler, because the equivalent information is expressed by means of more separate strings.

If an acceptable string t can no longer be dissolved into two or more acceptable strings, we call t a terminal or an atomic acceptable string. It may still be possible to transform an atomic string to another atomic string by means of a singulary transformation. We have different atomic strings which are equivalent to each other. We pick up one of them and call it a kernel sentence.

14.2

Transformation is a peculiar operation which changes the structure of phrase markers. If a string is given and its phrase marker is not given explicitly, the phrase marker is to be determined by virtue of concatenation rules. We say a string r is transformed to s and mean that the phrase marker of r is transformed to the phrase marker of s. If the string is ambiguous, the transformation rules are applied only to those phrase markers which meet the structural description.

The transformation is not always meaning preserving or truth-value preserving, or even acceptability preserving [8]. What if it is not meaning preserving, for instance? The features of the original string are modified, and we

shall have a transform which has no longer the same meaning as the original string. It is quite all right if exactly the intended feature is given to the string, so that the final meaning is determined by the series of transformations applied. Unfortunately, the meaning is too often distorted in an unexpected way.

What, then, is a transformation? What remains invariant throughout a transformation? We need not care what it may be as far as the formal properties are concerned. A transformation T is defined as an ordered pair (p,q) of the phrase marker p of the original string and the phrase marker q of the transform. The operation is called an inverse transformation if it changes q back to p:

$$T(p) = q\,, \qquad T^{-1}(q) = p\,.$$

This is not, however, exactly the way in which we understand a transformation. A transformation—passivization, for instance—changes many distinct strings to their corresponding passive counterparts. A transformation is, therefore, a set of ordered pairs of corresponding phrase markers:

$$T = \text{set } ((p_i,q_i)\colon i = 1, 2, \ldots)\,.$$

Given a set P of p_i's, we have the set Q of the corresponding q_i's:

$$\begin{aligned} Q &= \text{set } (q_i\colon (p_i,q_i) \text{ in } T, \text{ and } p_i \text{ in } P) \\ &= T''P \end{aligned}$$

which is the image of the set P under the relation T.

Let T be a transformation such that

$$T = \text{set } ((p,q), (p,q'), (p,q''), \ldots)\,.$$

The relation T includes such pairs as (p,q), (p,q'), (p,q''), and it gives three distinct phrase markers q, q', and q'' as the possible transforms of a single phrase marker p:

$$Q = T'' \text{ set } (p) = \text{set } (q, q', q'')\,.$$

If there are no such pairs as (p,q') or (p,q''), we have the unique transform q of p, and write:

$$q = T(p)\,.$$

If this is true for all p's in P, we write

$$Q = T(P)\,.$$

Suppose we have two transformations:

$$T' = \text{set } ((p_1,q_1), (p_2,q_2), \ldots)\,,$$

and

$$T'' = \text{set } ((r_1,s_1), (r_2,s_2), \ldots)\,.$$

We can define one transformation T which changes p_1 to q_1, p_2 to $q_2, \ldots$, r_1 to s_1, r_2 to $s_2, \ldots$:

$$\begin{aligned} T &= \text{set } ((p_1,q_1), (p_2,q_2), \ldots, (r_1,s_1), (r_2,s_2), \ldots) \\ &= T' \cup T''\,. \end{aligned}$$

We see that two distinct transformations can be brought together to be regarded as one transformation, and some transformations established separately can be included in a more general transformation. This is not always the case in practice. Given a set of phrase markers, we consider, for instance, that the phrase marker p becomes r by passivization T(pass) and it becomes s by nominalization T(nom). The union T of T(pass) and T(nom) does not give the sets of passivized and nominalized forms separately.

In practice, a transformation rule is not given as a set whose elements are listed explicitly. Instead, it is provided with the information to determine for every string whether or not the rule can be or must be applied. Two transformation rules prepared separately can be united only if a common description of applicability is prepared and the features of the transforms can be properly specified.

14.3

Sometimes it is considered linguistically more reasonable to assume that a string is not acceptable but that its transform is an acceptable string or a constituent of an acceptable string. In other cases, a string may be an acceptable string and its transform may not be an acceptable string or a constituent thereof. We can prepare the rules in such a way that a sequence of obligatory transformations is contracted to a single rule. This seems formally simpler and more consistent. However, it will result in a more entangled system of grammar. We admit some such strings as potentially acceptable and indicate this with a marker. This convention is sometimes useful not merely as a technique but also as a consistent and more plausible derivation of acceptable strings. It is known that some strings of a Chinese dialect marked potentially acceptable for the derivation of apparently inconsistent strings are quite acceptable in another dialect [9]. We say a string is acceptable if it is really or potentially acceptable. A phrase marker is acceptable if it underlies an acceptable string or an acceptable phrase marker.

14.4

Let A be a node of a phrase marker p. The node A is called the root of p if it dominates the whole structure of p. Let B be the root of another phrase marker q. We say the roots A and B belong to the identical part of p and q if A and B have the same label. Let A immediately dominate the nodes A_1 and A_2 and only these, and let B dominate the nodes B_1 and B_2: $A(A_1,A_2)$ and $B(B_1,B_2)$. We say the phrase markers p and q are identical up to the nodes A_1 and A_2, or up to B_1 and B_2, if and only if the nodes A_1 and B_1 have the same label, A_2 and B_2 have the same label, and A_1 and A_2 are arranged in the same order as B_1 and B_2. Every node of p and q is compared successively from the roots in this way. If they are identical up to their terminal nodes of lexical items, they represent the same syntactic interpretation of linear strings r and s which are also identical.

By "$p\ I\ q$" we mean the identical part of phrase markers p and q including the root. Given that p_1 becomes q_1 and that p_2 becomes q_2 by a transformation T, we expect that T be applicable to another phrase marker p_3 if (1) the identical part $p_1\ I\ p_2$ is a subtree of p_3 including the root, and (2) the labels on the corresponding nodes of p_3 and $p_1\ I\ p_2$ are the same. We also expect that the transforms q_1 and q_2 share certain features and that the features be found in the transform q_3 of p_3. In other words, $q_1\ I\ q_2$ is a part of q_3 and their corresponding node labels are also the same. If p_3 cannot undergo the transformation T or if q_3 does not share the features mentioned above, we consider that parts of p_1 and p_2 not in $p_1\ I\ p_2$ are not the same but still have features in common which permit the transformation T, while the corresponding part of p_3 does not meet the structural requirement of T. Therefore, it is desirable to describe a transformation in such a way that (1) the structural requirement of T be met by both p_1 and p_2 although they are not the same, and (2) p_3 not meet the structural requirement of T although it contains $p_1\ I\ p_2$ as a part of it. This means that it is not desirable to specify the structure in terms of identity of tree structures and node labels. The description should be prepared in terms of features of trees, so that the phrase markers p_1 and p_2 meet the specification and p_3 fails to meet it. It is also desirable to have a mechanism which permits the features of the original phrase markers to be transferred to their transforms so that a rule can cover a number of distinct phrase markers and still maintain a distinction among the transforms and a correspondence to their original phrase markers.

14.5

Most rules are accompanied by a number of restrictions imposed on the original strings and their transforms as well as some manipulations of strings. These are classified into a few types, and a subroutine must be prepared for each. Some of the restrictions which have been picked up sporadically from the rules for generating Chinese strings are listed below [10]:

1. Certain segments $r(h)$ and $r(i)$ in the original string must or must not share a certain feature in common and/or $r(j)$ must or must not have a certain feature.

2. The segment $r(i)$ of the original string and the segment $s(j)$ of the transform must or must not have the same feature.

3. Some segments in the transform must satisfy a condition similar to (1).

4. Absence and/or presence of particular segments must be checked.

5. Positions of certain segments in the string must be found.

6. A check of the derivational history sometimes decides the recursive application of some rules.

7. The tree structure of the transform is specified.

14.6

A generalized transformation rule consists of terms u and v, where

$$\begin{aligned} u &= (u',u'') \\ &= u(1)u(2)\ \ldots\ u(i)\ \ldots\ u(m)\ , \\ u' &= u'(1)u'(2)\ \ldots\ u'(i')\ \ldots\ u'(m')\ , \\ u'' &= u'(1)u''(2)\ \ldots\ u''(i'')\ \ldots\ u''(m'')\ , \\ m &= m' + m''\ , \\ u &\text{ becomes } v\ , \\ v &= v(1)v(2)\ \ldots\ v(j)\ \ldots\ v(n)\ . \end{aligned}$$

If the rule is not the one for generalized transformation, we set $m'' = 0$. The terms u and v are node labels of the roots dominating the entire phrase markers.

The rule says, if the string r has the feature

$$u = u(1)u(2)\ \ldots\ u(i)\ \ldots\ u(m)\ ,$$

then it is transformed to another string s which has the feature

$$v = v(1)v(2)\ \ldots\ v(j)\ \ldots\ v(n)\ .$$

What are these features? They must be defined on the basis of the answers of our normative device. The program must be consistent with the features defined. Once a program is written and accepted for usage, the program is the definition.

We say that the node labels are complete neighborhoods if the concatenation rules are written in terms of complete neighborhoods. If the concatenation rules are written in terms of distribution classes, we consider the node labels distribution classes.

14.7

We want to associate the node labels of phrase markers with the set of acceptable strings. The definition directly associated with the set of acceptable strings is rarely used in the description of transformations, because the strings of apparently identical structures can be the realizations of quite different logical or semantic content. For this reason, our universe of discourse is the set of all possible phrase markers including deep and surface structures, as well as the intermediate forms under transformational operation.

We confine ourselves to singulary transformations, and the generalized transformations are to be handled by the use of phrase markers in which the constituent phrase markers have been embedded or conjoined. We assume that the recursive elements are introduced to the base phrase markers before the transformations are applied: We do not want to generate such sentences as "the man who ate an apple ate an apple" or "the box is white and white" by syntactic transformations [11].

14.8

Take an acceptable phrase marker and suppose that the list of all its transforms and inverse transforms has been prepared. The list is in the form of a matrix. The names $T_1, T_2, \ldots, T_j, \ldots$ of transformations are given on the upper horizontal side, the names $p_1, p_2, \ldots, p_i, \ldots$ of the phrase markers are given on the left-hand side, and the name p_k of the transform of p_i by T_j is given at position (i,j). If the transformation T_j is not applicable to p_i, the position (i,j) remains blank. Take another phrase marker q and suppose all its transforms and inverse transforms are listed on the positions (i,j) of another matrix where the arrangement of T_j is identical with that of the matrix of p_k. The horizontal rows can be interchanged without losing the linguistic meaning of the matrix. If the matrix of q_k can take the same form as that of p_k, we say that the two matrices belong to the same class. Thus, we have a class N of matrices of the same structure but of different phrase markers. We take the phrase marker p_k at position (i,j),q_k at (i,j), and so on, and consider the set P_k of phrase markers:

$$P(k) = \text{set } (p_k, q_k, r_k, \ldots)$$
$$= \text{set } (p_k\text{: } p_k \text{ at } (i,j) \text{ of matrix in } N) .$$

Now we have a matrix M whose (i,j) elements are $P(k)$. If the matrix M' of one class differs from the matrix M'' of another class only in that M' has the set $P'(k)$ at the (i,j)th position while the corresponding position of M'' is blank, the two classes can be united by assuming a set $P''(k)$ of potentially acceptable phrase markers at position (i,j) of M'' which was blank. However, reckless application of this principle may result in a conclusion that every language has only one class and hence only one matrix M of $P(k)$, and the class symbols are eliminated for the price of marking many phrase markers as potentially acceptable. It is impossible to list all potentially acceptable phrase markers introduced by the above procedure: Again we need a classification of phrase markers so that the potentially acceptable phrase markers will be properly marked.

The matrix M can be represented by a network. Each node is labeled with a set $P(i)$ of phrase markers, and is connected to other nodes of $P(k)$ by the arrows $T(j)$: $P(k)$ is the set of transforms of the phrase markers in $P(i)$. The routes in the network show the sequences of transformations $T(j)$.

14.9

Not all possible transformations need be described in a grammar: Some of them can be realized as a sequence of several others. We break off the transformation routes in the network representation if there exist two or more routes from one node to another, rearrange the routes in a serial order, and then establish the new transformations which connect the top of a broken route to the tail of the preceding one. In the matrix representation, we eliminate the element $P(k)$ at position (i,j) if the transformation of $P(i)$ to $P(k)$ by $T(j)$ is to be prohibited, and establish a new column j' with the element $P(k')$ at i'th row if the tail $P(i')$ of a route segment is to be connected to the top $P(k')$ of another route segment by the transformation $T(j')$. A choice of transformation route means a choice of grammar. What is the principle of choosing a grammar?

14.9.1.—Simplicity of description can be a criterion. Some people prefer to apply a series of deformational operations to realize a transformation. The possible operations are limited to a few elementary ones for changing the structure of phrase markers. One may arrange the sets $P(k)$ of phrase markers in such a way that one can go over to another set by applying a sequence of elementary operations and cover all the sets in the network. Two distinct possible routes can be compared with respect to the number of elementary operations or the number of sets of potentially acceptable phrase markers, so that one may prefer one route to the other.

14.9.2.—Two or more routes from one set $P(k)$ to another set $P(k')$ were reduced to one by the preceding operation. We may still have a loop which comes back to the same node from which we started. Even the simplest sentences have quite a few alternatives of the same meaning. We can also think of many truth-value-preserving transformations which establish correct relations among the sentences [12]. We have no grounds for regarding any one of these transformations as the most fundamental and deriving the others from it.

It is possible to have a phrase marker and derive many sentences by applying transformations which are not meaning preserving or truth-value preserving. One might say that the sentence of active voice is more fundamental because it can be transformed to its passive form by an unambiguously formulated rule and the actor can be deleted if so desired. On no poorer ground, one might say that the passive sentence is more fundamental because it does not need to have two noun phrases explicitly, because, if necessary, one can be added at a later step of generation. Why should a sentence be passivized before it is nominalized or nominalized before it is passivized? The choice on the basis of economy seems only to be technical, subjective, or arbitrary from the purely syntactic point of view.

14.9.3.—The synthesis of a meaningful sentence or the encoding of the thought to be conveyed is realized only if the derivation is directed by information sufficient to specify the choice of lexical items, their dependence on each other, the syntactic and stylistic features of the sentence, and so on. We do not want to generate such sentences as "Bill asked, 'Did Jack kill the pig?' and John replied, 'Yes, it was'" [11]. One takes a deep structure, applies a transformation, keeps note of the change of meaning and the features of the phrase marker, checks the phrase marker against the specification of the desired meaning and features, and repeats the same procedure for the generation of the desired sentence. To perform this operation successfully, one must know which path to take on every branching point of the transformation routes. A chart must be available which shows the change of mean-

ing, modification of syntactic and stylistic features, etc., along all possible routes the phrase marker in question may take.

This is not exactly the way we use language. We do not remodel a sentence repeatedly until we come upon the desired form, except when we have made a wrong choice of words. We know what we want to mean, consciously or unconsciously, and we look for the proper way of expressing it. The whole sentence is synthesized according to what is to be expressed if it is not too complex. A part of a long sentence is generated first, and the remainder is generated similarly; the syntactic and stylistic features of the latter part are affected by the part already generated.

14.10

We assume that the fact to be encoded is known to the man who wants to express it, or that it is available to the machine in an appropriate form. One may call this, after stratificationalists, the sememic representation, or, after transformationalists, the deep structure with all the optional transformations specified. At any rate, we suppose that the transformational process of generation starts with this. Given this kind of representation with information enough to specify the sentence to be generated, the following operations are directed toward the final form of the sentence. That is, the process is the realization into the lower and lower strata, or the application of optional and obligatory rules toward the surface structure. The whole system is now oriented in a definite direction, and every step is a mapping of one set of phrase markers into another. The choice of rules at every step of mapping is made by the information already given to the phrase marker whose image is to be obtained.

Let us consider two types of rules for describing transformations: deformational transformations, and realizational transformations.

14.10.1.—We can describe a transformation by means of a sequence of deformational operations. When a phrase marker is given, it is checked to find if it meets the structural description of the transformation to be applied, and, if it does meet, the structure of the phrase marker is changed in the definite way as specified by the rules, and the node labels are also modified if necessary. The set P of original phrase markers is the set of all phrase markers which meet the structural description. In this case, the set P is not defined explicitly, and, therefore, the structural description has to be applicable to all possible phrase markers as the criterion. The node labels have to be defined over the set of all possible phrase markers so that they can always be checked against the criteria during the process of a sequence of transformations.

14.10.2.—Let us say a transformation is realizational if it is a mapping of an explicitly defined set P of phrase markers into the set of transforms. It does not matter if the practical procedure is broken down into a sequence of simpler operations, as in the case of deformational transformations. The set P is the set of all phrase markers to which the transformation T is applicable, and a phrase marker is a member of as many sets if it can undergo more than one transformation. Hence, the node labels are to be defined over each such set P.

15. Complete Neighborhoods and Transformations

Complete neighborhoods are defined on the basis of concatenated strings, and we have to associate them with the labels given to the nodes of phrase markers. Let us see what happens when the node labels are assumed to be complete neighborhoods.

15.1

Let p be an acceptable string, and let $r = r(1)r(2) \ldots r(i) \ldots r(m)$ be a segment of p. The string p is transformed to q by T, and the segment r appears in q as the segment $s = s(1)s(2) \ldots s(j) \ldots s(n)$. Some fragments may have been added and some may have been deleted. Let $x(i) = C(r(i))$ be the complete neighborhood of each fragment $r(i)$ of r. By virtue of the concatenation rules, we have

$$\begin{aligned} x &= C(r(1)r(2) \ldots r(i) \ldots r(m)) \\ &= x(1)x(2) \ldots x(i) \ldots x(m) , \end{aligned}$$

and similarly,

$$\begin{aligned} y &= C(s(1)s(2) \ldots s(j) \ldots s(n)) \\ &= y(1)y(2) \ldots y(j) \ldots y(n) . \end{aligned}$$

A string can be ambiguous and it can be the realization of more than one tree structure. When a particular tree is chosen, the complete neighborhood of a segment is a subset of the complete neighborhood it would have otherwise. Every segment belongs to one and only one distribution class of type J. Therefore, instead of writing the transform of r by T in the form

$$T(r(1) \ldots r(i) \ldots r(m)) = s(1)s(2) \ldots s(j) \ldots s(n) ,$$

we write

$$\begin{aligned} &T(J(x(1)) \ldots J(x(i)) \ldots J(x(m))) \\ &\qquad = J(y(1)) \ldots J(y(j)) \ldots J(y(n)) . \end{aligned}$$

Since all elements in a distribution class of type J have the same complete neighborhood, we rewrite the above as

$$T(x(1) \ldots x(i) \ldots x(m)) = y(1) \ldots y(j) \ldots y(n) .$$

This is rewritten again in the form

$$\begin{aligned} x &= x(1) \ldots x(i) \ldots x(m) , \\ y &= T(x) \\ &= y(1) \ldots y(j) \ldots y(n) . \end{aligned}$$

If we have a complete set of rules which gives the concatenation of complete neighborhoods of the language, we

can find the complete neighborhood x. The transformation takes place when x is changed to y. The string y is to be generated by virtue of the information brought forward from x and the structural requirement of y itself. A transformation is then interpreted as: "The complete neighborhood x of the node dominating the string of complete neighborhoods

$$x(1) \ldots x(i) \ldots x(m)$$

is transformed to another complete neighborhood y of the node dominating the string of complete neighborhoods

$$y(1) \ldots y(j) \ldots y(n)\ ."$$

This interpretation, however, suggests a few problems.

We know that

$$J(x(1)) \ldots J(x(m)) \subseteq J(x)\ ,$$
$$J(y(1)) \ldots J(y(n)) \subseteq J(y)\ .$$

The statement "x is transformed to y" is a generalization of the original fact, and this generalization is not always true. Two strings r and r' may replace the same nonterminal node to yield a longer acceptable string. However, when a transformation T is to be applied, the two strings must have the specified structure; thus, the string p with r as a segment in it may be transformed by T, while the string p', which differs from p only in that it has the segment r' in the place of r, may not. The lack of transform of p' by T implies that $C(r) \neq C(r')$.

We may say that the structure mentioned above is a representation of the derivational history. The history can be recorded by listing all the derivational steps the string has experienced. This representation, however, will not always be sufficient, because it is possible that the strings p and q of different histories result in an identical string, and that string is ambiguous since the string from p can undergo a sequence of transformations and the same string from q another, thus the structure itself can not be an absolutely reliable marker. We think it more practical to associate the rules with the features of the phrase markers to which the rules are applied. These features should correspond to the series of transformations applicable to the phrase markers in the case of synthesis and a series of inverse transformations in the case of analysis.

15.2

Because of this complexity involved in natural languages, we encounter a difficulty when we try to prepare a set of syntactic data for practical purposes. We confine ourselves to the set P of strings p to which the transformation T is to be applied and the set Q of transforms q of p by T, and do not consider the other strings which are not in P or Q: $C(r)$ is the set of all contexts of r defined over P, and $D(r)$ is the set of all contexts of r defined over Q, so that

$$C(r) \cup D(r) = E(r)\ ,$$

where $E(r)$ is defined over the union of P and Q. Let

$$p = p(1) \ldots p(i) \ldots p(m)$$

be a string in P, and let

$$q = q(1) \ldots q(j) \ldots q(n)$$
$$= T(p)$$

be the transform of p by T. By modifying the meaning of the notation, we set

$$x(i) = C(p(i)) \text{ over } P\ ,$$
$$y(j) = D(q(j)) \text{ over } Q\ .$$

The requirement that $p(i)$ should appear unchanged as $q(j)$ in q gives

$$p(i) = q(j)\ ,$$
$$C(p(i)) \neq 0\ ,$$
$$D(q(j)) \neq 0\ ;$$

if $p(i)$ does not occur in any q of Q,

$$x(i) = C(p(i)) \text{ over } P$$
$$= E(p(i)) \text{ over } P \cup Q\ ;$$

if $q(j)$ does not occur in any p of P,

$$y(j) = D(q(j)) \text{ over } Q$$
$$= E(q(j)) \text{ over } P \cup Q\ .$$

The relational conditions imposed on the segments $p(i)$ of the original string and $q(j)$ of the transform are indicated in terms of $E(p(i))$ and $E(q(j))$, or by a relation between $C(p(i))$ and $D(q(j))$.

15.3

Let us examine some examples in which the separation of the sets P and Q helps us understand the behavior of lexical items on the syntactic level.

Take the Japanese verbs *ataeru* and *jaru* (pronounced approximately as if they are written in the international phonetic alphabet) with their restricted meaning "to give [something to someone]." The strings

$$A \text{ ga } B \text{ ni } C \text{ o atae-ru} \qquad (1)$$

and

$$A \text{ ga } B \text{ ni } C \text{ o jar-ru} \qquad (2)$$

are acceptable, and they mean

$$A \text{ gives } C \text{ to } B\ , \qquad (3)$$

where *atae-ru* and *jar-ru* are realized as *ataeru* and *jaru*, respectively. Since both (1) and (2) are acceptable, the context

$$A \text{ ga } B \text{ ni } C \text{ o } (\ )\text{-ru} \qquad (4)$$

is acceptable to both *atae* and *jar*. One of the passive forms

$$B \text{ ga } C \text{ o atae-rare-ru} \qquad (5)$$

is acceptable and means

$$B \text{ is given } C \text{ by someone}, \qquad (6)$$

while the corresponding form

$$B \text{ ga } C \text{ o jar-rare-ru} \qquad (7)$$

is not: The context

$$B \text{ ga } C \text{ o ( )-rare-ru} \qquad (8)$$

is acceptable to *atae* but not to *jar*. We cannot describe the applicability of this transformation in terms of the complete neighborhoods defined over the set P of active forms. The distinction is made if *jar* is marked for its forbidden rule and *atae* is unmarked [4], or if the complete neighborhoods are defined over both P and Q.

The above formulation of the difference between *ataeru* and *jaru* is a description of the phenomenon as it appears on the syntactic level, and it is by no means a description of the essential difference. The difference is of a semantic nature: As the result of (1),

$$B \text{ becomes the owner of } C\,; \qquad (9)$$

while as the result of (2),

$$A \text{ no longer has } C \text{ and } C \text{ goes away from } A\,. \qquad (10)$$

In fact, the form (7), which is strange to most native speakers, means, if it is acceptable at all,

$$B \text{ is caused to have } C \text{ taken away}\,. \qquad (11)$$

Another example is seen when the strings

$$\text{John is eager to please} \qquad (12)$$

and

$$\text{John is easy to please} \qquad (13)$$

undergo some of their meaning-preserving transformations. The strings *eager* and *easy* are not the same in their syntactic function because they cannot be interchanged in the same constituent place of

$$\text{it is ( ) to please John} \qquad (14)$$

or

$$\text{John is ( ) to please some one [12]}\,. \qquad (15)$$

In our terminology, the complete neighborhoods of *eager* and *easy* are not the same if they are defined over the union of P and Q, so that the structural descriptions *eager* and *easy* meet are not the same [4].

The difference is the consequence of the semantic content of the two words. This fact should be taken into consideration when the deep structure is constructed on the stratum of semantics, and the following syntactic operations are to be directed in accordance with this information.

15.4

Now, let us consider the complete neighborhoods of nodes of phrase markers. Let $A(0)$ be the node dominating the whole phrase marker p, and let $A(h)$ be a node of p dominating a subtree whose terminal string is $r(h)$. The string $r(h)$ is, then, a segment of the terminal string $r(0)$ of p. We define the context $c(A(h)/p)$ of $A(h)$ in p as the part of p other than the subtree dominated by $A(h)$, and the complete neighborhood $C(A(h))$ of $A(h)$ as the set of all contexts of $A(h)$ in all acceptable phrase markers:

$$C(A(h)) = \text{set } (c(A(h)/p)\colon p \text{ is acceptable})\,.$$

Let $A(h)$ immediately dominate one or more nodes $A(i)$, $A(j)$, etc. Then the terminal strings covered by $A(i)$, $A(j)$, etc., yield, when they are concatenated, the terminal string of $A(h)$. We say, then, the node $A(h)$ is the concatenation of the nodes $A(i)$, $A(j)$, etc. The concatenation of nodes is not the same as the concatenation of linear strings: The same linear concatenation of strings can be generated as the terminal string of a few different trees. The complete neighborhood of a node dominating a subtree is not the same as the complete neighborhood of its terminal string.

We make use of this type of complete neighborhoods to describe transformations of phrase markers. Of course, we can also define them separately over the set P of phrase markers to which a transformation may be applied and over the set Q of transforms:

$$C(A(h)) = \text{set } (c(A(h)/p)\colon p \text{ in } P)\,,$$
$$D(B(h)) = \text{set } (c(B(h)/q)\colon q \text{ in } Q)\,,$$

where $B(h)$ is a node of the transform q of p. In this case, the phrase markers not in P or Q are not taken into consideration.

16. Complete Neighborhood Representation of Transformation Rules

We cannot describe a transformation by listing all pairs of phrase markers in it. We have to describe it in terms of the features of phrase markers which may undergo the transformation. We regard the complete neighborhood given to each node as the representation of the feature of the node, and a phrase marker is characterized by its tree structure and its node labels.

Let p be a phrase marker and let q be the transform of p. Let p consist of the nodes

$$A(i)\,, \quad i = 1, 2, \ldots, M\,,$$

whose complete neighborhoods are

$$x(i)\,, \quad i = 1, 2, \ldots, M\,,$$

and let q consist of the nodes

$$B(j)\,, \quad j = 1, 2, \ldots, N\,,$$

whose complete neighborhoods are

$$y(j), \quad j = 1, 2, \ldots, N.$$

We want to describe the transformations in terms of the features of $A(i)$'s and $B(j)$'s. We suppose the features of $x(i)$'s are described in the form $g(x(i);u(i))$ relative to the complete neighborhoods $u(i)$'s and the features of $y(j)$'s are described in the form $g(y(j);v(j))$ relative to the complete neighborhoods $v(j)$'s so that we have rules of the form

"if $g(x(i);u(i))$ for all nodes $A(i)$,
then $g(y(j);v(j))$ for all nodes $B(j)$"

for transformations and rules of the form

"if $h(y(j);v(j))$ for all nodes $B(j)$,
then $h(x(i);u(i))$ for all nodes $A(i)$"

for inverse transformations.

16.1

Let us assume we have a tree u whose nodes are labeled by the complete neighborhoods

$$u(i), \quad i = 1, 2, \ldots, M,$$

and another tree v with its nodes labeled by

$$v(j), \quad j = 1, 2, \ldots, N.$$

Let p be a phrase marker and let

$$x(i) = C(A(i)), \quad i = 1, 2, \ldots, M'$$

be the complete neighborhoods of its nodes $A(i)$ defined over P. Let q be the transform of p, and let

$$y(j) = D(B(j)), \quad j = 1, 2, \ldots, N'$$

be the complete neighborhoods of its nodes $B(j)$ defined over Q. We consider a set of rules of the form

If u is a subtree of p including the root and if $g(x(i);u(i))$ holds for every pair of corresponding nodes, then the transformation T is applicable to p and its transform q has the tree v as its subtree including the root and $g(y(j);v(j))$ holds for every pair of the corresponding nodes of q and v.

We need not have $v(j)$ in an explicit form, but we can make use of a number of complete neighborhoods $w(j,k)$ such that $f(v(j);w(j,k))$ as we did for the description of concatenation rules.

16.2

Inverse transformations make use of the rules of a similar form:

If v is a subtree of q including the root and if $h(y(j);v(j))$ holds for all corresponding nodes of q and v, then the original phrase marker p of q has the tree u as its subtree including the root and $h(x(i);u(i))$ holds for all corresponding nodes of p and u.

Each of such rules is accompanied by a set of relations $f(u(i);w(i,k))$: We need not have the explicit form of $u(i)$.

Sometimes, the phrase marker q is ambiguous and it can be regarded as a degenerate realization of two or more distinct base phrase markers. We then write p_1 (+) p_2 and mean an alternative choice as we did when we expanded a node into its constituent nodes.

16.3

With all the linguistic difference between the concatenation rules and transformation rules, they exhibit formal similarities when the labels are assumed to be sets of contexts. We would not repeat a similar discussion on the choice of $g(x(i);u(i))$, $f(v(j);w(j,k))$, $h(y(j);v(j))$, $f(u(i);w(i,k))$, or the algorithm for finding $y(j)$ or $x(i)$. The correspondence is shown in table 4. We can also compare concatenation with

TABLE 4

A. The Correspondence between the Concatenation Rules and the Transformation Rules When the Labels Are Assumed to be Sets of Contexts

Concatenation	Transformation
rs becomes t	p becomes q
$xy = C(rs)$ $z = C(t)$	$x(i) = C(A(i))$ for all i $y(j) = D(B(j))$ for all j
$xy = z$	the tree of $x(j)$'s becomes the tree of $y(j)$'s
if $g(x;u)$ and $g(y;v)$, then $g(xy;uv)$	if $g(x(i);u(i))$ for all i, then $g(y(j);v(j))$ for all j
$f(uv;w)$	$f(v(j);w(j,k))$

B. The Same for the Expansion Rules and the Inverse Transformation Rules

Expansion	Inverse Transformation
t becomes rs	q becomes p
$z = C(t)$ $xy = C(rs)$	$y(j) = D(B(j))$ for all j $x(i) = C(A(i))$ for all i
z becomes x_1y_1 (+) x_2y_2	the tree of $y(j)$'s becomes the tree of $x_1(i_1)$'s or the tree of $x_2(i_2)$'s
if $h(z;w)$, then $h(xy;uv)$	if $h(y(j);v(j))$ for all j, then $h(x(i);u(i))$ for all i
$f(w;uv)$	$f(u(i);w(i,k))$

inverse transformation, or expansion with transformation. The parallelism observed in this comparison is also a superficial one.

17. Distribution Class Representation of Transformation Rules

Let us make use of the sets R and S of phrase markers defined over P and Q, respectively, for the purpose of de-

scribing the transformation T without listing all pairs of phrase markers of T. Let r be a phrase marker and s its transform, and consider the rules of the form "if $r \in R$, then $s \in S$." If a number of rules are applicable, we have $r \in R_h \cap R_i \cap \ldots \cap R_k$ and $s \in S_h \cap S_i \cap \ldots \cap S_k$, so that the features of the transform be specified as precisely as desired. Let

$$R = \text{set } (r: g(r;r'))$$

be the set of phrase markers r defined over P, and let

$$S = \text{set } (s: g(s;s'))$$

be the set of phrase markers s defined over Q. We mean by these that (1) r' is a subtree of r including the root, (2) the node labels of r satisfy the same condition as the corresponding node labels of r' do, (3) s' is a subtree of s including the root, and (4) the node labels of s satisfy the same condition as the corresponding node labels of s' do. Let us make use of the following conventions:

$A'(i)$, $i = 1, 2, \ldots, M'$, are the nodes of r' ;

$A(i)$, $i = 1, 2, \ldots, M'$, are the nodes of r corresponding to those of $A'(i)$;

$B'(j)$, $j = 1, 2, \ldots, N'$, are the nodes of s' ;

$B(j)$, $j = 1, 2, \ldots, N'$, are the nodes of s corresponding to those of $B'(j)$;

$L(A)$ means the label of the node A .

The condition g is then reduced to the form

$$g(L(A(i));L(A'(i))) \text{ for all } i$$

and

$$g(L(B(j));L(B'(j))) \text{ for all } j .$$

We further consider that the nodes are characterized by their complete neighborhoods, and put:

$$L(A(i)) = C(A(i)) ,$$

$$L(A'(i)) = u(i) ,$$

$$L(B(j)) = D(B(j)) ,$$

$$L(B'(j)) = v(j) ,$$

$$R = \text{set } (r: g(C(A(i));u(i)) \text{ for all } A(i) \text{ of } r) = R(u) ,$$

$$S = \text{set } (s: g(D(B(j));v(j)) \text{ for all } B(j) \text{ of } s) = S(v) ,$$

where u and v denote the phrase markers whose node labels are $u(i)$'s and $v(j)$'s, respectively. The rules will take the form

$$\text{if } r \text{ is in } R(u), \text{ then } s \text{ is in } S(v) .$$

For the flexibility of description, we can make use of the trees w consisting of the nodes labeled by $w(j)$ under the relation $f(v(j);w(j))$, so that

$$\text{if } r \text{ is in } R(u), \text{ then } s \text{ is in } T(w) ,$$

where the type of the sets

$$T(w) = \text{set } (s: h(D(B(j));w(j)))$$

is to be chosen so as to describe the transformations adequately. The formal properties are parallel to those of the concatenation rules, and the algorithm can be reduced to that of complete-neighborhood representation (see table 5).

TABLE 5

THE PARALLELISM BETWEEN THE FORMAL PROPERTIES OF THE CONCATENATION RULES AND THE TRANSFORMATION RULES

Concatenation	Transformation
$R = \text{set } (r: g(r))$	$R = \text{set } (r: g(r;r'))$
$S = \text{set } (s: g(s))$	...
$R(u) = \text{set } (r: g(C(r);u))$	$R(u) = \text{set } (r: g(C(A(i));u(i)))$
$S(v) = \text{set } (s: g(C(s);v))$	...
if $r \in R(u)$ and $s \in S(v)$, then $rs \in R(u)S(v)$	if $r \in R(u)$, then $s \in S(v)$
$T(w) = \text{set } (t: h(C(t);w))$	$T(w) = \text{set } (s: h(D(B(j));w(j)))$
$f(uv;w)$	$f(v(j);w(j))$

18. Establishment and Representation of Complete Neighborhoods

A syntactic function is called a complete neighborhood if it is defined as the set of all acceptable contexts. We use conventional terms and redefine them as symbols assigned to complete neighborhoods.

18.1

When we establish the complete neighborhoods of a natural language, we regard a few of them as undefined terms and derive the others by hypothetical concatenation rules. Sometimes we have a choice among a few hypothetical rules. We take one of them to define a complete neighborhood and regard the others as the property of the complete neighborhood defined by the former. Thus, we distinguish two kinds of rules: definition rules and property rules. Let $axb = c$ and $xd = f$ be hypothetical rules. If one decides to regard the former as the definition of x, the latter is a property of x. Every time a definition rule is established as a hypothesis, it must be determined whether or not the rule contradicts any other definition rules. No property rules should contradict any other rules. Whenever a contradiction is found, the source of trouble can be found by tracing back the definition rules, and the hypothesis that has given rise to the trouble should be modified.

18.2

It seems adequate, for most natural languages, to admit two complete neighborhoods, nominals and verbals. Many other complete neighborhoods are derived from hypothetical concatenations that can occur in acceptable strings.

The prepositions in many European languages are subclassified according to the case of the nominals they govern, and the nominals according to their case, gender, and number. A rule for yielding prepositional phrases will be stated as follows: A preposition that governs nominals of the case c, followed by a nominal of the case c', of any gender and of any number, results in a prepositional phrase, provided the cases c and c' are the same. As suggested in this example, subclassification and desubclassification are useful in describing syntax. A number of indices are made use of in subclassifying a broadly defined complete neighborhood. The example above will be rewritten, by introducing the indices c for case, g for gender, and n for number, and a coefficient $d(c,c')$, in the form

$$\text{prep}(c)n(c';g;n) = d(c,c')\text{prep-}n\,,$$

where

$$d(c,c') = 1\,, \quad \text{if } c = c'\,,$$
$$= 0\,, \quad \text{if } c \neq c'\,.$$

Usually, the linguist will define complete neighborhoods broadly so that the majority of acceptable strings can be generated and recognized correctly. As his analysis proceeds further in detail, he will take an example that is not generated or recognized correctly by his broadly defined complete neighborhoods: Generation may give him some unacceptable strings, or the syntactic analysis may give him erroneous or unnecessarily ambiguous interpretations. He will then trace back the definitions and find that some of his rules hold in his example with respect to a subset of one of his complete neighborhoods. Suppose he has a set $R(xy)$ of rules to concatenate x and y. His new example may indicate that the rules are not always true. He will then establish the subsets x', x'', y', y'', and a new set of rules which allows $x'y'$ and $x''y''$, for instance, but not $x'y''$ or $x''y'$.

18.3

Let a broadly classified complete neighborhood be represented by a symbol, say, v. If a subclassification thereof is desired, we introduce an index p, such that

$$v = v(p_1) \cup v(p_2) \cup \ldots \cup v(p_n)\,.$$

When the subclassification is not necessary, we set $p = 0$:

$$v(0) = \bigcup v(p_i)\,, \quad i = 1, 2, \ldots, n\,.$$

The union of a few subsets is written as

$$v(p_1,p_3,p_5) = v(p_1) \cup v(p_3) \cup v(p_5), \text{ etc.}$$

If a complete neighborhood is subclassified from several different points of view, as many indices are introduced:

$$v(p;q),\ v(p;q;r), \text{ etc.}\,,$$
$$v(p_1,p_2;q) = v(p_1;q) \cup v(p_2;q)\,,$$
$$v(p;q_1,q_2) = v(p;q_1) \cup v(p;q_2)\,,$$
$$v(p;0) \cap v(0;q) = (\bigcup v(p;q_j)) \cap (\bigcup v(p_i;q))$$
$$= v(p;q), \text{ etc.}$$

Hence, for the distribution classes, we have

$$H(v(p_1,p_2;q)) = H(v(p_1;q)) \cap H(v(p_2;q))\,,$$
$$I(v(p;q)) = I(v(p;0)) \cap I(v(0;q))\,,$$
etc.

Sometimes, an index depends upon other indices:

$$v(p;q(r;s;t))\,,$$

for example. The meaning of r, s, and t depends on the meaning of q.

18.4

The above scheme can be further generalized. Let a complete neighborhood be represented by a number of indices

$$(a;b;c;\ldots;n)\,,$$

where the first broad class symbol v in the previous representation is one of the indices and each index represents a classification from a certain point of view. This kind of homogeneous representation, although it is redundant, enables us to describe the syntax of a language more systematically. Each digit can be regarded as an indication of a certain feature common to some elementary neighborhoods, and they are classified according to their specific features.

Suppose a concatenation rule $f(uv;w)$ is to be applied to a text string xy to determine $z = xy$, and the complete neighborhoods are represented with the indices in the form

$$x = (a(x);b(x);\ldots;n(x))\,,$$
$$y = (a(y);b(y);\ldots;n(y))\,,$$
$$z = (a(z);b(z);\ldots;n(z))\,,$$
$$u = (a(u);b(u);\ldots;n(u))\,,$$
$$v = (a(v);b(v);\ldots;n(v))\,,$$
$$w = (a(w);b(w);\ldots;n(w))\,.$$

If a rule indicates the relation between the pair $(i(u),j(v))$, and $k(w)$, and if all the others are independent of these, we have

$$u = (0;\ldots;0;i(u);0;\ldots;0)\,,$$
$$v = (0;\ldots;0;j(v);0;\ldots;0)\,,$$
$$w = (0;\ldots;0;k(w);0;\ldots;0)\,.$$

If the pairs $(i(x),i(u))$, $(j(y),j(v))$, and $(k(z),k(w))$ satisfy the condition specified by the grammar system being used, the rule is applied to xy and gives a z modified by this rule. The rule gives no information regarding the other indices. The information should not be lost if it is available in x or y. We have to indicate in the rule how to transfer the information to z from x or y. A simple method was used in a translation program [7].

The transformation rules require that certain features

of the original string be carried forward to its transform. This requirement is usually indicated by the identity of features of certain segments in the original string and its transform. The use of rules is to be programmed in such a way that the value of the index in the original string is transferred to the corresponding index of the transform and vice versa in case of an inverse transformation.

18.5

An extremely simplified example is given. The complete neighborhoods are no longer regarded as sets. The symbol "+" means "or." The symbol "=" does not necessarily mean an identity: It can be replaced by an arrow. The segments of the string

they are red planes
1 2 3 4

are represented in the form (h,k):

(1,1) = they ,

(1,3) = they are red ,

(2,3) = are red ,

etc.

Both $(h,i)(j,k)$ and (h,i) & (j,k) mean the concatenation of the segments (h,i) and (j,k). The following abbreviations are used: adj: adjective; adj-pred: adjectival predicate; compl: complement; m: masculine; n: nominal; nonh: nonhuman; n/n: modifier of nominal; nom: nominative; pl: plural; pn: pronoun; s: sentence; v: verbal.

Input Language (English)

(1,4)(v;s) = (1,1)(pn;3d;pl) & (2,2)(v;be;pres;3d;pl) & (3,4)(n;pl) ,

(3,4)(n;pl) = (3,3)(adj) & (4,4)(n;pl) .

Intermediate Representation

(1,4)(v) = (1,1)(pn;3d;pl;nom) & (2,2)(copula;pres) & (3,4)(n;compl;pl) ,

(3,4)(n;compl;pl) = (3,3)(n/n) & (4,4)(n;compl;pl) .

Output Language (Russian)

(1,1)(pn;3d;pl;nom) = on(pl;nom) = onji ,

(2,2)(copula;pres) = ( ) ,

(3,3)(red)(n/n) = krasn(adj) ,

(4,4)(plane)(n;compl;pl) = (rubank + samoljet)(n;m;pl;nom)
= (rubanki + samoljeti)(n;m;pl;nom) ,

(3,4)(n;compl;pl) = (3,3)(adj) & (4,4)(n;m;pl;nom)
= (3,3)-ije(4,4) ,

(1,4)(v) = (1,1)(2,2)(3,4)
= onji krasnije (rubanki + samoljeti) .

Output Language (Japanese)

(1,1)(pn;3d;pl;nom)
= (kare(human) + sore(nonh))(pn;pl;nom) ,

(2,2)(copula;pres) = ar(v;5;pres;final) = ar-ru = aru ,

(3,3)(red)(n/n) = aka(adj-pred;n/n) ,

(4,4)(plane)(n;comp;pl)
= (heimen + hikooki)(n;non-h;compl) ,

(3,4)(n;compl;pl) = ((3,3)(n/n) & (4,4)(n;nonh))(n;comp)
= ((3,3)-i(4,4))(n;nonh)-de ,

(1,4)(v) = (1,1)(human,nonh;pn;pl;nom)
& (3,4)(n;nonh)-de
& (2,2)(v;5;pres;final)
= (1,1)(non-h;pn;pl;nom) & (3,4)(n;nonh)-de
& (2,2)(v;5;pres;final)
= sorera (ga + wa) akai (heimen + hikooki) de aru .

18.6

We observe in the above example that the index of human or nonhuman objects affects the choice of a lexical element in Japanese while it is not relevant in Russian. This phenomenon may be considered syntactic in one language and semantic in another. Take two languages A and B, and suppose A has a syntactic marker of gender and B does not. The gender is considered syntactic in A and semantic in B. The syntactic genders are sometimes arbitrary and cannot always be preserved in the transfer process from one language to another. We must prepare two separate procedures for handling gender. Similar problems can arise with respect to other indices. The complicated syntactic case-number choice in Russian is another example.

The choice of lexical elements depends greatly upon the habitual usage of language. The situation is similar when we observe the combinations of longer constituents. The choice of constituents is limited for logical, semantic, or habitual reasons. Sometimes the choice is quite capricious. It seems more practical to handle this kind of information separately [13] corresponding to the separate normative devices the linguist has conjectured.

Acknowledgment

The need to define distribution classes was recognized when I was with the Machine Translation Project, University of California. The basic approach was worked out at the First Research Center, Defense Agency of Japan. Further development was made at the Project on Linguistic Analysis, Ohio State University, and a previous version of this

paper was read at the 1965 International Conference on Computational Linguistics held in New York. I have appreciated the encouragement of these organizations.

Received July 17, 1968

References

1. Parker-Rhodes, A. F. "A New Model of Syntactic Description." In *1961 International Conference on Machine Translation of Languages and Applied Language Analysis.* London: Her Majesty's Stationery Office, 1962.
2. Kulagina, Olga S. *Ob Odnom Sposobje Oprjedjeljenjija Grammatjičeskix Ponjatjij na Bazje Tjeorjiji Množestv Probljemi Kibjernjetjiki.* Vipusk 1. Moscow, 1958.
3. Rosen, Barry K. "Context-sensitive Syntax Analysis." In *Report No. NSF-18, Mathematical Linguistics and Automatic Translation.* Cambridge, Mass.: Computation Laboratory, Harvard University, 1967.
4. Lakoff, George. "On the Nature of Syntactic Irregularity." *Report No. NSF-16, Mathematical Linguistics and Automatic Translation.* Cambridge, Mass.: Computation Laboratory, Harvard University, 1965.
5. Lamb, Sydney. *On Alteration, Transformation, Realization, and Stratification.* Monograph Series on Language and Linguistics, no. 7, 1964.
6. Opler, A.; Silverstones, R.; Saleh, Y.; Hildebran, M.; and Slutzky, I. "The Application of Table Processing Concepts to the Sakai Translation Technique." *Mechanical Translation* 7, no. 2 (1963): 40.
7. Sakai, I. "Syntax in Universal Translation." *1961 International Conference on Machine Translation of Languages and Applied Language Analysis.* London: Her Majesty's Stationery Office, 1962.
8. Hillman, Donald J. "Grammars and Text Analysis." *Report No. 1, Computational, Phonological, and Morphological Linguistics and Retrieval Studies, Center for Information Sciences.* Bethlehem, Pa.: Lehigh University, 1965.
9. Wang, W. S. "Two Aspect Markers in Mandarin." In *Project on Linguistic Analysis,* report no. 8, 1964.
10. Hashimoto, Anne Yue. "Resultative Verbs and Other Problems." In *Project on Linguistic Analysis,* report no. 8, 1964.
11. Watt, W. C. "Acceptance, Acceptability, Grammaticality, Sentencehood." *National Bureau of Standards Report 9051,* 1966.
12. Charney, Elinor K. *Structural Semantic Foundations for a Theory of Meaning.* Mechanical Translation Group. Chicago: University of Chicago, 1966.
13. Matthews, P. H. "Problem of Selection in Transformational Grammar." *Journal of Linguistics,* no. 1 (1965).