SYNTACTIC ANALYSIS
FOR THE R2 SYSTEM

KENNETH BISS

UNIVERSITY OF ILLINOIS – URBANA, ILLINOIS
This work was supported by the U.S. Office of Education under Contract OE C-1-7-071213-4557; auxiliary support was provided by the Joint Services Electronics Program (U.S. Army, U.S. Navy, and U.S. Air Force) under Contract DAAB 07-67-C-0199.

Reproduction in whole or in part is permitted for any purpose of the United States Government.

Distribution of this report is unlimited.
ACKNOWLEDGMENT

I wish to express my gratitude to Professor R. T. Chien for his invaluable advice and guidance throughout the period of research for this report. Thanks are also extended to Jeff Schultz for supervising the construction of the dictionary for me and to Mrs. Julie Anson for the preparation of this document.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>PHRASE-STRUCTURE GRAMMAR</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>SOME PHRASE-STRUCTURE PARSING ALGORITHMS</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>DEPENDENCY GRAMMARS</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>SOME DEPENDENCY PARSING PROGRAMS</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>A DEPENDENCY PARSING PROGRAM FOR THE R2 SYSTEM</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>REFERENCES</td>
<td>34</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

The R2 system is being designed to automatically answer questions on the Illinois Driver's Manual--Rules of the Road. That is, the R2 system will accept any question based on Rules of the Road and, through some search procedure, pick the answer to the question. At some point in the search procedure it will be necessary to syntactically analyze the input question or some data text, or both. It is for this reason that we initiated the investigation which has led to this report.

In our search for a syntactic analysis program for the R2 system we came across many programs which were written in the last few years. Since we were interested in efficiency as well as effectiveness we studied the more basic immediate constituent analysis and dependency analysis leaving the transformational grammar programs for some later date.

In Sections Two through Five we will discuss some of the grammars and associated programs that were considered for our syntactic analysis program.

In Section Two we discuss phrase structure grammars and standard form phrase structure grammars showing how these grammars are used to generate the sentences of a language.

In Section Three we discuss immediate constituent analysis which is the analysis procedure usually associated with phrase structure
grammars and the predictive analysis program of Kuno\textsuperscript{10,11,12} which is based on a standard form grammar.

In Section Four we discuss dependency grammars and its relation to phrase structure grammars.

In Section Five we discuss some dependency grammar analysis programs, and finally, in Section Six, we discuss the result of our search for an analysis program, namely our own dependency grammar parsing program. This program is now operating on the CDC 1604.
2. PHRASE-STRUCTURE GRAMMAR

It is not known exactly how the speaker of a language picks a sentence to communicate an idea he wishes to express. In some way he picks a group of words from the vocabulary of the language he wishes to communicate in, and a structure which gives the relationships among the words he has picked. He then puts the words and the structure together, putting the words in an order which reflects the structure and modifying words where it is necessary to further reflect the structure. The listener then takes note of the words the speaker used, the order he used them in, and the modifications he made to the words. From this information the listener knows what idea the speaker was trying to communicate.

The rules which a listener uses to discover the underlying structure of the speaker's sentence are called the grammar of the language. These rules enable the listener to associate a structure with each sentence he hears, a structure which should be relatively independent of the lexical entries in the sentence.

The rules which the listener uses to parse sentences are the inverse of the rules which the speaker uses to generate sentences. Thus if we understand the process of sentence generation we should understand how to parse sentences. So let us now look at some grammars which enable us to generate sentences.
A context free phrase structure grammar consists of a non-terminal alphabet (e.g. NP-noun phrase, VP-verb phrase, etc.), a terminal alphabet (e.g. noun, verb, etc.) and a set of rewrite rules. These rewrite rules are of the form $S \rightarrow NP \ VP \ PD$ which means whenever we encounter the symbol $S$ we can replace it with the concatenated triple $NP \ VP \ PD$. For example, we might have the grammar with the following rules:

1) $S \rightarrow NP \ VP \ PD$
2) $NP' \rightarrow A \ N$
3) $PP' \rightarrow P \ NP'$
4) $NP' \rightarrow T \ N$
5) $VP \rightarrow V \ PP'$
6) $NP \rightarrow T \ NP'$
7) $A \rightarrow \text{adj}$
8) $V \rightarrow \text{verb}$
9) $N \rightarrow \text{noun}$
10) $T \rightarrow \text{def}$
11) $T \rightarrow \text{indef}$
12) $P \rightarrow \text{prep}$
13) $PD \rightarrow \text{pd}$

To generate sentences from the above grammar we successively apply the rules being sure to start the process with a rule whose left side is the initial symbol $S$. We can thus generate the sentence, "The old man sat on the bench." as is done on the next page.
\[
S \\
NP \ VP \ PD \\
T \ NP' \ VP \ PD \\
def \ NP' \ VP \ PD \\
def \ Adj \ N \ VP \ PD \\
def \ adj \ noun \ VP \ PD \\
def \ adj \ noun \ V \ PP'PD \\
def \ adj \ noun \ verb \ PP'PD \\
def \ adj \ noun \ verb \ P \ NP' \ PD \\
def \ adj \ noun \ verb \ prep \ NP' \ PD \\
def \ adj \ noun \ prep \ TN \ PD \\
def \ adj \ noun \ verb \ prep \ def \ noun \ period \\
\text{(the)(old)(man)(sat)(on)(the)(bench).}
\]

The above derivation can also be represented as a tree (fig. 1).
If we slightly modify the above rules we can form a standard form grammar whose rules are the following:

1) \( S \rightarrow \text{def NP VP PD} \)

1a) \( S \rightarrow \text{indef NP VP PD} \)

2) \( \text{NP} \rightarrow \text{adj N} \)

3) \( \text{N} \rightarrow \text{noun} \)

4) \( \text{VP} \rightarrow \text{verb PP'} \)

5) \( \text{PP'} \rightarrow \text{prep NP} \)

6) \( \text{NP} \rightarrow \text{def N} \)

6a) \( \text{NP} \rightarrow \text{indef N} \)

7) \( \text{PD} \rightarrow \text{pd} \)

Let us again generate "The old man sat on the bench." this time using the standard form grammar. This derivation is shown below and is represented by the tree of Figure 2.

\[
S \\
def \text{NP VP PD} \quad \text{rule 1} \\
def \text{adj N VP PD} \quad \text{rule 2} \\
def \text{adj noun VP PD} \quad \text{rule 3} \\
def \text{adj noun verb PP' PD} \quad \text{rule 4} \\
def \text{adj noun verb prep NP PD} \quad \text{rule 5} \\
def \text{adj noun verb prep def N PD} \quad \text{rule 6} \\
def \text{adj noun verb prep def noun PD} \quad \text{rule 3} \\
def \text{adj noun verb prep def noun pd} \quad \text{rule 7} \\
(\text{the})(\text{old})(\text{man})(\text{sat})(\text{on})(\text{the})(\text{bench}).
\]
The old man sat on the bench.
3. SOME PHRASE-STRUCTURE PARSING ALGORITHMS

Immediate constituent analysis is the inverse of context-free phrase structure sentence generation.

Instead of having rules which allow us to expand terms we have rules which allow us to combine terms. That is, the rewrite rules for an immediate constituent analysis are of the form $AjN \rightarrow NP'$, which states that an adjective and a noun which appear next to each other in a sentence may be combined to form a term called $NP'$. To analyze a sentence we first find the syntactic class of each word in the input string. Using the string of syntactic classes we search for adjacent words which the rules indicate can be combined. We put parentheses around the combined words and treat this parenthetic expression (p.e.) as a syntactic word class and search for new combinations.

Let us apply the idea of immediate constituent analysis to the sentence "The old man sat on the bench." assuming the rules of our analyses are the following:

1) $adj \text{ noun} \rightarrow \text{NP}$
2) $def \text{ noun} \rightarrow \text{NP}'$
3) $\text{NP VP PD} \rightarrow S$
4) $\text{prep NP'} \rightarrow \text{PP'}$
5) $\text{verb PP'} \rightarrow \text{VP}$
6) $def \text{ NP'} \rightarrow \text{NP}$

The analysis would then be:
Predictive analysis as used by Kuno\textsuperscript{10,11,12} is the inverse of standard form sentence generation. The rules used in the analysis are of the form \((S, \text{def})/\text{NP \ VP \ PD}\). This rule is a prediction that if a sentence starts with "def" it will be followed by NP VP PD. Some of the rules will be of the form \((N,\text{noun})/\Omega, \Omega \text{ being the empty string.}\)

The analysis starts by finding the syntactic classes of the words in the input string and putting the initial symbol \(S\) into a push down store (PDS). The machine then looks at the first word of the input string and tries to find a rule whose left side is \((S, \text{first word in input string})\). If no such rule is found the string is ill-formed. If a rule is found, the symbols of the right hand side of the rule replace \(S\) in the PDS and the analysis goes to the second word.

At the \(n\)\textsuperscript{th} step in the analysis, the symbol at the top of the PDS is compared with the \(n\)\textsuperscript{th} input symbol. If a rule whose left hand side is of the form \((\text{symbol at top of PDS}, n\text{\textsuperscript{th} input symbol})\) is not found, the string is ill-formed. If such a rule is found the right hand side of this rule replaces the symbol at the top of PDS and the analysis goes to the \((n+1)\textsuperscript{th}\) input symbol. If processing the last input symbol yields an empty PDS, we have a parsing.
As an example of predictive analysis let us parse the by-now familiar sentence "The old man sat on the bench," assuming the rules of our analysis system are as follows:

1) (S, def)/NP VP PD
1a) (S, indef)/NP VP PD
2) (NP, adj)/N
3) (N, noun)/\Omega
4) (VP, verb)/PP'
5) (PP', prep)/NP
6) (NP, def)/N
6a) (NP, indef)/N

The analysis would be:

The old man sat on the bench

contents of PDS

1) S
2) NP
   VP replacing S rule 1
   PD
3) N
   VP rule 2
   PD
4) VP rule 3
   PD
5) PP'                  rule 4
   PD
6) NP                    rule 5
   PD
7) N                     rule 6
   PD
8) PD                    rule 3
9) Ω                     rule 7

To form the phrase structure tree which underlies this sentence
we simply recall the rules which led us to step nine.
4. DEPENDENCY GRAMMARS

We say that word A depends on word B, or that B governs A, if A modifies or complements the meaning of B in a sentence. A grammar which is based on the idea of dependents and governors is called a dependency grammar.

Let us now summarize the formal theory of dependency grammars that Hays has developed.

A dependency grammar consists of a non-terminal alphabet (e.g. Npl, -Ving, etc.), a terminal alphabet whose members are words or morphemes, and a set of dependency rules. The rules are of the form $X_i(X_{i_1}, X_{i_2}, \ldots, X_{i_k}, *, X_{i_j}, \ldots, X_{i_m})$ where the $X_{i_1}$ belong to the non-terminal alphabet. This rule states that $X_{i_1}$ governs $X_{i_2}$, that $X_{i_1}$ will appear ahead of $X_{i_2}$, $X_{i_3}$ will appear ahead of $X_{i_4}$, and that $X_{i_5}$ will appear between $X_{i_6}$ and $X_{i_7}$. A rule of the form $*(X_k)$ indicates that $X_k$ need not have a governor, while a rule of the form $X_j(*)$ indicates that $X_j$ need not have any dependents.

To generate a sentence we first pick an element which appears in a rule of form $*(X_k)$. We then find rules of the form $X_k(X_{k_1}, \ldots, *, \ldots, X_{k_n})$ and attach the dependents of $X_k$ to $X_k$. We now search for rules which indicate dependents of the $X_{k_i}$ and make the dependency connections. We continue this process until we assign, to all units which have no dependents, a rule of the form $X_m(*)$. This completes the first step of the derivation. The second step of the derivation procedure replaces the non-terminal symbols with terminal symbols.
The process can be thought of as building a tree. The nodes of the tree are the symbols (first nonterminal, then replaced by terminal symbols). The connections between the nodes indicate dependency (i.e., if two words are directly connected in the tree, the lower node depends on the higher node). The independent symbol is the head of the tree, and the elements which appeared in rules of the form \(X_m(*)\) are at the bottom of the tree.

As a simple example let us derive the sentence "The old man sat on the bench," assuming we have the following rules:

1) \(I(V)\)
2) \(N(T, Aj, *)\)
3) \(N(T, *)\)
4) \(V(N, *, P)\)
5) \(T(*)\)
6) \(Aj(*)\)
7) \(P(*, N)\)

And assuming we have a function which carries out the following mapping:

\[V \rightarrow \text{sat, hit, run, etc.}\]
\[N \rightarrow \text{man, woman, bench, house, etc.}\]
\[P \rightarrow \text{on, in, etc.}\]
\[T \rightarrow \text{the, a, etc.}\]
\[Aj \rightarrow \text{old, new, large, etc.}\]

Applying the rules we get the following derivation:

\[\ast(V)\] rule 1
\[\ast(V(N, *, P))\] rule 4
This is equivalent to the tree of Figure 3.

![Figure 3](image)

Now applying the function to the tree of Figure 3 we get the tree of Figure 4.

![Figure 4](image)
Let us look more closely at the tree in Figure 4. If we project a line down from each node to a base line, we see that no projection line crosses a dependency link (Figure 5--the dashed lines are the projection lines).

![Figure 5](image1)

This properly is called projectivity and any sentence which satisfies this property is a projective sentence. Not all English sentences satisfy projectivity as Hays points out. An example due to Hays of a non-projective sentence appears in Figure 6. The point where the projection line and the dependency link cross is circled.

![Figure 6](image2)
It should be noted that an immediate constituent generation of the sentence in Figure 6 would show no relation between larger and than. Thus non-projective sentences are made up of non-immediate constituents. Thus the assumption that a sentence is projective is the same as assuming that the sentence is made up of immediate constituents.

Projectivity is equivalent to the statement: any word that occurs between two connected occurrences in a sentence must depend on either one of those words or the other.

Proof: Assume two words (X, Y) are connected in a projective sentence. Assume Z lies between X and Y in the sentence and assume Z does not depend on X or Y. Then Z must depend on some occurrence to the left of X or to the right of Y or Z is independent. If Z depends on something to the left of X then its dependency connection will cross the projection line of X (Figure 7).

If Z depends on something to the right of Y then Z's dependency connection crosses Y's projection line (Figure 8). If Z is independent then its projection line crosses the dependency connection between X and Y (Figure 9).
Before we give some examples of dependency parsing programs, let us see the relationship between projective dependency trees and immediate constituent analysis.

Hays has shown how we can derive a dependency parsing for a sentence given an immediate constituent parsing and vice versa. Let us now summarize how this is done.

Given a projective dependency tree let us derive an immediate
constituent analysis (i.e., a parenthetic expression).

First we embed the tree in a plane and drop projection lines from each node of the tree to the base line. Now insert parentheses around all points that are projected from complete subtrees and make all possible permutations of these p.e.'s.

If we now have p.e.'s with at least three other p.e.'s contained in them we can form optional p.e.'s. Consider a p.e.: we will call it A, which has \( N \geq 3 \) p.e.'s contained in it. A is the projection of a complete subtree. Now we take any 2, 3, ..., \( N-1 \) of the p.e.'s within A, including the head of the complete subtree which projects into A, and take any permutation of this new set of p.e.'s, making sure we keep the constituents contiguous. We can continue this process as long as we still have p.e.'s with at least three p.e.'s contained in them.

Let us look at an example:

![Figure 10](image-url)

\[
[((*) (*)& (*)&)] (*)& [(*)& (*)(*)] \]
Figure 10 shows the transformation from the tree to an immediate constituent analysis with the optional p.e.'s in brackets.

Now let us see how to go from a p.e. to a dependency tree.

First we strip away the outermost parenthesis leaving two or more p.e.'s. Choose one of these and make it the head of a tree with the others dependent on it. If every node corresponds to an elementary p.e. then we are done. If not, look at each node which corresponds to a non-elementary p.e. Drop the outermost parenthesis of the p.e. thus leaving two or more p.e.'s. Choose one of these to correspond to the given node. Continue this process until all nodes correspond to elementary p.e.'s.

As an example, let us reverse the procedure used in Figure 10. Figure 11 shows two dependency trees which were derived from the given parenthetic expression.
Figure 11
5. SOME DEPENDENCY PARSING PROGRAMS

It is now obvious that dependency analysis and immediate constituent analysis are related. In fact, Gaifman has shown that dependency grammars and immediate constituent grammars are weakly equivalent. That is, they generate the same set of sentences from the same initial vocabulary or, analytically, they classify the same set of sentences. However, dependency grammars and immediate constituent grammars are not strongly equivalent, which means that there is no isomorphism between the structural diagrams each associates with a given sentence. Thus, there will be some sentences which are ambiguous to immediate constituent grammars (i.e., there will be several structural descriptions assigned to the sentence) but unambiguous to dependency grammars, and vice versa. This means, as points out, that immediate constituent grammars obtain something of syntax which dependency grammars miss, and vice versa.

Let us now look at some parsing programs which have used dependency grammars. A dependency parsing consists of finding the dependency tree for the sentence being parsed. The main verb of the sentence is usually taken as the head node of the tree with the other words of the sentence appearing at the other nodes.

has written a dependency grammar parsing program as part of machine translation system. The first step in the program is to
assign every word in the input sentence its grammar code symbol; this is done by dictionary lookup. Using the string of grammar code symbols the program checks all possible pairs of words which could be connected in a projective sentence for a dependency relation. We say that two words could be connected in a projective sentence if they satisfy precedence where precedence is defined in the following way.

\[ x \text{ precedes } y \iff \]

1) \( x \text{ appears to the left of } y \) in the input sentence.
2) Any word that is between \( x \) and \( y \) depends on either \( x \) or \( y \) (this is to insure projectivity)
3) Neither \( x \) depends (directly or indirectly) on \( y \) or \( y \) depends on \( x \) (this is to insure no circular dependency connections).
4) Either \( x \) or \( y \) or both are independent (this is to insure that all elements in the sentence have only one governor).

When a pair of words are tested for and become a precedence pair, we can test the pair for a dependency connection. We will allow \( x \) to depend on \( y \) if

1) \( x \text{ is independent} \)
2) \( x \text{ precedes } y \) or \( y \text{ precedes } x \)
3) there is an entry in the table of dependency types which states that the grammar code symbol of \( x \) can depend on the grammar code symbol of \( y \).

If the above three conditions are satisfied we make the connection between \( x \) and \( y \) then modify the grammar code symbols of both \( x \) and \( y \) to
show the properties of the pair \(x,y\). This modification turns the \(x,y\) pair into a constituent of an immediate constituent analysis. In this way Hays obtained an immediate constituent analysis along with the dependency analysis. The final output of Hay's program is all possible dependency parsings of the input sentence compatible with the grammar of the system, along with the associated immediate constituent analysis. Hays decided to give both immediate constituent parsings and dependency parsings because, as we pointed out a few pages ago, each captures something of grammar which the other misses.

Simmons has also written a dependency parsings program called the "Pattern-Learning Parser" (P-LP) which "learns" the grammar rules used in the grammar from a set of hand-parsed sentences.

To be more specific, a set of sentences are hand parsed and each word in the sentences is assigned a number to show at what level it appeared in the dependency tree. For example:

\[
\begin{align*}
0 &-3 -3 -2 +1 +2 -4 +3 0
\end{align*}
\]

would be assigned the numbers which appear below the sentence (note that the numbers on the object side of the verb are positive). The P-LP then forms a word class for each word encountered in the hand parsed text, and a set of sentence pattern rules which indicate the structure of the dependency trees of the hand parsed text.

The word class for a particular word is of the form \(a/b/c\) where "a" is the depth in a dependency tree at which the particular word occurred, "b" is the depth at which the word to the left of "a" occurred,
show the properties of the pair x,y. This modification turns the x,y pair into a constituent of an immediate constituent analysis. In this way Hays obtained an immediate constituent analysis along with the dependency analysis. The final output of Hay's program is all possible dependency parsings of the input sentence compatible with the grammar of the system, along with the associated immediate constituent analysis. Hays decided to give both immediate constituent parsings and dependency parsings because, as we pointed out a few pages ago, each captures something of grammar which the other misses.

Simmons has also written a dependency parsings program called the "Pattern-Learning Parser" (P-LP) which "learns" the grammar rules used in the grammar from a set of hand-parsed sentences.

To be more specific, a set of sentences are hand parsed and each word in the sentences is assigned a number to show at what level it appeared in the dependency tree. For example:

. "The old man sat on the bench."
0 -3 -3 -2 +1 +2 -4 +3 0

would be assigned the numbers which appear below the sentence (note that the numbers on the object side of the verb are positive). The P-LP then forms a word class for each word encountered in the hand parsed text, and a set of sentence pattern rules which indicate the structure of the dependency trees of the hand parsed text.

The word class for a particular word is of the form a/b/c where "a" is the depth in a dependency tree at which the particular word occurred, "b" is the depth at which the word to the left of "a" occurred,
and "c" is the depth at which the word to the right of "a" occurred. If the word has occurred in several places of the hand parsed text then all of the word classes which have been assigned to the word are logically added. This yields an expanded word class of the "a,b,c/d,e,f,g,h,j" where a,b,c are the depths at which the word has occurred in the various trees, etc.

The sentence pattern rules formed from a hand parsed sentence indicate the structure of the dependency tree of the sentence above a certain level. For example, the word "The" of the sentence "The old man sat on the bench" generates the rule "-3/-3/-2/+1/+3/+3/0/0," "man" generates '2/+1/+2/00' and "bench" generates '+3/0/-2/+1/+2/0" (here the zeros are used as directional markers). When a sentence is given to the P-LP to parse, the program first finds, for each word in the input string, a word class. Then certain parts of adjacent words word classes are logically added. This logical addition allows the words to the left and right of the word in question to predict at what level the word will occur.

The new set of word classes for each word are then checked against a table of possible triads which indicate what 3-tuples of the form a/b/c were encountered in the sample text. If a particular triad was never encountered in the sample text then it is not allowed to occur in the word classes of the input sentence. To help reduce ambiguity at this stage, certain parts of the word classes are logically subtracted with their left and right neighbors (this is the reverse of the logical addition which was previously done).
The altered word class of each word of the sentence is now intersected with the original word class assigned to the word. If the intersection leaves the word class unchanged or reduces it to zero, no change is made in the words altered word class. If the class is changed by the intersection, the result of the intersection replaces the altered word class (this gives greater weight to the experienced depth code over neighbors predictions). The words of the input sentence now should be almost unambiguously assigned a depth in the dependency tree. The governor is easily found since a word at level-3 requires a governor at level +2, etc. If after looking for all governors for all words of the sentence ambiguity still exists, we apply the sentence pattern rules.

The P-LP outputs multiple parsings of a given input sentence of which the first parsing is chosen.
6. A DEPENDENCY PARSING PROGRAM FOR THE R2 SYSTEM

After careful consideration we decided to take a different approach than that suggested by Hays' and Simmons' programs. We decided not to take Hays' approach for two reasons. First, we felt that for the R2 system we did not need to impose the restriction that all sentences must be parsed as projective sentences. Second, we felt that we could get useful results by obtaining the most probable parsing for a given sentence. By allowing only one parsing we can save time in the program and we can eliminate the problems associated with multiple parsings.

We decided not to take Simmons' approach also for two reasons. First, we felt that the rules which the P-LP generates might be too general for the R2 system. Second, as we stated above, we wanted a program which would select the most probable parsing for any sentence.

We have written a dependency parsing program which will eventually be integrated into the R2 system. The program is written in CSL6 (the local form of the L6 Language) for programming ease.

We chose to use a dependency grammar parser for two reasons. First, it seemed to us that dependency analysis would lend itself to fast running, easily written programs. Second, it appeared to us that a dependency grammar would give us the most revealing structural description of the sentences in Rules of the Road.
Let us now explain our program. The program uses two tables. The first table is a dictionary containing words and their syntactic word classes. Any word is allowed to be a member of up to three syntactic classes. The second table is a table of grammar rules. Following Hays, this table lists the syntactic class of a dependent, the syntactic class which may govern the syntactic class of the dependent, and a number which indicates the order which the governor and its dependent must have (i.e., the entry is two if the dependent must be to the left of the governor, one if the governor is to the left of the dependent, and zero if there is no order restriction in the rule).

The input to the program is the words of the sentence to be parsed. The first step is to assign the syntactic classes to all the words of the input sentence; this is done by dictionary lookup. The program is now ready to start the actual parsing.

Initially, the first word of the input string is considered a dependent and the program searches for its governor. We first test the word to the immediate right of this particular word, then the second word to the right, etc., until we find a governor for the first word. To test for a dependent-governor pair we go to the table of rules. If the class of the first word of the sentence is the first word of a rule and the class of the candidate governor is the second word of a rule, and if the word order required by the rule is satisfied, then the first word is connected to this candidate governor and we go to the second word of the sentence. If we are at the $k^{th}$ word of the sentence we consider the $k^{th}$ word as a dependent and then check the words at distance
one from the dependent for governorship, checking the \((k+1)^{th}\) word first, then the \((k-1)^{th}\) word. If neither of these is the governor we go to a distance two from the \(k^{th}\) word, checking to the right first, then to the left. If no governor is found we continue to search further away from the \(k^{th}\) word until either the governor of the \(k^{th}\) word is found or until we have checked all other words in the sentence. If no governor is found the word is taken to be the head of the dependency tree underlying the sentence. If the \(k^{th}\) word has more than one syntactic class associated with it, then we search for a governor for each syntactic class. That is, if word \(j\) has syntactic classes "A" and "B" associated with it, we first search for a governor for the \(j^{th}\) word as above, assuming the \(j^{th}\) word is a member of syntactic class "A." Once this governor is found we start again at the \(j^{th}\) word, now assuming that the word is a member of syntactic class "B." If in searching for the governor of the \(m^{th}\) word we came across a governor candidate which has two or three syntactic classes associated with it, we first test for governorship assuming the governor candidate is a member of the first class assigned to it. If a dependency connection is made we go to the \((m+1)^{th}\) word and start over looking now for a governor of the \((m+1)^{th}\) word. If a dependency connection is not made we test the second syntactic class of the governor candidate for agreement with the \(m^{th}\) word. That is, if in searching for the governor of the \(m^{th}\) word we encounter the \(n^{th}\) word which has classes "C" and "D" associated with it, we check the \(m^{th}\) and the \(n^{th}\) word for agreement assuming the \(n^{th}\) word is a member of class "C." If a connection is made in this way
between the \( m^{th} \) and \( n^{th} \) words, we go to the \((m+1)^{th}\) word and start searching for its governor. If no connection is made we check for agreement between the \( m^{th} \) and \( n^{th} \) words assuming the \( n^{th} \) word is a member of syntactic class "D." If a connection is then made we go to the \((m+1)^{th}\) word and search for its governor. If no connection is made we leave the \( n^{th} \) word and search for the governor of the \( m^{th} \) word elsewhere.

Be searching in this fashion we emphasize that the governor for any word is most likely to occur very near the particular word in question.

To test our program we first compiled a dictionary containing all the words from *Rules of the Road*. We then selected a small set of simple, compound and complex sentences from the driver's manual, selecting several sentences from each chapter of the manual.

The results of this test indicate that the parsing we get has a fairly high probability of being the correct parsing. If this parsing is the wrong parsing, then we could attach dependents to the second candidate governor instead of the first. This would give another parsing keeping with the philosophy that dependents are close to their governors; although in general, this parsing would have a lower probability of being correct than the first parsing. This is, in fact, what we intend to do in the future. More precisely we will form a sequential procedure which employs a semantic-syntactic loop in which the parsings of a sentence will be ranked according to their probability of being correct. When a sentence is presented to this semantic-syntactic loop system it will be assigned its most probable parsing. This parsing
will then be interpreted semantically to see if it is indeed the correct parsing. If not, we will assign the sentence its second most probable parsing and check again. We will continue to do this until we get the correct parsing for the sentence.

We can now see in what way our approach to sentence structure determination is better than the approaches of Hays and Simmons. Hays and Simmons produced all possible parsings for a sentence with no rank to them. Therefore each parsing was considered to have equal probability of being correct. If this equal probability approach were used in a question-answering system we would have to produce one answer for each parsing of the question. By using our approach we eliminate the need to consider all the parsings of a sentence thus saving time, and we will eliminate most of the irrelevant answers to a question.

Let us now look at some of the sentences which we have parsed with our program (Figure 12 a, b, c).

The     D Shape
Eight-sided D Shape
Octagon  D Shape
Shape    D Means Means
Always   D Means
Means    D
Stop     D Means Means

Figure 12a
A D Sign
Stop D Sign Sign
Sign D Is
Is D
Red D Is
With D Red
White D Letters
Letters D With

Figure 12b

Signs D Posted Like
Like D Signs Signs Posted
This D Are Are
Are D Posted
Posted D
On D Posted
Main D Highways On
Highways D On

Figure 12c

Figure 12

The sentences in Figure 12a and Figure 12b are unambiguously parsed and assigned the tree shown in Figure 13a and Figure 13b respectively
The sentence of Figure 12c however is not unambiguously parsed since the program tells us that "signs" depends on either "posted" or "like." It will be noticed, however, that "this" is unambiguously
connected to the wrong governor. It is connected to the wrong governor because it finds the word "are" which is a possible governor and stops never getting to the word "like" which is the actual governor. We would like to point out though, that "this" is the only word in the sentence which is mis-connected due to our philosophy of parsing and that in most of the sample sentences which had faulty connections due to our method of parsing there was only one such faulty connection. Looking at the sentence in Figure 12c again, we also see that if we are allowed to make a second pass at the sentence, allowing "this" to look for its second most probable governor, the connection between "this" and "like" would be established and we would have the correct parsing of the sentence.


Distribution List as of November 1, 1968

1 Dr. K.V. Noble
   Air Force Automote Laboratory
   Wright-Patterson AFB, Ohio 45433

2 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

3 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

4 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

5 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

6 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

7 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

8 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

9 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

10 Lt Col Robert S. Liddell
    Chief, Electronic Division
    Air Force Office of Scientific Research
    Arlington, Virginia 22209

11 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

12 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

13 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

14 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

15 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

16 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

17 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

18 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

19 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

20 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

21 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

22 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

23 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

24 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

25 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

26 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

27 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

28 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

29 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

30 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

31 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

32 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

33 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

34 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

35 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

36 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

37 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

38 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

39 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

40 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

41 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

42 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

43 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

44 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

45 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209

46 Lt Col Robert S. Liddell
   Chief, Electronic Division
   Air Force Office of Scientific Research
   Arlington, Virginia 22209
SYNTACTIC ANALYSIS FOR R2 SYSTEM

The R2 system is being designed to automatically answer questions on the Illinois Driver's Manual--Rules of the Road. That is, the R2 system will accept any question based on Rules of the Road and, through some search procedure, pick the answer to the question. At some point in the search it will be necessary to syntactically analyze the input question or some data text, or both. It is for this reason that we initiated the investigation which has led to this report.
<table>
<thead>
<tr>
<th>KEY WORDS</th>
<th>LINK A</th>
<th>LINK B</th>
<th>LINK C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntactic Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automatic Sentence Structure Determination</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automatic Sentence Parsing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A Dependency Grammar Parsing Program</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review of Dependency Grammars</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>