In the following discussion I cover a range of topics concerning surface structure parsing in natural language, concentrating in particular on defining the procedures used by native speakers in computing the surface structure tree for a sentence. I assume, following the analogous precedent in phonology in which models of acoustic perception are constructed on an articulatory basis, that the procedures by which surface structures are assigned to input sentences are the same as those utilized in the construction of surface structures by speakers. Thus, I hypothesize that an adequate model is neutral between speaker and hearer. It may be the case that such an assumption is gratuitous on the epistemological ground that one cannot test how difficult it is to produce a sentence in the same way that one can, presumably, test perceptual complexity. However, I suggest that surface structure complexity is a matter of linguistic intuition just as are questions of grammaticality; if this is so, then the theory of surface structure complexity is genuinely neutral between speech production and speech perception.

The discussion of parsing leads naturally to a discussion of the interaction of semantic information with a parse routine. That is, it has long been hypothesized that a decision concerning surface phrase structure may be made on the basis of the semantic analysis of the sentence up to the point of that decision. A very simple example would be the contrast between sentences like (1a) and (1b):

(1) a. He hit the man with a bald head.
   b. He hit the man with a nightstick.
Sentence (1a) has only the reading in which *man with a bald head* is a phrase, while (1b) is ambiguous, the perhaps preferred reading being that in which *with a nightstick* is parsed as associated with the verb phrase (VP) of the sentence; i.e., (2a) versus (2b):

\[ \text{(2)} \]

\[ \begin{array}{c}
\text{VP} \\
\text{V} \\
\text{hit} \\
\text{NP} \\
\text{the man with a bald head} \\
\end{array} \quad \begin{array}{c}
\text{VP} \\
\text{V} \\
\text{hit} \\
\text{NP} \\
\text{the man} \\
\text{NP} \\
\text{with a nightstick} \\
\end{array} \]

Whereas it is clear that in such examples semantic information is brought to bear in determining the correct parse, no precise hypothesis has been made in the literature as to how the semantic information interacts with parsing choice. I try to formulate one in terms of the procedures of parsing outlined in this study.

The final section of the study is concerned with the universality of the theory of parsing presented here. In particular, I look at a language that is ‘backwards’ from English, namely, Japanese, to determine if those predictions of perceptual complexity that are borne out by English data are also successful in a language with a radically different surface arrangement of phrases.

**TWO CLASSES OF PARSERS**

**Predictive Analysis**

It is a common observation that it is frequently possible for native speakers to be able to ‘tell what’s coming next’ in a sentence. More precisely, native speakers may be able to predict, if not the exact word, the part of speech of the $n + 1$st word of a sentence on the basis of the previous $n$ words or, perhaps, the kind of major phrase (NP, VP, etc.) that will occur next in a sentence. [In fact, early researchers (cf. Hocket, 1958) attempted to utilize such notions in constructing a model of syntactic behavior. In such models the speaker was represented as knowing the set of words that could initiate utterances and, given the $n$th word of an utterance, the set of grammatical possibilities for the $n + 1$st word. Such a representation could not, in principle, specify the class of grammatical sentences in English (cf. Chomsky, 1957).]

To take some particularly simple examples, given the word *the*, the range of next possible words is limited to something like nouns, adjectives, and adjectival modifiers like *very*. Or, given an initial adverb like *yesterday*, the category of the next phrase is predictably S. (It is rather interesting that while there are definite indicators of the beginning of a new phrase, there seem
to be no phrase terminators, i.e., words that signify the end of a phrase. Phrasal termination is indicated in spoken English by stress.) I shall attempt to explain this ability on the part of speakers to make grammatical predictions in certain cases in terms of a model of how sentences are processed.

Frequently, models of parsing, both for computer and for natural language, are presented on the basis of a bottom-up algorithm. In such computations of surface trees, a parser recognizes the right-hand side, \( X \), of a rule of the form \( A \to X \) of the grammar. (There is an assumption here that there is a surface structure context-free grammar that the speaker employs in producing the correct parse structure; I will return to this question.) In the computation the string \( X \) is replaced by \( A \), and so on until the goal symbol is reached. Thus, for example, assuming that the parser reduces phrases left-to-right, the computation of the phrase structure of \textit{The pretty girl likes banana soup} would be as shown in (3), where successive stages of the parse are shown on successive lines:

(3)

\[
\begin{array}{ll}
\text{the} & \text{NP} \quad \text{V banana} \\
\text{Det} & \text{NP} \quad \text{V N} \\
\text{Det pretty} & \text{NP} \quad \text{V N soup} \\
\text{Det Adj} & \text{NP} \quad \text{V N N} \\
\text{Det Adj girl} & \text{NP} \quad \text{V NP} \\
\text{Det Adj N} & \text{NP} \quad \text{VP} \\
\text{NP} & \text{S} \\
\text{NP likes} & \\
\text{NP V} & \\
\end{array}
\]

In cases of right-branching structures, the whole string must be read into memory before any parsing computations (other than part-of-speech assignment) can be made. That is, the bottom-up parsing history of a structure like (4a) is shown in (4b):

(4)  

\[
\text{a.} \\
S \\
\hspace{1cm} \text{A} \\
\hspace{1cm} \text{B} \\
\hspace{2cm} \text{C} \\
\hspace{2cm} \text{d} \\
\text{b.} \\
\text{a} \\
\text{c.} \\
\text{b} \\
\text{abcC} \\
\text{ab} \\
\text{abc} \\
\text{abcd} \\
\text{S}
\]

For the analogous left-branching structure, the parsing procedure is shown in (5):

(5)  

\[
\text{a.} \\
S \\
\hspace{1cm} \text{A} \\
\hspace{2cm} \text{b} \\
\hspace{2cm} \text{C} \\
\hspace{3cm} \text{d} \\
\text{b.} \\
\text{d} \\
\text{Bb} \\
\hspace{1cm} \text{C} \\
\hspace{2cm} \text{A} \\
\hspace{3cm} \text{Cc} \\
\hspace{4cm} \text{Aa} \\
\hspace{5cm} \text{B} \\
\hspace{6cm} \text{S}
\]
Thus, in employing bottom-up parsing we predict that the memory load on the parsing device will be greatest with right-branching structures and least with left-branching ones, as shown in the respective computations (4b) and (5b). (It may be thought that with both, the whole tree resulting from the computation must be stored at the end, so that in that respect the memory load is equal. However, if we hypothesize that as each node is formed in the process of a parse, it is cleared out of the parser into semantic processing (the principle of Processing in Kimball, 1973), then this will not be the case. In fact, it is by this flow of parsed nodes into semantic analysis that it will be possible subsequently to postulate the relation between semantics and parsing. One indication, among many, that the completely parsed sentence is not passed whole to semantic analysis is the simple observation that, from the standpoint of parse, long sentences present no greater load than short ones, as long as they contain internally easily parsed structures.) Thus, the postulation that parsing in natural language is bottom-up leads to incorrect results, for left- and right-branching structures are of equal perceptual complexity, and both are much easier than center-embedded constructions.

It will be noticed that I have taken simple memory load as the measure of complexity of a surface form, and it is worthwhile to examine how this hypothesis applies in a number of cases. Consider each reading of (6a), rendered in (6b) and (6c):

(6)  
   a. Tom said that Bill had taken the cleaning out yesterday.  
   b. Tom said that yesterday Bill had taken the cleaning out.  
   c. Tom said yesterday that Bill had taken the cleaning out.

The most natural reading of (6a) is (6b) because of the string proximity of the adverb yesterday to the lower clause. Schematically, the different parsings that give each reading are shown in (7):

(7)  
   a.  
      S
      \  
     /  
    S   
   Tom said
   \  
   /  
  that Bill had taken
  \  
  /  
 yesterday
   b.  
      S
      \  
     /  
    S   
   Tom said
   \  
   /  
  that Bill had taken
  \  
  /  
 yesterday

The perceptually easy parsing is as in (7a); the more difficult parse is (7b). The reason, based on the complexity-as-memory-overload hypothesis, is that the top S must be held in the parser in (7b) long enough for the adverb to be located and attached to that S before that S is shipped off to semantics.
In the easier (7a), the top S, *Tom said NP* can be removed from the parser (without being recalled for further additions), leaving space open for allocation in computing the parse of the lower S.

It is easy to see from these considerations that we would deduce that the tendency in a parse is always to associate the terminal immediately inputted as a constituent of the lowest, rightmost nonterminal. That is, the higher nodes will tend to be closed and shipped off to semantics as soon as possible (the principle of Closure), and it would be costly in terms of computation space to bring them back to be reopened for reanalysis or additional constituents (Fixed Structure). The tendency to associate new terminals with the lowest, rightmost node is called Right Association.

If, as hypothesized, sentences become perceptually complex because of memory constraints on the parser, then we might seek to determine what these constraints are. Although it is not clear at the outset what the correct parameters are in terms of which we can measure memory allocation, there are indications that one sort of limit concerns the number of different sentences whose constituency can be computed at once. There is evidence that the limit is two (principle of Two Sentences), as shown by the difference between (8a), (8b) and (9a), (9b):¹

(8)

a. *The girl the boy kissed left.*  
b. *The girl the boy the man saw kissed left.*

(9)

a. *That Tom left bothered Sam.*  
b. *For that Tom left to bother Sam amused Mark.*

I will give an indirect argument that parsing in natural language must be top-down (or at least predictive, in a sense to be defined). This argument is predicated on the hypothesis that the memory limit in parsing is given by the principle that the constituent's of no more than two sentences can be parsed at once. The argument is that it is impossible to formulate this principle correctly for a bottom-up parser.

To see this, consider a bottom-up parsing routine applied to the following two surface structures, one of which is perceptually complex, the other of which is easily understood. We will consider cases like *the boy who the girl who the man saw kissed left* versus *the man saw the girl who kissed the boy who left*. A bottom-up parser will contain the constituents of all three clauses in both sentences before any reduction can take place. Thus, it is incapable of distinguishing between these two in complexity: The two-sentence principle would be violated in both, while in fact one is acceptable. A top-down

¹ A. Grosu (personal communication) offers the following as a counterexample to this principle: *The guy whom the secretary we fired slept with is a real lucky dog*. In this case, three clauses must be processed simultaneously.
or predictive analyzer would likewise have to hold constituents of three sentences at bay in the first case before making any reductions, but in the second sentence the clauses will be parsed top-down in order of occurrence left to right. In order to make the two-sentence principle work, then, parsing in natural language must be either top-down or at least predictive (to be defined).

It can be seen, at any rate, that the principles of parsing outlined earlier (with the exception of Two sentences) are really one principle, namely, Processing, plus the hypothesis that perceptual complexity comes from memory overload [not memory in the usual sense of cognitive recall of bit information, but in the special, nonconscious sense of amount of stored tree structure in the computation space of the parser. This space can, of course, be added to as a cognitive task in, say, computing with pencil and paper the meanings of sentences like (8b) and (9b)].

To work, however, Processing demands that parsing not operate bottom-up; that is, the tree structure must be built up to the goal symbol (S) from the terminals as they are read in. Not all phrase structure grammars allow this type of parsing. E.g., if a grammar has recursion on a left branch, with rules like \( S \to A \ a, A \to A \ a, A \to a \), upon reading the first \( a \) the parser has no way of telling how many \( A \)'s have intervened between this terminal and \( S \). A class of grammars that allows top-down parsing, in which structure is built down from \( S \), is the class of LL(\( k \)) grammars, defined and studied in Lewis and Stearns (1968) and Rosenkrantz and Stearns (1970). This class of grammars is too strong for natural language parsing, as will be discussed.

Now, it is clear that there is no context-free surface structure grammar of English. However, it does seem possible to construct an uncomplicated grammar of a superset of English surface structures. This grammar will have rules in it like \( VP \to V \ VP \), which, of course, will generate too much. However, when used in a recognition device for parsing, this grammar will assign the correct structures. In other words, this grammar will assign surface trees to English, and a lot of non-English also. But this feature should be of no concern, since the grammar has no role as a generating device, i.e., a device that defines the language.

There are a number of ways in which English is not an LL(\( k \)) language. E.g., in the initial stages of the top-down parse of the lazy dog that I saw..., we will have (10a):

\[
(10) \quad a. \quad S \to NP \to Det \ Adj \ N \to the \ lazy \ dog
\]

\[
(10) \quad b. \quad S \to NP \to Det \ Adj \ N \to the \ lazy \ dog
\]
However, when the relative clause is recognized, the additional NP must be inserted in line from the S down to the. Such examples can be continued indefinitely. E.g., notice the successively higher S's that must be produced in getting the correct parse for (11):

(11)

Thus, what we should seek in order to build a reasonable formal model of surface structure parsing in natural language is a class of algorithms that will operate like the top-down algorithms in connecting terminals with the highest symbol possible, but will not falter over left-recursive structures. These devices should not be restricted to deterministic (unambiguous) languages, as are the strictly LL(k) parsers (intended for use with computer languages), although in fact the jump from deterministic to nondeterministic languages is a relatively minor matter. I shall define even more restrictive classes of parsing algorithms that, I hypothesize, are operative in natural language and should explain the propensity of native speakers, noted at the outset of this discussion, to be able to predict the next word or phrase in a sentence. In the algorithms discussed the parser is given the latitude of peeking ahead no more than k terminals in a string (where k is fixed for the parser) in order to make a parsing decision; this is called k look-ahead.

The first class of devices I shall work with are called predictive analyzers with k look-ahead (PA(k) parsers), and they come in steps. A Step 1 PA(k) will, given a terminal string, determine what symbol dominates the initial symbol of that string. E.g., if the string is (12) with the given phrase structure, then upon reading a the machine will predict that the nonterminal immediately dominating a is A without having to complete the A phrase and reduce as in a bottom-up parser:

(12)

If all terminals in a language are immediately dominated by a nonterminal that does not branch (the part-of-speech indicator), then this machine is equivalent to part-of-speech look-up.

In application to (12), a Step 1 PA(k) machine, upon reading a and predicting A as the phrasal type, will read b, completing the parse of the A phrase. The A phrase being completed, this machine can now predict up one level
to determine the phrase type of which \( A \) is the first constituent, so it predicts an \( S \). Then \( b \) is read in, \( B \) predicted and parsed, and the computation completed. The order of parsing of nodes in this machine is, then, \( a, A, b, S, c, B \).

In a Step 2 machine, not only is the immediate dominant of \( a \) predicted but also the immediate dominant of the predicted node, \( A \). So the machine reads \( a \), predicts \( A \), and predicts \( S \). Then \( b \) is read, and \( A \) is finished (and, thus, is available for semantic analysis); \( c \) is read, \( B \) is predicted and parsed, and the computation ends. A class of predictive analyzers has been defined in the literature (Kuno and Oettinger, 1963a, 1963b) that differ from \( PA(k) \) machines. They rely on the grammar’s being in Greibach normal form (each production starting with a terminal on the right-hand side), and are top-down. No look-ahead is involved, since none is necessary because the beginning terminal of the right side of the production is used to identify which production is involved. That this class of parsers is inadequate for the full range of natural language surface structures is evident. To keep the terminology straight, we may distinguish the \( PA \) machines of Kuno and Oettinger from the \( PA(k) \) machines introduced here. (For discussion of \( PA \) and other types of parsers, cf. Griffiths and Petrick, 1965.)

The design of \( PA(k) \) machines is particularly simple. If \( \gamma \) is any symbol that is initial on the right side of a production \( A \rightarrow \gamma X \) (\( \gamma \) is either terminal or nonterminal), then a \( PA(k) \) machine will place \( A \) on a tape upon reading \( \gamma \). It is possible that there will be another production in the grammar, \( B \Rightarrow \gamma Y \), in which case the machine must use its look-ahead capabilities to determine whether to predict \( A \) or \( B \). That is, defining \( \Rightarrow \) to be the transitive closure of \( \rightarrow \), a choice between \( A \) and \( B \) is possible if \( X \Rightarrow \sigma, Y \Rightarrow \sigma' \), where \( \sigma \) and \( \sigma' \) are strings of terminals, and the first \( k \) symbols of \( \sigma \) are distinct from the first \( k \) symbols of \( \sigma' \).

There are cases in natural language in which \( k \) look-ahead will not uniquely determine a prediction for any finite \( k \). For example, in a sentence like \textit{Flying planes are dangerous}, the phrase \textit{flying planes} has a unique analysis. However, the determination of the correct analysis for this otherwise generally ambiguous phrase is made only when the main verb \textit{are} is reached. It is possible to intersperse indefinite amounts of material between \textit{flying planes} and \textit{are}, however, by the use, e.g., of a relative clause, as in \textit{flying planes which are constructed in any city with a population of more than the number of hairs on the heads of giraffes who live in states with a high ratio of railroads to pigeons... are dangerous}.

Thus, there are sentences for which no finite amount of look-ahead is sufficient to determine the correct parse. The conclusion to be drawn is that English is not a \( PA(k) \) language for any \( k \). Notice, however, that this observation is irrelevant to the hypothesis that speakers of English employ \( PA(k) \) parsers in analyzing surface structures. To correctly model speaker behavior,
a parser must balk, make incorrect or underdetermined decisions, etc., exactly where a native speaker would. If we say, e.g., that speakers internalize a PA(2) parser, then we predict that speakers may make parse errors in cases in which PA(2) analysis is inadequate to correctly determine the parse, i.e., where a look-ahead of more than two symbols is required.

Let us see how a PA(2) machine operates in the parse of a sentence like (13), which is perceptually complex:

(13) The man that the dog that the cat saw chased left.

The parse tree is shown in (14):

(14)

We assume, first, that available to the parser is the context-free grammar of a superset of English surface structure discussed earlier. This grammar will contain, among others, the following rules:

(15) a. \( S \rightarrow NP \ VP \)  
    b. \( S \rightarrow NP \ S \)  
    c. \( NP \rightarrow \text{Det} \ N \)  
    d. \( NP \rightarrow \text{NP} \ S \)  
    e. \( VP \rightarrow V \)  
    f. \( VP \rightarrow V \ NP \)  
    g. \( VP \rightarrow V \ NP \ NP \)  
    h. \( N \rightarrow \text{man, etc.} \)  
    i. \( \text{Det} \rightarrow \{ \text{the} \} \)

Assume now that the PA(2) machine in question is Step S, so that it will predict dominating nodes up to the first S over the nonterminal immediately dominating the terminal read. Upon reading the first terminal, the, the machine finds rule (i) in the grammar and places the node Det on a tape. Reading Det, then, it looks at all rules with this as the first symbol on the right-hand side—in the case of the grammar in (15), only rule (c)—and chooses one, placing its initial symbol, NP, on the tape to the left of Det. Now, NP starts the right-hand side of three rules, (a), (b), and (d), and a choice must be made. We could appeal to look-ahead here, saying the machine looks for
a \textit{wh} morpheme to tell it whether there is a relative clause, i.e., whether to pick rule (d) or not. This would work in the case at hand but will not work in general, since no finite look-ahead will resolve this kind of question because indefinitely many terminals can come between a determiner and its head noun. In fact, there are other such cases in natural language in which a PA(\(k\)) machine will have to decide, given a node \(A\), whether \(A\) dominates \(A\) directly in a path to the root \(S\). These cases arise from surface structure configurations that reflect conjunction, as in (16a), Chomsky adjunction on the right, as in (16b), or cases like (16c), generated in the base:

\[
\text{(16) a. } \quad \begin{array}{c}
\text{A} \\
\text{A and A}
\end{array} \\
\begin{array}{c}
\text{b. } \\
\text{A} \\
\text{A B}
\end{array} \\
\begin{array}{c}
\text{c. } \\
\text{NP} \\
\text{NP S S S S}
\end{array}
\]

In formal terms, the machine is seeking a path to the root, where certain symbols may be repeated. Let us define an equivalence relation between paths, so that two paths are equivalent if they differ only in repetition of a given symbol. Thus, the paths \(\text{Det NP S, Det NP NP S, and Det NP NP NP S S S S}\) are all equivalent. We can then pick a canonical representative of such an equivalence class to be the most collapsed string, in this case the first. (This is formalized in Wise and Kimball (to appear), where a rigorous definition of the class of grammars susceptible to the kind of analysis discussed here is presented.)

In other words, let us assume that the machine will always pick the shortest path to the root when faced with a choice like that described earlier between rules (a), (b), and (d); i.e., it picks the canonical representative of equivalence classes of paths to the root. Thus, it picks \(S\) to place on the tape to the left of \(NP, Det\). At this point the machine has picked a path to the root and is reading the next terminal in (14), \textit{man}. As the \(Det\) node is completely analyzed

\[
\text{Det} \quad \text{is removed from the parser and sent to semantics, although a copy of Det is left in the parser, to be removed only when the next node up is completely parsed. Man is recognized as an N and taken to semantics, and N is placed on the tape. The tape now has S NP Det N, since because Det N is an NP by rule (c), it can be cleared from the parser. Thus, } \quad \begin{array}{c}
\text{NP} \\
\text{Det N}
\end{array}
\]

\[
\text{is sent to semantics and the NP left in the parser but given a special mark, NP, to indicate that it is closed.}
\]

Now, if the next terminal were a \(V\), the analyzer would place \(V\) on the tape, and then \(VP\) to the left of \(V\) by reference to rules (e), (f), or (g)—which are the same rule by the collapsing conventions. The tape would
then contain \( S \) NP VP V. Since NP VP is an \( S \), however, by rule (a), the machine will assume (modulo look-ahead) that \( S \) is completely parsed and ship \( S \) off to semantics, leaving the \( S \) behind just as it left NP behind on the tape when \( NP, N \) was sent to semantics. If this \( S \) is in fact the top \( S \), it will be the sole occupier of the tape when the computation is finished. However, this \( S \) could be the first member of a conjunction in a configuration

\[
\begin{align*}
& \quad S \\
& \quad \quad \quad S & \quad \quad \text{and} & \quad \quad S
\end{align*}
\]

in which case it will be needed in the parser for further computation.

Let us now return to the computation of surface structure for (13). After the \emph{man} is read in, the tape has \( S \) NP, and the next terminal, \emph{which}, is read, a look-ahead determines that the NP dominating \emph{which} came from a rule like (b), so \( S \) is placed on the tape next to the NP. The tape now has \( S \) NP S NP, and the machine must make a decision as to the constituency of NP S. It is clear that they are sisters in some phrase; reference to the rules in (15) indicates that NP S can be either an \( S \) phrase or an NP. In this case, the \( S \) dominates an NP that dominates a \emph{wh} word (\emph{which}), indicating that the NP S is a relative clause and so is an NP constituent.

Thus, the phrase \( NP, S \) is sent from the parser into semantics, the top NP remains in the parser with a closed marker, NP, and the lower S remains, since its constituency has yet to be determined. Thus, the tape has on it \( S \) NP S NP, as before, but the machine is now working on the constituency of the lower S and so does not reparse NP S as an NP. The semantics now has the phrases

\[
\begin{align*}
& \quad NP, \\
& \quad \quad \quad Det, \quad N, & \quad \quad \quad NP
\end{align*}
\]

in it. The machine next reads in the \textbf{the of the dog} and by prediction places S NP Det on the tape, so the tape now has S NP S VP S NP Det; by reference to the grammar, the phrase NP S is parsed as an \( S \) and sent to
semantics, so that S NP S NP Det remains. *Dog* is read and parsed as an N, so the Det N can be parsed as an NP, which is then shipped off, leaving another NP on the tape. I.e., the tape now has S NP e S NP e, and the machine is ready to read *which*.

At this point, the machine goes through the same process of computation as that described in the preceding paragraph, except that after *the cat* has been read in and parsed, the tape now has S NP e S NP e S NP e on it, presuming that our tape has an indefinitely extendible length. In fact, however, human syntactic memory for the purposes of parsing seems to be limited to the contents of no more than two sentences, so our model will show this by failing to be able to hold the third S and the computation will block, as it does in the real case for normal speakers.

For the purposes of exposition, however, let us assume that the tape can hold an indefinite amount of material storage. When *saw* is read, the prediction of VP V is made and placed on the tape. A look-ahead indicates that the VP is complete, so the fragments $\frac{V}{\text{saw}}$ and $\frac{VP}{V}$ are cleared from the parser, leaving VP e on the tape. The lower S now precedes NP e VP e and is so parsed, being cleared. This happens in the same sequence two more times as the verbs *chased* and *left* are read into the parser, and the computation is complete.

Having illustrated the operation of a PA(2) machine for English surface structure, I would like to make some general comments on this model of natural language parsing. As mentioned in an earlier section of this study, it seems to be the case that all complexity in surface structure configuration (where in Kimball, 1973 I tried to define complexity in terms of violation of one or another principle of parsing) is the result of basically *one* limitation in human parsing; namely, limitation on memory for accreted nodes in the parser. I hesitate to call this short-term memory, since this term has been usurped by psychologists and others to cover much more general and perhaps disparate processes. It is reasonable to expect that the kind of memory involved in parsing is of a quite specialized and linguistic-particular nature, and has to do solely with the linguistic processing capabilities of the organism.

The construction of a PA(2) Step S parser is designed to give a direct measure of the amount of memory capacity required in parsing any sentence in terms of the amount of material written on its computation tape. If the model is correct, this will represent exactly what is stored in the parser end of human language processing.

A few points concerning the computation outlined here have been left vague, e.g., how the parser picks the rule that parses NP S as an NP and
not an S. It was remarked that this is done by reference to the presence of a \textit{wh} word in the lower S. In fact, there seems to be a rather general process of parsing a sequence $X \rightarrow Y$ as an $X$, which reflects the generative process either of Chomsky adjunction or a phrase structure rule of the form $X \rightarrow X \ Y$. This question is worked out formally in Wise and Kimball (to appear), where also is presented the construction of machines to parse PA(2) languages.

It is interesting to see why it is the case that structures in English like (17) are not difficult to parse:

(17) \textit{The cat that ate the rat that ate the cheese that sat in the cupboard has a fuzzy tail.}

Upon initial inspection, it looks as though such structures violate the principle of Right Association, which says that the optimal placement of a node is to the right of and possibly embedded in the node currently held in the parser being parsed. This is because when the main verb \textit{has} is reached, it is not tacked onto the phrase dominating \textit{cupboard} but is brought up under a VP attached to the main S. That no violation of Right Association is involved is shown by what is on the tape of the machine when \textit{has} is read. When \textit{cupboard} is read, a look ahead reveals that its NP and the next-higher VP and S are completed. Thus, they are closed and cleared from the parse routine, leaving on the tape only S NP, where the NP is the noun phrase dominating the initial phrase of the sentence. As it is closed, the node being parsed at this point is the top S, and so by Right Association the VP over \textit{has} will be attached under this S.

There is another and somewhat more general way of looking at this whole question. The length of the storage tape in the parser in some sense gives us a difficulty function that maps trees into the natural numbers, where the number assigned to each tree is its difficulty under the measure. Now, at each stage one wants the parser to make a choice in node assignment that will optimize the value of this function (i.e., make it as small as possible). The constraints on the choice, however, are determined in part by the rules of surface structure grammar that give boundary conditions for the choices, i.e., determine what choices will give rise to possible parse structures.

We can construct a \textit{gedanken} experiment, however, in which we look at the operation of an optimizing parser in which the boundary conditions are removed. That is, we can look at what kind of surface trees a parser will build over some arbitrary input string without being constrained to follow phrase structure rules, assuming that the parser proceeds at each step to minimize the difficulty function. The class of trees so defined will, then, be the optimal surface structure trees, and will provide a base line against which the actually occurring trees of natural language can be compared.
Assume that the machine is given a stock of nonterminals in terms of which to analyze any string it is given, and proceeds to parse the sequence $a\ b\ c\ d\ e$. The first symbol, $a$ is read in and assigned to a nonterminal, say $A$. The machine at this point could either (a) assume that $A$ is closed and clear the completed constituent $\frac{A}{a}$ from the parser, leaving $A_c$ in the

parser, but it must then place some node, $B$, to the left of $A$ in prediction of what phrase $A_c$ is the first element, so the tape would then have $B\ A_c$ on it; or (b) it could assume that $A$ is not closed, in which case the tape would contain $A\ a$. Notice that in both cases I am assuming that the parser must contain one, but no more than one, incomplete node, which it is 'working on.' In the case of natural language, one may have several unfulfilled nodes on tape.

For both options (a) and (b) the machine has essentially the same sort of configuration on tape, namely, $X\ Y$, where $Y$ is a completed node, so that the next terminal read in will not be attached to $Y$. Let us take (a) for a working example. The next terminal, $b$, is read in, and the options of the machine in this case are shown in (18):

(18) a. 
\[ \begin{array}{c}
\frac{A}{b} \\
A_c \\
\end{array} \begin{array}{c}
B \\
\end{array} \]

b. 
\[ \begin{array}{c}
\frac{A_c}{b} \\
A_c \\
\end{array} \begin{array}{c}
B \\
\end{array} \]

c. 
\[ \begin{array}{c}
\frac{A_c}{b} \\
A_c \\
\end{array} \begin{array}{c}
B \\
C \\
\end{array} \]

d. 
\[ \begin{array}{c}
\frac{A_c}{b} \\
A_c \\
\end{array} \begin{array}{c}
B \\
C \\
\end{array} \]

In case (18a), the $b$ is Chomsky adjoined to $A_c$, and here $\frac{A}{b}$ will be cleared from the parser as a completed node (otherwise, the parser would contain two incomplete nodes in storage, contrary to the minimizing hypothesis), again leaving $B\ A_c$ on the tape. In (18b) we should assume that $B$ is closed; otherwise, the storage tape will have $B\ A\ b$ on it, which is longer than if $B$ is assumed closed and assigned as the initial node of a new nonterminal, $C$, so the tape would have $C = B_c$ on it. In (18c), $b$ is assigned to a new nonterminal, $C$, which is subtended under $B$, and in (18d), $C$ is Chomsky adjoined to $A$. To minimize tape content, for (18c) the machine will close and clear $B$ if $C$ is open, leaving $C\ b$ on the tape. However, if $C$ is considered to be closed, then the machine can choose either to leave $B$ open, having $B\ A_c\ C_c$ on the tape, or to close $B$, subtending it as the first constituent of a new node upward, $D$. In the latter case, the tape would contain only $D\ B_c$. Since we are optimizing at each step, we would be forced to the latter choice, since a tape content of two is better than a content of three. In (18d), optimization forces us to close $C$ and $A$. 

We can summarize the results of the various choices open to the machine in (19), which shows the tape contents after each of the choices in (18) is made:

\[(19) \quad \begin{array}{cccc}
\text{a. } B A_c & \text{b. } C B_c & \text{c. } C b & \text{d. } B A_c \\
D B_c & & & \\
\end{array} \]

Notice that in each case we have a nonterminal plus its left constituent on the tape. Thus, we will proceed from any of the tape states shown in (19) to attach the next terminal, c, according to the choices provided in (18).

Notice, further, that choices (18a) and (18b) are essentially those of (18d) and (18c), respectively, as far as modification of the original tree is concerned, the only difference consisting in the addition of a node directly dominating the terminal (a preterminal node). Modulo this difference, we can now construct all possible optimal trees over an arbitrary input string, where 'optimal' is defined as 'parsable with minimum required space on the storage tape'. The minimum at each step is one nonterminal plus its left constituent. For example, all optimal trees over the string \( a b c d e \) are constructed in (20):

\[(20) \quad \begin{array}{cccc}
\text{a. } & \text{b. } & & \\
W & X & Y & D \\
Z & C & A & E \\
A & B & c & d \\
& b & & \\
\text{c. } & & & \\
W & X & Y & & \\
A & B & C & & \\
& d & & \\
\text{d. } & & & \\
W & X & Y & Z & E \\
A & B & C & D & \\
& c & d & e \\
\end{array} \]

As remarked earlier, the preterminals in these trees are optional and, in any case, do not affect the formal outline of the tree structure. The node labels \( W-Z \) are used as variables.

It is evident what principles govern the progression in this class of trees. Each tree has two parts, an initial left-branching prefix followed by the remainder, which is right-branching. In (20a), this prefix is the whole tree; in (20b), it is the first three symbols; in (20c), it is the first two symbols; and in (20d), it is the first only. Thus, in general, for a string of terminals of length \( n \), there will be \( n - 1 \) optimal trees, each constructed along a different line of growth from left to right. Each tree will grow upward on a left branch until the first decision is made to grow on a right branch, in which case the condition on optimality requires that the growth continue on the right branch. Each node will be binary branching, again by optimality.
In linguistics we are concerned not so much with the patterning of individual nodes in surface structure as with the organization of clauses, where the S node is taken as the only recursive (in the sense of Chomsky, 1965:225, n 11) symbol (up to conjunction). That is, if clause units are the basic units of sentence perception (for which there is evidence), then it is the organization of clausal units on surface structure that will determine parsing complexity. Thus, the optimal arrangement of a sentence with five clauses is given by (20) as follows:

(21) a. 

These tree structures are optimal in the sense that the parser never has to remember more than the S it is actually working on. The fact of the matter seems to be that humans can retain two S’s in storage for the purpose of parsing, but no more (as discussed previously). Thus, we may allow combinations of any of the one-S memory configurations of clauses under a clause and remain within the limits of human capabilities. For example, in English a structure like (22) is quite acceptable, as illustrated in (23):

(22)
(23) The boy who knew the girl that ate the egg that Tom’s chicken laid knew that Bill wanted Mike to ask Sam to leave.

A structure like (22) consists basically of two structures of type (21e) embedded under a single S. So, in shorthand, if all structures in (21) are of weight one, then (22) is of weight two, and weight two structures are the heaviest parsable by humans.

In concluding this section it should be remarked that there are several well-known transformational processes that reduce the number of underlying clauses in a sentence by breaking these up into pieces and incorporating the pieces as parts of higher clauses. Certainly subject raising and predicate raising are two such operations, as is relative clause reduction. Further, there are transformations that put constituents out on right branches, such as extraposition. These operations all conspire to reduce the perceptual complexity of a sentence, as per Chomsky’s hypothesis (1965: Chapter 1, sec. 2), and as elaborated in Kimball (1973: sec. 5).

Over-the-Top Parsing

In this section I will consider rather briefly a somewhat stronger type of parsing than predictive analysis, stronger in the sense that it applies successfully only to a more restricted subclass of languages. This type of parsing is called over-the-top parsing (terminology suggested by De Remer) and, in a sense, is the minimal extension of predictive analysis.

In a PA(k) parser, the machine determines for each phrase parsed whether it is the initial constituent of a higher phrase and, if so, what the higher phrase is. That is, in a Step 1 machine, in a configuration like that shown in (24), when the A phrase is completely parsed, the machine ‘looks up’ to see what phrase A is the initial constituent of:

(24)

Suppose, now, that we instruct the parser not only to look upward to B but also to look back down to see what the next constituent of B will be, i.e., C. (Note that it is not necessary that C would be recognized immediately by a PA(k) machine, since the first terminal of C could be separated from C by a long chain of nonterminals.) Thus, in a PA(k) machine, upon completing A we would know only that we were working on parsing a phrase of type B; in an over-the-top machine (with k look-ahead), OT(k), we would know, in addition, that we were working on a phrase of type C within the phrase of
type $B$. When $C$ is completed, the OT($k$) machine would again look down from $B$ to put $D$ on the tape.

It is quite evident that there are grammars subject to PA($k$) analysis but not to OT($k$) analysis. For example, if a language had a grammar with rules $B \rightarrow A C D$, $B \rightarrow A E D$, then we would not know whether to predict $C$ or $E$ over-the-top, unless some look-ahead revealed the difference. However, if both $C$ and $E$ were phrases allowing left recursion, then no amount of look-ahead would do the trick. E.g., OT($k$) analysis is impossible for the following grammar, while PA($k$) analysis is not:

$$
\begin{align*}
S & \rightarrow A \ B \ B \rightarrow D \ e \ D \rightarrow d \ D \ E \rightarrow d \ D \ E \rightarrow d \\
S & \rightarrow A \ C \ C \rightarrow E \ f \ D \rightarrow d \ A \rightarrow a
\end{align*}
$$

All strings of the language will be of the form $a d^n\{e, f\}$, where it is impossible to tell which rule rewrote $S$ until the last symbol of the string is reached. Thus, we have proved that OT($k$) grammars are properly included in PA($k$) grammars.

It is possible to conceive of an even stronger condition on grammars, namely, that upon reading each terminal symbol it is possible to determine the path of nonterminals that connect that terminal with the goal symbol, plus what rules applied at each point to expand those nonterminals. E.g., in a tree like (26), a parser would know, by looking at the initial terminal, not only the path to the root of nonterminals but also the rules applying in each case, i.e., that $S$ was expanded as $A \ B \ C$, that $A$ was expanded as $C \ d \ B$, and that $C$ was expanded as $a \ D \ D$:

```
(26)
```

Grammars that are subject to this type of analysis are called LL($k$) grammars. It is clear that these are properly included in OT($k$) grammars (for while one may be able to determine the constituents of a phrase one at a time, this may be impossible all at once on the basis of the initial constituent.)

To complete the picture, we may add that PA($k$) grammars are properly included in LR($k$) grammars (Knuth, 1965)—where this latter class consists essentially of grammars that are parsable bottom-up, left to right deterministically. E.g., any grammar that will generate a left-descending repetitive sequence of nodes, such as $S \ A \ B \ C \ A \ B \ C \ A \ B \ C \ldots$, i.e. $S \ (ABC)^n$ on a left branch, will be LR($k$) but not PA($k$). In summary, then, we have proved the following theorem:
THEOREM: \( LR(k) \not\subseteq PA(k) \not\subseteq OT(k) \not\subseteq LL(k) \)

The definition of \( OT(k) \) parsing is of interest with regard to natural language in that it is a formalization of the ability of native speakers to predict sequence such as that a *the* will be followed by a noun. This ability, insofar as it is real, is based not on speakers’ associating pairs of sequentially occurring words, like *the dog, the horse*, etc. but on their \( PA(k) \) type knowledge that *the* is a Det and that a Det begins an NP, and perhaps, in addition, on their \( OT(k) \) type knowledge that in an NP, an N may follow a Det.

We are left, however, with a decision as to whether human parsing in natural language is \( PA(k) \) or \( OT(k) \), if either. In a sense, it might be interesting if it were the latter, given the theorem just presented, for then we would have a restriction on the form of possible surface structure in natural language. However, given that the basic limitation in parsing seems to be on memory for parsed nodes rather than on types of information available from a surface structure grammar, it would be reasonable to hypothesize that \( PA(k) \) parsing serves as a more adequate model.

SEMANTICS IN PARSING

The type of computation procedures outlined earlier for building trees over surface structures top-down, including clearing completed phrases (a non-terminal plus all its immediate constituents) from the parser into a semantic analyzer, allows one to make some specific hypotheses about the nature of the interaction between parsing at any point in a string and the meaning of the previously parsed material. That such interaction exists is evidenced by cases like (1a) and (1b), and many other cases that can be constructed similarly. We might mention here, also, the intuition of native speakers that part of the process of understanding a sentence consists in at each point relating the new terminals to what has gone on before; the fact that in speaking and hearing it is possible for a sentence to be punctuated by a long pause or some digression, and then resumed without resulting difficulty of interpretation, is also relevant here.

The basic idea (suggested to me by Wise and Shapiro in personal communication) is that at each stage of a parse, after some completed phrase has been removed from the parser and placed in the semantic analyzer, the analyzer may be thought of as containing a function the argument of which is the next thing to be formed in the parser. We may call this the ongoing function hypothesis. To see how this idea works, consider a successive stage of the top-down parse of a sentence like (27):

(27) Tom wanted to ask Susan to bake a cake.
The surface structure of this tree is shown in (28):

After Tom is read, the structure left in the parser. When wanted is read, with a look-ahead of one, we will have the following phrases cleared to semantics:

presumably with pointers indicating how these pieces might be reassembled. What goes on in the semantic processor, or what the form of semantic representation is, is beyond the scope of this discussion. However, I conjecture that at this point we have in the semantics a function, which we might call (Tom want), that requires an argument. In a sense, then, the parser ‘knows’ that what it comes up with in the parse of the next node, basically to ask Susan to VP, must be a potential semantic argument for this function. Likewise when, at the next stage of the parse, there will be a function in semantics (ask Susan to) that will require a semantically suitable argument to be delivered from the parser. Thus, if the parser is forced to make a choice between alternative analyses, it may make reference in this choice to semantics in order to determine what would be a suitable argument for the function built at that time. As each argument is assimilated into semantics, successive functions are constructed there.

Now, in the case cited earlier the parse is unique, so the feedback from semantics has no discernible effect. However, if we consider a case of a potentially ambiguous sentence, the effect will be observable. Consider a sentence like (29):

(29) Joe encouraged the man to please Betty.

Such a construction has three meanings, depending on the attachment of the final infinitive phrase, as shown in (30):
In (30a), the man to please Betty is a phrase, originating in deep structure from a relative clause, the man who is to please Betty. In (30b), the man was encouraged to do something, namely, to please Betty, and in (30c), Joe encouraged the man for the purpose of pleasing Betty. It is, in fact, possible to find all three clauses present in the same sentence, as in Joe encouraged the man to be shot to make a prayer to please the priest. (In fact, this last is ambiguous between whether to make a prayer to please the priest is a phrase, where the future victim is pleasing the priest, or whether to please the priest is attached as in (30c), where Joe is pleasing the priest.)

Now, the principles of parsing of Kimball (1973) predict that of the readings (30a), (30b), and (30c), (30a) ought to be preferable, (30b) second in preference, and (30c) third in preference. The argument here is on the basis of an analogy to constructions like Tom said that Bill told Mary to leave yesterday, in which the adverb is read most easily as on the lower VP, leave, less easily as on the middle VP, told, and least easily as attached to the highest VP, said. Whereas these principles work successfully for the latter type of case, I think that for a sentence like (30), the (30b) and (30c) readings come to mind most readily for speakers and are of equal difficulty, while the (30a) reading is not so obvious, though perhaps not difficult to achieve for reasons of parsability. This does not necessarily spell doom for the principles mentioned previously, for it is reasonable to assume that these might be overshadowed by some principles having to do with areas of sentence processing other than parsing alone, e.g., semantics. In fact, I will try to show how the ongoing function hypothesis concerning the interaction of semantics and parsing could explain the preferred reading of (30) as (30b).

After Joe encouraged the man is read and parsed top-down, we have in the semantic processor the following tree chunks:
A fact available to semantics is that *encourage* may take a sentential object in an underlying structure of the form:

(31)

The S here will be reduced to a VP by the application of equi-NP deletion. So in semantics, the missing structure would be reconstructed, and we have a tree of the form:

(32)

In other words, the semantic processor is waiting for a VP to fill out the missing VP from the S being reconstructed. If the analysis did proceed along these lines, then (30b) would be the imputed surface structure.

There are a number of loose ends in the suggestion being made here. One of these is the suggestion that underlying structures are computed in the semantic processor, which is by no means obvious. In fact, if this is the case, then it can be shown that for certain sentences the computation of underlying structures must wait until the end of the sentence is read in. Such a sentence is (33), discussed first, I think, in Lakoff (1967):

(33) *The bagel was believed by Sam to be said by Harry to be thought by Tom to have been eaten by Mable.*

In the underlying structure *bagel* is the direct object of the most deeply embedded verb, *eat*; thus, if the semantic processor does compute underlying structure, it is clear from this and numerous other examples that can be brought forth that this computation cannot be on-line, so to speak, but can be effected only after the entire sentence is read in and parsed.

However, if it is by virtue of underlying structure that decisions about the parse of a sentence based on semantics can be made, then clearly this cannot be an ongoing interaction between semantic analysis and parse decisions, at least in the case of many sentences. If we drop the hypothesis that in the
semantic analyzer the semantic information is extracted from a string by
first reconstructing the underlying structure and argue instead that it is di-
rectly represented in the surface structure (which is one interpretation of the
significance of the condition that transformations preserve meaning), then
we can explain ongoing interaction of semantics and parsing. (In this case
we will be forced to reanalyze what goes on in a sentence like (30) to give
it the preferred reading on the basis of semantics.)

THE UNIVERSALITY OF PA(k) PARSE PRINCIPLES

The essential thrust of this study has been to construct a set of hypotheses
about surface structure parsing in natural language that will explain the cases
of perceptual complexity. As remarked earlier, in Kimball (1973) the pro-
cedure was to construct a metric of complexity in terms of the violation of
one or another of a set of principles governing the construction of parse
trees. The hypotheses presented here are intended to be more general, in the
sense of allowing these principles as consequences. The hypotheses themselves
are (a) surface parse trees are built by predictive analysis (Step S), and
(b) memory load is kept minimal, where the absolute limit on memory is
two sentences.

From these hypotheses it was possible to construct the notion of optimal
parse tree. The prediction made here is that it is only the form of the surface
tree that can contribute to surface complexity; in particular, the complexity is
proportional to the deviation of the surface tree from an optimal tree for the
same string. This differs from some other hypotheses that, e.g., com-
plexity is proportional to the length of the derivational history of the sentence
(in terms of application of transformations) or is to be stated in terms of the
difficulty of applying certain semantic interpretive principles (Bever, 1970).

Evidence on this question can be found by examining the surface structure
trees of other languages; from these we can determine the extent to which
the form of the tree contributes to perceptual complexity. I will take two
examples from Japanese to illustrate the point.²

Japanese is distinct from English with respect to the order of clauses in
underlying structure; main clauses are verb final, and relative clauses precede
their antecedents, as shown in (34):

(34) a. \[ S \rightarrow NP \rightarrow VP \] b. \[ S \rightarrow NP \rightarrow V \]

Consider, now, a sentence like (35), which is difficult in English and has a
surface structure like that shown in (14):

² Thanks are due to my informant, Yasuko Kamata, for judgments on these sentences.
(35)  The man the dog the cat bit hit saw the girl.

In Japanese such a sentence has the surface structure shown in (36):

(36)

(Here -ga and -wa mark subjects, and -o marks objects.)

Example (36) is a left-branching structure, and so by the hypotheses presented earlier should be easily parsed; it is judged in fact as being acceptable.

On the other hand, in English a sentence like (37) is acceptable because it is right-branching on the surface:

(37)  The man knows that the girl said that Bill wants Sam to leave.

However, its Japanese counterpart, shown in (38), is not:

(38)

In Japanese sentences where there are subordinate clauses, -wa marks the subject of the main clause and -ga the subject of subordinate clauses. This,
in effect, allows the Japanese speaker one degree more of center embedding within the range of acceptability, for the first -wa signals a special register, which one might call MAIN CLAUSE, and then the clauses signaled by -ga pile up in the tape of the parser in the way described earlier. In effect, this means that the main clause is not counted in the application of the twosentence memory limitation principle. Thus, a sentence of the form NP-wa NP-ga NP-ga V V V is acceptable, while one like (38) of the form NP-wa NP-ga NP-ga NP-ga V V V V is not. But modulo this difference of -wa signaling main clauses, the memory limitation in Japanese is the same as in English, so we have evidence for its universality.

REFERENCES


Kuno, S., and Oettinger, A. Multiple path syntactic analyzer. Information Processing, 1963, 62. (a)

