Chapter 16

Basic Concepts

16.1 Languages, grammars and automata

At one level of description, a natural language is simply a set of strings–finite sequences of words, morphemes, phonemes, or whatever. Not every possible sequence is in the language: we distinguish the *grammatical* strings from those that are *ungrammatical*. A *grammar*, then, is some explicit device for making this distinction; it is, in other words, a means for selecting a subset of strings, those that are grammatical, from the set of all possible strings formed from an initially given alphabet or vocabulary.

In this chapter we will consider two classes of formal devices which can function as grammars in this very general sense: (1) automata, which are abstract computing machines, and (2) string rewriting systems, which generally bear the name "grammar" or "formal grammar". The latter will be familiar to linguists inasmuch as grammars in this sense have formed the basis of much of the work in generative transformational theory.

We begin by considering certain properties of strings and sets of strings. Given a finite set A, a *string on* (or *over*) A is a finite sequence of occurrences of elements from A. For example, if $A = \{a, b, c\}$, then *acbaab* is a string on A. Strings are by definition finite in length. (Infinite sequences of symbols are also perfectly reasonable objects of study, but they are not suitable as models for natural language strings.) The set from which strings are formed is often called the *vocabulary* or *alphabet*, and this too is always assumed to be finite. The length of a string is, or course, the number of occurrences of symbols in it (i.e., the number of tokens, not the number of types). The string *acbaab* thus is of length 6.

Because we are dealing with tokens of an alphabet, there is an important difference between the linearly ordered sequences we call strings and a linearly ordered set. If the set $A = \{a, b, c\}$ were linearly ordered, say, as $b \rightarrow a \rightarrow c$, each element of A would occupy a unique place in the ordering. In a string, e.g., *acbaab*, tokens of a occur in the first, fourth, and fifth positions.

To be formal, one could define a string of length n over the alphabet A to be a function mapping the first n positive integers into A. For example, *acbaab* would be the function $\{\langle 1, a\rangle, \langle 2, c\rangle, \langle 3, b\rangle, \langle 4, a\rangle, \langle 5, a\rangle, \langle 6, b\rangle\}$. There is little to be gained in this case by the reduction to the primitives of set theory, however, so we will continue to think of strings simply as finite sequences of symbols. A string may be of length 1, and so we distinguish the string b of length 1 from the symbol b itself. We also recognize the (unique) string of length 0, the *empty string*, which we will denote e (some authors use Λ). Two strings are identical if they have the same symbol occurrences in the same order; thus, *acb* is distinct from *abc*, and strings of different length are always distinct.

An important binary operation on strings is concatenation, which amounts simply to juxtaposition. For example, the strings *abca* and *bac* can be concatenated, in the order mentioned, to give the string *abcabac*. Sometimes concatenation is denoted with the symbol " $\frown$ " thus, *abca* $\frown$ *bac*. Concatenation is associative since for any strings $\alpha, \beta, \gamma, (\alpha \frown \beta) \frown \gamma = \alpha \frown (\beta \frown \gamma)$, but it is not commutative, since in general $\alpha \frown \beta \neq \beta \frown \alpha$. The empty string is the identity element for concatenation; i.e., for any string α, $\alpha \frown e = e \frown \alpha = \alpha$.

Given a finite set A, the set of all strings over A, denoted A^*, together with the operation of concatenation constitutes a monoid. Concatenation is well-defined for any pair of strings in A^* and the result is a string in A^*; the operation is associative; and there is an identity element. $\langle A^*, \frown \rangle$ fails to be a group since no element other than e has an inverse: no string concatenated with a non-empty string x will yield the empty string. Since concatenation is not commutative, $\langle A^*, \frown \rangle$ is not an Abelian monoid.

A frequently encountered unary operation on strings is reversal. The reversal of a string x, denoted x^R, is simply the string formed by writing the symbols of x in the reverse order. Thus $(acbab)^R = babca$. The reversal of e is just e itself. To be formal, we could define reversal by induction on the

length of a string:

DEFINITION 16.1 *Given an alphabet A:*

(1) *If x is a string of length 0, then $x^R = x$ (i.e., $e^R = e$)*

(2) *If x is a string of length $k+1$, then it is of the form wa, where $a \in A$ and $w \in A^*$; then $x^R = (wa)^R = aw^R$.*

■

Concatenation and reversal are connected in the following way: For all strings x and y, $(x \frown y)^R = y^R \frown x^R$. For example,

(16–1) $(bca \frown ca)^R = (ca)^R \frown (bca)^R = ac \frown acb = acacb$

Given a string x, a *substring of* x is any string formed from contiguous occurrences of symbols in x taken in the same order in which they occur in x. For example, *bac* is a substring of *abacca*, but neither *bcc* nor *cb* is a substring. Formally, y is a substring of x iff there exist strings z and w such that $x = z \frown y \frown w$. In general, z or w (or both) may be empty, so every string is trivially a substring of itself. (Non-identical substrings can be called *proper* substrings.) The empty string is a substring of every string; i.e., given x we can choose z in the definition as e and w as x so that $x = e \frown e \frown x$.

An initial substring is called a *prefix*, and a final substring, a *suffix*. Thus, *ab* is a (proper) prefix of *abacca*, and *cca* is a (proper) suffix of this string.

We may now define a *language* (over a vocabulary A) as any subset of A^*. Since A^* is a denumerably infinite set, it has cardinality $\aleph_0$; its power set, i.e., the set of all languages over A, has cardinality $2^{\aleph_0}$ and is thus non-denumerably infinite. Since the devices for characterizing languages which we will consider, *viz.*, formal grammars and automata, form denumerably infinite classes, it follows that there are infinitely many languages–in fact, non-denumerably infinitely many–which have no grammar. What this means in intuitive terms is that there are languages which are such motley collections of strings that they cannot be completely characterized by any finite device. The languages which *are* so characterizable exhibit a certain amount

of order or pattern in their strings which allows these strings to be distinguished from others in A^* by a grammar or automaton with finite resources. The study of formal languages is essentially the investigation of a scale of complexity in this patterning in strings. For example, we might define a language over the alphabet $\{a, b\}$ in the following way:

(16–2) $L = \{x \mid x$ contains equal numbers of a's and b's (in any order)$\}$

We might then compare this language with the following:

(16–3) $L_1 = \{x \in \{a, b\}^* \mid x = a^n b^n (n \geq 0)\}$, i.e., strings consisting of some number of a's followed by the same number of b's
$L_2 = \{x \in \{a, b\}^* \mid x$ contains a number of a's which is the square of the number of b's$\}$

Is L_1 or L_2 in some intuitive sense more complex than L? Most would probably agree that L_2 is a more complex language than L in that greater effort would be required to determine that the numbers of a's and b's stood in the "square" relation than to determine merely that they were equal. In other words, a device which could discriminate strings from non-strings of L_2 would have to be more powerful or more "intelligent" than a device for making the comparable discrimination for L.

What of L_1 and L? Here our intuitions are much less clear. Some might think that it would require a less powerful device to recognize strings in L reliably than to recognize strings in L_1; others might think it is the other way around or see no difference. As it happens, the particular scale of complexity we will investigate (the so-called Chomsky Hierarchy) does regard L_2 as more complex than L but puts L_1 and L in the same complexity class. At least this is so for the overall complexity measure. Finer divisions could be established which might distinguish L_1 from L.

One linguistic application of these investigations is to try to locate natural languages on this complexity scale. This is part of the overall task of linguistics to characterize as precisely as possible the class of (potential and actual) natural languages and to distinguish this class from the class of all language-like systems which could not be natural languages. One must keep clearly in mind the limitations of this enterprise, however, the principal one being that languages are regarded here simply as string sets. It is clear that sentences of any natural language have a great deal more structure than simply the concatenation of one element with another. Thus, to establish a

complexity scale for string sets and to place natural languages on this scale may, because of the neglect of other important structural properties, be to classify natural language along an ultimately irrelevant dimension. Extending results from the study of formal languages into linguistic theory must therefore be done with great caution.