CHAPTER SIX: INTRODUCTION TO FORMAL LANGUAGES

It is a familiar and basic intuition that language somehow involves stringing things together. Examples include stringing phonemes together to form syllables or (phonologies of) morphemes, stringing morphemes together to form words, stringing words together to form phrases (including sentences), and stringing sentences together to form discourses. Indeed, in the early days of syntactic theory (early to mid 1950s), natural languages were modelled as sets of strings, and the notion of a grammar was identified with a mathematical device for listing the members of such sets. But what exactly is a string?

1 Strings

As we saw in Chapter Four, for every natural number n,

$$n = \{m \in \omega \mid m < n\}$$

Let us now consider, for some set A and some $n \in \omega$, the set A^n, i.e. the set of arrows (functions with specified codomains) from n to A. The members of this set are called the **A-strings of length n**. In a linguistic application, we would think of the members of A as linguistic entities of some kind (phonemes, morphemes, words, etc.) that we would like to be able to string together, and of a particular A-string of length $n > 0$, f, as one of the possible results of stringing n such entities together, namely the one starting with $f(0)$, then $f(1)$, then $f(2)$, etc. If $f(i) = a_i$ for all $i < n$, then we usually denote f by the string (in the informal sense) of symbols $a_0 \ldots a_{n-1}$. (But in working with strings, it is important to remember that, technically, a string is not really a bunch of symbols lined up from left to right on a page, but rather a function whose domain is a natural number.) Also, it's important to note that there is exactly one A-string of length 0, denoted by ϵ_A (or just ϵ when no confusion is possible).[1] The set of all A-strings of length greater than 0 is denoted by A^+.

For strings of length 1, a mild notational confusion arises: if $f \colon 1 \to A$ and $f(0) = a$, then the notation 'a' could refer to either a itself (a member of A), or to the length-one A-string f. It should be clear from context which

[1]In Chapter Three, we called this $\Diamond_A$, but the name ϵ_A is more usual when we are thinking of it as a string.

is intended. Note also that an A-string of length one is the same thing as a nullary operation on A.

The "infinite counterpart" of an A-string is called an **infinite A-sequence**; technically, an infinite A-sequence is a function from ω to A.

The set of all A-strings, i.e. the union of all the sets A^n, for all $n \in \omega$, is written A^*. Thus $A^* = A^+ \cup \{\epsilon_A\}$. When the identity of the set A is clear from context, we usually speak simply of strings, rather than A-strings. It should be obvious that there is a function from A^* to ω that maps each string to its length, and that the relation on strings of having the same length is an equivalence relation. Of course the sets in the partition induced by that equivalence relation are just the sets A^n. If A is a subset of another set B, then clearly there is an injection $\eta\colon A^* \to B^*$ that maps each A-string to a B-string just like it except that its codomain is B instead of A.

For each $n \in \omega$, there is an obvious bijection from $A^{(n)}$ to A^n. For $n \geq 2$, the bijection maps each n-tuple $\langle a_0, \ldots, a_{n-1}\rangle$ to the string $a_0 \ldots, a_{n-1}$. For $n = 1$, it maps each $a \in A$ to the length-one string that maps 0 to a; and for $n = 0$, it is the (only) function from 1 to $\{\epsilon_A\}$, i.e. the function that maps 0 to ϵ_A.

The binary operation of **concatenation** on A^*, written $\frown$, can be described intuitively as follows: if f and g are strings, then $f \frown g$ is the string that "starts with f and ends with g." More precisely, for each pair of natural numbers $\langle m, n\rangle$, if f and g are strings of length m and n respectively, then $f \frown g$ is the string of length $m + n$ such that

1. $(f \frown g)(i) = f(i)$ for all $i < m$; and
2. $(f \frown g)(m + i) = g(i)$ for all $i < n$.

It can be proven inductively (though the details are quite tedious) that for any strings f, g, and h, the following equalities hold:[2]

1. $(f \frown g) \frown h = f \frown (g \frown h)$
2. $f \frown \epsilon = f = \epsilon \frown f$

Usually concatenation is expressed without the "$\frown$", by mere juxtaposition; e.g. fg for $f \frown g$. And because concatenation is an associative operation, we can write simply fgh instead of $f(gh)$ or $(fg)h$.

[2]As we will see later, the truth of these equations means that A^* together with the nullary operation ϵ and the binary operation $\frown$ is an instance of a kind of algebra called a *monoid*; i.e. (1) $\frown$ is an *associative* operation, and (2) ϵ is a *two-sided identity* for $\frown$.

2 Formal Languages

A **formal** (A-)**language** is defined to be a subset of A^*. But when it is clear that we are talking about formal languages rather than natural languages, we will usually just speak of an A-language, or simply a language if the identity of A is clear from the context. In the most straightforward application of formal languages to linguistics, we mathematically model a natural language as a set of A-strings, where A is a set each of whose members is (a representation of) one of the words of the natural language in question. Of course this is a very crude model, since it disregards any linguistic structure a sentence has other than the temporal sequence of the words themselves. Additionally, once one speaks of a sentence as a string of words, one is immediately faced with the question of what counts as a word, or, to put it another way, what criterion of identity one is using for words. Is it enough to be homophonous (i.e. to sound the same), so that *meat* and *meet* count as the same word? Or to be homographic (written the same), so that *row* 'linear array' and *row* 'fight' count as the same word? Or must two words have the same sound, meaning, and 'part of speech' (whatever we think that is), so that *murder* counts as two words (one a noun and one a verb)? We will return to these and related questions in later chapters.

For the time being, we set such issues aside and assume we know what we mean by a 'word'. Assuming that, we can begin theorizing about questions such as the following: How many sentences (qua word strings) does the language have? Is there a way to list all its members? Is there a way to decide whether a given word string is a sentence of the language? Can we construct a plausible model of the process by which people who know the language recognize that a given string is a sentence of the language? Can the processing model somehow be extended to a model of how language users interpret utterances in context?

In order to address such questions, we need some techniques for defining formal languages. Since natural languages uncontroversially have an infinitude of sentences (how do you know?), it will not do to just make a list of A-strings. In due course we'll consider various kinds of *formal grammars*—mathematical systems for specifying formal languages—but we already have a powerful tool for doing just that, namely the Recursion Theorem (RT). One important way RT is used to specify an A-language L is roughly as follows: we start with (1) a set L_0 of A-strings which we know to be in the A-language we wish to define, and (2) a general method for adding more strings to any arbitrary set of strings, i.e. a function F from A-languages to A-languages. We can think of L_0 as the "dictionary" of the language we are trying to define and F as its "rules". We then define L as the union of the infinite sequence of languages $L_0, \ldots, L_n, \ldots$ where for each $k \in \omega$, L_{k+1} is the result of appying F to L_k.

To make this precise, it will help to introduce a little notation. First, suppose B is a set, $n \in \omega$, and $f \colon n \to B$ (in our applications, B will usually be $\wp(A^*)$.) Suppose also that for each $i < n$, $f(i) = x_i$. Then $\bigcup \mathsf{ran}(f)$ is written $\bigcup_{i<n} x_i$. If f is an infinite sequence in B, i.e. a function from ω to B, and $f(n) = x_n$ for all $n \in \omega$, then $\bigcup \mathsf{ran}(f)$ is written $\bigcup_{n \in \omega} x_n$. For example, it's not hard to see that $\bigcup_{n \in \omega} A^n = A^*$.

We now give a simple example of a recursive definition for a language. Intuitively, a *mirror image* string in A is one whose second half is the reverse of its first half. Informally, we define the language $\mathrm{Mir}(A)$ as follows:

1. $\epsilon \in \mathrm{Mir}(A)$;
2. If $x \in \mathrm{Mir}(A)$ and $a \in A$, then $axa \in \mathrm{Mir}(A)$;
3. Nothing else is in $\mathrm{Mir}(A)$.

Formally, this definition is justified by the RT as follows (here X, x, and F are as in the statement of RT in Chapter Four) we take X to be $\wp(A^*)$, x to be $\{\epsilon\}$, and $F \colon \wp(A^*) \to \wp(A^*)$ to be the function such that for any A-language S,

$$F(S) = \{y \in A^* \mid \exists a \exists x[a \in A \wedge x \in S \wedge y = axa]\}$$

.

RT then guarantees the existence of a function $h \colon \omega \to \wp(A^*)$ such that $h(0) = \{\epsilon\}$ and for every $n \in \omega$, $h(\mathrm{suc}(n)) = F(h(n))$. Finally, we define $\mathrm{Mir}(A)$ to be $\bigcup_{n \in \omega} h(n)$. Intuitively, $h(n)$ is the set of all mirror image strings of length $2n$.

3 Operations on Languages

Let A be a set, so that A^* is the set of A-strings, $\wp(A^*)$ is the set of A-languages, and $\wp(\wp(A^*))$ is the set whose members are *sets* of A-languages.

We introduce the following notations for certain particularly simple A-languages:

a. For any $a \in A$, $\underline{a}$ is the singleton A-language whose only member is the string of length one a (remember this is the function from 1 to A that maps 0 to a).

b. $\underline{\epsilon}$ is the singleton A-language whose only member is the null A-string (i.e. the unique arrow from 0 to A). An alternative notation for this language is I_A.

c. $\emptyset$ as always is just the empty set, but for any A we can also think of this as the A-language which contains no strings! An alternative notation for this language is 0_A.

Next, we define some operations on $\wp(A^*)$. In these definitions L and M range over A-languages.

a. The **concatenation** of L and M, written $L \bullet M$, is the set of all strings of the form $u \frown v$ where $u \in L$ and $v \in M$.

b. The **right residual** of L by M, written L/M, is the set of all strings u such that $u \frown v \in L$ for every $v \in M$.

c. The **left residual** of L by M, written $M \backslash L$, is the set of all strings u such that $v \frown u \in L$ for every $v \in M$.

d. The **Kleene closure** of L, written $\mathsf{kl}(L)$, has the following informal recursive definition (formalizing this definition will be an exercise):

   i. (base clause) $\epsilon \in \mathsf{kl}(L)$;
   ii. (recursion clause) if $u \in L$ and $v \in \mathsf{kl}(L)$, then $uv \in \mathsf{kl}(L)$; and
   iii. nothing else is in $\mathsf{kl}(L)$.

   To put it even less formally but more intuitively: the Kleene closure of L is the language whose members are those strings that result from concatenating together zero or more strings drawn from L.

e. The **positive Kleene closure** of L, written $\mathsf{kl}^+(L)$, has the following informal recursive definition:

   i. (base clause) If $u \in L$, then $u \in \mathsf{kl}^+(L)$;
   ii. (recursion clause) if $u \in L$ and $v \in \mathsf{kl}^+(L)$, then $uv \in \mathsf{kl}^+(L)$; and
   iii. nothing else is in $\mathsf{kl}^+(L)$.

   Intuitively: the positive Kleene closure of L is the language whose members are those strings that result from concatenating together one or more strings drawn from L.

4 Regular Languages

Linguists are often concerned not just with languages, but with *sets* of languages, e.g. the set of finite languages, the set of *decidable* languages (languages for which an algorithm exists that tells for any given string whether it is in the language), the set of *recursively enumerable languages* (languages for which an algorithm exists for listing all its strings while not listing any strings not in the language), etc. In computational linguistics applications, one of the most important sets of languages is (for a fixed alphabet A) the set $\mathsf{Reg}(A)$ of **regular** A-languages. As with many other important sets of languages, there are several different ways to define this set, all of which give the same result. For our purposes, the simplest way is a recursive definition. The informal version runs as follows:

a. For each $a \in A$, $\underline{a} \in \mathsf{Reg}(A)$;

b. $\mathsf{0}_A \in \mathsf{Reg}(A)$;

c. $\mathsf{I}_A \in \mathsf{Reg}(A)$;

d. for each $L \in \mathsf{Reg}(A)$, $\mathsf{kl}(L) \in \mathsf{Reg}(A)$;

e. for each $L, M \in \mathsf{Reg}(A)$, $L \cup M \in \mathsf{Reg}(A)$;

f. for each $L, M \in \mathsf{Reg}(A)$, $L \bullet M \in \mathsf{Reg}(A)$; and

g. nothing else is in $\mathsf{Reg}(A)$.

Note that in this definition, the first three clauses are base clauses and the next three are recursion clauses. The formalization of this definition using RT is left as an exercise. (Hint: remember that we are defining not a language, but rather a set of languages, and therefore the choice of X (as in the statement of RT in Chapter Four) is not $\wp(A^*)$ but rather $\wp(\wp(A^*))$.

5 Context-Free Grammars

Context-free grammars (CFGs) are a particular way of defining languages recursively that is very widely used in syntactic theory; in one form or another, CFGs play a central role in a wide range of syntactic frameworks (here ‘framework’ means, roughly, a research paradigm or community), including, to name just a few, all forms of transformational grammar (TG); many kinds of categorial grammar (CG); lexical-functional grammar (LFG); generalized phrase structure grammar (GPSG); and head-driven phrase structure grammar (HPSG). In due course it will emerge that CFGs are a rather blunt instrument for modelling natural languages, but they are a good point of departure in the sense that they can be elaborated, refined, and adapted in many ways (some of which we will examine closely) that make them more suitable for this task.

The basic idea behind CFGs is to *simultaneuously* recursively define a finite set of different languages, each of which consitutes a set of strings that have the same "distribution" or "privileges of occurrence" or "combinatory potential" in the whole language being defined, which is the union of that set of languages. The languages in that family are called the **syntactic categories** of the whole language.

Getting technical, a CFG consists of four things: (1) a finite set T whose members are called **terminals**; (2) a finite set N whose members are called **nonterminals**; (3) a finite set D of ordered pairs called **lexical entries**, each of which has a nonterminal as its left component and a terminal as its right component[3]; and (4) a finite set P of ordered pairs called **phrase structure rules** (or simply PSRs), each of which has a nonterminal as its left component and a non-null string of nonterminals as its right component[4].

Intuitively, the terminals are the words (or word phonologies, or word orthographies – see above) of the language under investigation. The nonterminals are names of the syntactic categories. The lexical entries make up the dictionary (or lexicon) of the language. And the PSRs provide a mechanism for telling which strings (other than length-one strings of words) are in the language and what syntactic categories they belong to. Once all this is made more precise, the CFG will specify, for each nonterminal A, a T-language C_A, and the language defined by the CFG will be the union over all $A \in N$ of the C_A.

We'll make all this precise in two stages, first using an informal recursive definition (the usual kind), and then a more informal or 'official' definition employing the Recursion Theorem (RT).

First, the informal version. As with all recursive definitions, a CFG has a base part and a recursion part. The base part makes use of the lexicon D and the recursion part uses the set P of PSRs. Starting with the lexicon, remember that formally a lexical entry is an ordered pair $\langle A, t\rangle \in D \subseteq N \times T$; but formal language theorists usually write entries in the form

$$A \to t$$

to express that $\langle A, t\rangle \in D$. In the informal recursive definition, the significance of a lexical entry expressed as follows:

[3]formal language theorists usually allow any T-string as the right component of a lexical entry, but we will not need this generality for our applications.

[4]Formal language theorists usually allow any $(N \cup T)$-string containing at least one nonterminal as the right component of a PSR, but again this generality goes beyond the needs of our linguistic applications.

$$\text{If } A \to t, \text{ then } t \in C_A.$$

That is: for any terminal a which the dictionary pairs with the nonterminal A, the string a of length one will be in the category which that nonterminal names.

Note that it is conventional to abbreviate sets of lexical entries with the same left-hand side using curly brackets on the right-hand side, e.g.

$$A \to \{t_1, t_2\}$$

abbreviates

$A \to t_1$

$A \to t_2$

As mentioned above, the recursive part of the (informal) recursive definition draws on the set P of PSRs. Technically, a PSR is an ordered pair $\langle A, A_0 \ \ldots A_{n-1}\rangle \in P \subseteq N \times N^+$, but formal language theorists usually write form

$$A \to A_0 \ \ldots A_{n-1}$$

to express that $\langle A, A_0 \ \ldots A_{n-1}\rangle \in P$. In the informal recursive definition, the significance of a PSR is expressed this way:

$$\text{If } A \to A_0 \ \ldots A_{n-1} \text{ and for each } i < n,\ s_i \in C_{A_i}, \text{ then } s_0 \ldots s_{n-1} \in C_A.$$

That is: if, for each nonterminal on the right-hand-side of some rule, we have a string belonging to the category named by that nonterminal, then the result of concatenating together all those strings (in the same order in which the corresponding nonterminals appear in the rule) is a member of the category named by the nonterminal on the left-hand side of the rule.

As with lexical entries, sets of rules with the same left-hand side can be abbreviated using curly brackets on the right-hand side.

Before going on to the formal, RT-based formulation of CFGs, we illustrate the informal version with a ‘toy’ (i.e. ridiculously simplified) linguistic example.

$T = \{\mathsf{Fido}, \mathsf{Felix}, \mathsf{Mary}, \mathsf{barked}, \mathsf{bit}, \mathsf{gave}, \mathsf{believed}, \mathsf{the}, \mathsf{cat}, \mathsf{dog}, \mathsf{yesterday}\}$

$N = \{\mathrm{S}, \mathrm{NP}, \mathrm{VP}, \mathrm{TV}, \mathrm{DTV}, \mathrm{SV}, \mathrm{Det}, \mathrm{N}\}$

D consist of the following lexical entries:

$\mathrm{NP} \rightarrow \{\mathsf{Fido}, \mathsf{Felix}, \mathsf{Mary}\}$

$\mathrm{VP} \rightarrow \mathsf{barked}$

$\mathrm{TV} \rightarrow \mathsf{bit}$

$\mathrm{DTV} \rightarrow \mathsf{gave}$

$\mathrm{SV} \rightarrow \mathsf{believed}$

$\mathrm{Det} \rightarrow \mathsf{the}$

$\mathrm{N} \rightarrow \{\mathsf{cat}, \mathsf{dog}\}$

$\mathrm{Adv} \rightarrow \mathsf{yesterday}$

P consists of the following PSRs:

$\mathrm{S} \rightarrow \mathrm{NP}\ \mathrm{VP}$

$\mathrm{VP} \rightarrow \{\mathrm{TV}\ \mathrm{NP}, \mathrm{DTV}\ \mathrm{NP}\ \mathrm{NP}, \mathrm{SV}\ \mathrm{S}, \mathrm{VP}\ \mathrm{Adv}\}$

$\mathrm{NP} \rightarrow \mathrm{Det}\ \mathrm{N}$

In this grammar, the nonterminals are names for the syntactic categories of noun phrases, verb phrases, transitive verbs, sentential-complement verbs, ditransitive verbs, determiners, and common noun phrases.[5] The lexical entries tell us, for example, that **Felix** (the length-one word string, not the word itself) is a member of the syntactic category C_{NP}, and the PSRs tell us, for example, that the string that results from concatenating two strings, one belonging to the syntactic category C_{NP} (e.g. **Felix**) and the other belonging to the syntactic category C_{VP} (e.g. **barked**), in that order (in this case, the length-two string **Felix barked**), belongs to the syntactic category C_{S}.

Finally, we show how to formalize the simultaneous recursive definition of the syntactic categories associated with a CFG, using the RT. As always when applying the RT, the key is making the right choice for the three pieces of data X, x, and F. Since we are defining not a language but rather a function from nonterminals to languages, the right choice for X is not

[5]The category names are a bit confusing, since the categories of noun phrases, verb phrases, and common noun phrases are allowed to contain length-one strings (intuitively, words).

$\wp(T^*)$ but rather $\wp(T^*)^N$; x will be a member of this set, and F will be a function from this set to itself.

So what is x? Intuitively, it should tell us, for each nonterminal A, which strings are in the syntactic category C_A by virtue of the lexicon alone, i.e. without appealing to the recursive part of the defnition (the PSRs). That is, x is the function that maps each nonterminal A to the set of strings t (all of which will have length one) such that $A \to t$ is one of the lexical entries.

What about F? What should be the result of applying F to an arbitrary function $L : N \to \wp(T^*)$? Well, for each $A \in N$, we will want $F(L)(A)$ to contain all the strings that were in $L(A)$, together with any strings that can be obtained by applying a rule of the form $A \to A_0 \ldots A_{n-1}$ to strings $s_0, \ldots, s_{n-1}$, where, for each $i < n$, s_i belongs to the language that L assigned to A_i. Another way to say this is that F maps each L to the function that maps each nonterminal A to the language which is the union of the following two languages: (1) $L(A)$, and (2) the union, over all rules of the form $A \to A_0 \ldots A_{n-1}$, of the languages $L(A_0) \bullet \ldots \bullet L(A_{n-1})$.

Given these values of X, x, and F, the RT guarantees us a unique function h from ω to functions from N to $\wp(T^*)$. Finally, for each nonterminal A, we define the corresponding syntactic category to be

$$C_A =_{\text{def}} \bigcup_{n \in \omega} h(n)(A)$$

A suggested exercise here is to calculate, for as many values of n as you have patience for, and for each nonterminal A, the value of $h(n+1)(A) \setminus h(n)(A)$ (that is, the set of strings that are added to C_A at the nth recursive step).