21

1965 International Conference on Computational Linguistics

SOME MATHEMATICAL ASPECTS ON SYNTACTIC DISCRIPTION

Itiroo Sakai

Project on Linguistic Analysis
Ohio State University
216 North Oval Drive
Columbus, Ohio 43210
U. S. A.

Abstract. The purpose of this paper is to help linguists contruct a consistent, sufficient and less redundant syntax of language.

An acceptable string corresponds to an expression or an utterance: it may be a natural text, a string of morphemes, a tree structure or any kind of representation. 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. 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. Two contexts are equivalent if they accept or reject any given string at the same time. An elementary neighborhood is the set of all contexts equivalent to one context.
Four simple distribution classes are proposed and their properties are discussed.

Concatenation rules of a language can be described in terms of concatenated complete neighborhoods or concatenated distribution classes. Some possible representations and their consequences are discussed.

Transformational rules are also described in a similar way. However, there is another problem of correspondence of original strings to their transforms. It is useful to establish subsets of elementary neighborhoods and this subclassification may contribute to a simplification of the clumsy representation of derivational history.

Finally, some trivial but practically useful conventions are described.

1. Introduction.

The grammar of a language should be consistent throughout its whole system. No features should be left unformulated in order that the grammar be a complete one. At the same time, it is desirable to prepare the grammar as compact as possible. These are important requirements especially when the grammar is a machine-oriented one. The knowledge on the formal properties of syntax will help us construct an objective system of grammar. Every term used in a description should be rigorously defined and no ambiguous expressions are allowed. If the consequence of grammar rules deviates from the proper usage of the language, we will be able to trace back the definitions and locate the source of trouble.

When the grammar rules are given in terms of concatenated symbols, we must know the formal definition of the symbols before writing a program by which the rules are applied to the text. If a grammar rule describes the nature of a P-marker, the label given to each node in the P-marker must have an unambiguous definition which relates the meaning of the symbol to the strings supplied as texts.

We need, at least, an objective criterion by which we can specify a language. This criterion will be a 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 or a part of speech. The syntactic function of a symbol string is considered as the set of all acceptable utterances in which the string occurs. 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 a complete neighborhood.

A distribution class can be defined as a set of strings whose complete neighborhoods are related to a given set of contexts in a specified way. We propose four simple definitions of distribution classes.

With these fundamental concepts of parts of speech and distribution classes, we can proceed to a more formal system of syntactic description. However, a few questions may be immediately raised. Is it really possible to construct a grammar in such an elementary way? How can we list the elements of a set picking them up out of 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? Etc. Etc.

We should be better off if we were to create a new language by preparing a grammar and a lexicon. Unfortunately the situation is quite contrary when we are to handle a natural language. The language exists. We want to find out a grammar that accounts for all and only the acceptable strings of the language. We regard a language L as a set of strings generated by a machine M, whose internal structure is not known to us. We can observe only a part of the set of generated strings in a limited length of time. We want to construct a hypothetical machanism 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 if it is an element of L, and strings are supplied to M' to see if 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. 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 a stratum to another before it becomes a string of natural language. An utterance has a few different forms corresponding 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 constuct a grammar will find it advantageous to establish rules that hold for a set of strings or for a set of relevant facts. A linguistic phenomenon may be analyzed from various points of view which will help him avoid listing a tremendous 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 common features in their markers. The procedure of using these rules consists of two parts. The one is a routine that compares a rule with the text and decides whether or not the rule is to be applied. The other is a 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 is unambiguously defined so that the code obtained in the text is exactly what the linguist wants to mean.

Some of his rules may account for a certain number 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 a natural text or by generating strings. The normative device will tell him whether or not a string supplied to it is acceptable but not tell him why. It is obvious that these procedures can not be carried out practically on every string that may be supplied to a machine in the future, and that nobody will be able to predict what can occur when an arbitrary string is supplied to the machine. Nevertheless, it is required that a grammar may deal with most of the texts supplied in the future.

His grammar is inevitably affected by the nature of the normative device. If the normative device is so strict as to reject every string which fails to meet such requirements as that its style must be just an ordinary one, the statement must be logically correct, the lexical usage must conform with the regular way of the language, etc., 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 it finds the internal relationship of the string is acceptable, regardless of the reality the string designates. If the grammar is to be applied to input texts whose structure is always grammatically correct and unambiguous, a grammar which satisfies the requirement of this device will be enough. However, it will give many unusual strings if it is used in random generation and many ambiguous alternatives if it is used for analysis. The second device may reject those strings whose structure shows 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 one wants to have more rigorous grammar that may be used for random generation of only non-surprising sentences, 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 number of digits will be assigned to the coded form of markers corresponding to each step of decision. The procedure will be programmed so as 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 a particular combination of codes is to obey a particular rule, the rule will be prepared independently and the general procedure will be prohibited. This is done by a simple technique in coding and programming.

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

2. Symbol; String; Language.

2.1. Symbol is an undefined term. Morphs, morphemes, lexes, lexemes, or some other units may be regarded as symbols. Any unit consisting of a number of symbols is called a string. All the strings are possible strings. If a string is considered meaningful, then it is an acceptable string. Each acceptable string is an undefined term.

These definitions are quite formal. If we confine ourselves to the problems in morphotactics, the symbols are morphs and the acceptable strings are what are called expressions or utterances. A symbol may 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 may regard a labeled tree called a P-marker as a string, and a labeled node as a representation of the substring dominated by the node, although the term string seems inadequate in this case. A node represents a P-marker consisting of all the terminal and non-terminal nodes it dominates. We can regard a P-marker as a tree-like string of P-markers dominated by the former. Another kind of branches may be added to the syntactic tree in order to indicate the relationship among the constituents. We call this representation a net, provisionally. We may regard a net as a string consisting of a number of labeled nodes, whose arrangement is shown by two kinds of branches.

We define a language as a set of acceptable strings. An acceptable string of a natural language is considered to have as many versions as the number of strata established by linguist. 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.

2.2. Suppose we have a linear string. We interrupt the string by deleting some of the symbols therein and inserting a symbol of absence to each point of deletion. If a symbol of absence is followed by another immediately, they are contracted to one. A linear string is continuous if it is not interrupted.

All the 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 tree-like string is continuous, if and only if (1) all the nodes of the string are included in one node D, and (2) there are no other nodes which are not included in D.

A net string is continuous, if and only if the syntactic tree is continuous and no branches of the second kind are broken off.

Any substring of a string is called a segment. It may be either continuous or discontinuous. A discontinuous segment consists of a few parts separated from each other. Each part of segment is called a fragment which is necessarily continuous (Parker-Rhodes, 1961).

3. Context; Neighborhood.

3.1. Context; Acceptable Context.

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

If the discussion is confined to a context-free phrase structure language, it seems more convenient to modify the concepts acceptable string and context: any immediate constituent of an acceptable string is also acceptable, and a context is acceptable to a string if the string, its context and the whole string are all acceptable. If the constituents are continuous, the situation becomes simpler. The context c = r()t is acceptable to s, if r, s, t, and rst are all acceptable. Either r or t may be absent.

3.2. Neighborhood.

A context is an interrupted string which becomes a continuous string if an appropriate segment is supplied to its points of interruption. Let

$$y = \mathrm{set}(c_1, c_2, ---, c_n)$$

be a set of contexts and let s be a string. If all the contexts in y become acceptable strings when s is supplied to them, then the set y defines a property of s. We call the set y an acceptable neighborhood of s. If y is an acceptable neighborhood of strings s_1, s_2, s_3, for instance, then we say y is an acceptable neighborhood of

$$S = \mathrm{set}(s_1, s_2, s_3),$$

and we consider the set y represents a syntactic property common to all the strings in S. Note that our neighborhood is not the same as the okrjestnostj (Kulagina, 1958). A set of acceptable strings with a string s is called a paradigm of s (Parker-Rhodes, 1961); our neighborhood is a paradigm in which the string 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 not acceptable to c_i or c_j. In this case, we can not tell the difference between c_i and c_j as far as the acceptance of the strings s and t are concerned. We say these contexts are equivalent to each other and write

$$c_i \text{ eqv } c_j,$$

if the condition "c_i is acceptable to a string s, if and only if c_j is acceptable to s" is satisfied for every possible string s of the language. The relation of equivalence is symmetric, reflexive, and transitive:

(1) c_i eqv c_j;

(2) if c_i eqv c_j, then c_j eqv c_i;

(3) if c_i eqv c_j and c_j eqv c_k, then c_i eqv c_k.

5. Complete Neighborhood.

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 the 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 context c' in y:

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

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 the contexts acceptable to it. We show that C(s) is complete.

(1) If $c \in C(s)$, then $c \in N(C(s))$; that is $C(s) \subseteq N(C(s))$.

(2) If $c \in N(C(s))$,
then c eqv c' for some c' in C(s),
then c eqv c' and c' is acceptable to s,
then c is acceptable to s,
then $c \in C(s)$,
therefore $N(C(s)) \subseteq C(s)$.

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.

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

The complete neighborhood C(s) of a string s is considered to correspond to the syntactic function or the part of speech of the string s. The elements of C(s) share a 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. This property of C(s) leads us to the application of complete neighborhood to a given set of contexts supplied as text.

Let S be an arbitrary set of contexts. Some elements of S may be accepted by s and some others may not. The elements accepted by s must, at the same time, belong to C(s), that is, to $C(s) \cap S$. If

$$C(s) \cap S = 0,$$

then the string s can not occur under the contextual condition defined by S, and vice versa. If

$$C(s) \cap S = C(t) \cap S,$$

then we have no means to distinguish the syntactic function of s and t with respect to the given S. If S is the set of all the possible contexts of the language, then

$$C(s) \cap S = C(s)$$

for any string s. If

$$C(s) = C(t),$$

then we have no means to distinguish the syntactic function of s and t so far as only the acceptability is concerned.

5.2. It occurs very often 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,

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

We put $x = C(r)$,

$y = C(s)$,

$z = C(t)$.

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

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

Taking the union of these two, we have

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

This means that r accepts every context in $S' \cap S''$ if it is acceptable to s or t. Now, we will 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 may be interpreted in terms of y and z, and that y and z may account for something lacking in x with respect to S.

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

6. Elementary Neighborhood.

6.1. We have seen above that a complete neighborhood

$$x = C(r)$$

is interpreted in terms of

$$y = C(s)$$

and $$z = C(t).$$

We can expect C(s) and C(t) may be a representation of a simpler and more specific syntactic function. If

$$C(r) = C(s) \cup C(t),$$
$$C(s) \neq C(t),$$
$$C(s) \neq 0,$$
$$C(t) \neq 0,$$

then, for some c_i in C(s) and some c_j in C(t), we have

$$c_i \text{ not eqv } c_j.$$

6.2. A set of all mutually equivalent contexts, called an elementary neighborhood, leads us to a 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

$$e(i) = \text{set}(c\colon c \text{ eqv } c_i).$$

Since the equivalence is symmetric, reflexive and transitive, any two distinct elementary neighborhoods have no elements in common.

6.3. Let x be a complete neighborhood and e(i) an elementary neighborhood. If an element c in x is a member of e(i), then

$$e(i) \subseteq x,$$

because x is complete. Take an element c_i in x; then there is an e(i) such that

$$c_i \in e(i).$$

Therefore, for any given complete neighborhood x, we have

$$x = \bigcup e(i)$$

for all e(i)'s having at least one element in x. Every elementary neighborhood is complete. An intersection of complete neighborhoods is complete. Every union of elementary neighborhoods is a complete neighborhood.

7. Distribution Class.

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 a string occurs. Given a context, we can classify the strings into two distinct categories: the one is a class of strings that can occur in the given environment and the other is the class of strings that can not occur therein.

If there exists at least one context c in which both s and t can occur, then $c \in C(s)$ and $c \in C(t)$,

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

We define the set of all strings t, that can replace s in some contexts, as

$$G(C(s)) = \text{set}(t: C(t) \cap C(s) \neq 0).$$

We introduce a convention

$$A\ (=)\ B$$

which means that the intersection of the two sets A and B is not empty:

$$G(C(s)) = \text{set}(t: C(t)\ (=)\ C(s)).$$

Suppose a string t can occur wherever s can occur, but s can not always occur in the contexts accepted by t. In this case,

$$C(t) \supseteq C(s).$$

We define

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

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

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

That the two strings s and t are mutually replaceable means that s can occur wherever t can occur and conversely t can occur wherever s can occur. In other words, any context c is accepted by t, if and only if it is accepted by s:

$$c \in C(t) \quad \text{if and only if} \quad c \in C(s),$$

or

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

We indicate the set of such strings t by

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

Other distribution classes are defined as sets of strings whose complete neighborhoods are related to a certain complete neighborhood in a specified way. Let x be an arbitrary complete neighborhood. The simple types of distribution classes mentioned above are written as

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

A distribution class is said to be real if it is not empty, and imaginary if it is empty. Suppose, for instance, 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 = flying,

s_2 = red,

s_3 = making

and their contexts

c_1 = they are () planes,

c_2 = a () saucer is an object,

c_3 = () planes is an industry.

The complete neighborhoods of the strings are

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

The distribution classes are determined by these neighborhoods. The simple types above are given in the table below.

i: s_1	$C(s_i)$	$G(C(s_i))$	$H(C(s_i))$	$I(C(s_i))$	$J(C(s_i))$
1: flying	(c_1,c_2,c_3)	(s_1,s_2,s_3)	(s_1)	(s_1,s_2,s_3)	(s_1)
2: red	(c_1,c_2)	(s_1,s_2,s_3)	(s_1,s_2)	(s_2)	(s_2)
3: making	(c_1,c_3)	(s_1,s_2,s_3)	(s_1,s_3)	(s_3)	(s_3)

The elementary neighborhoods

$$e(i) = \text{set}(c\colon c \text{ eqv } c_i), \qquad i = 1, 2, 3$$

are found by consulting the table below, where "+" on the i-th row and j-th column means "c_j is acceptable to s_i".

:	c_1	c_2	c_3
s_1:	+	+	+
s_2:	+	+	-
s_3:	+	-	+

$$e(1) = \text{set}(c\colon c \text{ eqv } c_1) = \text{set}(c_1),$$

$$e(2) = \text{set}(c_2),$$

$$e(3) = \text{set}(c_3).$$

Therefore,

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

$$C(s_2) = e(1) \cup e(2),$$

$$C(s_3) = e(1) \cup e(3).$$

<u>7.1</u>.

(1) $J(x) = H(x) \cap I(x);$

(2) $H(x) \cup I(x) \subseteq G(x), \quad x \neq 0.$

Proof.

(1) $t \in J(x),$

if and only if $C(t) = x,$

" $C(t) \supseteq x$ and $C(t) \subseteq x,$

" $t \in H(x)$ and $t \in I(x),$

" $t \in H(x) \cap I(x).$

(2) $t \in H(x) \cup I(x),$

if and only if $t \in H(x)$ or $t \in I(x),$

" $C(t) \supseteq x$ or $C(t) \subseteq x,$

then $C(t)\ (=)\ x$ for $x \neq 0,$

if and only if $t \in G(x).$

<u>7.2</u>. The equality $C(t) = C(s)$ of two sets is symmetric, reflexive and transitive. Therefore,

$J(x) = J(y)$

if and only if $J(x)\ (=)\ J(y).$

This means that any two different sets have no elements in common and, consequently, that every element belongs to one and only one set of the form $J(x)$.

7.3.

7.3.1. If x is an elementary neighborhood, then

$$G(x) = \mathrm{set}(t\colon C(t)\ (=)\ x) = \mathrm{set}(t\colon C(t) \supseteq x) = H(x)$$

if $x \neq 0$;

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

so that C(t) is also elementary.

7.3.2. If x is any complete neighborhood and if C(t) is elementary for all t, then

$$G(x) = \mathrm{set}(t\colon C(t)\ (=)\ x) = \mathrm{set}(t\colon C(t) \subseteq x) = I(x);$$

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

if $x \neq 0$,

so that x is also elementary.

7.3.3. If C(t) is elementary for all t and x is also elementary and non-empty, then

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

7.4. If $x = y \cup z$, then

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

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

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

Proof.

(1) $t \in G(x) = G(y \cup z)$,

if and only if $C(t)\ (=)\ x = y \cup z$,

" $C(t)\ (=)\ y$ or $C(t)\ (=)\ z$,

" $t \in G(y)$ or $t \in G(z)$,

" $t \in G(y) \cup G(z)$.

(2) $t \in H(x)$,

if and only if $C(t) \supseteq x = y \cup z$,

" $C(t) \supseteq y$ and $C(t) \supseteq z$,

" $t \in H(y)$ and $t \in H(z)$,

" $t \in H(y) \cap H(z)$.

(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 $t \in I(x)$.

7.5. If $x = y \cap z$, then

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

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

Proof.

(1) $t \in G(x)$,

if and only if $C(t) \cap x \neq 0$,

" $C(t) \cap y \cap z \neq 0$,

then $C(t) \cap y \neq 0$ and $C(t) \cap z \neq 0$,

if and only if $t \in G(y) \cap G(z)$.

(2) $t \in I(x)$,

if and only if $C(t) \subseteq x = y \cap z$,

" $C(t) \subseteq y$ and $C(t) \subseteq z$,

" $t \in I(y)$ and $t \in I(z)$,

" $t \in I(y) \cap I(z)$.

8. Concatenation.

8.1. Concatenation of Strings.

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

8.2. Concatenation of Contexts.

Let

$$r_1, r_2, ---, r_n$$

be segments of p with no fragments in common. The contexts

$$c_p(r_i) \quad \text{of} \quad r_i \text{ in } p,$$

$$i = 1, 2, ---, n$$

correspond uniquely to the segments r_i, respectively, and so does $c_p(t)$ to the concatenation

$$t = r_1 r_2 \text{---} r_n.$$

We write

$$c_p(r_1)c_p(r_2)\text{---}c_p(r_n) = c_p(t)$$

if and only if $t = r_1 r_2 \text{---} r_n$ in p.

8.3. Concatenation of Sets.

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

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

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

8.3.1. 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 as one of the three binary concatenations

$$A(BCD),\ (AB)(CD),\ (ABC)D$$

when the discussion is strictly binary. In a morphographemic description, however, this is not very important. One may assume one of these three acceptable and discard the other two as unacceptable. In a morphotactic description, some one of these three will be chosen so as to make the whole description of the language simpler. If any one of the sets which constitute a concatenation is empty, then the concatenation is also empty.

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 covered by

$$ABCD = A(BCD) \cup (AB)(CD) \cup (ABC)D.$$

Since we are to handle binary concatenations only, we consider two concatenations of elements are different if their structures are not the same:

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

Then, the condition

$$ABCD = (AB)(CD)$$

yields

(1) $(AB)(CD) \neq 0,$

(2) $A(BCD) = 0,$

(3) $(ABC)D = 0.$

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

Therefore,

(4) $A(BC) \neq 0$,

(5) $(AB)C = 0$,

because $A(BC) \cap (AB)C = 0$.

Similarly, $BCD = B(CD) \cup (BC)D = (BC)D$,

(6) $(BC)D \neq 0$,

(7) $B(CD) = 0$.

From (2), $A(BCD) = A(B(CD) \cup (BC)D) = 0$.

By (7) and (6), $A(BCD) = 0 \cup A((BC)D) = 0$,

or,

(8) $A \neq 0, \quad (BC)D \neq 0, \quad A((BC)D) = 0$.

From (3), $(ABC)D = (A(BC) \cup (AB)C)D = 0$.

By (4) and (5), $(ABC)D = (A(BC) \cup 0)D = (A(BC))D = 0$,

(9) $A(BC) \neq 0, \quad D \neq 0, \quad (A(BC))D = 0$.

Now, we can describe the syntax of these strings in terms of binary concatenations only, if we establish the rules numbered from (1) to (9).

<u>8.3.2</u>. The following formulas are frequently used.

(1) $AB = CD$, if and only if $A = C$ and $B = D$,

because, for any ab in AB,

$AB = CD$

if and only if ($ab \in AB$ if and only if $ab \in CD$)

" (($a \in A, b \in B$) if and only if ($a \in C, b \in D$))

" ($a \in A$ if and only if $a \in C$,
$b \in B$ if and only if $b \in D$)

" $A = C$ and $B = D$.

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

because $ab \in A(B \cup C)$

if and only if $a \in A$ and $b \in B \cup C$

" ($a \in A$ and $b \in B$) or ($a \in A$ and $b \in C$)

" $ab \in AB$ or $ab \in AC$

" $ab \in AB \cup AC$.

(3) Similarly, $(A \cup B)C = AC \cup BC$.

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

because $ab \in AB \cap CD$

if and only if $ab \in AB$ and $ab \in CD$

" $a \in A$ and $b \in B$ and $a \in C$ and $b \in D$

" $a \in A \cap C$ and $b \in B \cap D$

" $ab \in (A \cap C)(B \cap D)$.

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,

$$C(r_i) = x \quad \text{for all} \quad r_i \quad \text{in} \quad J(x)$$

and $$C(s_j) = y \quad \text{for all} \quad s_j \quad \text{in} \quad J(y).$$

Any string

$$p(r_i) = \text{---}r_i\text{---}$$

with the segment r_i in it is acceptable if and only if

$$p(r) = \text{---}r\text{---}$$

is acceptable, and the string

$$p(s_j) = \text{---}s_j\text{---}$$

is acceptable if and only if

$$p(s) = \text{---}s\text{---}$$

is acceptable. Suppose

$$p(r_i s_j) = \text{---}r_i\text{---}s_j\text{---}$$

is a string with both r_i and s_j in it. Any such string is acceptable if and only if the string

$$p(r_i s) = \text{---}r_i\text{---}s\text{---}$$

is acceptable, and $p(r_i s)$ is acceptable if and only if

$$p(rs) = \text{---}r\text{---}s\text{---}$$

is acceptable. Therefore, $p(r_i s_j)$ is acceptable if and only if $p(rs)$ is acceptable. That is

$$C(r_i s_j) = C(rs).$$

We define the concatenation $C(r)C(s)$ of complete neighborhoods as the complete neighborhood $C(rs)$ of the concatenated strings. Generally, we put

$$xy = C(rs), \quad r \in J(x), \quad s \in J(y)$$

for any complete neighborhoods x and y, where $J(x)$ and $J(y)$ may be real or imaginary. Note, however, that

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

then $xy = C(rs)$,

while $xy = C(rs)$

does not always result in

$$x = C(r) \quad \text{or} \quad y = C(s).$$

We have generalized and transferred the concatenation of 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 strings are syntactically ambiguous. The distribution class J(x) means the narrowest classification of strings and no further subclassification is possible, while its complete neighborhood x can be subclassified if x is not an elementary neighborhood. If

$$r \in J(x) \quad \text{and} \quad x = y \cup z,$$

then we can talk about imaginary strings r' and r", such that

$$C(r') = y \quad \text{and} \quad C(r'') = z.$$

These imaginary strings, always referred to implicitly in terms of distribution classes, can be discussed explicitly in terms of complete neighborhoods.

9.2. We make distinction between the concatenation

$$xy = C(r)C(s)$$

of complete neighborhoods and the complete neighborhood

$$z = C(rs).$$

The former means a set consisting of concatenated contexts. The properties of the language is introduced when it is written in the form

$$xy = z$$

or $C(r)C(s) = C(rs)$,

where the property x of r and the property y of s result in another property z of rs. Thus, z can be an empty set even if neither x nor y is empty, and ambiguous even if neither x nor y is ambiguous.

9.3. We find it advantageous to have a system which represents every complete neighborhood in a unified way. We saw that a complete neighborhood x can be represented by a union of elementary neighborhoods e(i):

$$x = \bigcup e(i) \quad \text{with} \quad x \cap e(i) \neq 0.$$

Let us introduce coefficients x(i), such that

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

and no other cases possibly occur. We put

$$\begin{aligned} x(i)e(i) &= e(i) \quad \text{if} \quad x(i) = 1, \\ &= 0 \quad \text{if} \quad x(i) = 0. \end{aligned}$$

In virtue of these coefficients, we can write

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

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

then
$$x \cup y = (\bigcup x(i)e(i)) \cup (\bigcup y(j)e(j)) = \bigcup (x(k) + y(k))e(k) = \bigcup z(k)e(k).$$

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

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) A concatenation of two elementary neighborhoods is a complete neighborhood, and it is also a union of elementary neighborhoods:
$$e(i)e(j) = \bigcup a(i,j,k)e(k).$$

Writing
$$z = xy = \bigcup\bigcup z(i,j)e(i)e(j) = \bigcup\bigcup\bigcup z(i,j)a(i,j,k)e(k) = \bigcup z(k)e(k),$$

we have $e(k) \subseteq z$

if and only if $e(i)e(j) \subseteq z$ and $e(k) \subseteq e(i)e(j)$.

Therefore, for the expression
$$z(i,j)a(i,j,k) = z(k),$$

we have $1 \times 1 = 1,$

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

10. Concatenation of Distribution Classes.

10.1.	$G(u)G(v) \subseteq G(uv)$,
because	$rs \in G(u)G(v)$
if and only if	$r \in G(u)$ and $s \in G(v)$
"	$C(r) \cap u \neq 0$ and $C(s) \cap v \neq 0$
"	$(C(r) \cap u)(C(s) \cap v) = C(r)C(s) \cap uv \neq 0$
then	$C(rs) \cap uv \neq 0$
if and only if	$rs \in G(uv)$.
10.2.	$H(u)H(v) \subseteq H(uv)$,
because	$rs \in H(u)H(v)$
if and only if	$r \in H(u)$ and $s \in H(v)$
"	$C(r) \supseteq u$ and $C(s) \supseteq v$
"	$C(r) \cap u = u$ and $C(s) \cap v = v$
"	$(C(r) \cap u)(C(s) \cap v) = C(r)C(s) \cap uv = uv$
"	$C(r)C(s) \supseteq uv$
then	$C(rs) \supseteq uv$
if and only if	$rs \in H(uv)$.
10.3.	$I(u)I(v) \subseteq I(uv)$,
because	$rs \in I(u)I(v)$
if and only if	$r \in I(u)$ and $s \in I(v)$
"	$C(r) \subseteq u$ and $C(s) \subseteq v$
"	$C(r) \cap u = C(r)$ and $C(s) \cap v = C(s)$
"	$(C(r) \cap u)(C(s) \cap v) = C(r)C(s) \cap uv = C(r)C(s)$
"	$C(r)C(s) \subseteq uv$
then	$C(rs) \subseteq uv$
if and only if	$rs \in I(uv)$.
10.4.	$J(u)J(v) \subseteq J(uv)$,
because	$rs \in J(u)J(v)$
if and only if	$r \in J(u)$ and $s \in J(v)$
"	$C(r) = u$ and $C(s) = v$
"	$C(r)C(s) = uv$
then	$C(rs) = uv$
if and only if	$rs \in J(uv)$.

11. Rules for Recognition and Generation.

Each rule of a grammar indicates the arrangement of a few items to be concatenated, accompanied by some other necessary informations. 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 a string of symbols.

In case a grammar is given in terms of complete neighborhoods, the input text is converted to a string of complete neighborhoods before the syntactic analysis begins. At the very end of generation, a terminal node accompanied by a complete neighborhood x is replaced by a string s whose complete neighborhood C(s) shares at least one elementary neighborhood with x.

When the syntactic rules are expressed in terms of sets of strings, the input text to be analyzed is replaced by a string of distribution classes. If a symbol string belongs to more than two sets of strings, their meet replaces the symbol string. At the end of a generation, the synthesized output string is obtained by replacing the set of strings on each terminal node by a string which is a member of the set.

11.1. An acceptable string can be generated and analyzed making use of a tree with its nodes marked by complete neighborhoods. The expansion of a node z to a concatenation xy of nodes x and y implies $z \supseteq xy$, because otherwise further expansion of x and y may yield a structure which can not be accepted by z. Transformational rules can be applied more freely because a transformation does not imply such a restriction. However, attention ahould be paid not to add any other contexts to the complete neighborhoods attached to the nodes already generated. Finally, each terminal node is replaced by a lexical element. The string obtained after applying all the obligatory rules must be an acceptable string.

The analysis is carried out by testing all the possible transformations and trying all the possible contractions. At any rate, both generation and analysis can be carried out if we have a set of rules which gives concatenation z = x---y for any x, ---,y of the language, and the transform y(1)y(2)---y(n) of any string $x(1)x(2)---x(_m)$ of complete neighborhoods.

11.2. Acceptable strings are also generated by starting from the node P(0) which is the set of all acceptable strings. It is replaced by its subset

$$P(1)P(2)---P(i)---P(m) \subseteq P(0)$$

which is a concatenation of nodes P(i)'s. Each node P(i) also represents a set of strings, and it may or may not be replaced again by

$$P(i1)---P(ij)---P(in) \subseteq P(i).$$

On each step of expansion, a choice is made by taking a subset of strings. The possible choice becomes narrower and narrower. It is expected that the string obtained by applying obligatory rules and by replacing each terminal node by a lexical element is an acceptable string.

This is not always true if the replacement of a node is independent of the other nodes already generated. This difficulty is overcome by executing a syntactic analysis after every step of expansion. If the analysis does not prove the possibility of obtaining an acceptable string, another subset should be chosen as a candidate. The check by analysis should be tried after a transformation if it is a local or a generalized one. All the nodes, terminal and non-terminal, are sets of symbol strings. A generated string of nodes is analyzed by tracing back the path of generation. If the analysis goes back to P(O) which covers the whole string, the generation is acceptable, and not acceptable if otherwise.

Any given string can be analyzed by applying rules to the string. In this case, however, the tree structure is not known. Rules should be tested on every possible combination of terminal and non-terminal nodes, so that the whole string may be covered by a single node and the possible derivational history may be accounted for by the concatenational and transformational rules.

<u>11.3</u>. The Rules for generation and those for recognition are essentially the same. They may be prepared in terms of complete neighborhoods or distribution classes. The rules will be prepared without any formal ambiguity if their definitions are carefully observed. Some formal systems are given in the following pages as examples of simple types of grammar.

<u>12</u>. <u>Complete Neighborhood Representation of Concatenation Rules</u>.

We say a set of concatenation rules is complete if it gives the concatenation

$$z = xy$$

of any complete neighborhoods x and y of the language. It is not necessary, however, to list all the possible x's and y's. Much less number of rules can cover all the possible complete neighborhoods if their use is properly programmed.

We consider a rule f(uv;w) represents a relation between the concatenated complete neighborhoods uv and another complete neighborhood w. Each rule will give information to xy if x (=) u and y (=) v:

$$(x \cap u)(y \cap v) = xy \cap uv;$$

which is a part of xy = z.

In order to obtain the given concatenation xy, we determine a set R(xy) of rules applicable to xy. Each rule is decided whether or not it 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 term w is 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 of finding z. We have to 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):

$$R(z) = \mathrm{set}(f(uv;w)\colon h(z;w)).$$

The situation is a little complicated in this case. We can possibly expect a case where both

$$z = x_1y_1 \quad \text{and} \quad z = x_2y_2$$

are true under the condition

$$x_1 \cap x_2 = 0 \quad \text{and/or} \quad y_1 \cap y_2 = 0.$$

Note that this is not the case of formal concatenation of sets

$$AB \cap CD = (A \cap C)(B \cap D).$$

The concatenations x_1y_1 and x_2y_2 happened to be z by the syntactic reason of the language being studied. A storage space is assigned to each x_iy_i as soon as any rule in R(z) proves a possibility, and x_iy_i is modified every time a rule is applied to it. However, if

$$x_i \subseteq x_j \quad \text{and} \quad y_i \subseteq y_j,$$

then either x_iy_i or x_jy_j is just trivial. The choice depends upon the type of rules and the program which applies the rules to the text. Finally, we have a set of x_iy_i accompanied by the subset R(z;i) of R(z). Possible types of rules for this purpose will not be discussed here, because the principle is similar to the case of finding z from x and y.

In order to see some properties of rules, we assume simple forms of f(uv;w):

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

$$uv \supseteq w,$$

$$uv \subseteq w,$$

$$uv = w.$$

The condition g will be assumed simply as

$$(=), \quad \supseteq, \quad \subseteq, \quad \text{or} \quad =.$$

The condition of constituents can be replaced by a condition imposed on the whole concatenation:

(1) $xy\ (=)\ uv$

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

" $x\ (=)\ u$ and $y\ (=)\ v$;

(2) $xy \supseteq uv$

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

" $u = x \cap u$ and $v = y \cap v$

" $x \supseteq u$ and $y \supseteq v$;

(3) similarly, $xy \subseteq uv$

if and only if $x \subseteq u$ and $y \subseteq v$;

(4) $xy = uv$

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

<u>12.1</u>. Suppose we have the rules of the form

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

applicable to xy if

$$x\ (=)\ u \quad \text{and} \quad y\ (=)\ v.$$

Then, for such a rule, we have

$$xy\ (=)\ uv\ (=)\ w.$$

We can also assume the rules are applicable if

$$xy \supseteq uv,$$

$$xy \subseteq uv,$$

$$xy = uv.$$

We can not decide which part of w belongs to uv, unless some other information is available.

<u>12.2</u>. If each rule represents the relation

$$uv \supseteq w$$

then $(x \cap u)(y \cap v) = xy \cap uv \supseteq xy \cap w.$

<u>12.2.1</u>. Let the rules of the form

$$uv \supseteq w$$

be applicable to xy if and only if

$$x \supseteq u \quad \text{and} \quad y \supseteq v.$$

Then $xy \supseteq uv \supseteq w.$

This is true for any rule in

$$R(xy) = \text{set}(uv \supseteq w\text{: } xy \supseteq uv).$$

If the set R(xy) has sufficient rules to give

$$xy = \bigcup w,$$

we can find xy by simply taking the union of all the w's in R(xy).

12.2.2. If the rules are applicable to xy when

$$x \subseteq u \quad \text{and} \quad y \subseteq v,$$

then $\quad xy \subseteq uv \supseteq w.$

12.2.3. We know that a concatenation xy of any two neighborhoods is broken down to the concatenations of elementary neighborhoods e(i)e(j) and that each e(i)e(j) is represented as a union of elementary neighborhoods.

If $\quad x = e(1),$

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

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

then $\quad xy \supseteq e(5) \cup e(6).$

These rules will 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 entries. However, this type of rules is preferred when a particular technique is available on machine (Opler et al., 1963).

12.3. Let us consider a set of rules of the form

$$uv \subseteq w.$$

We assume a rule is applicable to xy if

$$x\ (=)\ u \quad \text{and} \quad y\ (=)\ v.$$

We have, then,

$$(x \cap u)(y \cap v) = xy \cap uv \subseteq xy \cap w.$$

12.3.1. Suppose the rules of the form

$$uv \subseteq w$$

are applicable to xy if and only if

$$x \supseteq u \quad \text{and} \quad y \supseteq v.$$

For all the rules in the set R(xy) of applicable rules, we have

$$xy \supseteq uv \subseteq w.$$

12.3.2. Let the set R(xy) of applicable rules be

$$R(xy) = \text{set}(uv \subseteq w\colon x \subseteq u,\ y \subseteq v).$$

Then, for each rule in R(xy), we have

$$xy \subseteq uv \subseteq w.$$

Taking all the rules in R(xy), we can expect

$$xy = \bigcap w,$$

and, if the set of rules is prepared so as to meet this condition, we can find xy by taking the intersection of w's in R(xy).

12.4. Let the rules be given in the form

$$uv = w,$$

and let R(xy) be the set of rules such that

$$x\ (=)\ u \quad \text{and} \quad y\ (=)\ v.$$

12.4.1. If R(xy) is the set of all the rules satisfying the condition

$$x \supseteq u \quad \text{and} \quad y \supseteq v,$$

then we have

$$xy \supseteq uv = w$$

for all the rules in R(xy). Then,

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

where the union is to cover all the rules in R(xy); if the rules are prepared so that

$$xy = \bigcup uv,$$

then we can find the concatenation simply by taking the union of w's of the rules in R(xy).

12.4.2. If the rules are prepared so that they may be applied to xy when

$$x \subseteq u \quad \text{and} \quad y \subseteq v,$$

then $\quad xy \subseteq uv = w.$

If $\quad xy = \bigcap uv$

is true for all the rules in R(xy), then we can find the desired concatenation by

$$xy = \bigcap w.$$

12.4.3. If the rules are represented in terms of elementary neighborhoods in the form

$$e(i)e(j) = w(i,j),$$

then, in virtue of the 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, a rule is applicable to xy if

$$x(i) = y(j) = 1.$$

The result $z = xy$ is obtained as the union of all the $w(i,j)$'s of the applicable rules:

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

12.5. The rules are prepared and used more freely according to the given condition and requirement. In the following scheme (Sakai, 1961), a complete neighborhood is represented by a code consisting of a number of digits and each digit is checked, modified and transferred independently.

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 the possible contexts. x, y and z are transferred to a temporary storage space (x_1,y_1,z_1). A rule is applicable if

$$x \ (=)\ u, \quad y \ (=)\ v \quad \text{and} \quad z \ (=)\ w,$$

and the set (x_1,y_1,z_1) is modified everytime a rule is applied. If a rule proves

$$x_1 \ (=)\ u, \qquad y_1 \ (=)\ v, \qquad z_1 \cap w = 0,$$

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). Similar procedure is repeated over again on two languages simultaneously, so that the syntactic structure can be transferred from the tree structure in one language to that of another language. The form of the tree is preserved but their nodes are marked by the labels specific to each language, input, intermediate or output language.

13. Distribution Class Representation of Concatenation Rules.

Possible concatenation of a language can be formulated as concatenated sets of strings. Let

$$R = \text{set}(r: h(r))$$

and $S = \text{set}(s\text{: } h(s))$

be sets of strings satisfying the conditions h(r) and h(s), respectively, and let their concatenation have the property k(rs), so that

$$rs \in T = \text{set}(t\text{: } k(t)).$$

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 $r \in R_h \cap R_i \cap \text{---} \cap R_k$

and $s \in S_h \cap S_i \cap \text{---} \cap S_k$,

then as many rules are applicable to rs and they give

$$rs \in T_h \cap T_i \cap \text{---} \cap T_k = T'.$$

The intersection T' has less number of elements and, if the rules are precise, the character of the strings in it is determined as precisely as required. Of course, these procedures are not to be done by listing up all the 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 the string belongs to any given set. These codes are to be generated and attached to rs to indicate that it belongs to the set T'.

Practically, it is convenient to classify the strings in terms of their complete neighborhoods:

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

A grammar of concatenation will be given as a set of rules of the form

$$R(u)S(v) \subseteq T(w)$$

with a relation

$$f(uv;w),$$

and the rules can be described in a number of different ways according to the choice of R(u)S(v), T(w) and f(uv;w). In order to see the principle, we simplify the situation by making use of the distribution classes G, H, I and J, and by assuming the relation f(uv;w) as

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

$uv \subseteq w$,

or $uv = w$.

The type of $T(w)$ is chosen so that the grammar may describe the language adequately.

13.1. G Representation.

Put

$$R(u) = G(u), \qquad S(v) = G(v).$$

If $r \in G(u), \quad s \in G(v), \quad uv\ (=)\ w$,

then $rs \in G(u)G(v) \subseteq G(uv)$,

then $C(rs)\ (=)\ uv\ (=)\ w$.

If $r \in G(u), \quad s \in G(v), \quad uv \supseteq w$,

then $rs \in G(u)G(v) \subseteq G(uv)$,

then $C(rs)\ (=)\ uv \supseteq w$.

If $r \in G(u), \quad s \in G(v), \quad uv \subseteq w$,

then $rs \in G(u)G(v) \subseteq G(uv) \subseteq G(w)$.

If $r \in G(u), \quad s \in G(v), \quad uv = w$,

then $rs \in G(u)G(v) \subseteq G(uv) = G(w)$.

Even if a few rules are applicable to rs in these cases, that is,

$$rs \in G(w_h) \cap G(w_i) \cap \text{---} \cap G(w_k),$$

we have no simple way to find $C(rs)$ from w's. We can not specify a set of less members which adequately indicates the property of rs, unless more specific information is available.

13.1.1. Suppose, however, u and v are elementary.

If $C(r)\ (=)\ u$ and $C(s)\ (=)\ v$,

then $C(r) \supseteq u, \quad C(s) \supseteq v$.

That is, $r \in G(u) = H(u), \quad s \in G(v) = H(v)$.

For further discussion, see "H Representation", where u or v is not necessarily elementary.

13.1.2. 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 $s \in I(v)$.

For further discussion, see " I Representation", where no neighborhoods are necessarily elementary.

13.2. H Representation.

Put

$$R(u) = H(u), \qquad S(v) = H(v).$$

If $r \in H(u), \quad s \in H(v),$

then $rs \in H(u)H(v) \subseteq H(uv).$

<u>13.2.1</u>. If $uv\ (=)\ w,$

then $rs \in H(u)H(v) \subseteq H(uv),$

then $C(rs) \supseteq uv\ (=)\ w,$

then $C(rs)\ (=)\ w,$

then $rs \in G(w).$

We put

$$T(w) = G(w).$$

However, there is no simple procedure of finding the intersection of G(w)'s. We can not specify the features of the strings by finding more rules applicable to rs, unless more specific information is available.

<u>13.2.2</u>. If $uv \supseteq w,$

then $rs \in H(u)H(v) \subseteq H(uv) \subseteq H(w),$

because $H(uv) = H(w \cup w') = H(w) \cap H(w') \subseteq H(w).$

We put $T(w) = H(w)$

to have the rules of the form

$$H(u)H(v) \subseteq H(w).$$

If a number of rules are applicable and

$$rs \in H(u_h)H(v_h) \subseteq H(w_h)$$

$$rs \in H(u_i)H(v_i) \subseteq H(w_i)$$

--- --- ---

$$rs \in H(u_k)H(v_k) \subseteq H(w_k),$$

then $rs \in H(w_h) \cap H(w_i) \cap \text{---} \cap H(w_k)$

$= H(w_h \cup w_i \cup \text{---} \cup w_k),$

then $C(rs) \supseteq w_h \cup w_i \cup \text{---} \cup w_k.$

The rules of this type are essentially the same as the rules of complete neighborhoods

$$xy \supseteq uv \supseteq w,$$

although they are encoded as the sets of strings:

<u>13.2.3</u>. If $uv \subseteq w,$

then $C(rs) \supseteq uv \subseteq w,$

then $rs \in G(w).$

<u>13.2.4</u>. Put

$$uv = w.$$

Then $rs \in H(u)H(v) \subseteq H(uv) = H(w).$

The situation is the same as the case above, where $uv \supseteq w$.

13.3. I Representation.

Put

$$R(u) = I(u), \qquad S(v) = I(v).$$

If $r \in I(u), \quad s \in I(v),$

then $rs \in I(u)I(v) \subseteq I(uv).$

13.3.1. If $uv \ (=) \ w,$

then $C(rs) \subseteq uv \ (=) \ w.$

No relationship is relevant between $C(rs)$ and w.

13.3.2. If $uv \supseteq w,$

then $C(rs) \subseteq uv \supseteq w.$

No definite $T(w)$ is available, such that $I(u)I(v) \subseteq T(w)$.

13.3.3. We consider the rules of the type

$$I(u)I(v)$$

with $uv \subseteq w.$

If $r \in I(u), \quad s \in I(v),$

then $rs \in I(uv) \subseteq I(w).$

If a number of rules are applicable to rs,

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

Therefore, the rules of this type are equivalent to those of the type

$$xy \subseteq uv \subseteq w.$$

13.3.4. Put

$$uv = w.$$

Then $rs \in I(u)I(v) \subseteq I(uv) = I(w).$

This is the same to the case mentioned above.

13.4. J Representation.

Put

$$R(u) = J(u), \qquad S(v) = J(v).$$

This type of grammar is not practical because every real distribution class J of the language must be listed in the rules. This condition corresponds to the complete neighborhood representation of rules $f(uv;w)$ applicable to xy only if

$$x = u \qquad \text{and} \qquad y = v.$$

13.5. Practically, the rules can be written more freely and the program can be more flexible and efficient, provided that a more sophisticated scheme is introduced to the G Representation and the condition $f(uv;w)$. This is realized by representing the sets of strings by codes, so that the union and the intersection of any two sets are determined by the operation on the codes.

14. Some Remarks on Transformation.

14.1. It is generally agreed that we generate acceptable strings by starting with an axiom and expanding it repeatedly into a string of constituents. This procedure is taken care of by concatenation rules. After generating one or more strings by this procedure, they are transformed to yield another string.

Let us imagine another function of our normative device. We give it a pair

$$r = (r', r'')$$

of acceptable strings

$$r' = r'(1)r'(2)---r'(i')---r'(m')$$

and $$r'' = r''(1)r''(2)---r''(i'')---r''(m'').$$

The pair r will be referred to as a string

$$r = r(1)r(2)---r(i)---r(m)$$

with $$m = m' + m''.$$

We put $$m'' = 0$$

if the string r'' is absent. We then give it another acceptable string

$$s = s(1)s(2)---s(j)---s(n),$$

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 consider the string s is generated from r by a transformation. We call r the original string and s its transform. If it says "no", no such transformation exists. Conversely, we ask it whether or not r' and r'' are true if s is true. If the device says "yes", we consider an inverse transformation exists, such that s is expressed by r' and r''. 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 can not be retrieved unless appropriate, possibly non-linguistic, information is supplied. This situation is beyond the scope of syntactics.

A transformation or an inverse transformation is called singularly if r'' in r is absent, and it is a generalized one if both r' and r'' are present. If it is an embedding transformation, r' and r'' are called matrix and constituent strings, respectively.

If we understand the transformation in the sense mentioned above, the transfer of syntactic structure from one language to another is also a transformation (Gross, 1962).

14.2. If it is known that r is transformed to s, then this fact is used to generate a particular string. If r is known to be an inverse transform of s, then this is used to recognize s, giving a possible derivational history. If no other such transformations are found, r is the only nearest history. Otherwise, the ambiguous history is to be accounted for by other rules.

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 and write

r eqv s.

Obviously, this equivalence is symmetric, reflexive, and transitive. A transformation that transforms a string into an equivalent string is called an equivalence transformation. If we have a grammar consisting of equivalence transformations only, it can be used for both synthesis and analysis.

Let us confine ourselves to the equivalence transformations in order to simplify the discussion, and assume we have a set of rules or a normative device. 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 eqv (r(1),r(2));
r(1) eqv (r(11),r(12));
r(2) eqv (r(21),r(22));
r(11) eqv (r(111),r(112));
r(12) eqv (r(121),r(122));
     ---   ---   ---   .
```

If an acceptable string t can no longer be dissolved into two acceptable strings, we call t a terminal or an atomic acceptable string. Throughout this procedure, the strings are expected to become shorter and simpler, because equivalent information is expressed by many separate strings. It will be still possible to transform an atomic string to another atomic string by means of a singulary transformation. We have different atomic strings which are mutually equivalent. We may pick up one of them and call it a kernel string.

The sequence of inverse transformations is not always uniquely determined. There can be other orders of dissolving a given string into atomic strings. We can make the grammar less redundant by studying the possible sequences of inverse transformations. If the rules are all equivalence rules, there is no theoretical problem of ambiguity. The investigation of these problems requires quite a different treatment, and will not be included in this paper.

14.3. Sometimes, it is considered more linguistically reasonable to assume that a string is not acceptable but its transform is an acceptable string or a constituent of an acceptable string. In some other cases, a string may be an acceptable string and its transform may not be an acceptable string or a constituent thereof. In other words, a transformation is applied to an unacceptable string or a transformation results in an unacceptable string. We may prepare the rules in such a way that a sequence of obligatory transformations is contracted to a single rule. This seems formally simpler and consistent. However, it will result in a more entangled system of grammar. We admit some of such strings as potentially acceptable and indicate it by 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 a string of a Chinese dialect marked potentially acceptable for the derivation of apparently inconsistent strings is quite acceptable in another dialect (Wang, 1964).

14.4. A generalized transformational rule consists of terms u and v, where

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

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 subroutines are to be prepared for them. Some of the operations are listed below, which have been picked up sporadically from the rules for generating Chinese strings (Hasimoto, 1964).

(0) A routine supervising the subroutines takes care of the whole procedure of applying the rules to a string. If the rules are prepared in a definite format, they are automatically checked and applied to the given string.

(1) Certain segments r(h) and r(i) in the original string must or must not share a certain feature in common and/or a segment 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 specified by the rule.

(3) Some segments in the transform must satisfy the 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 the rule.

(7) The tree structure must or must not be changed by the final procedure of a transformation.

<u>14.5</u>. No rule describes a transformation of an individual string r into an individual string s. The rule says, if the string r has the feature

$$u = u(1)u(2)---u(i)---u(m),$$

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

$$v = v(1)v(2)---v(j)---v(n).$$

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 decided to be used, the program is <u>the</u> definition. If the program is modified, the rules and the lexicon are to be modified.

Since the transformations are applied to P-markers, a string is considered to be a tree-like string. If it is a linear string of terminal nodes, the other non-terminal nodes and the branches are to be determined by virtue of the concatenation rules. We consider the labels u(i) and v(j) 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, u(i)'s and v(j)'s are considered to be distribution classes.

<u>14.6</u>. The complete neighborhoods are defined on the basis of concatenated strings and we have to associate them with the labels given to the nodes of our transformational rules in order that the kernel strings can be transformed. Let us see what happens when the nodes are assumed to be complete neighborhoods.

Let

$$p = (p', p'')$$

be a pair of acceptable strings p' and p'', and let

$$r = r(1)---r(i)---r(m)$$

be a segment of p. The pair p is transformed by T into

$$q = T(p),$$

and the segment appears in q as

$$s = s(1)---s(j)---s(n).$$

Some strings may have been added and some others may have been deleted.
Put

$$x(i) = C(r(i)),$$
$$x = C(r),$$
$$y(j) = C(s(j)),$$
$$y = C(s).$$

By definition,

$$x = x(1)---x(i)---x(m),$$
$$y = y(1)---y(j)---y(n).$$

Any string belongs to one and only one distribution class J. Therefore, instead of

$$T(r(1)---r(i)---r(n)) = s(1)---s(j)---s(n),$$

we write

$$T(J(x(1))---J(x(i))---J(x(m)))$$
$$= J(y(1))---J(y(j))---J(y(n)).$$

Since all the elements in a J has the same complete neighborhood, we rewrite the above as

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

This is rewritten again by breaking down in the form

$$x = x(1)---x(i)---x(m),$$
$$y = T(x)$$
$$= y(1)---y(j)---y(n).$$

If we have a complete set of rules which gives the concatenation of any complete neighborhoods of the language, then we can find the complete neighborhood x. The transformation takes place when x is changed to y. The string y is to be generated in 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

$$x(1)---x(i)---x(m)$$

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

$$y(1)---y(j)---y(n).$$

This interpretation, however, suggests a few problems.

14.7. We know that

$$J(x(1))---J(x(i))---J(x(m)) \subseteq J(x),$$
$$J(y(1))---J(y(j))---J(y(n)) \subseteq J(y).$$

The statement "x is transformed to y" is a generalization of the original fact, and this generalization is not always true. The text should be checked before a transformational rule is applied to it. Some separate steps for this purpose will save the machine time.

(1) A text to be parsed must consist of segments specified by the rule. The correct segmentation can be done by finding the tree structure of the text. Therefore, the concatenation rules must be prepared so as to account for the structure of any acceptable string.

(2) Not all the trees of the specified form undergo the inverse transformation so that the derivational history may be traced back. The nodes are labeled. A tree of a form can correspond to a number of trees whose nodes have different labels.

(3) When a string is being synthesized, the text is given as a pair of P-markers. A rule can be applied only if the P-markers meet the condition specified by the rule.

14.8. We may regard the structure mentioned above as a representation of derivational history. The history can be recorded by listing all the derivational steps the string has experienced. This representation, however, will be redundant and inefficient, because it is likely to occur that an identical series of transformations is applied to strings of different history. On the other hand, it is also possible that the strings p and q of different histories result in an identical string s by a transformation and the string s is ambiguous in that the s from p can undergo a sequence of transformations and the s 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 in the P-marker to which the rules are applied. These features should correspond to the series of transformations applicable to the P-marker in case of synthesis and the series of inverse transformations in case of analysis. We have some rules with notes on the type of transformations to which the resultant strings may be exposed (Hasimoto, 1964).

15. Complete Neighborhoods and Transformational Rules.

Let us assume u(i)'s and v(j)'s are complete neighborhoods.

15.1. Two strings r and s may replace the same non-terminal node to yield a longer acceptable string. However, when a transformation T is to be applied, they must have the specified structure; thus the string p with r as a segment in it may be transformed by T, while the string q which differs from p only in that it has the segment s in the place of r may not. The lack of q by T means

$$C(r) \neq C(s).$$

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 refine the definition of complete neighborhood in such a way that C(r) of a string r is the set of all contexts of r which appear in the strings to which no transformations have ever been applied during their derivation. The difference between r and s is found in their internal structure, if the machine is given only the input string to be parsed. In order to indicate this difference, we put

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

where C(r) is defined over the set of kernel strings,
D(r) is defined over the set of transforms,
E(r) is defined over the set of kernel strings and transforms.

Let c(i) be an elementary neighborhood defined over the set of kernel strings, and let r be a real or imaginary string such that

$$C(r) = c(i).$$

Let d(i;j) be the elementary neighborhood defined over the set of all the possible transforms of which r is a segment, where j corresponds to the possible sequence of transformations. Putting

$$c(i) \cup d(i;j) = e(i;j),$$

we have the elementary neighborhood e(i;j) defined over the set of kernel strings and transforms. These e(i;j)'s are no longer necessarily disjoint:

$$e(i;j) \cap e(i;j') \supseteq c(i).$$

15.3. The separation of kernel strings and transforms still involves a considerable complexity. Let q be a transform. It is a transform generated by a transformation in a sequence of transformations and it can be an original string to be transformed by the following transformation.

A transformation is accompanied by the set P of original strings and the set Q of transforms:

$$P = \text{set}(p\text{: T is applicable to } p),$$

$$Q = \text{set}(q: q = T(p),\ p \text{ in } P).$$

We simplify the situation by defining the complete neighborhoods over P and over Q. The feature of T is shown more explicitly in this way. Let A be a node and imagine a derivation by the context sensitive rules

$$A \longrightarrow BC$$
$$B \longrightarrow F \ / \ \text{---}C$$
$$C \longrightarrow G \ / \ B\text{---}$$

where the symbols are assumed to be complete neighborhoods. Let B be replaced by F first to yield FC, and the third rule can no longer be applied because of the lack of its necessary environment B---. When these rules are to be used in analysis, none of the contexts ---C or B--- is relevant in the given string FG of complete neighborhoods. We can get rid of this difficulty by defining B and C over a set of strings and F and G over another, and by considering a transformation from BC to FG, prohibiting the operations on the strings FC and BG.

Let

$$p = p(1)\text{---}p(i)\text{---}p(m)$$

be a string in P, and let

$$q = q(1)\text{---}q(j)\text{---}q(n) = T(p)$$

be the transform of p by T. We define the complete neighborhood of p(i) over P and that of q(j) over Q. By modifying the meaning of the notation, we put

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

The requirement that p(i) should appear 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 Q, then

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

if q(j) does not occur in P, then

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

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)).

The set Q can include a part of the set P' of original strings to which another transformation T' can be applied. Thus, we can classify the strings with respect to possible transformations. We have no positive grounds to assume any natural language has a stratified system of layers arranged one over another.

15.4. Let

u = (u',u")
 = u(1)---u(i)---u(m)

be a pair of concatenations

u' = u'(1)---u'(i')---u'(m')

and u" = u"(1)---u"(i")---u"(m")

of complete neighborhoods u'(i')'s and u"(i")'s defined over P. If the string is linear, the non-terminal nodes are to be determined by concatenation rules. We assume the rules of the form

f(T(u);v)

mean, over Q, a relation between T(u) and v. We assume further a rule is applicable to the given pair of concatenated complete neighborhoods

x = (x',x")
 = x(1)---x(i)---x(m)

if the condition g(x;u) holds. That is,

if g(x;u) over P,

then f(T(u);v) over Q.

We expect to find the transform T(x) in terms of v of the rules in the set

R(x) = set(f(T(u);v): g(x;u))

of the applicable rules.

Given the rules of the same form and a string represented by a concatenation

y = y(1)---y(j)---y(n)

of complete neighborhoods, an inverse transformation is to be carried out by finding the set

R(y) = set(f(T(u);v): h(y;v))

of applicable rules.

With all the linguistic difference between the concatenation rules and transformational rules, they exhibit formal similarities when the labels are assumed to be the sets of contexts. We will not repeat a similar discussion on the choice of $f(T(u);v)$, $g(x;u)$, $h(y;v)$ or the algorithm for finding x or y.

16. Distribution Classes and Transformational Rules.

Let p be a string and T(p) its transform by the transformation T. Let P be a set of strings p to which T is applicable. We defined the transform T(P) of P as the set of all T(p)'s:

$$T(P) = \text{set}(T(p): p \text{ in } P).$$

A rule will be written in the form

$$f(T(P);Q)$$

to indicate a relation between the sets T(P) and Q.

In order to specify the sets a little closer to the form of rules usually prepared by linguists, we put

$$p = p(1)p(2)\text{---}p(i)\text{---}p(m)$$
$$q = q(1)q(2)\text{---}q(j)\text{---}q(n),$$

where p(i)'s and q(j)'s are segments in p and q, respectively. Then we put

$$P = P(1)\text{---}P(i)\text{---}P(m)$$
$$Q = Q(1)\text{---}Q(j)\text{---}Q(n),$$

which are to be understood as concatenated sets of strings.

A rule of the form f(T(P);Q) is applicable to the string p, if

$$p(i) \in P(i) \quad \text{for} \quad i = 1, 2, \text{---}, m,$$

giving $T(p) \in T(P)$,

so that $f(T(P);Q)$

provides us with the information governed by this rule. Each string in the lexicon and each constituent in the string under analysis or synthesis is given a marker which indicates whether or not it belongs to any set of strings, provided that the sets are established systematically. Because of the ambiguous property of real strings, the markers will be given in terms of complete neighborhoods defined over the set of (potentially) acceptable strings.

17. Establishment and Representation of Complete Neighborhoods.

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

17.1. In establishing a set of complete neighborhoods of a natural language, we assume a few of them as undefined terms and derive the others by hypothetical concatenation rules. Sometimes, there will be a choice among a few hypothetical rules. We take one of them to define a complete neighgorhood 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. This method is applied not only to phrase structure grammar but also to transformational grammar, because both transformations and inverse transformations are applied to a (pair of) P-marker(s) to yield another (pair of) P-marker(s).

Every time a definition rule is established as a hypothesis, it must be tested as to whether or not it contradicts any other definition rules. No property rules should contradict any other rules. Whenever a contradiction is found, the source of trouble must be found out by tracing back the definition rules, and the hypothesis that has given rise to the trouble must be modified.

17.2. The complete neighborhoods of all the acceptable strings (as distinguished from the other ambiguous interpretations of the same string) are identical to each other and consist of one element indicating that the strings are acceptable. It seems adequate, for most of the natural languages, to admit two complete neighborhoods, nominals and verbals, although there are no rigid grounds. Many others are derived from hypothetical concatenations that can occur in acceptable strings.

The prepositions in many European languages are subclassified by the case of the nominals they govern, and the nominals by their case, gender and number. A rule for yielding prepositional phrases will be stated as follows: a preposition that governs nominals of case c, followed by a nominal of 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 to describe 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

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

The indices g and n are arbitrary if the preposition in question takes nominals of any gender and of any number.

Usually, a linguist will define complete neighborhoods broadly so that the majority of acceptable strings may 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 out 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 will indicate that the rules are not always true. He may 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'.

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

$$v = v(p_1) \cup v(p_2) \cup --- \cup v(p_n).$$

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

$$v(0) = \bigcup v(p_i), \qquad i = 1, 2, ---, n.$$

The union of a few subsets are written as

$$v(p_1, p_3, p_5) = v(p_1) \cup v(p_3) \cup v(p_5),$$

etc.

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

$$v(p;q), \qquad v(p;q;r), \qquad \text{etc};$$

$$v(p_1,p_2;q) = v(p_1;q) \cup v(p_2;q),$$

$$v(p;q_1,q_2) = v(p;q_1) \cup v(p;q_2),$$

$$v(p;0) \cap v(0;q) = (\bigcup v(p;q_j)) \cap (\bigcup v(p_i;q))$$

$$= v(p;q),$$

etc.

Hence, for the distribution classes

$$H(v(p_1,p_2;q)) = H(v(p_1;q)) \cap H(v(p_2;q)),$$

$$I(v(p;q)) = I(v(p;0)) \cap I(v(0;q)),$$

etc.

Sometimes, an index depends upon other indices:

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

for example. The meanings of r, s and t depend upon the meaning of q.

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

$$(a;b;c;---;n),$$

where the broad class symbol is one of the indices and each index represents a classification from a certain point of view.

It will be of interest to compare these indices with the concept of "razbijenije", "okrjestnostj" (Kulagina, 1958) or "sememe" (Lamb, 1962). This kind of representation, used by many research groups, enables us to describe the syntax of a language systematically. Each digit can be regarded as an indication of a certain feature common to some elementary neighborhoods, and classifies them according to their specific features.

<u>17.4</u>. Suppose a concatenation rule f(uv;w) is to be applied to a text xy of complete neighborhoods to determine z = xy, and the complete neighborhoods are represented by the indices in the form

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

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

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

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 as for the other indices. This information should not be lost if it is 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 (Sakai, 1961).

A transformational rule requires that certain features of the original string are 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, if the rules are applicable to the string regardless of a certain index, 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.

17.5. An extremely simplified example is given. The complete neighborhoods are no longer treated 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 strings (h,i) and (j,k). The following abbreviations are used.

adj: adjective
adj-pred: adjectival predicate
anim: animate
compl: complement
inanim: inanimate
m: masculine
n: nominal
n/n: modifier of nominal
nom: nominative
pl: plural
pn: pronoun
s: sentence
v: verbal
-k: ends with k
-t: ends with t

Input Language (English)

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

(3,4)(n;pl) = (3,3)(adj) * (4,4)(n;pl)

Intermediate Representaion.

(1,4)(v) = (1,1)(pn;3rd;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;3rd;pl;nom) = on(pl;nom) = oni

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

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

(4,4)(plane)(n;compl;pl) = (rubank(-k) + samoljet(-t))(n;m;pl;nom)

    = (rubanki + samoljety)(n;m;pl;nom)

(3,4)(n;compl;pl) = (3,3)(adj;hard) * (4,4)(n;m;pl;nom) = (3,3)-yje(4,4)

(1,4)(v) = (1,1)(2,2)(3,4) = oni krasnyje (rubanki + samoljety)

Output Language (Japanese)

(1,1)(pn;3rd;pl;nom) = (kare(anim) + sore(inanim))(pn;pl;nom)

(2,2)(copula;pres) = ar(v;4;pres;final) = ar-u

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

(4,4)(plane)(n;compl;pl) = (heimen + hikooki)(n;inanim;compl)

(3,4)(n;compl;pl) = ((3,3)-i(4,4))(n;inanim)-de

(1,4)(v) = (1,1)(anim,inanim;pn;pl;nom) * (3,4)(n;inanim)-de *

            (2,2)(v;4;pres;final)

    = (1,1)(inanim;pn;pl;nom) * (3,4)(n;inanim)-de * (2,2)(v;4;pres;final)

    = sorera (ga + wa) akai (heimen + hikooki) de aru

17.6. We observe in the above example that the index of an animate or an inanimate object affects the choice of a lexical element in Japanese while it is not relevant in English. 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 can not be always preserved in the transfer process from one language to another. We will have to prepare two separate procedures for handling gender. Similar problems arise with respect to other indices.

The choice of lexical elements depends greatly upon the habitual usage of language. The situation is similar when we observe some combinations of longer constituents. The choice of constituents is limited by logical, semantic or habitual reasons as indicated by the branches of the second kind in the net strings. Sometimes the choice is quite capricious. It seems more practical to handle this kind of information separately (Matthews, 1965), corresponding to the separate normative devices the linguist has conjectured.

Acknowledgment.

The need of defining 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, and was refined and finished at the Project on Linguistic Analysis, Ohio State University. I appreciate the encouragement of these organizations.

References.

Gross, M.: On the Equivalence of Models of Languages Used in the Fields of Mechanical Translation and Information Retrieval, NATO Advanced Study Institute on Automatic Translation of Languages, Venice, 1962.

Hasimoto, A. Y.: Revised Rules of Mandarin Grammar, Project on Linguistic Analysis, Ohio State University, Columbus, Ohio, 1964.

Kulagina, O. S.: Ob Odnom Sposobje Oprjedjeljenija Grammatičeskix Ponjatij na Bazje Tjeorii Množestv, Probljemy Kibjernjetiki, Vypusk 1, Moskva, 1958.

Lamb, S. M.: Outline of Stratificational Grammar, University of California, Berkeley, California, 1962.

Matthews, P. H.: Problems of Selection in Transformational Grammar, private circulation, Indiana University, to appear in the Journal of Linguistics, No. 1, 1965.

Opler, A.; Silverstone, R.; Saleh, Y.; Hildebran, M.; Slutzky, I.: The Application of Table Processing Concept to the Sakai Translation Technique, Mechanical Translation, vol. 7, No.2, 1963.

Parker-Rhodes, A. F.: A New Model of Syntactic Description, 1961 International Conference on Machine Translation of Languages and Applied Language Analysis, Her Majesty's Stationary Office, London.

Sakai, I.: Syntax in Universal Translation, 1961 International Conference (See above).

Wang, W. S.: Two Aspect Markers in Mandarin, Project on Linguistic Analysis (See above), Report No. 8, 1964.

Appendices.

A-1. Sets.

$a \in A$; a in A: a is an element of the set A; a belongs to A; a is in A.

$a \notin A$; a not in A: $a \in A$ is not true.

A (=) B: there is at least one element which belongs to both A and B.

$A \subseteq B$; $B \supseteq A$: if $a \in A$, then $a \in B$; A is a subset of B; B is a superset of A.

$A = B$: $a \in A$ if and only if $a \in B$; $A \subseteq B$ and $A \supseteq B$.

$A \neq B$: $A = B$ is not true.

$A = 0$: there is no element in the set A; the set A is empty.

A = set(a,b,c,d): A is a set whose elements are a,b,c and d.

A = set(a_i: i = 1,2,---): A = set(a_1,a_2,---).

A = set(a: f(a)): $a \in A$ if and only if f(a) is true.

$A = B \cup C$: A = set(a: $a \in B$ or $a \in C$); A is the union of B and C.

$A = \bigcup B_i$, i = 1,2,---: $A = B_1 \cup B_2 \cup$ --- .

$A = \bigcup B$ for f(B): A is the union of all B's satisfying f(B).

$A = B \cap C$: A = set(a: $a \in B$ and $a \in C$); A is the intersection or meet of B and C.

$A = \bigcap B_i$, i = 1,2,---: $A = B_1 \cap B_2 \cap$ --- .

$A = \bigcap B$ for f(B): A is the intersection of all B's satisjying f(B).

A-2. Boolean Coefficients.

We introduce coefficients which indicate 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 $a + b$ and the product $ab = a \times b$ are determined by

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

and

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

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

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

Table of Contents