Information Retrieval and Language Processing
C.A. Montgomery Editor

The Restriction Language for Computer Grammars of Natural Language

Naomi Sager and Ralph Grishman
New York University

Over the past few years, a number of systems for the computer analysis of natural language sentences have been based on augmented context-free grammars: a context-free grammar which defines a set of parse trees for a sentence, plus a group of restrictions to which a tree must conform in order to be a valid sentence analysis. As the coverage of the grammar is increased, an efficient representation becomes essential for further development. This paper presents a programming language designed specifically for the compact and perspicuous statement of restrictions of a natural language grammar. It is based on ten years' experience parsing text sentences with the comprehensive English grammar of the N.Y.U. Linguistic String Project, and embodies in its syntax and routines the relations which were found to be useful and adequate for computerized natural language analysis. The language is used in the current implementation of the Linguistic String Parser.

Key Words and Phrases: natural language, parsing, grammar, programming languages
CR Categories: 3.42, 3.79, 4.22

This research was supported in part by the National Science Foundation, Division of Social Sciences, Grants GS2462 and GS27925. Authors' address: New York University, Linguistic String Project, 2 Washington Square Village, 2B, New York, New York 10012.

Introduction

The Restriction Language (RL) was developed by the Linguistic String Project at New York University in order to provide a convenient form for expressing a large-coverage, detailed grammar of English. The language and its compiler are presently operative on the CDC 6600, and they function as part of a highly developed system for parsing English scientific texts. The successive implementations of the English parsing system, as well as its large grammar of English, have been described elsewhere [1–5]. This paper will describe the design and special features of the symbolic language in which the grammar is written.

While the major features of the language grew out of the needs of our particular type of grammar (linguistic string grammar), we have tried to respond to these needs by adding the most general mechanisms consistent with reasonable efficiency. For example, those parts of the metalanguage which depend on the choice of grammatical theory have been relegated for the most part to user-defined routines. In this way, we believe we have constructed a vehicle flexible enough to serve in a variety of language analysis tasks, which may use different linguistic theories, with little or no change in the system.

Background

The possibility of processing natural language texts as part of computerized information systems was one of the early goals (along with machine translation) of research projects in computational linguistics in the late 1950's. As early as 1957, a project at the University of Pennsylvania was initiated to develop a sentence analysis program based on transformational theory for application in information retrieval [6]. Syntactic segmentation of sentences, using string analysis [7], was to be the first step in this process. The Linguistic String Project inherited this task, and it took, in fact, a number of years to make the string segmentation step operational on a broad scale. In the foreign language area, pioneer work on computerized analysis of Russian texts was done at the Bureau of Standards in the late 1950's [8, 9], leading to a later generalization and application to English in the Harvard Predictive Analyzer [10].

These and other early projects in computational linguistics had large, real-world, technological goals, and drew upon a large reservoir of linguistic theory, tools, and data. What was unfortunately lacking at that time was a computational formulation of all the necessary language detail, as well as the programming tools to deal with such a large and complex data array as a natural language grammar. We believe that these two obstacles have been largely overcome in the past decade. Linguistic knowledge has accumulated, and a great deal of effort has gone into casting linguistic facts into a computable form. At the same time, the development of higher level programming languages and compilers makes it possible to deal with complex data on several levels, reserving the highest level with the briefest statements for human use. In the case of the natural language system, this is essential because of the very large amount of detail.

The language described here is our answer to the need for a higher level tool for natural language processing. Its design was heavily influenced by our experience in using a large grammar to parse long and complex sentences, such as one finds in scientific texts. We found that the key to successful text parsing lay in the portion of the grammar which deals with the special detail of natural language: in our system, the restriction component of the grammar. The major focus of the Restriction Language, then, is to provide a succinct and powerful vehicle for expressing the detailed grammatical and semantic constraints that operate in sentences. We will illustrate how the language accomplishes this task in the Linguistic String Parsing (LSP) system.

The LSP grammar has two principal components: a context-free grammar and a set of restrictions. The context-free grammar associates with each input sentence a set (possibly empty) of parse trees. The restrictions state conditions on a parse tree which must be met in order for the tree to be accepted as a correct analysis of the input sentence. The restrictions are used to express detailed wellformedness constraints that are not conveniently statable in the context-free component. These may be contextual constraints on the linking of different subtrees or, in the case of natural language, may involve attributes of the particular lexical items which are associated with terminal symbols of the parse tree. Thus, for a natural language parse tree which terminates in a well-formed sentence string consisting of Noun + Verb + ".", e.g. "John worries.", there are many word sequences conforming to this categorization which are not well-formed sentences, such as "John worry." (violates subject-verb agreement in number), "John wears." (verb occurs without necessary object), and "John occurs." (inappropriate subject noun for verb). Restriction rules specifying constraints must be applied if the output of the parsing program is to contain no incorrect parses.

The restrictions in the grammar are procedures which look at the parse tree and the attributes of the sentence words and return an OK or a not OK signal. This basic strategy of grammar design, in which a context-free framework is augmented by a set of conditions written as procedures, has become quite popular; it is used, for example, in the systems of Woods [12] and Winograd [11]. Our own use of such an organization dates from the first implementation of the Linguistic String Parser in 1964–65 [13]. This approach was motivated by a need to provide an adequate English grammar for parsing scientific texts. It was clear that such a grammar would be large and would have to be able to absorb increasing levels of detail without exploding. The two-component organization has made this possible: the context-free grammar, which defines the broad construction patterns of sentences, remains relatively stable, while the restrictions, which cover the detailed constraints, are gradually refined. The restrictions are relatively independent of one another, so that these refinements can be made locally, without a major revision of the restrictions at each step.

Even so, the development of a large-coverage grammar is a lengthy procedure because of the amount of detail which must be included. The restrictions in the earlier versions of our system were prepared in a list-structure format similar to LISP. This level of description is comparable to those used in other current systems, such as Winograd's PROGRAMMAR and the LISP procedures employed by Woods. At this level, however, our grammar ran to several thousand lines. As a result, to facilitate further experimentation and development of the grammar, we chose to design a language especially suited to such linguistic tasks; its first application and test of adequacy would be the writing of restrictions. A syntax-directed translator which compiles this Restriction Language into an internal list-structure form is included in the most recent implementation of the Linguistic String Parser.

An Overview of the Language

The design of the RL has been affected by a number of considerations which differentiate it from most other programming languages. These considerations include
1. The primary application: stating predicates on trees.
2. Interaction of the restrictions with the context-free parsing process.
3. The solution of problems inherent in natural-language grammar.

The first consideration has motivated a choice of style: since the restrictions state conditions, the general form of the language has been made declarative rather than procedural. Specifically, the basic statements of RL appear as simple English declarative sentences, such as: THE CORE OF THE SUBJECT IS NOT PLURAL. Generally the subject of the RL statement locates a node in the parse tree or an attribute of a word definition, and the predicate performs some test on that node or attribute. These basic statements may be combined into restrictions of arbitrary complexity by the use of the logical connectives BOTH . . . AND . . . , EITHER . . . OR . . ., IF . . . THEN . . ., etc. A provision for what are in effect procedures without arguments is also provided. Any RL statement may be assigned a name beginning with a $ and then be referenced from any point in another statement where the original statement could appear. In this way, tests common to several restrictions need appear only once, and complex restrictions can be divided into a sequence of statements.

Procedures which may take arguments are also included, but under a different guise and for a different purpose. The components of a sentence which are tested by any single restriction are related to each other by one of a small number of structural relationships (this is a consequence of the use of linguistic string theory as a framework for the grammar). Rather than have the sequences of tree-climbing operations which realize these relationships stated explicitly in each restriction which uses them, they are placed in procedures called *routines*. The restrictions refer only to these routines, and never directly to individual tree-climbing operations. The restrictions are thereby made easier to read and independent of the particular tree structures used to represent the string analysis.

Underlying our description of the semantics of the RL is the notion of an implicit pointer, which may point to any node in the parse tree, any attribute, or any word of the input sentence. This pointer is moved by the tree-climbing operations in the routines, and thus indirectly by each statement in a restriction. The pointer will be referred to by such phrases as "where the restriction is" or "the node which has been located." In addition, each statement and routine set an implicit flag to signal success or failure. This flag is used by the logical connectives to determine the flow of control and, ultimately, to determine whether the restriction succeeds or fails.

In the next section, we shall use an example to introduce some basic concepts and to illustrate the language features. After that, we shall describe some of the individual features in greater detail.

An Example

Figure 1 shows the source form of restriction WSEL1 (W: wellformedness, SEL: selection) taken from the LSP grammar of English. The intent of the restriction, stated in the comment card (beginning with *), is to insure that an appropriate noun is selected as the object of a given transitive verb. "Appropriate" here means normally acceptable in written English, excluding such special literary contexts as fairy tales, poetry, and the like. For example, in parsing the sentence "John eats each day.", WSEL1 rejects the parse which has "each day" as the object of "eats." This sentence has only one appropriate parse, with "John eats" as the main clause and "each day" as a time adjunct of the main clause.

To understand this restriction a few facts about the English parsing program and grammar should be stated. All words in a text to be parsed are defined in a word dictionary, which is supplied as a separate input to the parser. Each word definition is a sequence of categories, such as N (noun), PRO (pronoun), TV (tensed, i.e. inflected, verb), representing the major parts of speech to which the given word belongs. Each category may have associated with it a set of attributes (such as SINGULAR,

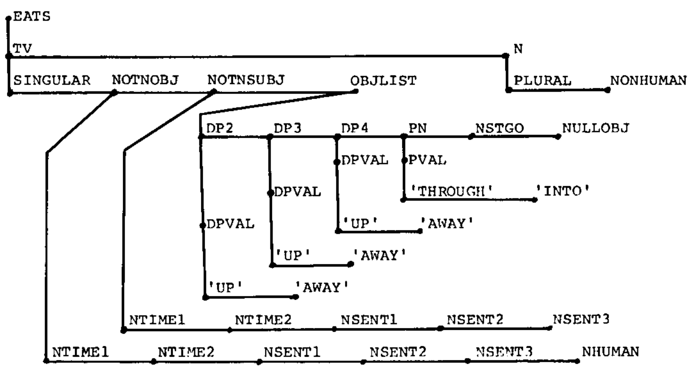
Fig. 1. A restriction from the English grammar.

```
*WSEL1:  Suitable object N for given verb

WSEL1 = IN OBJECT AFTER NSTGO:
        IF ALL OF $OBJECT-NOUN, $GOVERNING-VERB,
        $FORBIDDEN-NOUN-LIST ARE TRUE, THEN $NO-
        COMMON IS TRUE.

$OBJECT-NOUN = THE CORE X3 OF THE OBJECT X10
        IS N OR PRO.

$GOVERNING-VERB = AT X10 THE VERB-COELEMENT
        X4 EXISTS.

$FORBIDDEN-NOUN-LIST = THE CORE OF X4 HAS
        THE ATTRIBUTE NOTNOBJ X5.

$NO-COMMON = X3 AND X5 HAVE NO COMMON ATTRIBUTE.
```

Fig. 2. Entries from the word dictionary.

```
       EATS
       TV: .12, N:  .11.
 .12 = SINGULAR, NOTNOBJ: .1, NOTNSUBJ:  .2, OBJLIST:  .3.
  .1 = NTIME1, NTIME2, NSENT1, NSENT2, NSENT3, NHUMAN.
  .2 = NTIME1, NTIME2, NSENT1, NSENT2, NSENT3.
  .3 = DP2:  .6, DP3: .6, DP4: .6, PN: .5, NSTGO, NULLOBJ
  .5 = PVAL:  ('THROUGH', 'INTO').
  .6 = DPVAL:  ('UP', 'AWAY').
 .11 = PLURAL, NONHUMAN
```

Tree Structure



```
       DAY
       N:  .11.
 .11 = SINGULAR, NTIME1, NUNIT, NONHUMAN.
```

Tree Structure

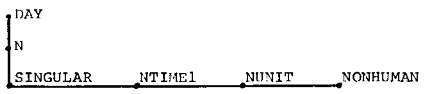
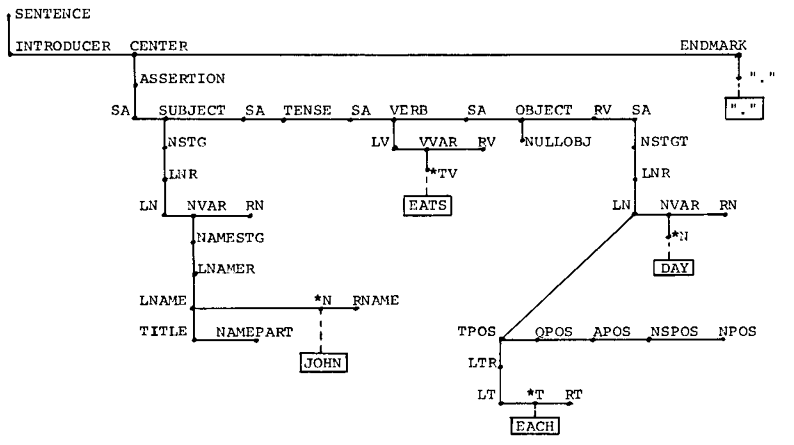


Fig. 3. Parse tree for "John eats each day." Note: Terminal symbols are flagged by an asterisk in front of the symbol name. Other nodes with no descendants indicated in the parse trees actually have a single descendant, the symbol NULL (the empty string).



PLURAL, NOMINATIVE, NTIME1, NTIME2) and attributes of the attributes organized into a tree structure for that category. In parsing, when a category is matched by a terminal symbol of the parse tree, a pointer is created from the terminal node to the attribute tree of that category. In this way, restrictions can test attributes and subattributes of the sentence words associated with particular terminal nodes of the parse tree.

Relevant to our example, dictionary entries for "eats" and "day" are shown in Figure 2, along with the tree diagrams for these definitions. It will be seen in Figure 2 that "day" has the category N (noun) with the attribute NTIME1 among other attributes, and that the verb "eats" has the category TV (tensed verb), with attribute NOTNOBJ (among others), whose attributes in turn are NTIME1, NTIME2, NSENT1, NSENT2, NSENT3, and NHUMAN. NOTNOBJ is assigned in the word dictionary to transitive verbs; its attributes for a given verb are those noun subclasses of the grammar which are not appropriate noun objects of the given verb (in scientific writing). Thus the appearance of NTIME1 as an attribute of NOTNOBJ in the entry for "eats" indicates that nouns having the attribute NTIME1 (e.g. "day") are not acceptable as the noun object of forms of "eat."

The parse tree for "John eats each day." is shown in Figure 3. In this parse the main clause (the value of CENTER) is an ASSERTION string, whose definition, as it appears in the context-free component of the grammar, is:

$$\langle\text{ASSERTION}\rangle ::= \langle\text{SA}\rangle\ \langle\text{SUBJECT}\rangle\ \langle\text{SA}\rangle\ \langle\text{TENSE}\rangle\ \langle\text{SA}\rangle\ \langle\text{VERB}\rangle$$
$$\langle\text{SA}\rangle\ \langle\text{OBJECT}\rangle\ \langle\text{RV}\rangle\ \langle\text{SA}\rangle.$$

In the parse tree, sibling nodes (corresponding to elements of a single definition) are shown connected horizontally to stress their sequential order, with only the leftmost sibling connected to the parent. The four syntactic types, SUBJECT, TENSE, VERB, and OBJECT, are said to be required, since they take on a null value only under special conditions. The other nodes correspond to positions where modifiers, or adjunct strings, may occur; five are of the type SA (sentence adjuncts), modifiers of the assertion as a whole, and one is of the type RV (right adjuncts of the verb) occurring after the verb object (e.g. "fast" in "He made up his mind rather fast."). In the grammar, *string definitions*, such as ASSERTION, have a single option with several elements, whereas *adjunct set definitions*, such as SA and RV, have several options, each of one element.

In Figure 3, all of the SA nodes in ASSERTION except the last, as well as the RV node, have the value NULL, which means that no sentence words are subsumed. NULL is included as one option of all adjunct set definitions and represents the fact that modifiers are optional. The OBJECT node in Figure 3 has a null value (NULLOBJ) of different linguistic significance; it satisfies OBJECT when the governing verb (here "eats") occurs intransitively. The TENSE node also has a NULL value; this node is nonempty only when a modal ("will," "can," etc.) occurs. If the tense is morphologically tied to the verb, as is the case in "eats," then the terminal symbol TV occurs under VERB and the TENSE node is empty.

Space does not permit a detailed discussion of the substructures of the parse tree in Figure 3. (The LSP grammar is described in [14, 15].) However, it can be noted that the SUBJECT node and the final SA node in ASSERTION both dominate subtrees which terminate in nodes labeled N, corresponding to nouns in the sentence ("John" and "day," respectively). Both of these subtrees contain an entity LNR whose three elements, LN, NVAR, RN, stand for the *L*eft adjuncts of the *N*oun, followed by a *N*oun or one of its *VAR*iants, followed by the *R*ight adjuncts of the *N*oun. This type of structure corresponds to a type of definition in the string grammar, called an LXR-type definition. This type of structure is seen again in Figure 3 under VERB, LNAMER, and LTR. The LXR-type definition is the standard representation in the LSP grammar for a sentence word with its associated modifiers. The introduction of standard definition types into the grammar has facilitated the writing of powerful tree-climbing routines whose path through the parse tree can be stated in terms of node types, such as LXR, ADJUNCT SET, STRING, rather than individual nodes, such as LNR, RN, ASSERTION, whenever this amount of generality is needed [16].

This can be illustrated with reference to Figure 3 by introducing a routine which is invoked in the restriction we are about to describe. In Figure 3 one observes that each of the nonnull nodes, SUBJECT, VERB, and post-object SA in ASSERTION, has a special relation to a particular terminal node in its subtree, respectively to N ("John"), TV ("eats"), and N ("day"). If a breadth-first downward scan of the subtree is made from the higher-level node of each pair, passing through all nodes except those corresponding to an adjunct set, and terminating at the first terminal node encountered, then in each case, the terminal node in question will be the one reached. The importance of this is that the words corresponding to these terminal nodes are each the carrier in the sentence of the linguistic relation named by the higher level node of the pair; e.g., "John" is the subject in the main assertion in the sentence, "eats" is the main verb, and "day" (with adjunct "each") has the relation of sentence adjunct to the other elements. This relation between node-pairs, called *core*, is crucial to the operation of restrictions. By means of the routine CORE, which executes the breadth-first scan described above, constraints which are expressed in terms of higher level grammatical relations, such as subject, object, and governing verb, can be applied to the words in the sentence which satisfy these grammatical relations.[1]

With these preliminary remarks, we may now delve

[1] In some cases, the core of a node is a string. For example in the sentence "What John eats is fish.", the core of the subject is a string which subsumes the words "What John eats." Thus, a more precise definition of the core of a node α (α not of type STRING) is the unique node of type STRING or ATOMIC (terminal node) in the subtree below α, which is not below a node of type ADJUNCT SET in that subtree.

into the restriction itself (Figure 1). The restriction begins with the line

WSEL1 = IN OBJECT AFTER NSTGO:

which identifies the restriction and says when and where it should be executed. In our system a top-down parser interprets the context-free component of the grammar and produces the parse trees for a sentence sequentially. In principle, all the restrictions could be applied after each parse tree had been completed. It is more efficient, however, to execute each restriction as soon as enough of the parse tree has been built for it to apply. For this reason each restriction lists one or more definitions and indicates whether the restriction is to be executed before a particular option is tried or after a subtree has been completed below a particular option or element. If a restriction fails, the parser backs up and tries for an alternate analysis. Thus, the restriction WSEL1 will be executed immediately after a subtree corresponding to the option NSTGO (*N*oun *ST*rin*G* in *O*bject) of OBJECT has been completed. This will occur in the analysis of the sentence "John eats each day." when the parser constructs a subtree below OBJECT subsuming "each day"; the appearance of the parse tree when this subtree is completed is shown in Figure 4. The restriction begins at the node in the parse tree corresponding to the first definition named in the initial portion of the restriction; in this case, at the OBJECT node.

The body of the restriction is an implication with three conjoined premises and one conclusion. Each of the four basic statements has been assigned a name beginning with a $:

IF ALL OF $OBJECT-NOUN, $GOVERNING-VERB, $FORBIDDEN-NOUN-LIST ARE TRUE,
THEN $NO-COMMON IS TRUE.

The first premise,

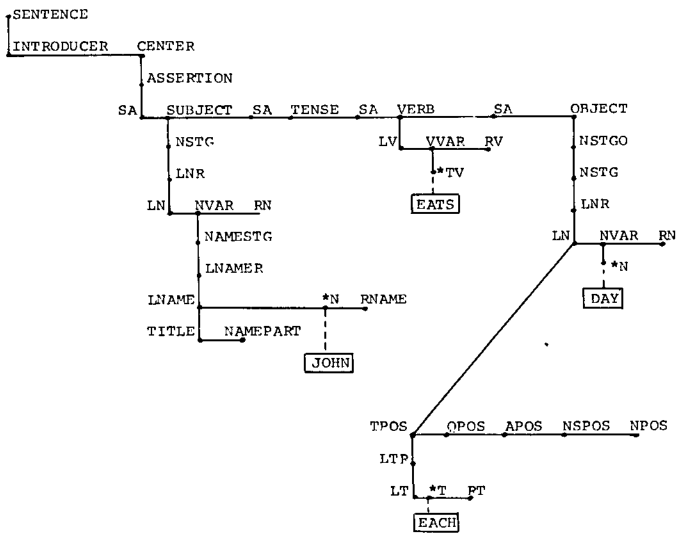
$OBJECT-NOUN = THE CORE X3 OF THE OBJECT X10 IS N OR PRO.

tests whether the core of the object is a noun or pronoun, and in addition stores a pointer to the node OBJECT in register X10 and a pointer to the core in register X3. Operating on the tree in Figure 4, this will place in X3 a pointer to the atomic node N ("day"). Registers, symbols consisting of an "X" followed by an integer, are the variables of the RL; they may contain pointers to any node, attribute, or sentence word. Should this first premise fail, the restriction is satisfied and a success signal will be returned to the context-free parser; the other statements will not be evaluated. This is as it should be, since if the object is not a noun or pronoun the object cannot be in an unacceptable noun or pronoun subclass.

The second premise,

$GOVERNING-VERB = AT X10 THE VERB-COELEMENT X4 EXISTS.

Fig. 4. Partial parse tree rejected by restriction WSEL1.



also has as its aim to locate and store a node on which further processing will be done. This node is the verbal sibling of OBJECT, in this case VERB. In other strings the verbal element may have a different name and different substructure (e.g. a *Ving* form, as in "is eating"), so that it is convenient to define a generalized routine, VERB-COELEMENT, which locates the desired verbal sibling in all cases. The $GOVERNING-VERB test, applied to the tree in Figure 4, starts at OBJECT (the value of X10), executes the VERB-COELEMENT routine, which brings it to the node VERB, and stores a pointer to VERB in X4. Again, if this test fails, the restriction is trivially satisfied, since this means there is no governing verb present.

The third premise,

$FORBIDDEN-NOUN-LIST = THE CORE OF X4 HAS THE ATTRIBUTE NOTNOBJ X5.

invokes the CORE routine starting at the node VERB (the value of X4) to obtain the governing verb itself (here TV, "eats"). It then searches the attribute list of category TV in the dictionary definition of "eats" for NOTNOBJ. If the search is successful, a pointer to the attribute NOTNOBJ is placed in X5.

The conclusion,

$NO-COMMON = X3 AND X5 HAVE NO COMMON ATTRIBUTE

compares the attribute list of category N of "day" (the value of X3) with the attribute list of NOTNOBJ in "eats" (the value of X5). If these lists have no element in common then "day" is acceptable as an object of "eats." In this case they have the attribute NTIME1 in common, so that the test $NO-COMMON fails, and the partial parse tree shown in Figure 4 is rejected.

Principal Constructions of the Restriction Language

Since the chief form of statement in the Restriction Language is a declarative sentence consisting of a subject and a predicate, we shall begin our presentation with a description of the various forms the subject and predicate can take. Thereafter we shall consider how statements may be combined with logical connectives and how register references and assignments can be inserted in statements. From there we shall turn to some statement forms not fitting the subject-predicate mold: the imperative forms used in the routines and some of the more unusual features of the language.

The Subject. As we noted earlier, the subject of a restriction language statement generally locates a node, attribute, or sentence word which is then tested by the predicate. Since the routines carry the burden of locating nodes in the parse tree, the routine invocation is the most frequently used type of subject. If a routine has no argument, it is invoked by simply writing its name: CORE. If a routine takes an argument, the argument is written immediately after the routine name. COELEMENT SUBJECT calls the routine COELEMENT with the argument SUBJECT; this will search the siblings of the current node for a node named SUBJECT. If a node name appears alone, without any routine name: OBJECT, a routine called STARTAT is invoked with the node name as argument. STARTAT examines the current node and the nodes one level below the current node for the node name given as argument. Frequently more than one routine must be executed to get from the starting point to the node to be tested. This can be accomplished by combining several routine calls with the word "OF."
For example,

CORE OF THE VERB-COELEMENT OF OBJECT[2]

would first execute STARTAT with argument OBJECT, then VERBCOELEMENT (which would begin where STARTAT finished) and finally CORE (which would begin where VERB-COELEMENT finished). If any routine invoked by a statement fails, execution of the entire statement is immediately terminated with a failure indication. Operations are also provided for locating a word in the sentence, such as the first or last word subsumed by a node or the next word to be matched in the parsing process.

The Predicate. The predicate tests whether the node, attribute, or word located by the subject has a particular property. The chief types of tests which can be made are the following.

1. Test of node name. One can test if the current node has a certain name with the IS predicate:

... IS N.

Any set of node names can be defined as a *type* of node. One can then test whether the name of the current node is in a given set with the IS OF TYPE predicate:

... IS OF TYPE ADJSET.

[2] The articles A, AN, THE are ignored by the RL compiler and may be inserted freely to enhance readability.

2. Test for attribute. To test if an atomic node or an attribute has a particular attribute, the HAS ATTRIBUTE predicate can be used:

... HAS ATTRIBUTE PLURAL.

It is frequently necessary to test both an atomic node's name and one of its attributes, or an attribute and one of the attributes of the attribute. Such conditions can be expressed succinctly by following the atomic node name or attribute by a colon (read "which has attribute") and the second attribute:

... IS N:PLURAL
... HAS ATTRIBUTE NOTNOBJ:NTIME1.

3. Test for name of word. The IS predicate also serves to test the name of the word matched to an atomic node: ... IS 'THE'. The same predicate serves to test the name of a sentence word located by the subject of a restriction statement.

4. Test for subsumed word. The SUBSUMES predicate tests whether any of the words subsumed by a node meets a stated condition. Any test which can be applied to a single sentence word (test for name, for category, or for category and attribute) can also be used in the SUBSUMES predicate:

... SUBSUMES ','
... SUBSUMES N:NTIME1.

In order to keep the semantics of the RL as simple and explicit as possible, we have endeavored to compile the more complex test operations into calls on routines in the grammar whenever feasible. Since the process of iterating through the words subsumed by a node can be expressed in terms of more elementary operations already in the RL, the SUBSUMES predicate is compiled into a call on the SUBSUMERT routine. The test which must be performed on each word is passed as an argument in the form of a procedure to the routine SUBSUMERT (in this respect SUBSUMERT differs from the routines invoked in the subject, which take a node name as argument). The special case where a node subsumes no sentence words can be tested for with the IS EMPTY predicate.

5. No test. Occasionally one only wants to determine whether a node specified in the subject can be located in the parse tree; no further test on that node is required. One instance of this occurred in the second premise of WSEL1. For this purpose RL provides the EXISTS predicate, which always succeeds (the statement fails only if the subject fails).

6. Parse tree tests. Any condition regarding the structure of the parse tree can be expressed entirely in the subject of a statement. For example, starting at the node NAMESTG, to determine whether it occurs as part of a tree dominated by SUBJECT, one can execute:

IMMEDIATE SUBJECT OF NAMESTG EXISTS.

where IMMEDIATE is a routine which looks up in the parse tree for the node named in the argument. Since an important consideration in the design of the RL has been the readability of the grammar, however, we allow the user some alternative formulations. First, the final routine may be invoked in the predicate, using the HAS predicate:

NAMESTG HAS IMMEDIATE SUBJECT.

Second, in the case of a few routines there is an alternative wording which is much more natural in the predicate position; for the routine IMMEDIATE we also permit:

NAMESTG IS OCCURRING AS SUBJECT.

The negation of any predicate may be expressed simply by adding NOT to the verb:

. . IS NOT N
. . . DOES NOT SUBSUME ','.

Wherever a word or symbol may appear in a predicate, the disjunction of two or more words or symbols may appear instead:[3]

. . . IS N OR PRO
. . . DOES NOT SUBSUME 'A' OR 'AN'.

A few of the tests which must be made in the grammar do not fit neatly into the subject-predicate organization, and so are included as separate types of statements. One of these is the COMMON ATTRIBUTE test, which was used (in its negated form) in WSEL1. Another is the THERE IS . . . AHEAD, which tests whether any sentence word not yet matched by a node in the parse tree meets a stated condition; for example,

THERE IS AN N:NTIME1 AHEAD.

This statement is used in optimization restrictions, to avoid trying an option if a required category or attribute is not present in the remainder of the sentence.

Connectives. The RL provides a full range of connectives for specifying logical combinations of statements and thereby in effect for specifying the flow of control within a restriction. The logical operators include NOT, AND, OR, NEITHER . . . NOR . . . , IF . . . THEN . . . , and can be nested to any depth. The operator evaluates only as many statements as are required to determine the value of the operator; thus, in a statement of the form BOTH *a* AND *b*, if *a* fails the entire statement fails, and *b* is not executed. Iteration operators, for constructing loops with a termination test either at the beginning or end, are also available, although in the current grammar they are used only inside the routines.

Any construction which can be used in the subject of a statement to locate a node can also be used in an AT . . . or IN . . . phrase before a statement, to specify where a statement should begin. For example,

AT OBJECT BOTH THE CORE X3 IS N OR PRO AND THE
  VERB-COELEMENT X4 EXISTS.

is equivalent to the first two premises of WSEL1.

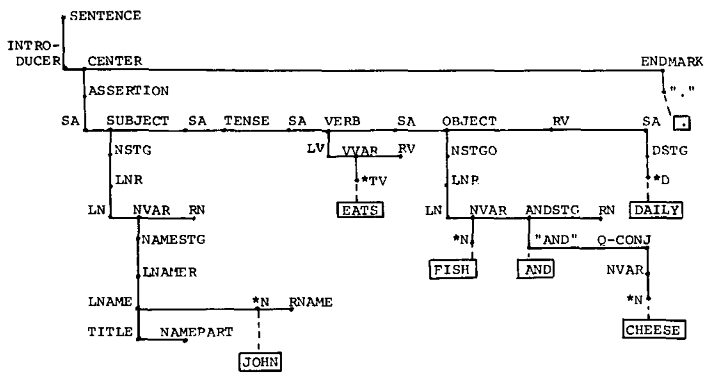
Registers. As we noted earlier, registers are the variables of the RL. A register may be assigned a value by writing its name after any subject, predicate, or routine invocation. The value assigned to the register is a pointer to the node, attribute, or word at which the restriction is located after the subject, predicate, or routine is executed. The register may be referenced by using it in the subject of a statement or in an AT phrase; both usages are illustrated in WSEL1. Registers are used frequently in the grammar in order to avoid having to locate the same node several times in one restriction.

Commands. The declarative statement format, which has proven so convenient for stating the restrictions, is not particularly suitable for the routines, especially the low-level routines which deal in terms of elementary tree motion and testing operators. The RL therefore provides an imperative format, similar to that used in procedural languages, for writing routines. In this format each statement or command specifies an individual elementary operation, such as GO UP, GO DOWN, GO LEFT, GO RIGHT (in the parse tree); TEST FOR SUBJECT; STORE IN X5; DO CORE (routine invocation). Commands may be strung together with semicolons (this is equivalent in effect to combining them with BOTH . . . AND . . .).

Monitoring the Parsing Process. Although the foregoing discussion would indicate that the restrictions evaluate each parse tree independent of the course of the parsing process to that point, this is not quite correct. Both for reasons of efficiency and to avoid unlikely analyses when more common analyses are available, several statements are included which can monitor the overall parsing process. The simplest such statement tests whether

A PARSE HAS BEEN OBTAINED.

Fig. 5. Parse tree for the sentence "John eats fish and cheese daily."



[3] The fact that disjunction may be expressed in predicates whereas conjunction requires the combining of entire statements is a consequence of the list structure into which the RL is translated: each elementary test operation takes a list of symbols as its argument, and succeeds if any one of the symbols can be found. In practice, testing for a conjunction of symbols is a rare occurrence in the grammar, so this has proved no inconvenience.

The parser also provides a more sophisticated method of selecting a preferred analysis by defining nested subsets of the English grammar. The input sentence is first analyzed with respect to the smallest subset, a grammar containing the common English constructions. If no parse is obtained, the sentence is reanalyzed with respect to the next larger subset, which includes some infrequent constructions. This process is repeated until an analysis is obtained or the entire grammar has been used. These subsets are specified in the grammar by including restrictions which test one of a set of switches. These switches are all off in the initial attempt at analyzing a sentence, and are turned on one by one in the subsequent stages of the process. Because of the richness of the grammar, this mechanism also makes an important contribution to parsing efficiency. The current grammar defines two subsets; the smaller, the "non-rare" grammar, is still adequate for most of the sentences in scientific texts. As a result, the time lost on reanalyzing the more unusual sentences is less than the time saved in analyzing the more common sentence types with a smaller grammar.

One further statement provided in the RL makes it possible to find out how far the context-free parser has backed up since the last parse tree for the sentence was found. This test is used as part of a scheme to avoid generating several parse trees in cases of permanent predictable ambiguity, such as when an adjunct string can be attached at several points in the tree (see [1] for further explanation).

Node Attributes. The RL includes a facility for assigning and testing node attributes. A node attribute is in effect a variable associated with a particular node in the parse tree; these variables may be assigned either the value true or false, or a pointer to some other node in the parse tree. Unlike assignments to registers, node attributes are not erased when a restriction is finished. Node attributes simplify the task of writing the grammar and can make the restrictions and routines much more efficient. For example, if a lengthy routine is frequently executed, one can save time by recording the node located by the routine as a node attribute of the starting node the first time the routine is executed at that node, and referring to the node attribute thereafter. Another application arises when one wants to know whether a particular node occurs in some subtree. Rather than search the entire subtree, one can assign a node attribute to the root of the subtree when the node in question is attached, and test that node attribute later.

A complication arises because a restriction housed on one node can assign an attribute to some other node. An instance of this arises in the relative clause construction. When an ASSERTION string occurs in a relative clause, the omitted element (e.g. the OBJECT in "what I eat" and the SUBJECT in "what eats me") is marked by assigning it the value NULLWH. When the ASSERTION is completed, a restriction checks that precisely one element has been omitted. This could be done by having the restriction search all possible points of omission. A more efficient procedure, employed in the LSP grammar, is to have a restriction housed on NULLWH assign the node attribute DIDOMIT to the ASSERTION node. The restriction on the ASSERTION need then merely check for the presence of this attribute.

Suppose, however, that the sentence contained a clause without omission, such as "that I eat cheese." The parser might try the value NULLWH for OBJECT (and hence assign DIDOMIT to the ASSERTION above). Because of this (incorrect) choice, the parser would eventually get stuck, unable to complete the parse; it would then back up and try an alternate option for OBJECT (in this case, "cheese"). At this point the node attribute DIDOMIT should be removed, since the ASSERTION no longer contains an omission. This erasure is performed automatically by the parser: if a node attribute is assigned by a restriction invoked when a node N is completed, the attribute will be erased when the parser "backs up" into node N. Similarly, if a restriction were executed when node N was attached to the tree, any node attributes assigned by that restriction will be erased when node N is detached from the tree. Just as the execution of restrictions and hence the assignment of node attributes is synchronized with the forward progress of the context-free parser, the undoing of these assignments must be synchronized with the backup of the parser.

An example of the use of this feature in connection with conjunctions is given below.

Conjunctions. The extensive treatment of conjunctional constructions in the English grammar has given rise to several unusual features of the RL. For example, when one attempts to provide for all the different structures which can be headed by conjunctions, it becomes apparent that the grammar would be overburdened by explicit definition of all conjunctional strings. The definitions of the conjunctional strings are therefore generated dynamically when a conjunction is encountered in the parsing process. The generated conjunction string consists of a conjunction followed by a sequence of elements which repeats a sequence of elements already present in the parse tree. Which particular sequence is generated depends on where the conjunction string is to be placed in the parse tree. For example, in the parse tree for the sentence "John eats fish and cheese daily" shown in Figure 5, the conjunctional string has been inserted to the right of the node NVAR in the LNR dominated by OBJECT; the "repeating" portion (Q-CONJ) of the string here consists of the node NVAR, so that the whole local construction consisting of the first NVAR plus the conjunctional string containing the second NVAR covers the word-sequence "fish and cheese." Had the word-sequence been, for example, "John eats fish but dislikes cheese," the conjunctional string would have been inserted after OBJECT and the value of Q-CONJ would be the sequence of elements: VERB, SA, OBJECT. A more complete description of the algorithm for generating conjunctional strings is to be found in earlier LSP papers [1, 5].

Two features of the parser make the dynamic generation of definitions possible. The first is an interrupt mechanism, which is activated when the parser reaches a word with a special conjunction flag in its definition. The definition specifies a node which is inserted into the parse tree after the last completed node. Expansion of this node triggers the execution of special restrictions which construct the conjunctional string definition. In this way the process of generating the conjunctional string definitions is made explicit in the grammar and can be easily changed by the grammar writer. Actual construction of the definition is made possible by a GENERATE command in the RL, which causes a new option with specified elements to be added to the definition of the current node.

Once a conjunctional string has been placed in the parse tree, it is efficient to cross-reference the two structurally similar elements on each side of the conjunction. This is done by a restriction which assigns node attributes. We assign to the pre-conjunction element (the first NVAR in Figure 5) a node attribute POSTCONJELEM whose value is a pointer to the structurally similar element occurring after the conjunction (the second NVAR in Figure 5). The node attribute PRECONJELEM is the counterpart assigned to the post-conjunction element; it points to the structurally similar pre-conjunction element. The automatic erasure feature of the node attributes assures that these pointers will be deleted when the conjunctional string is detached.

Conjunctional strings present a problem in the execution of restrictions. For example, contrast the sentences "John eats fish and cheese daily." with "John eats fish and calls daily." Clearly "calls" is intended here as a second verb whose subject is "John" and not as a conjunction of "fish" (similar to the status of "cheese" in Figure 5). The restriction which accepts "cheese" in Figure 5 but would reject "calls" as the object of "eats" is the same WSEL1 which rejected "each day" as the object of "eats" in the parse tree of Figure 4. In the parse tree of Figure 5, however, the restriction must recognize that the conjoined N is a second core-value of OBJECT, and must execute the tests $OBJECT-NOUN and $NO-COMMON twice, once with the CORE OF THE OBJECT equal to N ("fish") and once with it equal to the conjoined N ("cheese"/"calls").[4]

How can we arrange for the restrictions to be reexecuted in this fashion? One way, clearly, is to rewrite all the restrictions to test for conjunction, but this is an enormous task. What we would like, rather, is to allow the routines to be multivalued in the case of conjunction. This has been achieved in the parser by incorporating a nondeterministic programming mechanism [17]. If a routine such as CORE returns two values, the remainder of the restriction is first executed using the first value; if that succeeds, the portion of the restriction following the call on CORE is reexecuted using the second value. Only if both executions succeed is the entire restriction considered successful. Modifying the routines in this way greatly reduces the number of changes that have to be made to the restrictions for conjunctions.

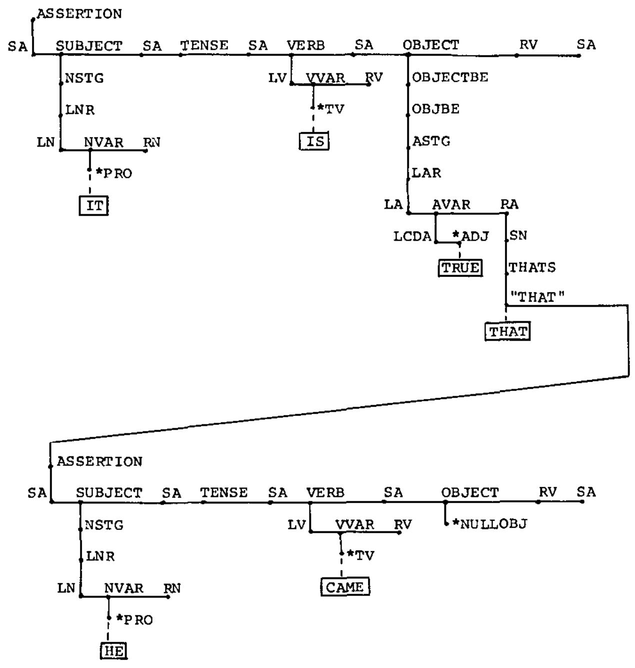
The nondeterministic programming mechanism is invoked through the STACK command. The principal value of a routine is simply the point in the tree located by the last statement in a routine. To return some alternate value as well, the routine has to execute the STACK command while positioned at that alternate node. In this way, the parser does not assume any particular structures for handling conjunctions, so the grammar writer retains the maximum possible flexibility.

[4] WSEL1 fails when the OBJECT is "cheese and calls" because "calls" is an NSENT1 ("the call for people to assemble"), and NSENT1 appears on the NOTNOBJ list of "eats."

Fig. 6. Another restriction from the English grammar. Note: A restriction statement marked GLOBAL is available for use in other restrictions. The global statement $NHUMAN referenced above, but not shown here, tests whether the node stored in X9 is or could be a "human" noun.

```
*DSN2:  AN SN STRING OR ASSERTION OCCURS AS THE RIGHT ADJUNCT OF
*       AN ADJECTIVE RA ONLY FOR CERTAIN SUBCLASSES OF ADJECTIVES
*       AND VING (IT IS TRUE THAT HE CAME, *IT IS ROUND THAT HE
*       CAME, IT IS SURPRISING THAT HE LEFT, *IT IS SURROUNDING
*       THAT HE LEFT; SHE IS ANXIOUS FOR YOU TO KNOW (ASENT3)).
*       IN ALL CASES BUT ASENT3 THE ULTIMATE SUBJECT MUST BE "IT".
*       FOR ASENT3 THE ULTIMATE SUBJECT MUST BE HUMAN.

DSN2 = IN RA RE SN, ASSERTION:  ALL OF $HOST, $NOCOMMA, $NOT-
       ADJINRN, $IT, $HUMAN ARE TRUE.

$HOST = HOST X1 IS ASENT1 OR ASENT3 OR VSENT1.

$NOT-ADJINRN = RA IS NOT OCCURRING IN ADJINRN.

$NOCOMMA = THE PRECEDING WORD IS NOT ",".

$IT = IF EITHER X1 IS VSENT1 OR BOTH X1 IS ASENT1 AND X1 IS NOT
      ASENT3, THEN $SUBJIT.

$SUBJIT = THE CORE X3 OF THE ULTIMATE-SUBJECT X10 IS "IT".  (GLOBAL)

$HUMAN = IF X1 IS ASENT3 THEN AT THE CORE X9 OF THE ULTIMATE-
         SUBJECT $NHUMAN [WPOS22] IS TRUE.
```

Fig. 7. ASSERTION parse tree for "It is true that he came."



Expressive Power of RL

The grammar writer gains two major advantages from using the Restriction Language and its associated routines. First, the grammar is relatively short and readable. This means it can be easily inspected and the interrelation of its parts kept in mind rather readily. The current LSP English grammar consists of less than 200 BNF definitions and about the same number of restrictions, together totaling about 2000 lines. The second gain is the ability to focus on the global logic of the linguistic operations without attending to the details of each computation, which are relegated to the level of the execution of routines. This means not only that the burden of writing a grammar is lighter but that the grammar itself can be richer. More complex linguistic operations can be undertaken than would otherwise be possible, simply because the tools for formulating and executing them are available.

This point would best be illustrated by tracing one of the longer and more intricate restrictions of the grammar, such as one governing relative clauses with embedding, or the agreement of subject and verb. Unfortunately, the long restrictions require too much explanation of the linguistic material. However, some impression of the expressive power of the RL can be gotten by considering an LSP grammar restriction of moderate length, the restriction DSN2 shown in Figure 6.

DSN2 is one of the restrictions concerned with the placing of sentential complements (called SN strings in the LSP grammar) of the type illustrated by "that he came," "for him to leave," "whether they left or not." In order for an occurrence of an SN string following a predicate adjective to be well formed ("It is true that he came"), various conditions must be satisfied, as stated in the text of DSN2. The first condition is stated in the restriction subpart $HOST. When executed, $HOST tests whether the adjectival element associated with the SN string is of the appropriate subclass (*"It is round that he came"). This adjectival element stands in the relation of *host* to the SN string in question. The execution of the HOST routine starting at the SN node locates the adjectival element which is to be tested. This can be illustrated by referring to the ASSERTION parse tree for the sentence "It is true that he came", shown in Figure 7.

In the upper right portion of Figure 7, several levels below the OBJECT node of ASSERTION is the LXR-type node LAR, standing for an adjective (or other adjectival element, such as VING or a compound adjective) with its left and right adjuncts. The adjectival element in this case is the terminal symbol ADJ ("true") and the value of the right adjunct (RA) node is SN.[5] Given the three-element standard form for all LXR type definitions, the routine HOST goes from any node in either the rightmost or leftmost subtree of an LXR configuration to the core of the center element. Thus, its execution starting from SN ends at ADJ, the correct element to be tested.

The test $NOT-ADJINRN rules out SN occurrences following an adjective which is occurring as a right adjunct of a noun. Thus, it would be wellformed to have an adjective in this position followed by adjuncts other than SN ("The implication, clear to all, was that he was present") but not by an SN string (*The implication, clear that he was present, was understood by all). In the BNF part of the English grammar, there is a definition ADJINRN for the occurrence of an adjective with its optional adjuncts (LAR) as a right adjunct of a noun. Hence, in order to verify that an SN in RA is not occurring in this position, the test $NOT-ADJINRN employs the predicate IS NOT OCCURRING IN ADJINRN. This predicate looks upward in the parse tree for ADJINRN (but not above the first STRING type node); if an ADJINRN is found the restriction fails.

The test $NOCOMMA rules out, e.g. *"It is true, that he came," in which a comma intervenes between the predicate adjective and the complement. The execution of THE PRECEDING WORD locates the sentence word last attached to the parse tree. DSN2 is executed when the option SN of RA is about to be attached, so in "It is true, that he came" THE PRECEDING WORD is ",".

The remaining tests of DSN2, $IT, $SUBJIT, and $HUMAN apply constraints to the subject, depending on which subclass of adjective or VING is present in the predicate adjective position. The comments preceding DSN2 in Figure 6 give some indication as to these subclasses, and detailed definitions of all adjective, noun, and verb subclasses in the LSP grammar are found in [18]. The most interesting feature in the test $SUBJIT is the use of the routine ULTIMATE-SUBJECT. By means of this routine, the same restriction DSN2 which checks that the subject is "it" in "It is true that he came," also checks for "it" in "It seems to be true that he came," in "It seems to be considered to have appeared probable that he came," etc., to an arbitrary depth of embedding. There are several extended scope routines of this type in the grammar. Being of the same order of complexity as some of the longer restrictions, their formulation is also considerably facilitated by writing them in the Restriction Language.

Conclusion

Five years ago, after several years of development, the Linguistic String Project possessed a grammar capable of analyzing a wide spectrum of English sentences. In the course of this development, the restrictions constituting the bulk of the grammar gradually grew into a mass of nonperspicuous list-structure code; further refinement of the code became increasingly difficult.

[5] SN strings are not modifiers of the elements they adjoin, but it is convenient to analyze them as occurring in adjunct position vis à vis these elements since they are regularly associated with the occurrence of particular subclasses in the element position. It is also to be noted that the subject "it" in "It is true that he came" appears in the parse tree as a PRO, though it is not functioning as a pronoun here.

The response to this problem was the language we have described in this paper.

The unusual features of the Restriction Language are a consequence of the special requirements of our application. The basic nature of the application—stating restrictions on parse trees—is reflected in the declarative syntax and the repertoire of basic operations. The special requirements of natural language grammar and the need to interact with the context-free parser have led to a number of features: switches which define grammar subsets, node attributes with automatic erasure, interrupts, dynamically generated definitions, nondeterministic programming, and a set of routines corresponding to the fundamental grammatical relations. As we noted in earlier sections, these features were responses to specific problems that we faced in working with a large string grammar in a text processing situation. In designing the Restriction Language, however, we have tried to separate as much as possible what is general to the problem of treating natural language from what specifically relates to our theoretical linguistic framework. The latter appears explicitly in the definitions of the routines as part of the grammar.

The next step in sentence analysis is transformational decomposition—the reduction of a sentence to a set of kernel sentences connected by transformations. We are starting this decomposition process from the linguistic string analyses currently produced by our program. Initial work has been done on specifying the inverse transformations and the conditions under which they should be applied. This work has shown that the Restriction Language is well suited to stating the inverse transformations themselves. The parsing program has recently been expanded to handle these transformations, and the transformations themselves are being gradually assembled.

Received February 1974; revised November 1974

References

1. Sager, N. Syntactic analysis of natural language. In *Advances in Computers*, No. 8, F. Alt and M. Rubinoff (Eds.), Academic Press, New York, 1967.
2. *String Program Reports*, Nos. 1–5, Linguistic String Project, New York U., 1965–1969.
3. Sager, N. The string parser for scientific literature. In *Natural Language Processing*, R. Rustin (Ed.), Algorithmics Press, New York, 1973.
4. Grishman, R. Implementation of the string parser of English. In *Natural Language Processing*, R. Rustin (Ed.), Algorithmics Press, New York, 1973.
5. Grishman, R., Sager, N., Raze, C., and Bookchin, B. The linguistic string parser. Proc. NCC, AFIPS Press, Montvale, N.J., 1973.
6. Harris, Z.S. Linguistic transformations for information retrieval. Proc. Conf. on Scientific Information, 1958, 2, NAS-NRC, Washington, D.C., 1959.
7. Harris, Z.S. *String Analysis of Sentence Structure*. Mouton and Co., The Hague, 1962.
8. Rhodes, Ida. A new approach to the mechanical syntactic analysis of Russian. Mechanical Translation Group, U.S. Dep. of Commerce, Applied Math. Div., Nat. Bur. Std. Rep. 6595, 1959.
9. Alt, F.L., and Rhodes, Ida. The hindsight technique in machine translation of natural languages. *J. Res. Bur. Std. B66*, 2 (1962), 47–51.
10. Kuno, S., and Oettinger, A.G. Multiple-path syntactic analyzer. In *Information Processing 1962*, North-Holland Pub. Co., Amsterdam, 1963, pp. 306–312.
11. Winograd, T. Procedures as a representation for data in a computer program for understanding natural language. MAC TR-84, MIT Proj. MAC, Cambridge, Mass., 1971.
12. Woods, W.A. Transition network grammars for natural language analysis. *Comm. ACM 13*, 10 (Oct. 1970), 591-606.
13. Sager, N., Morris, J., and Salkoff, M. First report on the string analysis programs. Dep. of Linguistics, U. of Pennsylvania, 1965. Expanded and reissued as String Program Rep. No. 1, 1966.
14. Sager, N. A computer string grammar of English. String Program Rep. No. 4, 1968.
15. Sager, N. *A Formal Grammar of English and Its Computer Applications.* (To be published by Gordon and Breach in the series Mathematics and Its Applications.)
16. Sager, N. A two-stage BNF specification of natural language. *J. of Cybernetics 2*, 39 (1972).
17. Raze, Carol. A computational treatment of coordinate conjunctions. 12th Ann. Meet. of Assoc. of Computat. Ling., Amherst, Mass., July 26, 1974. (To appear.)
18. Fitzpatrick, Eileen, and Sager, N. The lexical subclasses of the linguistic string parser. *Amer. J. of Computat. Ling.*, Microfiche 2 (1974). (Also available as String Program Rep. No. 9.)