COUNTING LANGUAGES

Ian C. Stirk

The Chomsky hierarchy of grammars and languages is discussed in every introductory work on mathematical linguistics. Context-sensitive, context-free and finite-state languages are relatively easy to deal with, since concrete examples can be used. It is straightforward enough to show that that the language a^nb^n (that is, the language whose sentences consist of a string of a's followed by an equal number of b's) is context-free but not finite-state, while $a^n b^n c^n$ is context-sensitive but not context-free. The rest of the hierarchy is generally mentioned rather perfunctorily in texts for linguists, since the normal treatment is entirely mathematical and abstract. I have tried here to present it in a rather more "linguistic" way, using strings of symbols for all the examples, instead of the transfinite arithmetic favoured in mathematical texts. Almost the only mathematics left, I think, is the use of the integers for counting, whence the title of this essay. I certainly found the issues greatly clarified in my own mind during the effort of writing. I hope some reader will find the same clarification in reading!

Most of the notation, as well as the inspiration for the proofs, is taken from Arto Salomaa's excellent book "Formal Languages" (Academic Press, 1973).

We had better start with some definitions. A *grammar* may be defined as an ordered quadruple (V_N, V_T, X_O, F), where V_N and V_T are the non-terminal and terminal vocabularies respectively. X_O is a designated member of V_N, the initial symbol. F is a set of ordered pairs, the rules of the grammar. In each ordered pair $(P, Q) \varepsilon F$, P and Q are strings of symbols from the set $V = V_N \cup V_T$, subject only to the condition that no terminal symbol may appear in P but not in Q. This ensures that no terminal symbol may be rewritten by a rule.

It is usually convenient to write the pairs $(P, Q) \varepsilon F$ in the form $P \rightarrow Q$. Non-terminal symbols are represented by capital letters, usually X_0, X_1 etc.

Small letters represent terminal symbols, e.g. a, b. Here is an example of a typical grammar, according to the definition:

$$(X_0, \{a, b\}, X_0, \{X_0 \rightarrow aX_0\ b, X_0 \rightarrow \lambda\})$$

The symbol "λ" is used as a *null symbol:* thus the second rule in F here just deletes X_0.

The rules F of a grammar G define a relation between strings which will be written $\Rightarrow$. If R and S are strings over V, then $R \Rightarrow S$ if and only if R has the form $\alpha P \beta$, and S the form $\alpha Q \beta$, where $P \rightarrow Q$ is a rule of F, and α and β are (possibly null) strings over V.

A *derivation* from the grammar G is a series of strings over V, $P_1, P_2, \ldots P_n$, such that $P_1 = X_0$, and for $i = 1$ to $n-1$, $P_i \Rightarrow P_{i+1}$. Furthermore, there must be no rule $P \rightarrow Q$ of G such that $P_n = \alpha P \beta$, where again α and β are (possibly null) strings over V. $\Rightarrow^*$ is the *ancestral* of the relation $\Rightarrow$, so that instead of

$$P_1 \Rightarrow P_2 \Rightarrow \ldots P_{n-1} \Rightarrow P_n$$

we may just write $P_1 \Rightarrow {}^*P_n$.

The *language* generated by the grammar G may now be defined as the following set of strings:

$$\{P:\ P \varepsilon W(V_T),\ X_0 \Rightarrow {}^*P\}$$

Here $W(V_T)$ is the set of all possible strings over V_T.

It is most important to remember that these are supposed to be *precise* definitions. Intuitive notions about what constitutes a language or a grammar should not be taken as part of them. For instance, it is nowhere stated that *all* of the terminal symbols mentioned in V_T should actually appear in some sentence or sentences of the language generated by a grammar. I shall use that fact tacitly in what follows. I shall also take advantage of the fact that the null *string*, that is, string of zero length, may be a member of $W(V_T)$, according to the usual ideas of set theory.

COUNTING LANGUAGES

The first important question is, can there be languages which have no grammars, that is, grammars of the kind we are considering? One way to answer this question would be to show that there are more possible languages than there are possible grammars. In that case, there could not be enough grammars to go round, and some of the languages would be left without. But surely, some would argue, there is an infinite number of possible languages, and an infinite number of possible grammars, so how could there be more of one thing than the other? Nevertheless, we shall proceed by trying to count the number of languages and the number of grammars.

The first step is to count the number of different strings over the terminal vocabulary, that is, every permutation of finite numbers of terminal symbols. These strings can be regarded as *possible sentences.* Any of them might be a sentence in some language, and every sentence of every language will be included among them. If you like to think less abstractly, imagine that the terminal vocabulary contains every different word of every human language which has ever been used up to the present moment. An extravagant notion, but one which would still result in a finite vocabulary. The various permutations of the symbols of this vocabulary would include every sentence of every human language, plus ungrammatical ones and a lot of mixed strings which are sentences of none.

Now for the counting. For the sake of a more manageable example, suppose that the terminal vocabulary has just three members, a, b, and c. The first string over this vocabulary that we shall list will be the string of zero length, that is, λ:

1. λ

This is just a matter of mathematical convenience: we have already seen that the null string may be part of a language, so it must be put somewhere among the list of possible strings. At the beginning is the best place, too, for next should come the three possible strings of *unit* length, namely:

2. a

3. b
4. c

They have been counted in alphabetical order, and obviously with this vocabulary there can be no more than these three strings of unit length. Now we count the strings of length two. There are nine of these, since we have three choices for the first element and three for the second. They can be listed in alphabetical order too:

5. aa
6. ab
7. ac
8. ba
9. bb
10. bc
11. ca
12. cb
13. cc

The next step is to list the 27 strings of length three, and so on. There are always just a finite number of strings of any particular length, and they can always be organized in alphabetical order. Clearly any string of any length will appear in the list somewhere, and it does not involve very difficult mathematics to work out a formula which gives the number in the listing corresponding to a given string, and vice-versa. The same method will work for any finite vocabulary. Number one in any such list will of course be the null string.

We know now that the number of possible strings is the same as the number of positive integers 1, 2, 3 etc. It remains to be seen if this is more useful than just knowing that the number is infinite. Sets like this set of possible strings, whose members can be counted by the positive integers, or *enumerated*, are known as *recursively enumerable* sets.

Now on to grammars. Do they form a recursively enumerable set, to use the new technical vocabulary? We can find out by trying to enumerate them.

Grammars as they were defined before do not look promising entities for easy listing, but we can try. For one thing, the set of rules could be written out as a string, provided we adopt some punctuation to show where one rule ends and another begins. One grammar we have seen before might look like this:

$$X_0 \rightarrow ;X_0 \rightarrow aX_0b$$

Here ";" has been used to provide the punctuation. Notice that the null sign "λ" has just been omitted. It is clear without it that the right-hand side of the first rule is zero. A slightly more complicated example:

$$X_0 \rightarrow X_1X_2; X_0 \rightarrow X_1X_0X_2; X_1 \rightarrow a; X_2 \rightarrow b$$

These strings could be ordered alphabetically, like possible sentences, provided that the signs ";", "$\rightarrow$", and the non-terminal symbols were added to the alphabet. The non-terminal symbols cause some difficulty here, since there could be any number of them in the form X_i. We do not want an infinite alphabet. Instead of using this notation, let us try representing X_0 as X, X_1 as X′, X_2 as X″, and so on. The numerical suffix becomes a string of the same number of "′" signs. "′" itself can be regarded as a separate symbol in the alphabet. If we stick to the same terminal vocabuloary a, b, c that we used in discussing possible sentences, then the seven symbol alphabet

$$a, b, c, X, ', \rightarrow, ;$$

will be enough for listing grammars in alphabetical order. Not all the possible strings over this vocabulary are grammars, of course. "″a" is one of the strings, for instance, and it is meaningless as a grammar. In fact the first grammar in the list is the 35th string, "X$\rightarrow$". This represents the grammar we would otherwise write in the form "$X_0 \rightarrow \lambda$". The language generated by this is the rather uninteresting one which has a string of zero length as its only sentence, but nevertheless it is a grammar. The 238th string in the list is "X′$\rightarrow$", but we shall not count it as a grammar, since "X", alias "X_0", is the

only initial symbol allowed. The 240th string "X→a" is a slightly more interesting grammar, and it is the second in the list. As for the previous examples, "X→; X→aXb" is somewhere about the 4,100,000th string, while "X→X′ X″; X→X′ XX″; X′ →a; X″→b" is around the 4.6×10^{22} position, to use a more compact arithmetical notation! Obviously it would be absurd to list grammars like this in practice, but what we have achieved is a purely mechanical method of doing so. A computer could easily be programmed to check through the list of strings one by one, and decide whether they represented grammars or not, according to the definition of grammar given before. Remember it was to be taken *literally*! Any grammar discovered would be added to a list of grammars, which would clearly begin

1. X →
2. X → a

and so on. As every possible string will eventually be examined, no grammar could possibly be missed. In fact, many grammars will be counted more than once, because their rules may appear in different orders. For instance, "X→a; X→b" is clearly equivalent to "X→b; X→a", but they would be listed separately. Other things listed as grammars would generate nothing at all, for instance, "X→X′; X″→a". It might be possible to devise more sophisticated computer programs to eliminate some of this excess, but there is no great point in bothering. The list of entities we obtain is clearly recursively enumerable, and all the grammars we would count as "reasonable" are included in it. There has been enough argument to show that, like possible sentences, grammars form a recursively enumerable set. Bear in mind that, although we have continued to use a terminal vocabulary a, b, c in all the examples, similar arguments will apply to any finite terminal vocabulary.

What about languages? Can they be enumerated? Just writing them down causes trouble for a start. Possible sentences and grammars have the virtue of being finite, but languages may be infinite sets of sentences. Finite languages can be presented as a list of sentences, but not infinite ones. One uniform way to represent a language might be as a series of O's and I's. The

series begins with a O if the first possible sentence is *not* in the language, with a 1 if it is. We follow with a O or 1 according to whether or not the second possible sentence is in, and so on. For instance, to go back to the example of possible sentences over the set a, b, c, the finite language {b, c, ab} would be represented as follows:

$$0\ 1\ 1\ 0\ 1\ 0\ \ldots$$

The dots of course represent the beginning of what should be an infinite series of 0's, since this language is finite. This might seem a poor notation, but it will do nicely for what follows.

Suppose that somehow we could find a way of enumerating languages. Then, even though we can't write them out completely, there would in principle be a way of listing these infinite series of 0's and 1's as follows:

$$\begin{array}{lllllll} 1. & a_{11} & a_{12} & a_{13} & a_{15} & a_{14} & \ldots \\ 2. & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & \ldots \\ 3. & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & \ldots \\ 4. & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & \ldots \end{array}$$

and so on. Each a_{ij} represents either a 0 or a 1. Now let us suppose that the symbol "$\bar{a}_{ij}$" represents 0 if $a_{ij}=1$ and 1 if $a_{ij}=0$. Then the following series is also a language:

$$\bar{a}_{11}\ \bar{a}_{22}\ \bar{a}_{33}\ \bar{a}_{44}\ \ldots\ \bar{a}_{ij}\ \ldots$$

Which language in the list is it? The list is supposed to be complete, so it must be there somewhere. It isn't No. 1, though, since $\bar{a}_{11}$ differs from a_{11}. It isn't No. 2 either, since $\bar{a}_{22}$ and a_{22} are not the same. It isn't No. 3, because of $\bar{a}_{33}$ and a_{33}, and so on. This language does not appear anywhere in the list! No matter how we try to enumerate languages, there will be languages "left over" which can't be fitted into the enumeration. I say languages, though only one has been described, because there are others too, such as:

$$\bar{a}_{12}\ \bar{a}_{23}\ \bar{a}_{34}\ \bar{a}_{45}\ \ldots\ \bar{a}_{i,j+1}\ \ldots$$

A similar argument shows that this one is not on the list either, and obviously there is an infinite number of others like it! It has been demonstrated that languages do *not* form a recursively enumerable set, or, to put it more crudely, that there are more languages than there are positive integers. This means that there are more languages than grammars, too, since grammars do form a recursively enumerable set. This fact is important enough to state as a theorem:

Theorem 1 There are languages which do not have grammars.

The kind of argument that was used in the proof was first devised by Georg Cantor, in an arithmetical context. It is generally known as "Cantor's diagonal proof". The reason for using the word "diagonal" should be clear enough.

There is another approach to proving this theorem, which at first sight looks rather different. Suppose that G(i) is the ith grammar in the list of grammars described before. We define a language L such that the ith sentence in the list of possible sentences is in the language if and only if it is *not* generated by G(i). Now if all languages have grammars, this one must too. Suppose it is the kth grammar in the list, G(k). This seems uncontroversial until we think about the kth in the list of possible sentences. According to the definition of L, it is in the language if it is *not* generated by G(k). On the other hand, if G(k) generates L, then the kth possible sentence *must* be generated by G(k)! The paradox can only be resolved if L cannot be generated by any of the grammars in the list, meaning that L does not have a grammar at all.

Somehow this proof does not seem quite so satisfactory as the first one. Grammars may generate infinite sets of strings as languages, and that set might need to be checked through completely in order to be sure that some possible sentence is not in the language generated. We seem to need a god's eye view of the various languages! Oddly enough, this proof is actually very similar to the diagonal one we used before. If we use the notation "L(G(i))" to

stand for "the language generated by grammar i in the list", we can construct an enumeration of this sort:

$$
\begin{array}{lccccccccc}
L(G(1)). & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & . & . & . & . \\
L(G(2)). & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & . & . & . & . \\
L(G(3)). & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & . & . & . & . \\
L(G(4)). & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & . & . & . & .
\end{array}
$$

Once again, the first possible sentence is in L if and only if it is not in L(G(1)). Representing L as a series of 0's and 1's, as before, we see that the first term in the series will be $\bar{a}_{11}$. Similarly for the following terms. Once again a diagonal is constructed. That diagonal cannot be the kth line in the enumeration, for there would be a "clash" in the kth place, where the diagonal crosses the line:

$$
\begin{array}{lccccccccc}
L(G(1)). & (a_{11}) & a_{12} & a_{13} & a_{14} & a_{15} & . & . & . & . \\
L(G(2)). & a_{21} & (a_{22}) & a_{23} & a_{24} & a_{25} & . & . & . & . \\
 & & . & . & . & . & . & . & . & . \\
L(G(k)). & a_{k1} & a_{k2} & a_{k3} & a_{k4} & a_{k5} & . & . & (a_{kk}) & .\ .
\end{array}
$$

According to the definition of L, the kth place would need to contain both a_{kk} and $\bar{a}_{kk}$.

That "god's eye view" will be considered again later. Meanwhile, something has really been tacitly assumed in that second version of the proof which needs justification. Do grammars really generate recursively enumerable sets of strings? It seems only reasonable, but it had better be proved explicitly, and stated as

Theorem 2 Grammars generate recursively enumerable sets of strings.

In a previous example, strings over the alphabet a, b, c, X, ′ , →,; were used to help provide a list of grammars. Suppose we drop the symbol "→", and just consider strings over the remaining six symbols. Now the list of such strings will contain *possible derivations* according to the various grammars in the list of

grammars. For instance, after starting in this way:

1. a
2. b
3. c
4. X
5. ′
6. ;
7. aa
8. ab

the list will have as its 3lst entry the string "X;", which represents the first derivation according to the grammar

$$X_0 \rightarrow \lambda$$
$$X_0 \rightarrow aX_0b$$

namely

X_0
λ

Somewhat later in the enumeration, namely at around position number 1,400,000 comes a second derivation according to the same grammar, that is, "X;aXb;ab", corresponding to

X_0
aX_0b
ab

I think it is plain that all possible derivations of all possible grammars will appear somewhere in this enumeration. Give a particular grammar, it would be a purely mechanical process to decide whether an item in the list was a

proper derivation or not. The definition of "derivation" given before confirms this. The mechanical process could be used to build up a list of sentences generated by the grammar, since the sentences are just the final strings of terminal symbols after the last occurrence of ";" in the correct derivations. There is just one slight difficulty: certain sentences might occur more than once in the list. For instance,

$$X;X'\,X'';aX'';ab \quad \text{and}$$
$$X;X'\,X'';X'\,b;ab$$

are both derivations of the sentence "ab" generated by the grammar

$$X_0 \rightarrow X_1X_2$$
$$X_0 \rightarrow X_1X_0X_2$$
$$X_1 \rightarrow a$$
$$X_2 \rightarrow b$$

This problem is easily solved just by adding to the mechanical procedure. Each time a sentence is extracted from a correct derivation, the list of sentences already obtained is checked through to see if that sentence has already occurred. This can always be done, as the list is finite at any stage. Only if the sentence has not occurred before is it added to the list.

That argument is enough to prove Theorem 2. Notice that in fact it was not strictly necessary to build up a list of derivations distinct from the list of grammars we had already developed. According to the way it was constructed, the possible derivations will occur interspersed among the grammars.

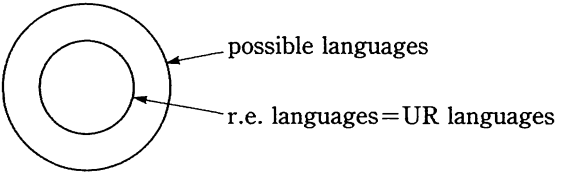
Languages are sets of sentences, so we could restate Theorem 2 by saying that grammars generate only recursively enumerable languages. Now we might speculate about the converse of Theorem 2, that is, do all recursively enumerable languages have grammars? In fact there is no definite mathematical answer to this, not so surprising when you consider the vagueness of the notion of recursively enumerable that we have been using. A

set is recursively enumerable if it can be listed by any "mechanical procedure", whatever that may be. Mathematicians formalize the notion of mechanical procedure by considering it to be anything that a *Turing machine* could be programmed to perform. A Turing machine is an idealised computer, and if you imagine it as a personal computer which you can program in some language like Basic, you are not far wrong. Anything you can program the computer to do is a mechanical procedure. It is not difficult to show that grammars as we have defined them are equivalent to Turing machines. In other words, any computer program you could devise for generating sentences could be written in the form of a grammar. Obviously if "recursively enumerable" means "enumerable by a Turing machine", and if grammars are equivalent to Turing machines, then any recursively enumerable language can be generated by a grammar: the answer to the original question is "yes".

But a problem still remains: is the Turing machine notion equivalent to our intuitive notion of "mechanical procedure"? This is a philosophical question rather than a mathematical one. The claim that the two notions are equivalent is known as Church's thesis. Most mathematicians accept it, because all kinds of mathematical formalizations of mechanical procedures have turned out to be equivalent to Turing machines.

When we think of human beings and their languages, the philosophical problem is worse, because of course then we are led to wonder whether or not human mental powers are equivalent to Turing machines. This leads to debate of the mind-body problem, and even to ethical questions. In what follows we assume that Church's thesis is true, and that human languages are recursively enumerable.

In that case, Theorem 1 also shows that there are languages which are not recursively enumerable. Representing sets as circles, the results so far may be summed up in this way:



"UR" stands for "unrestricted rewriting". Grammars as described so far are unrestricted rewriting, or UR grammars, since there is no particular restriction on the set F of rules, beyond that about terminal symbols. The languages that can be generated by UR grammars are UR languages.

Now that the set of recursively enumerable languages is properly established, it is time to consider its subsets. One thing we might expect a grammar to do for us is to determine whether or not some possible sentence P is grammatical or not. This is one of a class of problems known as "decision problems". When we want to decide whether or not some sentence is grammatical, all we can do in general is enumerate the sentences generated by the grammar, and hope that the given sentence will appear in the list. No matter how far we go, we can never be certain that the sentence will not turn up further down the list. It is impossible ever to brand some sentences as definitely ungrammatical, unless we have an infinite amount of time to spend, or the "god's eye view" mentioned before in the alternative proof of Theorem 1. The decision problem is the reason why that version of the proof seems somehow unsatisfactory.

Obviously there is no such decision problem when we are considering a *finite* language. An arbitrary sentence is grammatical if it is a member of the finite set, otherwise ungrammatical. And a finite set needs only a finite amount of checking. Languages with no decision problem are called *recursive* languages. Having defined them, we need to know now whether there are any infinite recursive languages, and indeed whether there are any recursively enumerable languages which are *not* recursive. We can't be sure yet. One important theorem is surprisingly easy to prove:

Theorem 3 A language L is recursive if and only if both L and −L are recursively enumerable.

−L means, reasonably enough, the set of all possible sentences which are *not* members of L. First of all, consider the case where L is recursive. Then we could check through the list of all possible sentences, and decide for each one whether or not it is in L. If it is in L, we could add it to a list of "sentences

of L", if not, we could and to a list of "sentences of −L". In this way, both L and −L can be enumerated. To complete the proof, we need to show that if L and −L are recursively enumerable, then L is recursive. Consider any possible sentence P. The lists of L and −L cover between them all possible sentences, of course, so after some finite time P will appear on one of them. If it is in the L list, P is grammatical, if in the −L list, it is ungrammatical. The decision can always be made, showing that L is recursive, by definition.
Q.E.D.

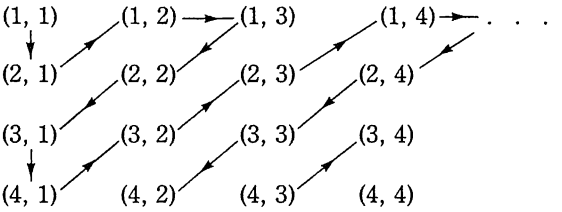
Theorem 3 can be used right away to prove this:

Theorem 4 There is a recursively enumerable language which is not recursive.

If we could find some recursively enumerable language L whose complement −L was not recursively enumerable, the theorem would follow immediately.

Now in the alternative proof of Theorem 1, we described a language which certainly does not have a grammar, even if we do have a god's eye view of its construction. According to Church's thesis, that means it is not recursively enumerable. Let us use it as −L. The ith possible sentence is a member of −L, then, if and only if it is not generated by the ith grammar, in those lists of possible sentences and possible grammars. (Of course the astute reader will have noticed that possible sentences and possible grammars could be combined in one list, if desired!) The ith possible sentence would be in the complementary language L, in that case, only if it *was* generated by the ith grammar. There does not seem to be anything paradoxical about such a language, but we must make sure that it is recursively enumerable. First of all, suppose we pick any pair of integers (i, j)—for example, (3, 16) or (415, 1066). We could mechanically check whether or not the jth possible derivation in the list of such things was a proper derivation according to the ith grammar in the list. If it was, we could go on to check if the sentence it culminated in was the ith sentence in the list of possible sentences. If it was the ith sentence,

it could be added to an enumeration of sentences of L. The whole task is mechanical provided that the number pairs (i, j) can themselves be enumerated, so that any of them would eventually be considered. Well, number pairs can obviously be placed in a grid as follows:



and enumerated by following the diagonals as shown by the arrows, counting 1, 2, 3 etc. Counting is such a way will obviously eventually bring you to any number pair whatever. This makes the whole business mechanical, meaning that L is a recursively enumerable language. The theorem follows. Notice, in passing, that counting is not such an easy and obvious business as it might at first appear. Suppose we started counting the number pairs by following the first row of the table, like this:

(1, 1) →	(1, 2) →	(1, 3) →	(1, 4) → . . .
(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 1)	(3, 2)	(3, 3)	(3, 4)
(4, 1)	(4, 2)	(4, 3)	(4, 4)

In that case, we would never be back, in finite time, to begin on the second row! The exact *way* of counting is often of crucial importance.

The next step is to enquire what kinds of grammar might generate recursive languages. A feature of UR grammars is that they may include *deletion* rules, rules whose right hand side is shorter than the left, for instance:

$$X_0 \rightarrow \lambda$$
$$X_1X_2X_3 \rightarrow X_4$$

When such rules are applied in the course of a derivation, then naturally the following line will be shorter than the line to which the rule applied. The length of a possible sentence is no guide to the complexity of a possible derivation: even a short sentence may have an enormously long line in its derivation, which is later reduced by deletion. This consideration might lead us to guess that grammars without deletion rules might only generate recursive languages. Given any possible sentence P, we might find that only a finite number of possible derivations could end in sentences as short as P. If P is generated by one of these, it is grammatical, otherwise ungrammatical. Let us try to find a rigorous proof of this conjecture.

Firstly we define a *length-increasing* grammar as one whose set of rules F contains only ordered pairs (P, Q) such that len(Q) (that is, the length of Q) is greater than or equal to len(P). Now the theorem may be stated:

Theorem 5 Length increasing grammars generate only recursive languages.

The proof proceeds by showing that only a finite number of possible derivations needs to be checked in order to determine whether or not some sentence P is grammatical. The list of possible derivations needs to be described more exactly this time. For any particular length-increasing grammar we consider, there will of course be only a finite number of different non-terminal symbols. If there are $r+1$ of them, they can be ordered alphabetically X_0 to X_r, without needing to invoke an extra symbol like "′". As usual, the possible derivations are ordered in alphabetically arranged blocks of increasing length. Using the symbols A_1 to A_k to represent the k symbols in the necessary alphabet of both non-terminal and terminal symbols, consider derivations of this form:

$$A_1;A_2;\ \ldots\ ;A_k;A_1A_1;A_1A_2;\ \ldots\ ;A_kA_k;A_1A_1A_1;\ \ldots\ \underbrace{A_kA_k\ \ldots\ A_k}_{n}$$

This kind of derivation begins with k lines each consisting of a single symbol,

followed by k^2 lines consisting of all possible pairs of symbols. Then come all possible triples, and so on until we finish with all possible n-tuples, where n is the length of the possible sentence P being tested. Since all the possibilities are included, then any length-increasing possible derivation longer than these would either end in a string of symbols longer than n, which could not possibly be P, or there would have to be a repetition of some x-tuple somewhere. Now derivations in which a line is repeated—which contain a "loop", in other words—are obviously equivalent to a shorter derivation without the loop. To take a concrete example, the derivation

$$X_0;X_1X_2;X_1X_3;X_4X_3;X_4X_2;X_1X_2;X_1b;ab$$

is equivalent to

$$X_0;X_1X_2;X_1b;ab$$

Thus we do not need to consider possible derivations longer than a certain length in the search for a derivation of P. There can only be a finite number of possible derivations up to a certain length, so they can all be checked. It is possible to determine whether or not P is a sentence of the language, which must then be recursive by definition.

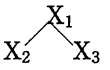
Now we need to know whether *all* recursive languages can be generated by length-increasing grammars. The next theorem provides the answer:

Theorem 6 There are recursive languages which do not have length increasing grammars.

This proof is rather simple, since we can use enumerations developed already. To start with, we can check the list of possible grammars with some terminal vocabulary V_T, and enumerate just the length-increasing ones, because of course it is a mechanical task to check that the rules obey the length restriction. We also have a list of possible sentences, and can define a language in a rather familiar way: the ith sentence in the list belongs to L just in

case it is *not* generated by the ith length-increasing grammar. L is recursive, because by Theorem 5 we can check whether or not some possible sentence is generated by some length-increasing grammar. But, in the usual way, if the kth grammar in the list, G(k), generates this language, it generates the kth possible sentence only if it does not generate the kth possible sentence.
Q.E.D.

Linguists are more often concerned with *context-sensitive* (CS) grammars than merely length-increasing ones. In CS grammars, all rules $P \rightarrow Q$ must be such that $P = \alpha X_i \gamma$ while $Q = \alpha \beta \gamma$, where α and γ are (possibly null) strings over V, while β is a string over V of length at least one. Such rules are clearly length-increasing, but only one non-terminal symbol X_i is "changed" by each rule. The reason for this restriction concerns the mechanical construction of tree structures from derivations. A derivation containing the lines, say, $\alpha X_1 \beta \Rightarrow \alpha X_2 X_3 \beta$ will clearly give rise to a subtree of the form:



Successive lines of a merely length-increasing derivation, though, might look like this:

$$\alpha X_1 X_2 \beta \Rightarrow \alpha X_3 X_4 \beta$$

on the application of rule $X_1 X_2 \rightarrow X_3 X_4$. In such a case, it would be uncertain, without further information, just how the branches of a tree should connect X_1 and X_2 with X_3 and X_4. However, it is easy enough to prove this theorem

Theorem 7 To every length increasing grammar there corresponds a CS grammar which generates the same language.

The proof shows how the equivalent CS grammar may be constructed. Any rules of the length-increasing grammar which already obey the CS restriction can be retained. For any rule of the form $X_1 X_2 \ldots X_m \rightarrow Y_1 Y_2 \ldots Y_n$,

where $2<=m<=n$, m *new* non-terminal symbols Z_1, ...Z_m are added to V_N. The rule itself is replaced by the following series of rules:

$$X_1X_2...X_m \rightarrow Z_1X_2...X_m$$
$$Z_1X_2...X_m \rightarrow Z_1Z_2...X_m$$

.

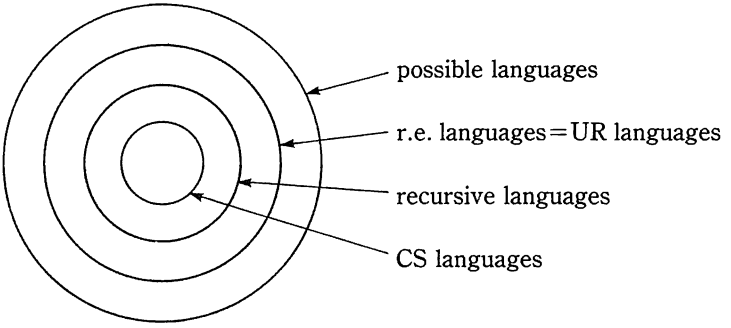
$$Z_1Z_2...Z_{m-1}X_m \rightarrow Z_1Z_2...Z_{m-1}Z_mY_{m+1}...Y_n$$
$$Z_1Z_2...Z_mY_{m+1}...Y_n \rightarrow Y_1Z_2...Z_mY_{m+1}...Y_n$$

.

$$Y_1...Y_{m-1}Z_mY_{m+1}...Y_n \rightarrow Y_1...Y_{m-1}Y_m...Y_n$$

Clearly each of these rules obeys the CS condition, and the whole series has the same effect as the single rule it replaces. Proceeding in this way, all the non-CS rules in the length-increasing grammar can be replaced, and the result will be a CS grammar generating the same language. Q.E.D.

The last few theorems enable us to extend the set inclusion diagram given before:



Now that we have left the abstractions for the more concrete world of context-sensitive languages, it is time to end.