IMPLEMENTATION OF THE STRING PARSER OF ENGLISH

Ralph Grishman
New York University

The organization of the latest implementation of the string parser for scientific literature is described. The steps involved in compiling and running a grammar are explained. Alternative schemes for increasing the speed of the parser are considered.

This paper describes the most recent implementation of a system developed at the Linguistic String Project of New York University for the analysis of English sentences. This system is based on a string grammar of English which is described in the previous paper of this volume [1]. The current implementation, in FORTRAN for the Control Data 6600, supercedes earlier implementations in IPL-V [2] and in FAP for the IBM 7094 [3].

The fundamental structure of the string grammar — a context-free component plus a set of restrictions — has remained unchanged during the project. In each implementation, the context-free productions and restrictions have been encoded into a list structure which is then used by an English

sentence analyzer consisting of a syntax-directed parser and an interpreter for the restrictions. In the earlier implementations the grammar had to be manually encoded, with only the substitution of numbers for symbol names being performed automatically. As the grammar grew, this encoding process became an increasingly heavy burden on the developers of the grammar. The encoded grammar was difficult to read, and any significant modification was a formidable task. Consequently, one of the basic objectives in developing the current implementation was to provide a user-oriented language for the grammar, which would be comparatively easy to write, read, and modify.

The translation of the English grammar from the user-oriented language to list structure is performed in the current system by a syntax-directed compiler. The compiler requires its own list-structured grammar, in this case the grammar of the user-oriented language. The latter grammar may be generated by compiling a grammar of the user-oriented language written, say, in BNF. This in turn requires a list-structured grammar of BNF; to avoid an infinite regress, this (very small) grammar must be manually encoded.

Thus, the system has now become a process involving three successive stages, as shown in Figure 1. This arrangement provides great flexibility, since the user can easily alter the specifications of his "user-oriented" language as his needs change. So, while this paper will describe the external format of the grammar as currently being used by the Linguistic String Project, the system may easily be adapted for use with an English grammar of very different external appearance by changing the input to the first stage of the process.

The core of the program is a very simple top-down parser for context-free grammars, which generates multiple parses of ambiguous sentences sequentially using a back-up mechanism. This parser, together with a table-driven lexical processor and a control card processor which invokes all the other

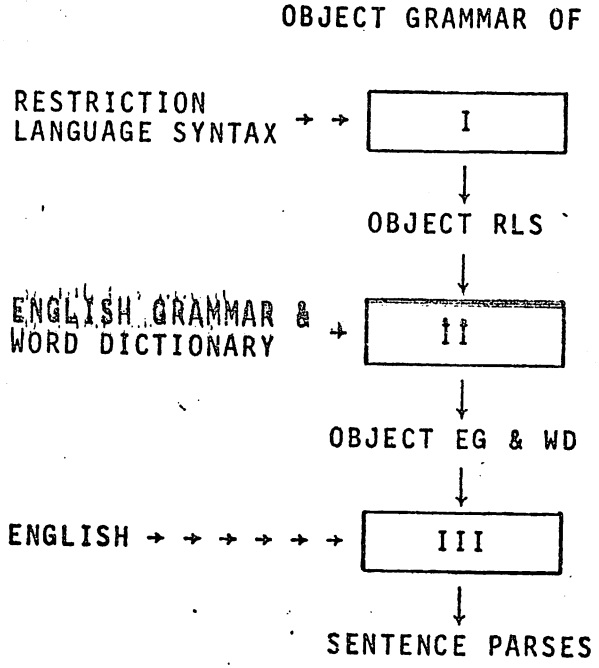

Figure 1

The Three Stages of the Parser

system components, is present in the program for each of the three stages of the system.

As indicated in Figures 2 and 3, the parser has "hooks" on it to permit various routines to be invoked during the top-down parse. In the first two stages, the parser is used as a syntax-directed compiler which translates the grammar of the grammar of English (i.e., the grammar of the user-oriented language) or the grammar of English from its input text form to list structure. Hence the routines invoked by the parser in the stage I and stage II programs are code generators which construct the

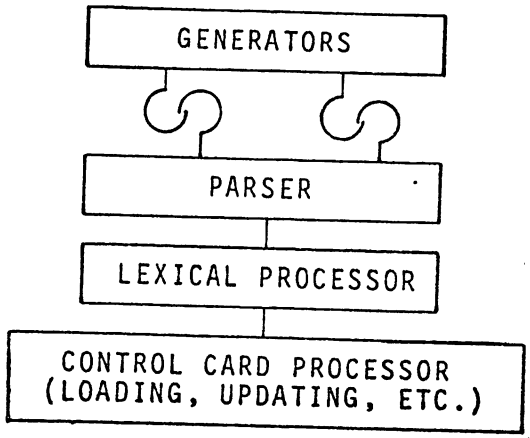

Figure 2
Organization of Stages I and II

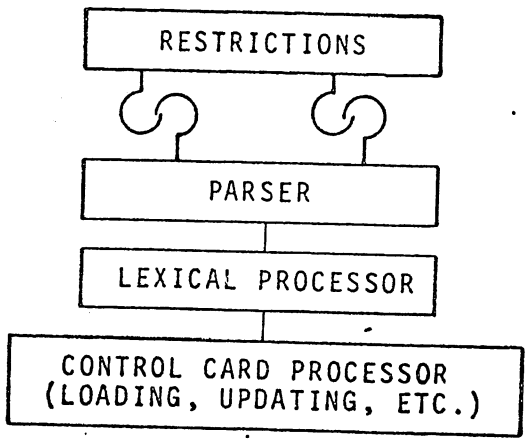

Figure 3
Organization of Stage III

requisite list structure during the top-down analysis (Figure 2).

For stage III the generators are replaced by a restriction interpreter (Figure 3). As the preceding paper described, the grammar of English consists of a context-free component plus a set of restrictions, each of which is associated with one or more productions in the context-free component. These restrictions state conditions which the parse tree must meet if the analysis is to be accepted. Each time a node is added to the parse tree, and each time a level in the tree is completed, the parser invokes the restriction interpreter to execute those restrictions appearing in the corresponding production; the restriction interpreter returns a success or failure indication to the parser. If the restriction has succeeded, the parser continues normally (i.e., as if there had been no restriction). If the restriction has failed, the parser must either try an alternate option, or, if all options in a production have been exhausted, dismantle part of the parse tree.

The system has been implemented on the Control Data 6600 at New York University. It consists of approximately 8000 source lines in about 100 subroutines, only some of which are included in the program for any one stage of the system. It is written almost entirely in FORTRAN, with only a few assembly routines for manipulating part-word fields. Each stage occupies approximately 55,000 60-bit words (corresponding roughly to 400,000 8-bit bytes).

When operating as a compiler (in stages I and II) the system has a throughput of about 1500 to 3000 cards per minute, depending on the complexity of the grammar. With an English grammar of about 20,000 source lines, including the word dictionary, a fast compiler is clearly an asset. Still, recompiling the grammar for seven or eight minutes each time a change is made is rather expensive, so an updating system was included in the program. In the output of stages I and II the source text and generated list structure are combined in a single

file called an *object grammar* (named in analogy with the object program produced by a compiler). Once an object grammar has been initially created, the user may specify modifications to it on a statement-by-statement basis. The system will compile the new statements and will insert, delete, or replace the source text and corresponding list structure in parallel.

The function of the various stages, and the format of the source input to these stages, will now be described.

Stage I parses the grammar of the grammar of English, which is a set of BNF statements describing the syntax of the four components of the English grammar: the context-free component, the lists, the restrictions, and the word dictionary. Of the four components, only the restrictions are syntactically complex; the BNF specification of the restrictions requires about 250 lines, the other components just a few lines apiece. Consequently, the entire input to this stage, including the syntax of all four components, is normally referred to as the Restriction Language Syntax, or RLS.

A small portion of the RLS is shown in Figure 4. This figure shows a few of the alternative expansions for <STATEMENT>, which is the basic construct in restrictions. A <STATEMENT> is either one of a set of "canned phrases" called <MONOSENT>s or is a restriction subject (<RSUBJ>) followed by a <PREDICATE>. The subject can take many different forms: it may be the name of a NODE (<*NODE>: terminal symbols are indicated by an asterisk on the left), the name of an ATTRIBUTE, the value of a node (<VALOF>), an element of a string (<ELEMX>), the coelement of a node (<COELX>), the CORE of a node, etc. The option <CORE> consists of the word CORE optionally followed by OF <OPCORE> (square brackets enclose optional elements). The operand of CORE (<OPCORE>) is a non-string argument (<NONSTGARG>) which may be either a non-string definition name (<*NONSTGDEF>) or the value of another node, or one of several other options. One possible form of


```
<STATEMENT> ::= <MONOSENT>/<RSUBJ><PREDICATE>

<RSUBJ>     ::= <*NODE> .ROUTX.(STARTAT) /
                <*ATTRIBUTE> .ROUTX.(ISIT) /
                <VALOF> / <ELEMX> /
                <COELX> / <CORE> / ...

<CORE>      ::= CORE [ OF <OPCORE> ]
                  .ROUTE.(CORERT)

<OPCORE>    ::= <NONSTGARG>

<NONSTGARG> ::= <*NONSTGDEF> .ROUTX.(STARTAT)
                  / <VALOF> / ...

<PREDICATE> ::= IS ((<BEPRED> / OCCURRING
                    <OCCPRED>) / NOT ...

<BEPRED>    ::= <ATOMAT> / <ATTRB> / ...

<ATTRB>     ::= <*ATTRIBUTE> .MARK
                  .GENSYM.(6) .ATTRBX
                  <ATT*> ( <ATRAOR> /
                  <*NULL> .UNMARK)
```

Figure 4

A Portion of the RLS

<PREDICATE> is IS <BEPRED> where <BEPRED> is <ATTRB>; <ATTRB> consists of the name of an ATTRIBUTE followed by zero or one or more occurrances of <ATT> (an asterisk on the right indicates optional repetition), specifying subsidiary attributes, followed optionally by <ATRAOR>, which allows for conjoined attribute predicates, with both AND and OR. An example of a restriction using these constructions will be considered shortly.

Note that the RLS has, in addition to those constructions normally appearing in BNF, certain

names, such as ROUTX and MARK, which are preceded by periods. These are the names of code generators which are to be invoked by the parser during the analysis of a restriction, after the symbols they follow have been successfully matched (the notation used here has been adapted from Cocke and Schwartz [4]). Each generator name may be followed by a single argument, which is enclosed in parentheses and preceded by a period; for example, .ROUTX (STARTAT) indicates that the generator ROUTX is to be invoked with the argument 'STARTAT'.

Stage I parses this RLS using the hand-encoded syntax of extended BNF and generates an object RLS. This object RLS is then used in stage II to parse the grammar of English (refer to Figure 1).

As noted earlier, the grammar of English has four components: a context-free component, a set of lists, the restrictions, and the word dictionary. The context-free component is a set of BNF statements; this time without any generator calls. One example, the definition of the ASSERTION string, is shown in Figure 5; the functions of the various elements in this string are described in the previous paper. The second component of the grammar is a set of lists of symbols appearing in the BNF grammar; these lists are referenced in the restrictions. One such list is shown in Figure 6.

The third component is a set of restrictions and routines. One of the simple test restrictions, WT9, which checks for number agreement between subject and verb, is shown in Figure 7. The restrictions invoke a number of routines for moving about the parse tree; these routines correspond to the basic string relations described in the previous paper. Routines are analogous to procedures in ALGOL: routines may invoke other routines, they may be invoked recursively, and they may be passed arguments.

The CORE routine, CORERT, which is invoked by restriction WT9, is shown in Figure 8. This routine checks whether the current node is an ATOM (terminal


```
<ASSERTION>  ::=  <SA> <SUBJECT> <CA2>
                  <SA> <VERB> <CA4>
                  <SA> <OBJECT> <CA6>
                  <RV> <SA>
```

Figure 5
A BNF Definition from the Grammar of English

```
TYPE ADJSET1 = SA, LT, LN, LW, LV, LQ,
     LA, LP, LCS, LPRO, RN, RV, RQ,
     RA, RA1,RD, LWR, LVSA, LCDA,
     LCDVA, LCDN, MLNR1, MLNR2, MLNR3.
```

Figure 6
A List from the Grammar of English

```
WT9 = IN ASSERTION IF THE
CORE OF THE VERB IS PLURAL,
THEN THE CORE OF THE SUBJECT
IS NOT SINGULAR.
```

Figure 7
Restriction WT9

```
ROUTINE CORERT=
  DO EITHER $1 OR $D OR $S.
$1 = TEST FOR ATOM.
$D = DESCEND TO ATOM NOT
  PASSING THROUGH ADJSET1.
$D = DESCEND TO STRING NOT
  PASSING THROUGH ADJSET1.
```

Figure 8

The CORE Routine

node), and, if not, goes down through the parse tree looking for a node which is either an ATOM or is on the STRING list. In going down the tree, the routine will not go through any node on the list ADJSET1 (which appeared in Figure 6). Note that the restrictions and routines employ two basically different styles for describing operations on the parse tree: the restrictions are declarative sentences, stating conditions the tree must meet, whereas the routines use imperative sentences, akin to most procedural languages, which explicitly state the operations to be performed.

The fourth and by far the largest component of the grammar is the word dictionary. Each word definition is a sequence of word *categories* (such as noun, tensed verb, adjective, adverb) together with a set of *attributes* (such as singular, plural, collective, concrete) organized into a tree structure associated with each category. The categories are matched by terminal symbols in the context-free component; the attributes are tested by the restrictions. A small fragment of the dictionary is shown in Figure 9. Each numbered line following a definition is an attribute list of that word. To save work in coding the dictionary, standard sets of word categories and attributes may be replaced by a reference to a *canonical form*, which appears in parentheses to the left of the word. The canonical forms are defined at the beginning of the word dictionary


```
(VTVPL) DEVOTE.
.12  =  OBJLIST: .3,NOTNSUBJ: .2.
.3   =  NPN: .15,PNN: .15, VINGSTGPN: .15,
          NPVINGO: .15.
.2   =  NTIME1, NSENT1, NSENT2, NONHUMAN.
.15  =  PVAL: (↓TO↓).

(TVVEN) DEVOTED    ↑.
.14  =  OBJLIST: .3, NOTNSUBJ: .2, POBJLIST: .4.
.4   =  PN: .15, PVINGSTG: .15, PVINGO: .15.

(TVSI)  DEVOTES    ↑.

(ING)   DEVOTING   ↑.

(VTVPL) DEVOUR.
.12  =  NOTNOBJ: .1, NOTNSUBJ: .2, OBJLIST: .3.
.1   =  NTIME1.
.2   =  NTIME1, NSENT1, NSENT2.
.3   =  NSTGO.

(TVVEN) DEVOURED   ↑.
.14  =  NOTNOBJ: .1, NOTNSUBJ: .2, OBJLIST: .3,
          POBJLIST: .5.
.5   =  NULLOBJ.

(ING)   DEVOURING  ↑.

(TVSI)  DEVOURS    ↑.

(NSIX)  DEXTRAN.
.11  =  COLLECTIVE, NONHUMAN, NCHEM.

(NPLX)  DEXTRANS.
.11  =  NONHUMAN, NCHEM.
```

Figure 9

A Fragment of the Word Dictionary

(Figure 10). As a further saving, words with several attribute lists in common may be grouped together; this is normally done for words with the same stem. All words in the group except the first are marked with an up-arrow after the word; only those attribute lists of the following words which differ from those of the first word need then be given explicitly.

```
(NSIX)= N: .11,SINGULAR.
(NPLX)= N: .11, PLURAL.
(NSI) = N: (SINGULAR).
(NPL) = N: (PLURAL).
(ADJE)= ADJ.
(ADJX)= ADJ: .10.
(INGSI) = VING: .12. SINGULAR.
(CHSI)  = N: (SINGULAR, COLLECTIVE, NONHUMAN,
             NCHEM, CONCRETE).
(CHPL)  = N: (PLURAL, NONHUMAN, NCHEM, CONCRETE).
(NAMEX) = N: (NAME, NHUMAN, SINGULAR).
```

Figure 10

Some of the Canonical Form Definitions

As each statement in the English grammar is parsed during stage II, the generators specified in the RLS are invoked, generating the list structures of the object grammar of English. As an example of this process, consider the restriction WT9 (Figure 7), for which most of the pertinent RLS is given in Figure 4. The list structure generated for this restriction is shown in Figure 11. The IF...THEN... construction in the restriction is translated into an IMPLY operation with the two statements as arguments. The first statement, THE CORE OF THE VERB IS PLURAL, is compiled into a list of three operations; first, a call (calls on routines are performed by the EXECUTE operation) on the routine STARTAT with argument VERB, which finds the VERB

OBJECT RECORD:

```
(WT9 [17408],
IMPLY [((EXECUTE [(STARTAT [(VERB)])],
         EXECUTE [(CORERT)],
         ATTRB[(PLURAL)]),
        (EXECUTE [(STARTAT [(SUBJECT)])],
         EXECUTE [(CORERT)],
         NOT [(ATTRB [(SINGULAR)] )]))] )
```

Figure 11

List Structure Generated for Restriction WT9

node in the parse tree; second, a call on the routine CORERT, which finds the word in the sentence which is the core of the VERB; third, the operation ATTRB with argument PLURAL, which checks whether the current word has the attribute PLURAL. The second statement is translated similarly, with the addition of a NOT operation.

The parse tree for the first statement is shown in Figure 12.

(The word THE is ignored in the parsing and so does not appear on the parse tree.) The generators, which have been copied from the RLS onto the parse tree, are executed when the elements they follow have been completed. Thus they are executed in the following order:

generator	*List structure generated*
.ROUTX.(STARTAT)	EXECUTE[(STARTAT[(VERB)])]
.ROUTE.(CORERT)	EXECUTE[(CORERT)]
.MARK	
.GENSYM.(6)	ATTRB[(PLURAL)]
.ATTRBX	
.UNMARK	

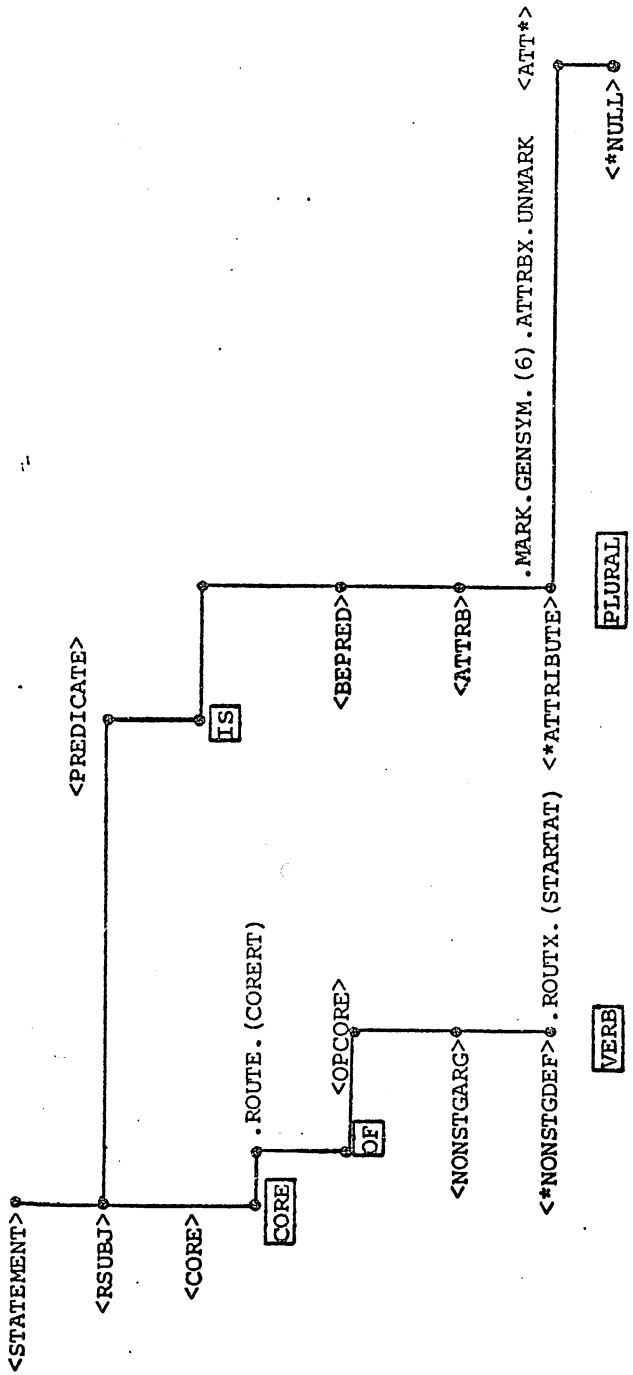

Figure 12

Parse Tree of "THE CORE OF THE VERB IS PLURAL"

Finally, when an object grammar of English has been generated by stage II, it may be used as input to stage III to parse English sentences. During the analysis of a sentence, various traces may optionally be printed. One of these is a trace of the operation of the restrictions; such a trace of the execution of restriction WT9 appears in Figure 13. The + or - after each operation indicates whether the operation succeeded or failed. The trace indicates how the top-level operator, IMPLY, begins by performing the first operation in its first argument, a call on the routine STARTAT with the argument VERB. The trace then shows how STARTAT moves about the tree looking for the VERB, testing each node it comes to with the operation IS(VERB). When the VERB has been located, CORERT is invoked to find its core and then ATTRB(PLURAL) is executed. ATTRB returns plus, indicating that the core of the verb is indeed plural. Because the first argument to IMPLY completed successfully, IMPLY now transfers control to its second argument, which proceeds similarly. The second argument (which tests that the core of the subject is not singular) also succeeds, so the IMPLY operation and hence the restriction is successful.

```
RESTRICTION BEING EXECUTED IS WT9 IN ASSERTION
IMPLY((STARTAT(VERB): ORPTH(IS(VERB)-DOWN+
     ITERT(ORPTH(IS(VERB)-NAMEX-)-RIGHT+
     ORPTH(IS(VERB)-NAMEX-)-RIGHT+
     ORPTH(IS(VERB)-NAMEX-)-RIGHT+
     ORPTH(IS(VERB)+)+)+)+)+
  (CORERT: ORPTH(DNTRN(ATOM,0,ADJSET1)+)+)+
  ATTRB(PLURAL)+
  (STARTAT(SUBJECT): ORPTH(IS(SUBJECT)-
     DOWN+ITERT(ORPTH(IS(SUBJECT)-
     NAMEX-)-RIGHT+ORPTH(IS(SUBJECT)+)+)+)+)+
  (CORERT: ORPTH(DNTRN(ATOM,0,ADJSET1)+)+)+
  NOT(ATTRB(SINGULAR)-)+)+
```

Figure 13

Trace of Restriction WT9

Finally, after many thousands of lines of trace (which are normally supressed), a parse is obtained, such as the one shown in Figure 14 for the sentence "Digitalis acts on the heart."

```
DIGITALIS ACTS ON THE HEART.

     PARSE 1

1. SENTENCE  = INTRODUCER CENTER ENDMARK
                        2.     .
2. ASSERTION = SA SUBJECT * SA VERB * SA OBJECT * RV SA
                  DIGITALIS    ACTS      3.

3. PN        = LP P  NSTGO
                  ON 4. HEART

4. LN        = TPOS QPOS APOS NSPOS NPOS
                  THE
```

Figure 14
A Parse

The thousands of lines of trace which are produced for such a simple sentence are indicative of a basic problem which arises with a large grammar of English: the large number of alternative analyses which must be explored before all parses of the sentence are found. These many alternatives arise in part because nearly every word can have several different word categories, so that each substring of the sentence admits of many different analyses, few of which fit any analysis of the entire sentence. The many different analyses are due in part also to the fact that in this grammar one string can often appear as any of several elements of a containing string; in particular, that an adjunct string can be adjoined to a host string at any one of several different points. This causes particular inefficiencies for a pure top-down

parsing strategy, since the tree for the adjunct string will be dismembered and then built up again from scratch below the new element.

These wandering adjunct strings greatly increase the number of parses of the sentence which are finally obtained; these are the "permanent predictable ambiguities" discussed in the previous paper. As a simple example, the prepositional phrase "on the heart" in "Digitalis acts on the heart" could enter the ASSERTION string as the OBJECT, as the right adjunct of the verb (RV), or as a sentence adjunct (SA). To avoid explicitly generating all these parses, the grammar includes some restrictions which check whether the parse currently being built differs from a parse already obtained only in that some adjunct string has been moved from one point of adjunction to another; if so, the current analysis is abandoned and some other alternative is explored. This mechanism avoids the reconstruction of the tree for the adjunct string, and so significantly reduces the time required to find all parses.

Still, such mechanisms can at best go only part way toward alleviating this basic problem of serial top-down analysis, that the same subtree must be rebuilt many times in different tree contexts. Similar problems which arose with the Harvard Predictive Analyzer were overcome by introducing a saving mechanism [5]. When a tree spanning a certain set of words was successfully completed for the first time, the root symbol of the tree and the words it spanned were recorded in a table; a different entry was made if no tree could be built from that symbol starting at that word. When the same symbol was subsequently encountered while looking at the same word (after backing up and coming forward again) it was sufficient to consult the table to determine whether a tree could be constructed from this symbol; it was never necessary actually to construct the tree a second time.

A saving mechanism on the same basic principle was developed independently for the first implemen-

tation of the Linguistic String Project system [8] and was also included in the second implementation [3]. The mechanism is made much more complex, however, by the presence of restrictions which move about the parse tree. A context-free grammar, such as the one used in the Harvard system, allows a subtree with a particular root symbol which has been constructed in one context to be moved to any other context beginning at the same word where that symbol appears. The string grammar, on the other hand, includes some of the linguistic constraints which may make a subtree built in one context unacceptable in another context beginning with the same word. These constraints are implemented as restrictions which either start in the subtree and look outside of it, or start outside and look in; these restrictions must be reexecuted before one can be sure that a subtree built in one context is valid in a new context. Consequently, whereas the Harvard system need only save the *fact* that a subtree was successfully built, the string parser must save the subtree itself, together with a list of all restrictions which begin inside the subtree and look out.

Since a good deal of information must be stored for each subtree, only a few of the subtrees built during a parse can be saved (whereas the Harvard system can save the success/failure datum for each subtree). In practice, only trees dominated by a "noun string" node were saved. These trees could all be saved in core (on an IBM 7094), and were sufficient for a drastic reduction in parse time (better than an order of magnitude for large sentences). The first parse of a sentence required typically five seconds, with all parses found in less than one minute. The current implementation has about the same basic speed in building nodes and executing restrictions; the overhead introduced by a more flexible version written in FORTRAN roughly compensates for the increase in speed from an IBM 7094 to a CDC 6600. The current implementation, however, does not include a saving mechanism; we are currently considering alternative approaches toward speeding up the parsing process.

Because of the complexity of the saving algorithm, our recent interest has centered on the development of an alternate parsing algorithm which would obtain all parses of a sentence in a single left to right scan. One such algorithm, the immediate constituent analysis (ICA) algorithm developed by Cocke [6], has been extensively used for natural language analysis. We are currently investigating the *nodal spans* parsing algorithm developed by Cocke and Schwartz [4], which is essentially equivalent to the algorithm described by Earley [7]. In this algorithm, a given symbol spanning a given substring of the sentence will be recorded only once, even if it is used in different contexts in different analyses of the sentence. From the point of view of efficiency, this is equivalent to saving all subtrees — no subtree is ever built twice. Some preliminary experiments have been performed in which a nodal span parser was used to construct all analyses of a sentence according to the context-free component of the grammar, after which the restrictions were applied to eliminate invalid analyses. The results of these experiments were entirely negative: eight-word sentences exceeded the memory capacity of the computer.

This is not entirely surprising, since the restrictions are an integral part of the grammar and a lot of spurious analyses are produced if they are omitted. If the nodal spans algorithm is to have any chance at all, the restrictions must be executed during the parsing process, so that the large number of spurious parses will not be stored. For the restrictions to execute efficiently with the new parsing algorithm, however, much of the grammar will have to be rewritten; this is a considerable task, on which some work is just now beginning.

Until this task is completed — until the grammar is rewritten, the new parser is debugged, and some long sentences are run — we have no way of knowing whether this approach will bear fruit. With the top-down parser with saving there is a

continuous space-time tradeoff: the more space it is given, the more subtrees it can save, and the faster it runs, but it is able to run (albeit slowly) with any memory allocation sufficient to contain the current parse tree. In contrast, with parsers performing a single left-to-right scan, such as nodal spans, the situation is all-or-nothing: either a parse is rapidly obtained or memory overflows. For example, we could conceivably end up with a parser which will process sentences of fewer than 30 words at impressive speed but be unable to process longer sentences. This would be of little use in our current project, which aims to process scientific text (containing many long sentences), although it may be of considerable interest to future research in parsing algorithms. On the other hand, if our effort is successful, and memory requirements can be brought within the capacity of the CDC 6600, we should obtain a very fast parser for the string decomposition of English sentences.

REFERENCES

[1] Sager, Naomi, "The String Parser for Scientific Literature," this volume

[2] ——, James Morris, Morris Salkoff, and Carol Raze, "Report on the String Analysis Program," *String Program Reports 1*, New York University, Linguistic String Project, 1966.

[3] Raze, Carol, "The FAP Program for the String Decomposition of Scientific Texts," *String Program Reports 2*, 1967.

[4] Cocke, John and Schwartz, Jacob T., *Programming Languages and Their Compilers*, New York Univ.

[5] Kuno, Susumo, "The Predictive Analyzer and a Path Elimination Technique," *CACM 8*, 1965.

[6] Hays, David G., "Automatic Language Data Processing," *Computer Applications in the Behavioral Sciences*, ed. H. Borko, Prentice-Hall, 1962.

[7] Earley, Jay, "An Efficient Context-Free Parsing Algorithm," *CACM 13*, 1970.

[8] First Report on the String Analysis Program, Linguistic String Project, Department of Linguistics, University of Pennsylvania, 1965.

ACKNOWLEDGMENT

This research was supported in part by Research Grant No. LM00720-01, from the National Library of Medicine, National Institutes of Health, DHEW, and in part by Research Grants GS-2462 and GS-27925, from the National Science Foundation, Division of Social Sciences.