Computational Linguistics

A. G. OETTINGER, Editor

The Predictive Analyzer and a Path Elimination Technique

SUSUMU KUNO
Harvard University, Cambridge, Massachusetts*

Some of the characteristic features of a predictive analyzer, a system of syntactic analysis now operational at Harvard on an IBM 7094, are delineated. The advantages and disadvantages of the system are discussed in comparison to those of an immediate constituent analyzer, developed at the RAND Corporation with Robinson's English grammar. In addition, a new technique is described for repetitive path elimination for a predictive analyzer, which can now claim efficiency both in processing time and core storage requirement.

1. The Predictive Analyzer

A predictive analyzer produces for a given sentence all possible syntactic interpretations compatible with the current version of the predictive grammar. The predictive grammar G' comprises rules of the form:

$$(Z, c) \mid Y_1 \cdots Y_m \qquad (m \geq 1),$$

$$(Z, c) \mid \Lambda \qquad (m = 0),$$

where Z, Y_i are intermediate symbols (i.e., syntactic structures), c is a terminal symbol (i.e., syntactic word class) and Λ denotes the absence of any symbol. (Z, c) is called an argument pair. (SE, prn) | VP PD, for example, indicates that a sentence (SE) can be initiated by a prn (personal pronoun in the nominative case) if the prn is followed by a predicate (VP) and a period (PD). A fragment of our current English grammar is shown in Kuno and Oettinger [1, 2]. It has been proved by Greibach [3, 4, 5] that G' is an exact inverse[1] of a standard-form grammar G whose rules are of the form:

$Z \rightarrow cY_1 \cdots Y_m$ where $(Z, c) \mid Y_1 \cdots Y_m$ is a rule in G', or

$Z \rightarrow c$ where $(Z, c) \mid \Lambda$ is a rule in G'.

Since Greibach has proved that every context-free language can be generated by a standard-form grammar, the Harvard analyzer could accept any context-free language given a suitable predictive grammar.[2]

Consider a predictive grammar which does not contain more than one rule with the same argument pair, and an input string of words each of which is associated with a unique terminal symbol. The analysis of the sequence of terminal symbols $c_1 \cdots c_n$ is initiated with a pushdown store (PDS) containing some designated initial symbol. At word k in the course of the analysis of the string, an argument pair (Z, c_k) is formed from the intermediate symbol Z topmost in the PDS and the current terminal symbol c_k. If a rule with this argument pair is not found in the grammar, the input string is ill-formed (ungrammatical). If it is found, we say that Z has been fulfilled by the rule $(Z, c_k) \mid Y_1 \cdots Y_m$ (or $\mid \Lambda$), or simply by c_k. A set of new intermediate symbols $Y_1 \cdots Y_m$ or Λ then replaces the topmost intermediate symbol Z of the pushdown store and the analysis flow goes to $k+1$. The input string is well formed (grammatical) if the last terminal symbol c_n is processed yielding an empty PDS. A set of standard-form rules corresponding to the predictive rules used for the analysis of the string gives the derivational history of the string in the original standard-form grammar.

Actually, a grammar may have more than one rule with the same argument pair. Also, a word in an input string may be associated with more than one terminal symbol. Therefore, a mechanism for cycling through all possible combinations of these rules and terminal symbols must be superimposed on the simple pushdown store machine described in the previous paragraph.

Let $W(k)$ be a set of terminal symbols corresponding to word k in a given sentence and $S(k)$ a set of rules in the grammar whose argument pairs contain a member of $W(k)$. $S(k)$, then, defines a subset of the grammar which will be used for the processing of the kth word of this particular sentence. There are various techniques for referencing all the members of $S(k)$, but for the description of the cycling mechanism of the predictive analyzer, let it suffice to say that the first member of $S(k)$ is located at address $I(k)$ and its last member at $L(k)$.

The program utilizes two sets of registers: $A(i)$ and $B(i)$ for $i = 1, 2, \cdots, n$. Prior to the initiation of the analysis of the sentence, $A(i)$, the $S(i)$ boundary register, is set to $[I(i), L(i)]$. The active rule pointer $B(i)$ is set to $I(i)$ for all the values of i (see Table 1). Let $Z(k)$ be the topmost symbol in the PDS prior to the processing of a rule at k, and let $sr(B(k))$ be the rule currently to be used at k. The flow of the analysis procedure is shown in Table 2 (the explanation of "reconstruct the PDS configuration of k" in step 7 is given in the next paragraph).

This work has been supported in part by the National Science Foundation under grants GN-162 and GN-329.

* Computation Laboratory.

[1] If an analyzer with grammar A accepts all and only those sentences that are generated by B, we call A an exact inverse of B.

[2] Given a context-free grammar G'', we can automatically construct a standard-form grammar G' which generates the same language as G'' does. However, the structural descriptions assigned to a given sentence by G' are usually different from the ones assigned to the same sentence by G''. See Section 2(5) for further discussions on structural descriptions.

Assume that, at k in the course of the analysis of a sentence, the PDS contains two intermediate symbols $\boxed{P \mid R}$, and that, due to the rule $(P, a) \mid C_1 \cdots C_m$, we have the new PDS configuration $\boxed{C_1 \mid \cdots \mid C_m \mid R}$ at $k+1$. In step 7 of the analysis flow, the analysis path has to go back from $k+1$ to k, and the PDS configuration $\boxed{P \mid R}$ has to be reconstructed out of the current configuration $\boxed{C_1 \mid \cdots \mid C_m \mid R}$. In order to make this process efficient, the program uses two additional sets of registers $\alpha(i)$ and $\beta(i)$. $\alpha(i)$ is used to store $Z(i)$, and $\beta(i)$ is used to store the position of $Z(i)$ in the PDS counting from the bottom. Therefore, in the example under discussion the four registers at k and $k+1$ contain the information shown in Table 3. Now, given $\boxed{C_1 \mid \cdots \mid C_m \mid R}$ at $k+1$, the PDS configuration at k can be restored by erasing $\beta(k)$-th, $(\beta(k)+1)$-st, $\cdots$, $(\beta(k+1))$-st symbols in the current PDS and then by storing $\alpha(k)$ in the $\beta(k)$-th position in the PDS.

TABLE 1. Registers Used for Cycling Through Rules

i		1	$\cdots$	k	$\cdots$	n
$S(i)$ Boundary Register	$A(i)$	$[I(1), L(1)]$		$[I(k), L(k)]$		$[I(n), L(n)]$
Active Rule Pointer	$B(i)$	$I(1)$		$I(k)$		$I(n)$

TABLE 2. Basic Flow of a Predictive Analyzer

Step 0. Store the initial symbol X in the PDS.
Set $k = 1$.
Step 1. Does the argument pair of $sr(B(k))$ contain $Z(k)$? (Note: $Z(1) = X$.)
If no, go to step 6.
Step 2. Replace $Z(k)$ in the PDS by $sr(B(k))$'s set of new intermediate symbols $Y_1 \cdots Y_m$ (possibly Λ).
Step 3. $k + 1 \rightarrow k$
$k > n$? If no, go to step 1.
Step 4. Is the PDS empty? If no, go to step 7.
Step 5. Output $sr(B(i))$ for $i = 1, 2, \cdots, n$. Go to step 7.
Step 6. $B(k) + q \rightarrow B(k)$.
Note: q is a constant to be added to $B(k)$ to obtain the next rule in $S(k)$.
$B(k) > L(k)$? If yes, $I(k) \rightarrow B(k)$, and go to step 7.
If no, go to step 1.
Step 7. $k - 1 \rightarrow k$.
$k = 0$? If yes, the sentence has been exhaustively processed. Exit.
If no, reconstruct the PDS configuration of k, and go to step 6.

TABLE 3. Registers for the Reconstruction of PDS Configuration

i	k	$k+1$
$A(i)$	$[I(k), L(k)]$	$[I(k+1), L(k+1)]$
$B(i)$	XXX	XXX
$\alpha(i)$	P	C_1
$\beta(i)$	2	$2 + m - 1$

2. Characteristic Features of a Predictive Analyzer

As has been shown in the previous section, given a PDS configuration at word k, the number of analysis branches which emanate from k and reach $k+1$ is equal to $f(S(k+1))$, the number (possibly zero) of rules in $S(k+1)$ which have the same intermediate symbol in the argument pair as the PDS's topmost symbol $Z(k)$. From each of the PDS configurations thus formed at $k+1$, $f(S(k+2))$ (possibly zero) analysis branches emanate. With the use of the current English grammar, the number of analysis branches grows exponentially with sentence length for natural English sentences. In order to reduce this exponential effect to a linear one, a new repetitive path elimination technique has been devised and incorporated in the basic flow of the predictive analysis procedure described in Table 2. Before going into the details of this technique, it is worthwhile to compare a predictive analyzer with the immediate constituent (IC) analyzer of the RAND Corporation[3] to clarify some salient differences between the two systems.

It is shown below that a predictive analyzer, in its basic flow, achieves efficiency in core storage requirement at the sacrifice of efficiency in processing time, while an IC analyzer[4] achieves efficiency in the latter at a sacrifice of efficiency in the former. It is shown in Section 3 that the technique for repetitive path elimination that has been devised for a predictive analyzer is a means for making the predictive analyzer efficient both in core storage requirement and processing time.

The IC analyzer of the RAND Corporation is based on Cocke's [7] parsing algorithm[5] and utilizes Robinson's [9] IC grammar for English. The grammar is an exact inverse of a binary grammar: $(A, B) \mid Z$ is a rule in an IC grammar where $Z \rightarrow A\ B$ is a rule in the corresponding binary grammar, where Z is an intermediate symbol and A and B are either intermediate or terminal.

The parsing algorithm of an IC analyzer works as follows: initially, all two-word combinations are tested to see if they form syntactic constitutes.[6] If yes, the intermediate symbols representing the constitutes are stored in core together with the identification of the position of their constituents. Next, all three-word combinations are tested. The string k, $k+1$, $k+2$, for example, consists of immediate constituents $((k, k+1), k+2)$ or $(k, (k+1, k+2))$, and so on. If $(k, k+1)$ has not been identified

[3] The predictive analyzer at Harvard and the IC analyzer at the RAND Corporation are the only currently operational systems with considerably detailed English grammars. Models of context-free grammars are also used in connection with programming languages. See Floyd [6] for procedures of syntactic analysis developed for translation of programming languages.

[4] Throughout this paper, the term "an IC analyzer" is used specifically to refer to a context-free recognition procedure based on the same principle as the RAND analyzer is.

[5] Sakai [8] has independently developed an IC analysis algorithm very similar to Cocke's.

[6] Two substrings α and β, corresponding to A and B, respectively, of $(A, B) \mid Z$ are called (immediate) *constituents*, and the entire string $\alpha\beta$ corresponding to Z is called a *constitute*.

previously as a constitute, $((k, k+1), k+2)$ is rejected immediately. If it has, the intermediate symbol representing $(k, k+1)$ and the one representing $k+2$ are looked up in the grammar to see if they form a constitute. If they do, the intermediate symbol representing the whole constitute is stored in core with the identification of its two constituents $(k, k+1)$ and $k+2$. The same analysis procedure applies to $(k, (k+1, k+2))$, and so forth.

For the processing of the four-word combination k, $k+1$, $k+2$, $k+3$, the possible bracketings are $(k, (k+1, k+2, k+3))$, $((k, k+1), (k+2, k+3))$, and $((k, k+1, k+2), k+3)$. It is to be noted that if two-word combinations $(k, k+1)$ and $(k+2, k+3)$, and three-word combinations $(k+1, k+2, k+3)$ and $(k, k+1, k+2)$, contained in this four-word combination are syntactic constitutes, they must have been previously identified as such and must have been recorded in core with their identified intermediate symbols. The analysis proceeds to larger and larger constitutes until the n-word combination (n is the number of words in the sentence) has been processed. If there is an intermediate symbol of "sentence" identified for this n-word combination, the history of the path which has led to this designated symbol corresponds to an immediate constituent analysis of the sentence.

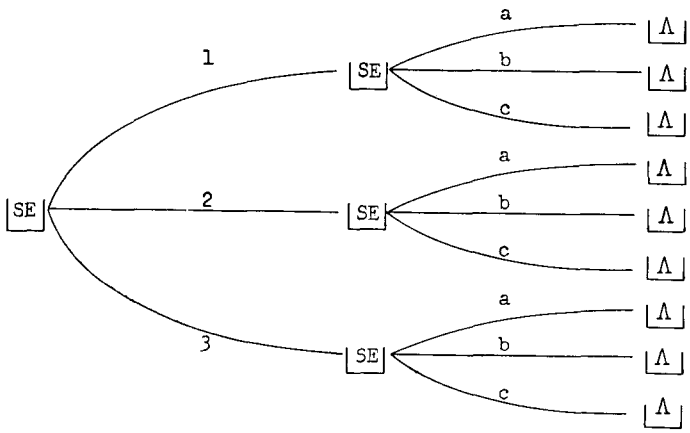
Now we are ready to compare some of the characteristic features of a predictive analyzer and an IC analyzer.

(1) A predictive analyzer, due to the cycling mechanism described in the previous section, keeps track of possible analysis paths utilizing a single PDS at any one time. Given the maximum value of n allowable for a given system ($\max n = 72$ for the current system on an IBM 7094), there is no problem of core storage overflow. An IC analyzer, on the other hand, stores all the intermediate constitutes in core for easy access in the processing of larger constitutes. This produces a problem of core storage overflow, and sometimes the current RAND analyzer, with the current Robinson grammar, actually causes overflow for sentences with 30 words. This is considered by the author to be the major drawback of an IC analyzer as compared with a predictive analyzer.

(1a) An IC analyzer recognizes, and can list as by-products, all the well-formed substrings contained in the sentence, while a predictive analyzer, in its basic flow of logic, does not recognize well formed substrings.

TABLE 4. Repetitive Analysis Paths

TIME FLIES LIKE AN ARROW AND THEY ARE FLYING PLANES.



(2) In an IC analyzer, once a given combination of words is identified as a syntactic constitute with a certain set of intermediate symbols assigned to it, the same combination will be regarded as a unit for the processing of larger constitutes; that is, no internal analysis of the constitute will be repeated. In a predictive analyzer, on the other hand, whenever an intermediate symbol becomes topmost in the PDS more than once at a given word position k, all rules in $S(k)$ are processed repeatedly.

For example, assume that the input sentence "TIME FLIES LIKE AN ARROW AND THEY ARE FLYING PLANES." is to be analyzed by a predictive analyzer. The two clauses in the sentence are triply ambiguous and their ambiguities are mutually independent.

TIME FLIES LIKE AN ARROW

1. Time passes as quickly as an arrow.
2. A species of flies called "time flies" are fond of an arrow.
3. You shall time the flies which are like an arrow. (or You shall, as quickly as an arrow, time the flies.)

THEY ARE FLYING PLANES

a. They are planes which are flying.
b. The facts are that the flying planes (over the hill). cf. The facts are that the flying kills.
c. These people are flying the planes.

After the first interpretation of the first clause is obtained and the coordinating conjunction is processed, the PDS contains SE as the topmost intermediate symbol (see Table 4). Three interpretations a, b and c are obtained for the second clause. The analysis path eventually backs up to the beginning of the sentence, and the second interpretation for the first clause is obtained. SE becomes topmost in the PDS again after the processing of "AND". Now, all the analysis branches that were traversed before when SE became topmost for the first time have to be traversed again. The three interpretations of "THEY ARE FLYING PLANES" are rediscovered one by one. The same situation arises when the third interpretation of "TIME FLIES LIKE AN ARROW" is obtained and when SE becomes topmost in the PDS for the third time at the same word position.

Since the number of active paths at a given word position can easily reach the order of 10,000, and since there are approximately 80 distinct intermediate symbols currently recognized, a given rule at a given word position may very easily be processed hundreds of times in the current English analyzer. This is considered by the author to be the major drawback of a predictive analyzer as opposed to an IC analyzer.

(3) A predictive analyzer produces multiple analyses of a given sentence one by one whenever the end of the sentence is reached with an empty PDS. An IC analyzer, on the other hand, produces multiple analyses of the sentence simultaneously when the n-word combination has been processed with one or more resulting intermediate

symbols. In cases when it is desirable to terminate the analysis procedure of a given sentence (a) after the first analysis has been obtained, (b) after a limited processing time is exceeded, or (c) after an analysis has been obtained which is regarded as desirable by man-machine communication or by some arbitrary criterion, a predictive analyzer is quite flexible and can meet any one of these three requirements. An IC analyzer, on the other hand, produces either no analysis or all analyses of the whole sentence. If the maximum processing time is exceeded and processing of the sentence has to be terminated, no analysis of the sentence can be obtained. However, as was mentioned in (1a), the well-formed substrings of the length that has been reached so far can be obtained.

(4) A standard-form grammar, of which a predictive grammar is an exact inverse, usually has more rules than the context-free grammar from which it is derived and to which it is equivalent. Consider a context-free grammar which contains the following set of ten rules:[7]

1. SE → NP VP PD (cf. *"This is true."*)
2. SE → NP PA SE (cf. *"It being Sunday, we have no school."*)
3. SG → NP VP (cf. "I think that *it is true.*")
4. SG′ → NP VP′ (cf. "This is what *I like.*")
5. NP → T N (cf. *"the boy"*)
6. NP → A N (cf. *"good boys"*)
7. NP → prn (cf. *"they"*)
8. NP → nou (cf. *"boys"*)
9. T → art
10. A → adj[8]

An equivalent standard-form grammar can be constructed by replacing the leftmost symbol Y_1 of the right-hand expression of a rule, if it is intermediate, by the right-hand expression of each of the Y_1-rules. The process is repeated until the leftmost symbol of the right-hand expression of all the rules becomes terminal. For example, the intermediate symbol NP is the leftmost symbol in the right-hand expression of rule 1. Therefore, NP is replaced by the right-hand expression of NP-rules 5, 6, 7 and 8.

1-5. SE → NP VP PD → T N VP PD
1-6. SE → NP VP PD → A N VP PD
1-7. SE → NP VP PD → prn VP PD
1-8. SE → NP VP PD → nou VP PD

Since T and A of rules 1–5 and 1–6 are still intermediate symbols, they are replaced by the right-hand expressions of rules 9 and 10, respectively.

1-5-9. SE → NP VP PD → T N VP PD → art N VP PD
1-6-10. SE → NP VP PD → A N VP PD → adj N VP PD

The standard-form grammar equivalent to the one mentioned above contains the following 20 rules:

1′, 2′, 3′, 4′. $\text{SE} \rightarrow \begin{Bmatrix}\text{art N}\\ \text{adj N}\\ \text{prn}\\ \text{nou}\end{Bmatrix} \text{VP PD}$

5′, 6′, 7′, 8′. $\text{SE} \rightarrow \begin{Bmatrix}\text{art N}\\ \text{adj N}\\ \text{prn}\\ \text{nou}\end{Bmatrix} \text{PA SE}$

9′, 10′, 11′, 12′. $\text{SG} \rightarrow \begin{Bmatrix}\text{art N}\\ \text{adj N}\\ \text{prn}\\ \text{nou}\end{Bmatrix} \text{VP}$

13′, 14′, 15′, 16′. $\text{SG}' \rightarrow \begin{Bmatrix}\text{art N}\\ \text{adj N}\\ \text{prn}\\ \text{nou}\end{Bmatrix} \text{VP}'$

17′, 18′. $\text{NP} \rightarrow \begin{Bmatrix}\text{art}\\ \text{adj}\end{Bmatrix} \text{N}$

19′. NP → prn

20′. NP → nou

If a rule of a context-free grammar has a terminal symbol c in the right-hand expression, but not as its leftmost symbol, c is replaced by a new intermediate symbol C, and a new rule $C \rightarrow c$ is added to the standard-form grammar.

The algorithm described above cannot be successfully applied when the original context-free grammar contains a rule or a set of rules which allows unbounded left-branchings. For example, if the grammar with rules 1 through 10 also contained

11. NP → NP AND NP
12. AND → and

an infinite number of standard-form rules would be generated by the conversion algorithm:

NP → art N
NP → art N AND NP
NP → art N AND NP ⋯ AND NP

Greibach [3, 5] has shown that there is a method for converting unbounded left-branching structures to unbounded right-branching structures, preserving the same degree of ambiguity. In the example under discussion, the loop AND NP will be given a new name, say ANN. Rules 5, 6 and 11 of the original context-free grammar will be replaced by the following set of standard-form rules:

$$\text{NP} \rightarrow \begin{Bmatrix}\text{art}\\ \text{adj}\end{Bmatrix} \text{N}$$

$$\text{NP} \rightarrow \begin{Bmatrix}\text{art}\\ \text{adj}\end{Bmatrix} \text{N ANN}$$

ANN → and NP
ANN → and NP ANN

In an IC grammar, a set of NP-rules appears only once, while in a predictive grammar, it appears as many times as the number of constitutes whose first constituent is an NP (SE, SE, SG, SG′, and NP in the example under discussion). Therefore, with regard to the size of grammars, a predictive grammar is usually less simple and less economical than an equivalent IC grammar.

(5) A standard-form grammar does not impose any theoretical limitation on the number of intermediate

[7] Terminal symbols are represented by small letters and intermediate symbols by capital letters.

[8] SE: sentence, NP: noun phrase, VP: predicate, VP′: predicate with a missing object, PD: period, PA: participial phrase, SG: declarative clause, SG′: declarative clause with a missing object, T: article, A: adjective, N: noun.

symbols in the right-hand expressions of its rules:

$$Z \rightarrow cY_1 \cdots Y_m \qquad (m \geqq 1)$$
$$Z \rightarrow c$$

This feature provides the predictive analyzer with some advantage over an IC grammar. It is not always easy to write a grammar for a natural language with only binary branching rules. For example, the context-free rewrite rule:

SE → MORE NP′ VP THAN-CLAUSE

for sentences such as "More people came than expected." will have to be replaced by a set of binary rules such as:

$\text{SE} \rightarrow \text{NP}_{\text{more}}\ \text{VP}_{\text{than}}$
$\text{NP}_{\text{more}} \rightarrow$ MORE NP′
$\text{VP}_{\text{than}} \rightarrow$ VP THAN-CLAUSE

while it can be represented by a single rule in a predictive grammar:

(SE, more) | NP′ VP THAN-CLAUSE

In general, both a standard-form grammar and an IC grammar require more rules than an arbitrary context-free grammar to which they are equivalent. However, it is important to notice the difference in the nature of the expansion in size of two types of grammars. In the case of a standard-form grammar, as has been explained in (4), the expansion is due to replacing Y_1 of $Z \rightarrow Y_1 \cdots Y_m$ by Y_1-rules. Where the original context-free grammar contained p Y_1-rules and q rules whose leftmost symbol of the right-hand expression is Y_1, an equivalent standard-form grammar contains at least $p \times q$ rules. In the case of an IC analyzer, as mentioned in (5), if the original context-free grammar contained a rule with more than two symbols in the right-hand expression, only this rule is replaced by a set of new binary rules, and the expansion does not extend over other rules in the grammar. While a standard-form grammar has redundancy whenever $q > 1$, an IC grammar does not have any redundant information in it. For grammars of natural languages, the expansion of the size of grammar due to (4) seems to be a more serious problem than that due to (5).

It should also be noticed that the structural descriptions assigned to a given sentence by an arbitrary context-free grammar are not usually equivalent to those assigned to the same sentence by either an equivalent standard-form grammar or an equivalent IC grammar. Therefore, when a predictive analyzer or an IC analyzer is used not only as an acceptance device for sentences of a context-free language, but as a device for assigning structural descriptions to accepted sentences, we need a mechanism for mapping the distorted structural descriptions of either analyzer into the correct structural descriptions which would have been assigned by the original context-free grammar.

(6) Partly to make up for the drawback of repetitive path processing, efficiency techniques such as (a) a simple shaper test (due to W. Bossert and D. Isenberg) and (b) a generalized shaper test (due to W. Plath) have been developed for the predictive analyzer.[9] In these two techniques, a rule applicable at a given word position on a path is rejected for this particular path if its new set of intermediate symbols produces a PDS configuration which can immediately be identified as leading to a nonempty PDS after the remaining portion of the sentence has been processed. One can associate with each intermediate symbol the minimum number of terminal symbols required to fulfill it entirely, that is, required to eliminate entirely from the PDS the intermediate symbol itself and its new contribution of intermediate symbols to the store. In the course of analysis of a sentence, each time a new PDS configuration is formed, the simple shaper test compares the number of remaining words with the sum of such minimum numbers over all the intermediate symbols in the PDS. If the former is smaller than the latter, the rule that has produced the current PDS configuration is rejected immediately as abortive.

The generalized shaper test is basically an extension of the simple shaper test concept to include a number of specific tests on particular intermediate symbols and terminal symbols. In order for some intermediate symbols to be entirely fulfilled, certain special terminal symbols are required. For example, the intermediate symbol COMMA requires the presence of the terminal symbol "comma" in the sentence. If the PDS contains two COMMA's, and there is only one "comma" left to be processed in the sentence, the current PDS will never be emptied. The rule which produced the current PDS configuration is immediately rejected, and the analysis flow goes back to step 7 (Table 2). In the current English analyzer, generalized shaper tests are performed on the intermediate symbols COMMA ("comma"), AND ("and/or"), PA ("participle"), etc., resulting in a considerable reduction of the processing time (see Section 4 for the effect of the generalized shaper tests).

Assume that a rule has been rejected at word k due to the simple shaper or generalized shaper tests. This does not preclude, however, the use of the same rule at the same word position the next time the same intermediate symbol has become the topmost symbol in the PDS. This time, the PDS to which the new intermediate symbols of the rule are added may be such that the resulting PDS configuration may be capable of leading to an empty PDS. Therefore, the simple shaper test and the generalized shaper test apply to each use of the rule at a given word position. In contrast, consider an IC analyzer. Since a given rule at a given word position for a given length of substring in the sentence is used only once, such tests obviously cannot be and need not be invoked.

3. A Technique for Repetitive Path Elimination

As was pointed out in the previous section, the major drawback of a predictive analyzer as compared with an IC

[9] Efficiency techniques described here, and a technique for repetitive path elimination described in Section 3 of this paper, can be applied to other context-free recognition procedures (for example, Irons' error-correcting parse algorithm [10]) based on the use of a pushdown store.

analyzer is the repetitive processing of the same substrings in the course of the analysis of a given sentence, and in connection with this problem, its incapability of identifying well-formed substrings. The new technique for repetitive path elimination which is subsequently described is a method for remedying this drawback of a predictive analyzer without introducing the problem of core storage overflow. As previously mentioned, the predictive analyzer is presently free of the problem of overflow, while the IC analyzer has to cope with it either by using an extended storage device such as magnetic tape or by abandoning the analysis. It will be shown that, according to this technique, an applicable rule at a given word position is blocked from use after the first unsuccessful attempt to employ it. Furthermore, well-formed substrings (wfs's) of various lengths corresponding to an intermediate symbol topmost in the PDS at a given word position, once identified as such, are treated as if each of them consisted of a single word. As will be shown below, the consequence of this technique is that a rule abortive at a given word position is processed no more than once, and a nonabortive rule twice, at most. The basic flow of logic for the analysis of a sentence is the same as that described in Table 2, and the new technique is superimposed on the basic flow without concern for core storage overflow.

The repetitive path eliminator uses a 72-bit *word-position register* associated with each rule in the grammar (see Table 5). Word-position registers are initially set to 1's for all rules in the grammar. In the course of analysis, if a given rule used at word position k for the first time turns out to be abortive, that is, to have led to no entire fulfillment of all of its new intermediate symbols, a 0-bit is stored in the kth bit position $R_k(sr(B(k)))$ of the word position register of the rule $sr(B(k))$. This 0-bit in the kth bit position prevents the use of the rule at k in any subsequent paths of analysis for this particular sentence.

Table $W(j, i)$, called "wfs length table", with 72 sets of approximately 80 72-bit registers, is used to record the lengths of identified wfs's corresponding to all the intermediate symbols at each word position. 72 is the maximum number of words in a sentence, while 80 is the approximate number of intermediate symbols currently recognized in the grammar. The ith set of registers in the wfs length table corresponds to the ith word ($i = 1, 2, \cdots, 72$) in the sentence; the jth register ($j = 1, 2, \cdots, 80$) in a given set corresponds to the jth member of an alphabetically ordered set of approximately 80 intermediate symbols. Since the first member of the intermediate symbol set is represented in the computer by a binary "0000001", the second member by "0000010", the third member by "0000011", and so on, the intermediate symbol itself is represented by its order in the intermediate symbol set. $W_m(j, i)$, the mth bit position ($m = 1, 2, \cdots, 71$) of the jth register of the ith set, is set equal to 1 if the jth symbol, which is topmost in the PDS at word position i, has been identified as entirely fulfillable by a wfs of length m. $W_{72}(j, i)$, originally set to zero, is set equal to 1 when the jth symbol becomes topmost in the PDS at word i for the first time. Table 6 is an illustrative layout of the ith set of registers of the wfs length table. The 1001011_2th symbol (TX) has been entirely fulfilled by a wfs of length 3 and 6; hence 1 appears in $W_3(1001011_2, i)$ and $W_6(1001011_2, i)$. Similarly, $W_4(1001101_2, i) = 1$ since the 1001101_2th symbol (VX) has been entirely fulfilled by a wfs of length 4. $W_m(0000010_2, i) = 0$ for all the values of m and $W_{72}(0000010_2, i) = 1$ indicates that although the 0000010_2th symbol 33 has become topmost at word i, no wfs of any length has been found thus far at this word position. $W_{72}(0000001_2, i) = 0$ with $W_m(0000001_2, i) = 0$ for all the values of m shows that the 0000001_2th symbol (1X) has not become topmost in the PDS at word i yet. Given a topmost symbol $j = \alpha(i)$ at a given word position i, the register at location $(c + (80 \times 2)i + j)$ directly indicates how many wfs's of what length have been found for the symbol j, where (80×2) is the size of a set of registers for a word position and c is a constant pertaining to the address of the first location of the wfs length table.

The question, then, is how to recognize whether or not a rule has been entirely fulfilled in the course of analysis, and if so, how to identify the length of the wfs which has entirely fulfilled the rule.

It was mentioned in Section 1 that, in the predictive analyzer, at word position k in the course of analysis, the following information is available in core to identify the current path leading to k:

- $A(i)$: the address of the first and the last rules of $S(i)$, for $i = 1, \cdots, n$
- $B(i)$: the address of the rule used at i, for $i = 1, \cdots, k$
- $\alpha(i)$: the fulfilled intermediate symbol at i, for $i = 1, \cdots, k$
- $\beta(i)$: the position, counting from the bottom, of $\alpha(i)$ in the PDS, for $i = 1, \cdots, k$

Three additional registers $\gamma(i)$, $\delta(i)$ and $F(i)$ will be added to the cycling mechanism for use by the repetitive path eliminator:

- $\gamma(i)$: the word position at which $\alpha(i)$ was introduced to the PDS
- $\delta(i)$: the position, counting from the bottom, of $\alpha(i)$ in the rule which introduced it in the PDS
- $F(i)$: is reset to 0 when processing of $sr(B(i))$ is initiated at i and is set to 1 if a path has been found in which all the new intermediate symbols introduced by $sr(B(i))$ have been entirely fulfilled at $k(k \geqq i)$

For example, if rules used at word positions 1, 2, 3 and 4 are:

1. $(X, a) \mid B\ C\ D$
2. $(B, b) \mid E\ F$
3. $(E, e) \mid \Lambda$
4. $(F, f) \mid \Lambda$

the registers of the cycling mechanism will contain the information shown in Table 7 (it is assumed for simplicity of illustration that the input string consists of unambiguous terminal symbols $abef \cdots$, and that there is not more than one rule in the grammar with the same argument pair).

The entire fulfillment of the rule can be detected in the following way. When a given intermediate symbol becomes topmost in the PDS for the first time at k, all the rules belonging to $S(k)$ which have the same intermediate

symbol in their argument pairs are checked one by one to see if the following condition is satisfied.

Condition 1. Is $sr(B(k))$ of the form $(Z, c) \mid \Lambda$?

If not, the normal routine (step 2, Table 2) is followed. If it is, the rule has been entirely fulfilled. Before going to step 2, $F(k)$ is set to 1, and the length of the identified wfs is recorded in the wfs length table.

In Table 7, $F(3) = 1$ and $F(4) = 1$ because the rules used at word positions 3 and 4 satisfied Condition 1. In order to identify the length of the wfs, the word position of the left-end of the wfs is stored in a register called L (for "left-end"). Since the right-end of the wfs is k, the length of the wfs is equal to $k - L + 1$. Under Condition 1, the Λ symbol was introduced to the PDS at k and was entirely fulfilled at k, so to speak. Therefore, k is stored in L, and the length of wfs $= 1$ is obtained. A 1-bit is stored in $W_{k-L+1}(\alpha(L), L)$, namely, in the first (i.e., $(k - L + 1)$-st) bit position of the $\alpha(L)$-th register at the Lth-$(L = k)$ word position. Next, a test is made to see if Condition 2 is satisfied.

Condition 2. $\delta(L) = 1$?

A fulfilled symbol at $L(= k$ initially) with $\delta(L) = 1$ (see $\delta(4)$ of Table 7) signifies that the bottom intermediate symbol of the rule used at word $\gamma(L)$ (in the example, Table 7, $\gamma(4) = 2$) has been entirely fulfilled. Therefore, the rule at word $\gamma(L)$ is entirely fulfilled. The program flags $F(\gamma(L)) = 1$ (see $F(2) = 1$ of the table) in order to indicate that the rule $sr(B(\gamma(L)))$ (i.e., rule 2 of the example above) can lead to at least one partial analysis in which all of its new intermediate symbols are fulfilled entirely. L is set to $\gamma(L)$, and a 1-bit is stored in $W_{k-L+1}(\alpha(L), L)$. Condition 2 is tested again to see if the fulfilled intermediate symbol at the new value of L is the bottom symbol of some previously used rule. This process continues until either δ of the last item flagged $\neq 1$ or $L = 0$.

When a rule at word position i has been exhaustively processed and is ready to be replaced by the next rule in $S(i)$ (see Step 6 of Table 2), the content of $F(i)$ is tested. If $F(i) = 0$, the rule has never had its intermediate symbols entirely fulfilled. Therefore, a 0-bit is stored in $R_i(sr(B(i)))$, namely in the ith bit position of the word-position register of $sr(B(i))$.

In the course of analysis of a sentence, a new PDS configuration is formed at word position $k-1$ $(k = 1, 2, \cdots, n)$ as dictated by step 2 of Table 2, and the analysis flow moves to k following step 3. If k is not greater than n, a test is made to see if the topmost symbol $\alpha(k)$ has been processed previously as the topmost symbol at the same word position ($W_{72}(\alpha(k), k) = 1$?). If not, all the members of $S(k)$ have to be processed with $\alpha(k)$; therefore, step 1 is taken. On the other hand, if $\alpha(k)$ has been processed previously at k, $W(\alpha(k), k)$ is checked to see if there is any wfs corresponding to the symbol $\alpha(k)$. If there is none, the current path is abandoned, and the analysis flow goes back to the previous branching point. If there are some wfs's corresponding to $\alpha(k)$, they are processed one at a time, with $\alpha(k)$ removed from the PDS and the analysis skipped by the length m of the wfs. Such a path, when it leads to a successful analysis of the entire sentence, then contains an unanalyzed wfs of length m. In some cases, the analysis of this wfs may be found in the corresponding word positions of some previously obtained analysis of the whole sentence. In such cases, a cross reference is made to that portion (identified by word position and length) of the previous analysis (identified by analysis number) with the consequence that the rules pertaining to $\alpha(k)$ of length m are processed no more than once.

In many cases, $\alpha(k)$ of length m is not found in any of the previously obtained analyses of the full sentence.[10] Then, the analysis of the substring of that length is performed with $\alpha(k)$ as the initial designated symbol over all the members of $S(j)$ for $j = k, \cdots, k+m-1$. It is at this time that word position registers R_j for the rules concerned are used in order to prohibit use of abortive rules. When a rule compatible with the current topmost symbol is found at the jth word position (step 1 of Table 2), a test is made to determine if $R_j(sr(B(j))) = 0$. If so, the rule is abortive and therefore is immediately abandoned. Since all the rules originally applicable to the current substring have already been identified as either abortive or nonabortive, no flagging of $F(j)$ is needed.

[10] This happens when the wfs of length m corresponding to $\alpha(k)$ was recognized on a path that aborted before the end of the sentence was reached.

TABLE 5. Rule and Word-position Register

Rule: Argument Pair	Rule: New Intermediate Symbols	Word-Position Register (R)							
		1	2	3	4		k		72
(Z, c)	$Y_1 \cdots Y_m$	0	0	1	0	$\cdots$	1	$\cdots$	0

TABLE 6. The ith Set of Registers of wfs Length Table

jth symbol		mth bit: 1	2	3	4	5	6	7	$\cdots$	71	72
0000001	(1X)	0	0	0	0	0	0	0		0	0
0000010	(33)	0	0	0	0	0	0	0		0	1
0000011	(4X)	0	0	0	0	0	0	0		0	0
1001011	(TX)	0	0	1	0	0	1	0		0	1
1001100	(UX)	0	0	0	0	0	0	0		0	1
1001101	(VX)	0	0	0	1	0	0	0		0	1
1001110	(WX)	0	0	1	0	0	0	0		0	1
1001111	(XC)	0	0	0	0	0	0	0		0	1

TABLE 7. The Registers of Analysis Path Record for Repetitive Path Elimination

i	1	2	3	4	$\cdots$
$A(i)$	[1, 1]	[2, 2]	[3, 3]	[4, 4]	
$B(i)$	1	2	3	4	
$\alpha(i)$	X	B	E	F	
$\beta(i)$	1	3	4	3	
$\gamma(i)$	0	1	2	2	
$\delta(i)$	1	3	2	1	
$F(i)$	0	1	1	1	

When the first analysis, for example, of "TIME FLIES LIKE AN ARROW AND THEY ARE FLYING PLANES." is obtained, all the wfs's contained in the analysis are stored in a table called *wfs reference table.* While the *wfs length table* $W_m(j, i)$ is used to store all the wfs's thus far identified, the wfs reference table is used to store only those wfs's which have appeared in some analysis of the whole sentence. Let $Q(i)$ represent the ith column of the wfs reference table. "NQ-2-1" at $Q(4)$ of Table 8,[11] for example, indicates that NQ is a wfs of length 2 ("AN ARROW") in Analysis Number 1. Similarly, "SE-5-1" at $Q(7)$ indicates that "THEY ARE FLYING PLANES." is identified as a wfs of length 5 in Analysis Number 1. "PD-9-1" is contained in $Q(3)$ because PD was the topmost and the only intermediate symbol in the PDS at word 3 (the sentence would have been terminated with a period at this position), and was eliminated from the PDS when the eleventh word, the period, was processed. The analysis shown in Table 8 corresponds to the interpretation of the sentence in the sense that time passes as quickly as an arrow and they are planes which are flying.

When the second analysis of the sentence is obtained in which "THEY ARE FLYING PLANES" has a structure similar to that of "the facts are smoking kills", it is indicated that the analysis of the period at word 11 has been skipped because it was identified as having a wfs of length 1 (see $Q(11)$ of Table 8). The analysis of "ARE FLYING PLANES.", which is the part of the sentence that has been reinterpreted, is written on the output tape with a cross reference to PD-1-1 at word 11. The length of the newly identified wfs's is computed: VX: "ARE FLYING PLANES", SG: "FLYING PLANES", and VX: "PLANES". VX-3-2, SG-2-2, and VX-1-2 are stored in $Q(8)$, $Q(9)$ and $Q(10)$ with the rightmost "2" representing the analysis number.

After the third analysis corresponding to "These people are flying (the) planes" is obtained, the program eventually "backs up" to the initial word position and processes "TIME FLIES LIKE AN ARROW" as "a species of flies called time flies are fond of an arrow". When PD becomes the topmost symbol at word position 6, $W_6(\text{PD}, 6) = 1$ indicates that a wfs of length 6 can fulfill PD entirely. Therefore, the end of the sentence is reached with an empty PDS. PD-6 is found in $Q(6)$ (see Table 8), therefore a cross reference to Analysis Number 1 is made in the analysis output record of the whole sentence. The registers of the analysis record indicate that VX at word position 3 was skipped because of a wfs of length 3: "LIKE AN ARROW". $Q(3)$ is scanned for a VX-3, but no match is found. The 3-word wfs is reprocessed with the initial symbol VX. When an analysis of the 3-word wfs is obtained, the complete analysis of "TIME FLIES LIKE AN ARROW" is given as Analysis Number 4 with a cross reference to word position 6 of Analysis Number 1 for the wfs of length 6 ("AND THEY ARE FLYING PLANES."). The wfs reference table now contains the information shown in Table 9.

The program continues its task of determining if "LIKE AN ARROW" corresponding to VX is ambiguous. When all the possible analysis branches pertaining to VX of length 3 at word 3 are exhaustively examined, the cycle is transferred back to the main analysis program, which "backs up" to word position 6, and checks to see if PD at 6 has any other wfs of different length ($W_m(\text{PD}, 6) = 1$ for any m greater than 6?). Eventually, the program interprets "TIME FLIES LIKE AN ARROW AND THEY" as a compound subject as if it meant "a species of flies called time flies which are similar to an arrow, and they". Cross references are made to "VX-3-1", "VX-3-2", "VX-3-3" at word 8, and to "PD-1-1" at word 11. Finally, the program interprets "TIME FLIES" as an imperative sentence, meaning "record the time of flies", with the cross reference given to PD-9-1 at word position 3.

The repetitive path eliminator, as it is presented here, excludes the use of the number agreement test between the subject and the predicate of a sentence, and also the use of such subsidiary tests for efficiency as "simple shaper test" and "generalized shaper test". If a given rule were rejected due to any one of these tests, the rule should not be blocked from use because the same rule at the same word position while following an alternate path might enter a different PDS configuration which has a different grammatical number specification, a smaller number of intermediate symbols, or different sets of special intermediate symbols. Similarly, if one of the paths originating from a given rule were to be terminated due to any one of these tests, which is probably the case with most rules used in the course of analysis of a sentence, the rule could not be regarded as abortive because the path which was terminated might have led to the entire fulfillment of the rule if the PDS configuration had been different.

TABLE 8. THE STATUS OF $Q(i)$ AFTER THE FIRST ANALYSIS

i	1 TIME	2 FLIES	3 LIKE	4 AN	5 ARROW	6 AND	7 THEY	8 ARE	9 FLYING	10 PLANES	11 .
$Q(i)$	SE-11-1	VX-1-1	PD-9-1	NQ-2-1	N5-1-1	PD-6-1	SE-5-1	VX-3-1	N3-2-1	N6-1-1	PD-1-1

TABLE 9. THE STATUS OF $Q(i)$ AFTER THE FOURTH ANALYSIS

i	1 TIME	2 FLIES	3 LIKE	4 AN	5 ARROW	6 AND	7 THEY	8 ARE	9 FLYING	10 PLANES	11 .
$Q(i)$	SE-11-1	VX-1-1 7X-1-4	PD-9-1 VX-3-4	NQ-2-1 N2-2-4	N5-1-1	PD-6-1	SE-5-1	VX-3-1 VX-3-2 VX-3-3	N3-2-1 SG-2-2 PA-2-3	N6-1-1 VX-1-2 N2-1-3	PD-1-1

[11] SE: sentence, VX: predicate, PD: period, NQ: noun object, N5: modified object, N3: noun complement, N6: modified complement, 7X: subject master, N2: object, SG: declarative clause, PA: participle.

Assume, for example, that the input sentence is "A boy with a telescope and a dog are coming.", and that a path corresponding to the interpretation of the noun phrase as "A boy (with a telescope and a dog)" is traversed first. When the ninth word "are" is ready to be processed, the PDS contains a singular VX as the topmost intermediate symbol. If the number agreement test were to be performed in the course of analysis, all the rules pertaining to VX and "are" would fail the test, and therefore, no wfs's corresponding to VX at word position 9 would be found. W_m(VX, 9) would remain zero for $m = 1, \cdots, 71$, although W_{72}(VX, 9) would be set to 1. Eventually, the analysis flow would back up and process the initial noun phrase as "a boy (with a telescope) and a dog". In this path, when "are" is ready to be processed at word position 9, the PDS would contain a plural VX as the topmost intermediate symbol. But W_{72}(VX, 9) = 1 and W_m(VX, 9) = 0 for $m = 1, \cdots, 71$ would indicate that there was no wfs found corresponding to VX at this word position. The current path would be terminated, and no analysis of the whole sentence would be obtained. Of course, it is possible to set $W_{72}(j, i) = 0$ in case a path originating from the jth intermediate symbol at word i is terminated due to a failure in the number agreement test or the subsidiary tests for efficiency, but this would have the effect of almost completely nullifying the performance of the repetitive path eliminator, because chances of one or more paths which originate from a given intermediate symbol at a given word position not being terminated by any of these tests are extremely slim. Therefore, all of these tests are inactivated in the main analysis routine. However, when a skipped wfs is reprocessed, since all the rules originally applicable to this wfs have already been identified as either abortive or nonabortive, the tests can be applied without any effect on the word position registers.

TABLE 10. Comparison of Processing Time of the Three Versions of the Predictive Analyzer

Sentence Number	*Number of Words*	*Number of Analyses*	*1963-FJCC (min.)*	*NEW SHARE (min).*	*wfs Syntactic Analyzer (min.)*
1	17	1	0.0	0.0	0.0
2	18	1	0.1	0.1	0.1
3	25	5	1.1	0.8	0.1
4	35	12	9.0	2.2	0.1
5	16	40	0.1	0.1	0.2
6	17	4	0.1	0.1	0.0
7	14	4	0.0	0.0	0.1
8	16	3	0.1	0.0	0.0
9	23	7	0.2	0.0	0.1
10	23	31	1.5	0.4	0.1
11	30	118	7.6	3.2	0.3
12	25	72	0.5	0.5	0.2
13	32	18	9.8	3.9	0.1
14	38	94	42.4	13.2	0.2
15	25	136	7.2	0.7	0.3
16	27	1	1.7	0.3	0.1
17	30	17	1.2	0.3	0.2
18	20	71	2.7	1.8	0.2
19	29	2	1.5	0.0	0.1
20	20	16	0.7	0.1	0.1
			Total 87.5	27.7	2.6

One of the advantages of the repetitive path eliminator is that, although it requires a considerable amount of core storage, the amount is still finite and of a practicable order of magnitude on an IBM 7094 computer. It will not cause core storage overflow once the wfs length table has been placed in core. The wfs reference table, although it may theoretically require an unbounded number of registers, can in practice be fairly small because the table is used to store only those wfs's which have appeared in previously obtained analyses of the whole sentence. In the current experimental program, $(77 - i)$ entries are reserved for $Q(i)$, where $i = 1, 2, \cdots, 72$. Even if the wfs reference table overflows, there is no catastrophic effect on the analysis procedure: a wfs that is to have been stored in $Q(k)$ is simply ignored. Therefore, if the same wfs appears at the same word position in a subsequent analysis of the sentence, it is not found in $Q(k)$ and therefore is reprocessed.

4. Comparison of Processing Time

An experimental program of the predictive analyzer with the repetitive path elimination is presently in operation and has well proved the efficiency of the new technique.[12] A routine for the number agreement test between the subject and the predicate of a sentence has not yet been incorporated; therefore, the program occasionally produces superfluous analyses which could easily be eliminated by the incorporation of such a test. Since the agreement test is to be performed only on the analyses of the whole sentence and not on all the analysis paths, the time needed for such a test is expected to be negligible. The program also does not have "simple shaper" and "generalized shaper" tests in its routine for reprocessing skipped wfs's. Such tests, when incorporated, would reduce the processing time to some degree. The performance of the "exhaustive shaper test," due to R. Abbott [11] and S. Cohen [12] and programmed by M. McAnulty, which blocks from use certain rules prior to the initiation of the predictive analysis of each input sentence, is still to be tested, in conjunction with the repetitive path eliminator. The output format for the experimental program of the repetitive path eliminator is still tentative, devised mainly for the purpose of debugging. However, it is expected that the "cleaned-up" version of the program will work as efficiently as, if not more efficiently than, the current experimental version.

[12] The author is greatly indebted to Carlton DeTar, who organized the whole program on the basis of the author's rather sketchy description [13], to Bernard Higonnet, who continued DeTar's work and contributed to the modifications of the system. The author is also indebted to Maynard Maxwell, who programmed a routine for identifying and preventing the use of abortive rules at each word position and worked on an experimental output editing routine. All of them not only collaborated together in programming and debugging, but also contributed to developing the system down to the minute details and bringing it to the current working version.

The execution time of sentences processed by three versions of the Harvard English predictive analyzer is shown in Table 10. The column with the heading "1963–FJCC" shows the speed of the analyzer which was operational in 1963, described by Kuno and Oettinger [2] at the 1963 Fall Joint Computer Conference. This version of the analyzer is currently available through the SHARE Distribution Agency. The column with the heading "New SHARE" shows the speed of the new version of the analyzer which is being packaged now for submittal to SHARE. This new version is different from the previous version in that it contains the generalized shaper test and works approximately three times as fast as the previous one. The rightmost column with the heading "wfs Syntactic Analyzer" shows the speed of the experimental program of the analyzer with the repetitive path eliminator. Sentence 14, for example, which previously took 42.4 minutes and 13.2 minutes using the old and new SHARE versions, respectively, now takes 0.2 minutes (more exactly, 5.9 seconds for internal processing and 10 seconds for input/output). The 20 sentences, which used to take 87.5 minutes and 27.7 minutes, respectively, using the two SHARE versions, now take 2.6 minutes, approximately two-thirds of which time is for input and output.

20 Test Sentences

1. The increase in flow stress was attributed to vacancies, which have appreciable mobility at $-72°$.
2. Single strain reversals at $-72°$ not only produced the n effect but also increased the flow stress.
3. Slime formation is dependent on size of particles formed by mechanical means, amount of metal in the amalgam, and purity of solutions.
4. A shear stress applied during the recovery had no effect on the amount of recovery, if the stress was less than the instantaneous yield point, irrespective of the direction of the stress.
5. People who apply for marriage licenses wearing shorts or pedal pushers will be denied licenses.
6. Nearly all authorities agree that this will be the first practical, large-scale use of space.
7. Economic studies show that it could be a billion-dollar-a-year business by the 1970's.
8. A clutch of major companies has been pressing to get such a system into being.
9. The U.S. has reached a momentous point of decision in a project that only a few years ago would have seemed improbable.
10. Technologically speaking, there are three basic contending schemes, with a number of variations, for orbiting a communication satellite system.
11. Each of the parties to this treaty undertakes to prohibit, to prevent, and not carry out any nuclear weapon test explosion or any other nuclear explosion at any place under its jurisdiction or control.
12. This applies in the atmosphere, beyond its limits, including outer space, or under water, including territorial waters or high seas.
13. This applies in any other environment if such explosion causes radioactive debris to be present outside the territorial limits of the state under whose jurisdiction or control such explosion is conducted.
14. It is understood in this connection that the provisions of this subparagraph are without prejudice to the conclusion of a treaty resulting in the permanent banning of all nuclear test explosions, including such explosions underground.
15. Gravely concerned with spreading racial violence, President Kennedy used his press conference to issue counsel to both sides in the struggle.
16. If a farmer produces 14 bushels of oats on one acre of land, how many acreas will he need to produce 700 bushels of oats?
17. A man who takes three yellow and two white capsules daily finds that a bottle of white capsules contains twice as many capsules as a bottle of yellow ones.
18. It is a mistake to think that the indeterminacy of boundaries makes our central notions less clear and valid.
19. Causes of slime formation and its reduction in the process of cementation of X by Y amalgam from Z solution containing waste from an N factory were investigated.
20. It will serve to illustrate what has been said if we apply it to the question of free will.

A corpus of 13 sentences, not listed here, which range from 40 to 56 words in length, takes 3.8 minutes for internal processing using the wfs syntactic analyzer. These are the sentences which we could not even try to analyze using either of the two SHARE versions for fear of excessive processing time.

We have operational, for the first time, a syntactic analysis system fast enough to enable us to process a large number of sentences. On the basis of studies performed on the analysis output, we can properly evaluate the performance of the current English grammar and find clues for the refinement of the grammar.

Since sentences of ordinary length can now be processed in real time, the designing of a system for information retrieval, question answering, or mechanical translation developed around the analyzer with man-machine interaction incorporated at proper places can be discussed on a practical basis.

REFERENCES

1. Kuno, S., and Oettinger, A. G. Multiple-path syntactic analyzer. In *Information Processing 1962*, North-Holland, Amsterdam, 1963.
2. ——, and ——. Syntactic structure and ambiguity of English. *Proc. 1963 Fall Joint Comput. Conf.*, Spartan Books, Baltimore, 1963.
3. Greibach, S. Inverses of phrase structure generators (doctoral thesis). *Mathematical Linguistics and Automatic Translation*, Rep. No. NSF-11, Harvard Comput. Lab., Cambridge, 1963.
4. ——. Formal parsing systems. *Comm. ACM 7* (Aug. 1964), 499–504.
5. ——. A new normal-form theorem for context-free phrase structure grammars. *J. ACM 12* (Jan. 1965), 42–52.
6. Floyd, R. W. The syntax of programming languages—a survey. *IEEE Trans. EC-13* (Aug. 1964), 346–353.
7. Hays, D. Automatic language-data processing. In *Computer Applications in the Behavioral Sciences*, Borko, H. (Ed.), Prentice-Hall, Englewood Cliffs, N.J., 1962, 394–421.
8. Sakai, I. Syntax in universal translation. *Proc. 1961 Int. Conf. on Machine Translation of Languages and Applied Language Analysis*, Her Majesty's Stationery Off., London, 1962, 593–608.
9. Robinson, J. Preliminary codes and rules for the automatic parsing of English. RM-3339-PR, RAND Corp., Santa Monica, Dec. 1962).
10. Irons, E. T. An error-correcting parse algorithm. *Comm. ACM 6* (Nov. 1963), 669–673.
11. Abbott, R. An exhaustive shaper test as preliminary analysis technique. *Mathematical Linguistics and Automatic Translation*, Sec. IX, Rep. No. NSF-13, Harvard Comput. Lab., Cambridge, 1964.
12. Cohen, S. Application of the exhaustive shaper test as preliminary analysis technique. *Mathematical Linguistics and Automatic Translation*, Sec. X, Rep. No. NSF-13, Harvard Comput. Lab., Cambridge, 1964.
13. Kuno, S. New techniques for repetitive path elimination. *Mathematical Linguistics and Automatic Translation*, Sec. XI, Rep. No. NSF-13, Harvard Comput. Lab., Cambridge, 1964.