CSE6390 3.0 Special Topics in AI & Interactive Systems II
Introduction to Computational Linguistics
Instructor: Nick Cercone - 3050 CSEB - nick@cse.yorku.ca
Winter Semester, 2010

Early Syntactic Theory and Computer Programs

1. Introduction

In spoken or written language we employ one-dimensional strings of symbols to convey measuring. The dimensionality is limited by our need for presenting the symbols in a single time sequence. This should ***not*** imply that information is stored in our minds in this manner; evidence indicates the contrary position. Thus, a speaker/writer apparently transforms stored information into a linear output string, and a listener/reader decodes this linear string. Two conditions prevail: (1) the universe of discourse is approximately the same for both speaker/writer and listener/reader; and (2) the decoding process is an approximate inverse of the encoding process.

Two major kinds of information processing are apparently involved in constructing and deciphering sentences (there are other considerations). *Semantic* processing analyzes the meaning of individual symbols (words); *syntactic* processing analyzes words as members of word classes and with the structural relationships between classes. There is difficulty in drawing sharp distinctions between the two, especially when definition of membership in a class of words is dependent upon the meaning of a word.

We are only going to concern ourselves in the latter category for the time being. Many different models have been proposed for syntactic processing of English sentences and a number of these methods have been implemented on digital computers. All such processes associate additional structures with the sentences of a language. It is useful to digress at this point to discuss formal languages first, in particular the Chomsky hierarchy, since they make for explicit models with well formed rules, easily implementable on digital computers, and have been used in connection with natural languages.

2. Formal Languages

Formal languages sprang to life around 1956 when Noam Chomsky gave a mathematical model of a grammar in connection with his study of natural language. Thus we define a formal language abstractly as a mathematical system. This will allow us to make vigorous statements about formal languages and develop a body of knowledge which can be applied to them. So we make the following definitions:

alphabet (or vocabulary) - any finite set of symbols

sentences - over an alphabet, a sentence is any string of finite length composed of symbols from the alphabet. Synonyms for sentences are *string* and *word.* The empty sentence, E, in the sentence consists of no symbols. If V is an alphabet, then V* denotes the set of all sentences composed of symbols of V, including E. We use V+ to denote the set V*-{ E} . Thus, if V = { 0,1} , then V* = { E, 0, 1, 00, 01, 10, 11, 000, ...} and V+ = { 0, 1, 00, 01, 10, 11, 000, ...).

language - any set of sentences over an alphabet.

2.1 Formal Notion of a Grammar

Formally, we denote a *grammar* G by (V_n, V_t, P, S). The symbols Vn, Vt, P, and S are, respectively, the *variables, terminals, productions,* and *start symbol.* V_n, V_t, and P are finite sets. We assume that V_n and V_t contain no elements in common; i.e.,

$V_n \leftrightarrow V_t = \phi$

where ϕ denotes the empty set and, conventionally

$V_n \approx V_t = V$

The set of productions P consists of expressions of the form $\alpha \rightarrow \beta$, where α is a string in V+ and β is a string in V*. Finally, S is always a symbol in V_n.

Customarily, we shall use capital Latin alphabet letters for variables. Lower case letters at the beginning of the Latin alphabet are used for terminals. Strings of terminals are denoted by lower case letters near the end of the Latin alphabet, and strings of variables and terminals are denoted by lower case Greek letters.

Given the grammar $G = (V_n, V_t, P, S)$, we define the language it generates as follows. If $\alpha \rightarrow \beta$ is a production of P and γ and δ are strings in V*, then

$$\gamma\alpha\delta \rightarrow_G \gamma\beta\delta.$$

In English we say $\gamma\alpha\delta$ *directly derives* $\gamma\beta\delta$ in *grammar G*. We say that the production $\alpha \rightarrow \beta$ is applied to the string $\gamma\alpha\delta$ to obtain $\gamma\beta\delta$. Thus $\rightarrow_G$ relates two strings exactly when the second is obtained from the first by the application of a single production.

Suppose that $\alpha_1, \alpha_2, ..., \alpha_m$ are strings in V*, and $\alpha_1 \rightarrow_G \alpha_2$, $\alpha_2 \rightarrow_G \alpha_3$, ..., $\alpha_{m-1} \rightarrow_G \alpha_m$. Then we say $\alpha_1 \rightarrow \alpha_m$ (or α_1 *derives* α_m *in grammar G*). In simple terms, we say for two strings α and β that $\alpha \rightarrow_G \beta$ if we can obtain β from α by application of some number of productions of P. By convention, $\alpha \rightarrow_G \beta$ for each string α.

We define the *language generated by G* [denoted L(G)] to be $\{ w \mid w \text{ is in } V_t^* \text{ and } S \rightarrow_G w \}$. That is, a string is in L(G) if:

1. The string consists solely of terminals.
2. The string can be derived from S.

A string of terminals and non terminals α is called a *sentential* form if $S \rightarrow_G \alpha$.

We define grammar G_1 and G_2 to be *equivalent* if $L(G_1) = L(G_2)$.

Example:

Let $V_n = \{ S \}$, $V_t = \{ 0,1 \}$, $P = \{ S \rightarrow 0S1, S \rightarrow 01 \}$. Here, S is the only variable, 0 and 1 are terminals. There are two productions, $S \rightarrow 0S1$, and $S \rightarrow 01$. By applying the first production n-1 times, followed by an application of the second production, we have

$$S \rightarrow 0S1 \rightarrow 00S11 \rightarrow 0^3S1^3 \rightarrow ... \rightarrow 0^{n-1}S1^{n-1} \rightarrow 0^n1^n.$$

Furthermore, these are the only strings in L(G). Thus, $L(G) = \{ 0^n1^n \mid n>0 \}$.

2.2 Chomsky Hierarchy

Type 0 grammars are the types of grammars we have defined. Certain restrictions can be made on the nature of the productions of a grammar to give three other types of grammars, sometimes called types 1, 2, and 3.

Let $G = (V_n, V_t, P, S)$ be a grammar. Suppose that for every production $\alpha \rightarrow \beta$ in P, $|\beta| >= |\alpha|$. [We use $|x|$ to stand for the length, or numbers of symbols in the string x.] Then the grammar G is *type 1* or *context sensitive*.

Some authors require that the productions of a context sensitive grammar be of the form $\square_1 A \square_2 \overline{s\text{-}a} \square_1 \square\square_2$, with $\square_1$, $\square_2$ and $\square$ in V*, $\square \sim= E$ and A in V_n. It can be shown that this restriction does not change the class of languages generated. However, it does motivate the name context sensitive since the

production □1 A □2 $\overline{s\text{-}a}$□1 □□2 allows A to be replaced by □whenever A appears in the context of □$_1$ and □$_2$.

Let G = (V_n, V_t, P, S). Suppose that for every production □$\overline{s\text{-}a}$□in P

1. $\overrightarrow{OA}$is a single variable
2. □is any string other than E.

Then the grammar is called *type 2* or *context free*. Note that a production of the form A $\overline{s\text{-}a}$□allows the variable A to be replaced by the string □independent of the context in which A appears. Hence the name context free.

Let's consider an interesting example of a context-free grammar. It is G = (V_n, V_t, P, S), where V_n = { S, A, B} , V_t = { a, b} , and P consists of the following

S $\overline{s\text{-}a}$AB	A $\overline{s\text{-}a}$BAA
S $\overline{s\text{-}a}$bA	B $\overline{s\text{-}a}$b
A $\overline{s\text{-}a}$a	B $\overline{s\text{-}a}$bS
A $\overline{s\text{-}a}$aS	B $\overline{s\text{-}a}$aBB

The grammar G is context-free since for each production, the left-hand side is a single variable and the right-hand side is a non-empty string of terminals and variables. The languages L(G) is the set of all words in V^*_t consisting of an equal number of a's and b's. We shall prove this statement by *induction* on the length of a word. For w in V^*_t,

1. S *$\overline{s\text{-}a}$w iff w consists of an equal number of a's and b's.
2. A *$\overline{s\text{-}a}$w iff w has one more a than it has b's.
3. B *$\overline{s\text{-}a}$w iff w has one more b than it has a's.

The *reduction hypothesis* is certainly true if |w| = 1, since A *$\overline{s\text{-}a}$a, B *$\overline{s\text{-}a}$b, and no other terminal string of length one is derivable from S. Also, no strings of length one, other than a and b are derivable from A and B, respectively. Suppose that the inductive hypothesis is true for all w of length k-1 or less. We shall show that it is true for|w|=k. First, if S *$\overline{s\text{-}a}$w, then the derivation must begin with either S $\overline{s\text{-}a}$aB or S $\overline{s\text{-}a}$ bA. In the first case, w is of the form aw_1, where |w_1| = k-1 and B *$\overline{s\text{-}a}$$w_1$. By the induction hypothesis, the number of b's in w_1 is one more than the number of a's, so w consists of an equal number of a's and b's. A similar argument prevails if the derivation begins with S $\overline{s\text{-}a}$bA. We must now prove the "only if" of part (1), that is, if |w|=k and w consists of an equal number of a's and b's, then S *$\overline{s\text{-}a}$w. Either the first symbol of w is a or it is b. Assume that w=aw_1. Now|w_1|=k-1, and w1 has one more b than a. By the inductive hypothesis, B *$\overline{s\text{-}a}$w1. But then S *$\overline{s\text{-}a}$aB *$\overline{s\text{-}a}$aw_1 = w. A similar argument prevails if the first symbol of w is b. Our task is not done. To complete the proof, we must show parts (2) and (3) at the inductive hypothesis for w of length k. These parts are proved in a manner similar to our method of proof for part (1).

Let G = (V_n, V_t, P, S) be a grammar. Suppose that every production in P is of the form A $\overline{s\text{-}a}$aB or A $\overline{s\text{-}a}$a, where A and B are variables and a is a terminal. Then G is called a *type 3* or *regular grammar*.

It should be clear that every regular grammar is context free; every context free grammar is context sensitive; every context sensitive grammar is type 0. We call a language that can be generated by a type 0 grammar or type 0 language, and so forth.

In summary, let us say that formal grammars present a natural way to order languages by the restrictions which can be placed on the rewriting rules. Ordering from the most powerful to the least, we have:

A *type 0 language* or *unrestricted rewriting* system is generated by a grammar in which all rules are of the form x $\overline{s\text{-}a}$y where x and y are strings in V*.

A *type 1 language* or *context sensitive language* is generated by a grammar in which all rules are of this form and, in addition, the number of symbols in y (written |y|) is not less than the number of symbols in x. The force of this restriction is best appreciated by examining a rewriting rule which violates it. It can be shown (although we will not do so) that a language generated by a context sensitive grammar can also be generated by a grammar in which all rules are of the form xAy $\overline{s\text{-}a}$xwy where w, x, and y are strings in V* and A is a variable symbol. Then we can interpret xAy $\overline{s\text{-}a}$xwy as "string w may be derived from A if A appears in the context of x and y".

Context free or *type 2 languages* are generated by grammars in which all the rules are of the form A $\overline{s\text{-}a}$x where, as before, A is a variable and x is a string in V*.

Finally, *regular* or *type 3 languages* are generated by grammars whose rules are written either A $\overline{s\text{-}a}$aB or A $\overline{s\text{-}a}$a where a is a terminal symbol and B is a variable symbol.

Since all type 3 languages are type 2 languages, all type 2 languages are type 1 languages, and type 1 are type 0 languages, it is not surprising to find that very powerful automata are required to accept type 0 languages, somewhat less powerful to accept type 1 languages, and so forth.

3. Syntactic Theories

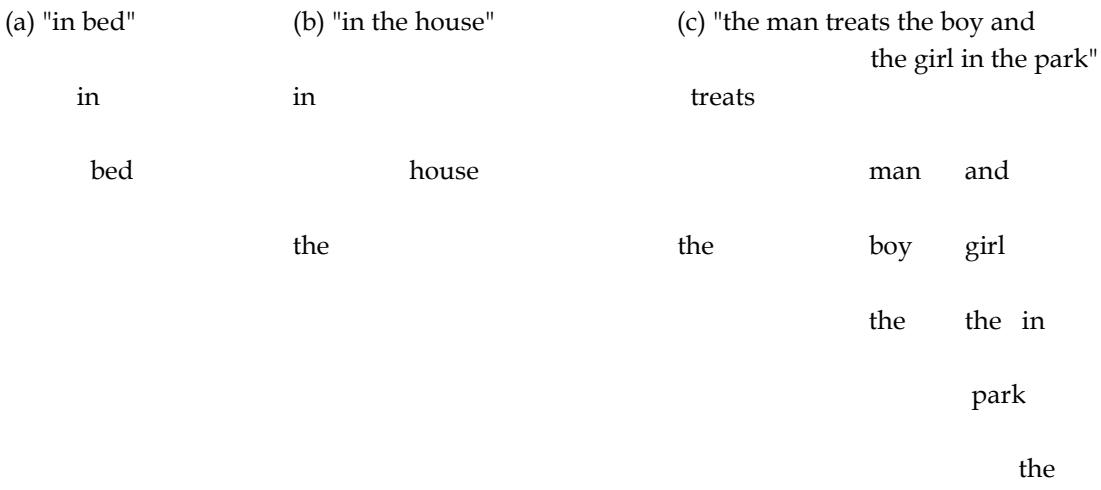
3.1 Dependency Grammars

Conceptually, one of the simplest grammars is the dependency grammar developed by David G. Hays. According to this grammar, a sentence is built up from a hierarchy of dependency structures, where each word in the sentence, except an origin word, usually the main verb, is related to the sentence by dependence on another word in the sentence.

For example, the string "the house" is made up of two elements, with "the" dependent on "house". The article "the" delimits "house", and is, thus, dependent upon "house" for its meaning in the sentence. This is a very pragmatic use of the word "meaning". In the phrase "in bed", "bed" is dependent upon the preposition "in" to connect it to the rest of the sentence, and thus depends upon this preposition.

A word can have more than one dependent. In the phrase "boy and girl", both "boy" and "girl" are dependent upon the *governor* of the phrase, the conjunction "and". Similarly, in the phrase "man bites dog", both "man" and "dog" are dependent upon the verb "bite".

A graphic representation of the syntactic structures associated with some strings by a dependency grammar is shown below.



These structures are downward branching trees. Each node of the tree is labeled with a word from the string. There is no limit on the number of branches from a node. A word is directly dependent upon any word immediately above it in the tree.

Such trees are constructed by investigating all possible connections between words in the initial string. The defining postulate of a dependency grammar is that "two occurrences can be connected only if every intervening occurrence depends directly or indirectly on one or the other of them." Thus, local connections must be made first, and then more distant connections may be tested for validity. The localization assumption is convenient for computer processing.

Another important property of a dependency grammar is the isolation of word order rules and agreement rules. The structure tables for the grammar define allowable sequences of dependencies in terms of word classes. For example, a noun followed by a verb may be in subject-verb relationship. If this word-order criteria is met, agreement in number may then be checked.

If, for each successful connection made, the rule that generated the dependency connection is re cord ed, the *use* of a particular word occurrence is the sentence (e.g., as subject, object, object of proposition, etc.) can be attached to the tree.

3.2 Immediate Constituent Grammars

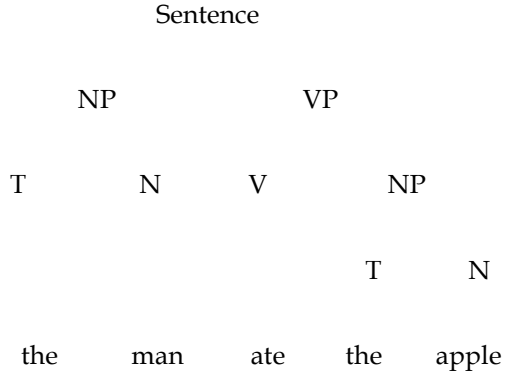
Another type of grammar used to describe English syntax is an immediate constituent grammar. The basic premise of this grammar is that contiguous substrings of a sentence are syntactically related. Brackets or labeled brackets are used to demark syntactically significant substrings. These brackets may be nested, but they may not overlap. The sentence is enclosed in the outermost bracket pair. [Chomsky calls this type of grammar a context-free phrase structure grammar.] Consider the sentence "the man ate the apple." Bracketing the syntactically significant phrases we get:

((the man) (ate (the apple)))

Those unlabeled brackets demark the three principal substructures of the sentence. Usually, when bracketing is done with a phrase structure grammar, the brackets are labeled in some way. For ex ample, we can use "{ } " to enclose a sentence, "[]" to enclose a verb phrase, and "()" to enclose a noun phrase. Then the bracketed sentence would be:

{ (the man) [ate (the apple)]}

The more common way to represent the constituent structure of a sentence is with a tree diagram:



The following simple grammar illustrates this process:

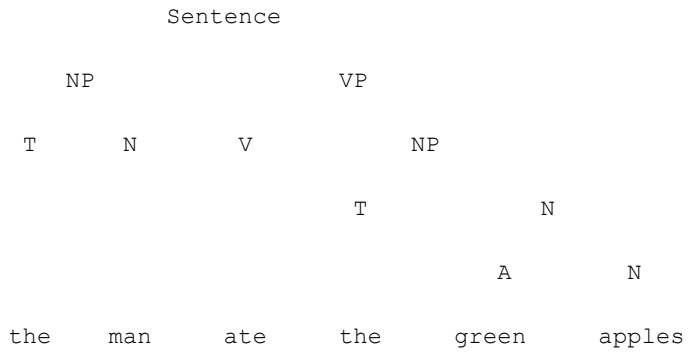
(i)	Sentence	s-a	NP + VP
(ii)	NP	s-a	T + N
(iii)	N	s-a	A + N
(iv)	VP	s-a	V + NP
(v)	T	s-a	the
(vi)	A	s-a	green, red
(vii)	N	s-a	man, apples
(viii)	V	s-a	ate, the

The following lines are a *derivation* of the sentence "the man ate the green apples". The numbers to the right refer to the rules used in generating this sentence from the "rewrite" rules.

SENTENCE
NP + VP (i)
T + N + VP (ii)
the + N + VP (v)
the + man + VP (vii)
the + man + V + NP (iv)
the + man + ate + NP (viii)
the + man + ate + T + N (ii)
the + man + ate + the + N (v)
the + man + ate + the + A + N (iii)
the + man + ate + the + green + N (vi)
the + man + ate + the + green + apples (vii)

The second line of this derivation is formed in accordance with rule (i) by rewriting Sentences as NP + VP, and so on.

The derivation can be represented by the following tree structure:



A substring is called a "constituent" of a sentence if, from all the words of the substring (and only those words), we can trace back to some single nodes of the tree. If this node is labeled Q, then we say that this substring is a "constituent of type Q." note that "the green apples" is a constituent of type NP, but "the green" is not a constituent of the sentence, because both these words cannot be traced back to a single node from which only they originate.

It is sometimes possible to construct two correct distinct diagrams for the same sentence. Chomsky calls this phenomena "constructional homonymity". When this occurs, the sentence in question is *ambiguous*.

3.3 Categorical Grammars

Syntactic analysis performed on the basis of rules of an immediate constituent grammar involves two independent dictionary lookup operations. Parsing operations handle words as members of classes. Therefore, on a first pass, each word occurrence in a sentence is associated with its possible syntactic categories. Subsequent references to the list of grammar rules determine which adjacent constituents in a string can be combined into higher level constituents. This lookup operation is iterated each time a new word category replaces two lower level syntactic markers.

With large vocabulary lists and many grammar rules, these lookups take a disproportionate amount of time, and this time increases rapidly with list size. Thus a goal of avoiding a lookup operation for grammar rules can be identified. If we can assign "grammatical types" to the words of English in such a way that the grammatical correctness of a sentence can be determined by a computation, we can avoid some of the difficulty caused by the lookup operations.

Obviously, such a language coding could not be commutative. For example, the sequence "the boy" is allowable as a syntactic unit in a sentence and "boy the" is not. However, some coding for parts of speech have been developed (due to Bar-Hillel). Let us illustrate this class coding with an example. Recall that a pronominal adjective has the property that the resulting adjective-noun string can again be treated in the same way at the original noun. Bar-Hillel assigns the noun the grammatical code n, and an adjective the code n/[n]. The string has type n/[n] . n (where "." indicated concatenation). Performing a quasi-arithmetic cancellation from the right, we compute the code for the string type as

n/[n] . n = n

As another example, an intransitive verb, such as "eats" in "John eats" is given type s/(n). The string "John eats" therefore has type

n . s/(n) = s

The indicated resulting type is s, or sentence, after cancellation.

If the basic grammatical categories are denoted by s, n_1, n_2, ..., m_1, m_2, ..., then the operator categories of a grammar are denoted by:

$s / (n_1)(n_2)...(n_i)[m_1]...[m_j]... ; i+j >= 1$

As indicated, a term enclosed by parenthesis, e.g., (n_k), can only be cancelled from the left; a term enclosed by brackets, e.g., $[m_j]$, can be cancelled from the right. Some examples:

phrase : very large house

types : n/[n] . n/[n] . $n^{\overline{s-a}}$ n/[n] . $n^{\overline{s-a}}$ n

[n[n]]

Note that this algebra is not associative. The derivation starts by combining the two left hand terms to get a derived type of n/[n] for the substring "very large". Then the right hand cancellation is made, yielding the grammatical type n for the entire string. If, in attempting to compute the type of this substring, we combined the right hand pair first, we would be left to find the type of a string with two constituent codes:

n/[n]
------- . n
[n/[n]]

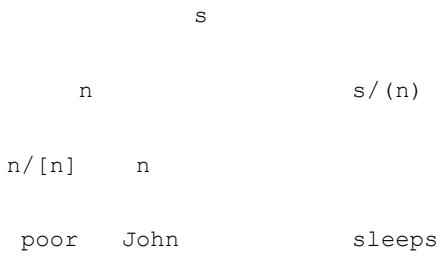
This pair is not further reducible. Thus the pairing must go the other way if this substring is to house a single derived category.

A derivation loading to a single operator category or single basic category is called a *proper derivation*. *A pairing* for a sentence is any proper derivation whose terminal symbol is s. The figure below shows the only proper derivation, and also a pair, of the simple sentence "Poor John sleeps."

```
Poor  John  sleeps
n/[n] . n . S/(n)
        n . S/(n)
            S
```

In a categorical grammar, one defines a substring t to be a constituent of a sentence s if in a proper derivation of s there is included a proper derivation of t. This definition is equivalent to our earlier definition of *constituent* in terms of nodes of a tree.

The derivation of "Poor John sleeps" can be represented by the tree that follows:



Each substring of a constituent can be traced back to a single node. Note that the boundaries of a constituent are dependent on the context of the substring. For example, "John sleeps" has an immediate derivation to a single category marker, S; however, in the context "poor John sleeps" there is NO proper derivation in which "John sleeps" is reduced to a single constituent.

3.4 Predictive Syntactic Analysis

Predictive syntactic analysis is based upon a restricted form of immediate constituent grammar. The restrictions are associated with the order in which words in the input string are scanned during analysis. A predictive parser analyses a sentence in one left-to-right scan through the words.

When a person reads 'the' in a construction, he expects a noun to follow. Similarly, he predicts, from the appearance in an initial noun phrase, the later occurrence of a verb phrase. Predictive analysis works in a similar manner. An initial prediction is made that the string to be scanned is a sentence. From this prediction and the initial word in the sentence, further, more detailed predictions are made of the expected sentence structure. For example, if the first word in a sentence is 'they' the grammar table states that with an initial pronoun the prediction of a sentence s may be replaced by predictions of a predicate and then a period, or by a prediction of, successively, an adjective phrase, a predicate, and then a period - or by seven other sets of predictions.

One set of these predictions at a time is placed on a pushdown list. The prediction that must appear at the top of this list is the one that must be satisfied first. As each successive word of a sentence is scanned, its syntactic type s_j and the topmost prediction on the stack p_i are compared. It may happen that this prediction p_i can be completely satisfied by a member of the class s_j. For example, if the prediction is 'noun phrase', the proper noun 'Tom' completely satisfies the prediction, and 'noun phrase' would be removed from the top of the stack. If not, new predictions compatible with s_j and p_i are generated, and these new predictions are pushed onto the stack. If no new predictions can be made, we infer that earlier predictions were incorrect and an alternative path must be tried. If the terminal punctuation mark of a sentence is reached and all the predictions made have been satisfied, then this set of predictions represents a parsing of the sentence. If some predictions remain unsatisfied, or if there are no more predictions on the stack, but words remain, then this parsing has failed. If all sets of predictions have failed, the sentence is *ungrammatical*.

Each set of predictions for a word class marker s_j and a top prediction p_i is a form of an immediate constituent binary rewrite rule for the predicted structure p_i. The two constituents of p_i are always of the following form: the *left* element is always a terminal marker (in fact s_j); this element can only be rewritten

to give a single symbol, i.e., a word of the sentence. The *right* subconstituent of p_i is a complex symbol - a list of further predictions.

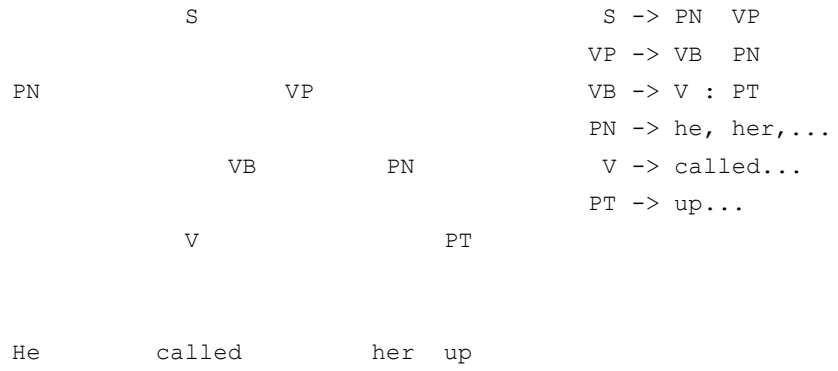
3.5 Phrase Structure Grammars with Discontinuous Constituents

The phrase structure grammars we have examined thus far have allowed only contiguous elements of a string to be syntactically related. In generating a sentence, a single syntactic constituent may be replaced by two or more contiguous subconstituents. Rewrite rules of the form A $\overline{s\text{-}a}$B+C are sufficient to generate most syntactic structures found in English. However, consider the following sentence: *He called her up*. The word up is part of the verb structure, and intuitively we think of 'call up' as one constituent.

To provide a concise notation to describe this type of discontinuous syntactic form, which appears in many languages, we add binary discontinuous rewrite rules. They take the form:

A -> B:C

The interpretation for such rewrite rules is the following. If, in generating a sentence, we have a string XAY, where Y is a single syntactic marker, we may rewrite the string, using the rewrite rule above, as XBYC. In general, when using the rule shown, A is replaced by B, and C is inserted in the string to the right of B, but separated from B by exactly one constituent marker. This rewrite rule is undefined if A is the right hand element in the string. The set of rules for the sentence 'He called her up.' is shown below.



The structural diagram is no longer a simple tree diagram. The relationships within the sentence can no longer be shown by just bracketing.

3.6 Transformational Grammars

All syntactic theories that we have considered thus far are *weakly equivalent* to what Chomsky calls a context-free phrase structure grammar (weak equivalence of two grammars means that both grammars recognize exactly the same set of strings as the set of grammatical sentences, or, equivalently, generate the same strings). He raises several objections to such grammars and proposes additional rules to be added to phrase structure grammars.

First, he proposes that the form of the rewrite rules be generalized to ZXW $\overline{s\text{-}a}$ZYW. Z and W are the *context* of the single symbol X, and Y may be a string of one or more symbols. Although Z and W may be null, the set of elements Y that may be substituted for X is usually dependent upon Z and W. A grammar with such rules is called a *context-sensitive phrase structure grammar*.

Another more serious objection to the phrase structure grammar is the mathematical limitation on the type of strings producible by such grammars. Strings of indefinite length of the form a b c a' b' c', in which a' is dependent on a, b' on b, and c' on c, can't be produced by a phrase structure grammar. An example is the sentence "Tom, Jane, and Dan are a man, woman, and programmer, respectively."

Another objection raised by Chomsky is that sentences such as "The man drives the car" and "The car is driven by the man" is not intuitively related by phrase structure grammars.

Chomsky proposed an additional set of rules, beyond the phrase structure rules, to solve these

problems, Chomsky proposed that after the generation of sentences by a phrase structure grammar, there be *transformation* rules that can transform one sentence into another sentence in the language, e.g., from active to passive voice. One such transformation would transform the two sentences "The boy stole my wallet" and "The boy ran away" into the complex sentence "The boy stole my wallet and ran away".

As Chomsky points out, such transformations have sets of P-markers as domain and ranges, that is, tree structures associated with strings with a phrase structure grammar. They are not defined on terminal strings. In addition to specifying how a terminal string is to be changed, a transformation must specify the "derived" P-marker of the new sentence.

The introduction of transformation rules simplifies the basic phrase structure grammar. Only a simple set of "kernel sentences" need be generated. All other complex sentences can be generated by applying transformation rules to these sentences. In addition, if certain semantic restrictions are to be included in the grammar (e.g., "frightens" may have "John" as an object but "sincerity" may not), these restrictions need only be listed once. For a phrase structure grammar, such restrictions would have to be listed explicitly for both active and passive voice.

3.7 String Transformational Grammars

String transformational grammars are an intermediary between constituent analysis and transformational analysis. The basis assumption underlying the analysis of a sentence is that the sentence has one "center", an elementary sentence that represents the basic structure of the sentence. Additional words within the sentence are adjuncts of these basic words, or of structures within the sentence. Analysis consists of identifying that center and adjoining the remaining words, in segments, to the proper elements of the sentence. For example, "Today. automatic trucks from the factory which we just visited carry carry coal up the sharp incline."

Trucks carry coal is the center, elementary sentence; *today* is an adjunct to the left of the elementary sentence; *automatic* is an adjunct to the left of the truck; *just* is an adjunct to the left of visit; etc. In analysis, each word is replaced by a marker for its syntactic category. Several constituents are strung together in such a way that the resulting pluri-constituent can be replaced by a marker with a constituent within it. This endocentric construction (i.e., one expanded from an elementary category by adjoining) can then be split into this head and its adjuncts. Iterating over all segments of the input string, one obtains the center of the string.

The results of string analysis resemble the results of Chomsky's transformational grammar analysis. A sentence is resolved into a number of kernel sentences such that each main verb of the sentence is part of its own kernel. Some phrases containing implicit verbs are also resolved; for example, "the violinist arrived late" is resolved into "N, plays the violin". These kernels are identified only from the string, not from the structure of the associated syntactic tree (as in a Chomsky transformational analysis).

In general, string transformational process works roughly as follows:

(1) Dictionary hookup of each word, and replacement of the word by its category marks or mark.

(2) Resolution, where possible, of multiple category marks for single words by the use of local context.

(3) Multiple scans through the string - some passes from the left, some from the right. Each scan tries to segment the sentence into "first order strings".

For example, to find noun phrases, the text is scanned from right to left. Whenever a noun is found, a noun-phrase bracket is opened on the right. The scan continuous to the left, accepting all words that can be a part of this phrase. When the left delimiter is found, such as an article, the phrase is closed and the scan is continued until no more groupings into the first order strings can be made. The form of this string of symbols (zero and first order) is then checked against a set of standard patterns. Alternative segmentations are checked and all resultant successful pairings are given.