LSP-9

The linguistic string parser*

by R. GRISHMAN, N. SAGER, C. RAZE, and B. BOOKCHIN

New York University
New York, New York

The linguistic string parser is a system for the syntactic analysis of English scientific text. This system, now in its third version, has been developed over the past 8 years by the Linguistic String Project of New York University. The structure of the system can be traced to an algorithm for natural language parsing described in 1960.[1] This algorithm was designed to overcome certain limitations of the first parsing program for English, which ran on the UNIVAC 1 at the University of Pennsylvania in 1959.[2] The UNIVAC program obtained one "preferred" grammatical reading for each sentence; the parsing program and the grammar were not separate components in the overall system. The 1960 algorithm obtained all valid parses of a sentence; it was syntax-driven by a grammar consisting of elementary linguistic strings and restrictions on the strings (described below). Successive implementations were made in 1965,[3] in 1967,[4] and in 1971.[5] The system contains the largest-coverage grammar of English among implemented natural language parsers.

Implementation of a large grammar by several people over a period of years raises the same problems of complexity and scale which affect large programming projects. The main thrust in the development of the current version of the parser has been to use modern programming techniques, ranging from higher-level languages and subroutine structures to syntax-directed translation and non-deterministic programming, in order to structure and simplify the task of the grammar writer. In this paper we shall briefly review the linguistic basis of the parser and describe the principal features of the current implementation. We shall then consider one particularly thorny problem of computational linguistics, that of conjunctions, and indicate how various features of the parser have simplified our approach to the problem. Readers are referred to an earlier report[6] for descriptions of unusual aspects of the parser incorporated into earlier versions of the system.

Our approach to the recognition of the structure of natural language sentences is based on linguistic string theory. This theory sets forth, in terms of particular syntactic categories (noun, tensed verb, etc.) a set of elementary strings and rules for combining the elementary strings to form sentence strings.

The simplest sentences consist of just one elementary string, called a *center string*. Examples of center strings are *noun tensed-verb*, such as "Tapes stretch." and *noun tensed-verb noun*, such as "Users cause problems." Any sentence string may be made into a more complicated sentence string by inserting an *adjunct string* to the left or right of an element of some elementary string of the sentence. For example, "Programmers at our installation write useless code." is built up by adjoining "at our installation" to the right of "programmers" and "useless" to the left of "code" in the center string "programmers write code." Sentences may also be augmented by the insertion of a *conjunct string*, such as "and debug" in "Programmers at our installation write and debug useless code." Finally, string theory allows an element of a string to be replaced by a *replacement string*. One example of this is the replacement of *noun* by *what noun tensed-verb* to form the sentence "What linguists do is baffling."

The status of string analysis in linguistic theory, its empirical basis and its relation to constituent analysis on the one hand and transformational analysis on the other, have been discussed by Harris.[7] More recently, Joshi and his coworkers have developed a formal system of grammars, called string adjunct grammars, which show formally the relation between the linguistic string structure and the transformational structure of sentences.[8] The string parser adds to linguistic string theory a computational form for the basic relations of string grammar. In terms of these relations the arguments of grammatical constraints (i.e., mutually constrained sentence words) can always be identified in the sentence regardless of the distance or the complexity of the relation which the words have to each other in that sentence.[9]

Each word of the language is assigned one or more word categories on the basis of its grammatical properties. For example, "stretches" would be classed as a *tensed verb* and a *noun*, while "tape" would be assigned the three categories *tensed verb*, *untensed verb*, and *noun*. Every sequence of words is thereby associated with one or more sequences of word categories. Linguistic string theory claims that each sentence of the language has at least one sequence of word categories which is a sentence string,

* The work reported here was supported in part by research grants from the National Science Foundation: GN559 and GN659 in the Office of Science Information Services, and GS2462 and GS27925 in the Division of Social Sciences.

i.e., which can be built up from a center string by adjunction, conjunction, and replacement.

However, not every combination of words drawn from the appropriate categories and inserted into a sentence string forms a valid sentence. Sometimes only words with related grammatical properties are acceptable in the same string, or in adjoined strings. For example, one of the sequences of word categories associated with "Tape stretch." is *noun tensed-verb*, which is a sentence string; this sentence is ungrammatical, however, because a singular *noun* has been combined with a plural *tensed-verb*. To record these properties, we add the subcategory (or attribute) singular to the category *noun* in the definition of "tape" and the subcategory plural to the category *tensed-verb* in the definition of "stretch." We then incorporate into the grammar a restriction on the center string *noun tensed-verb*, to check for number agreement between noun and verb.

The number of such restrictions required for a grammar of English is quite large. (The current grammar has about 250 restrictions.) However, the structural relationship between the elements being compared by a restriction is almost always one of a few standard types. Either the restriction operates between two elements of an elementary string, or between an element of an elementary string and an element of an adjunct string adjoining the first string or a replacement string inserted into the first string, or (less often) between elements of two adjunct strings adjoined to the same elementary string. This property is an important benefit of the use of linguistic string analysis; it simplifies the design of the restrictions and plays an important role in the organization of the grammar, as will be described later.

IMPLEMENTATION

As the preceding discussion indicates, the string grammar has three components: (1) a set of elementary strings together with rules for combining them to form sentence strings, (2) a set of restrictions on those strings, and (3) a word dictionary, listing the categories and subcategories of each word. Component 1. defines a context-free language and, for purposes of parsing, we have chosen to rewrite it as a BNF grammar.

The approximately 200 BNF definitions in our grammar can be divided into three groups. About 100 of these are single-option *string* definitions; each of these corresponds to one (or occasionally several) strings. For example,

ASSERTION: : = <SA><SUBJECT><SA><VERB>
<SA><OBJECT><RV><SA>

contains the required elements SUBJECT, VERB, and OBJECT (corresponding to the three elements in such center strings as *noun tensed-verb noun* and *noun tensed-verb adjective*) and the optional elements SA, indicating where an adjunct of the entire sentence may occur, and RV, for right adjuncts of the verb appearing after the object. SA and RV are two of the approximately 20 adjunct set definitions; these definitions group sets of strings which may adjoin particular elements. The remaining definitions, including those for SUBJECT, VERB, and OBJECT, are collections of positional variants; these define the possible values of the elements of string definitions.

Once component 1. has been rewritten in this way, it is possible to use any context-free parser as the core of the analysis algorithm. We have employed a top-down serial parser with automatic backup which builds a parse tree of a sentence being analyzed and, if the sentence is ambiguous, generates the different parse trees sequentially.

The parse tree for a very simple sentence is shown in Figure 1. A few things are worth noting about this parse tree. Most striking is the unusual appearance of the parse tree, as if it had grown up under a steady west wind. We have adopted the convention of connecting the first daughter node to its parent by a vertical line, and connecting the other daughter nodes to the first by a horizontal line. This is really the natural convention for a string grammar, since it emphasizes the connection between the elements of a string definition. More interesting is the regular appearance of "LXR" definitions: a <LNR> below the subject, a <LTVR> below the verb, and a <LAR> below the object. Each LXR has three elements: one for left adjuncts, one for right adjuncts, and one in the middle for the *core* word. The *core* of an element of a definition is the word category corresponding to this element in the associated elementary string in the sentence; e.g. the core of SUBJECT (and of LXR) in Figure 1 is the noun "trees"; it is the one terminal node below the element in question which is not an adjunct. In some cases the core of an element is itself a string. LXR definitions and linguistic string definitions play a distinguished role in conjoining, as will be described later.

Each restriction in the grammar is translated into a sequence of operations to move about the parse tree and test various properties, including the subcategories of words attached to the parse tree. When a portion of the parse tree containing some restrictions has been completed, the parser invokes a "restriction interpreter" to execute those restrictions. If the restriction interpreter returns a success signal, the parser proceeds as if nothing

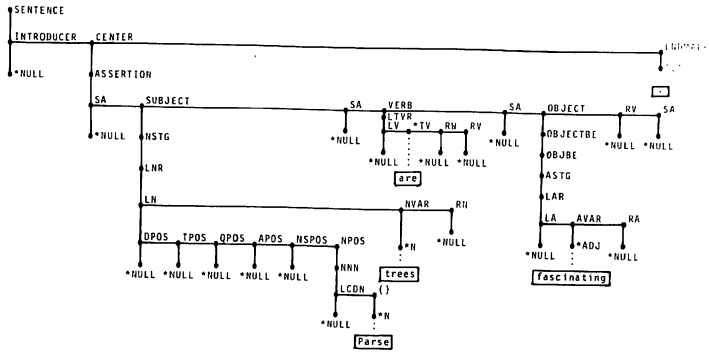

Figure 1—Parse tree for "Parse trees are fascinating"

had happened; if it returns a failure signal, the parser backs up and tries to find some alternate analysis.

The first version of the system was written in the list-processing language IPL-V for the IBM 7094. Because IPL-V was an interpretive system rather than a compiler, this implementation proved to be too slow for parsing a large corpus of sentences, and it was replaced by a system written in assembly language for the IBM 7094 (FAP). The speed of these systems was considerably enhanced by a mechanism which saved the subtrees below certain specified nodes the first time they were constructed, so that they would not have to be repeatedly built up. If restrictions on the saved subtrees had accessed nodes outside the subtree, these restrictions were re-executed when the subtree was re-used. With this saving mechanism the second version of the parser was able to obtain a first parse in a few seconds and all parses in under a minute for most sentences.

In both systems, the entire grammar (context-free component, restrictions, and word dictionary) was encoded in a uniform list-structure format. This format was easy to convert to internal list structure, but, particularly for encoding the restrictions, it left something to be desired with regard to perspicuity and brevity of expression. As the grammar was gradually refined and expanded, and especially when the restrictions were modified to handle conjunctions, these deficiencies increasingly burdened the grammar writer.

Therefore, when work was begun on a new version for the CDC 6600 in 1969 we set as our goal, in addition to the creation of a relatively machine-independent FORTRAN implementation, the development of a higher-level language suitable for writing restrictions. Because we realized that the language specifications would evolve gradually as new features were added to the system, we decided to use a syntax-directed compiler in translating the language into the internal list structure required by the program. This decision actually simplified the design of the overall system, since several modules, including the basic parser, could be used in both the compiler and the analyzer of English.

The restriction language we have developed looks like a subset of English but has a fairly simple syntax. The basic statement form is subject-predicate, for example

THE OBJECT IS NOT EMPTY.

This hypothetical restriction might be "housed" in ASSERTION (whose definition is given above); that is, it would be executed when the parse tree below ASSERTION was completed, and it would begin its travels through the parse tree at the node ASSERTION. The restriction looks for an element of ASSERTION named OBJECT (that is, it searches the level in the tree below ASSERTION for the node OBJECT). When it finds it, it verifies that the node is not empty, i.e., subsumes at least one word of the sentence.

Nodes which are not elements of the current string (the string in which the restriction is housed) can be referenced by using one or more of a set of "tree-climbing" routines. These routines correspond to the basic relations of the string grammar, such as CORE, LEFT ADJUNCT, RIGHT ADJUNCT, HOST (which goes from an adjunct to the element it adjoins), and COELEMENT (which goes from one element of a string definition to another). For example, a restriction sentence starting at ASSERTION,

THE CORE OF THE SUBJECT IS NHUMAN.

would begin by looking for the element SUBJECT. It would then invoke the routine CORE to descend to the core of the SUBJECT and test whether the sentence word corresponding to that node has the attribute NHUMAN.*

Because all grammatical restrictions test elements bearing one of only a few structural relationships to the current string (as described above), it is possible to formulate all the restrictions in terms of a small set of about 25 such "locating routines." The routines are coded in terms of the basic tree operations, such as going up, down, left, and right in the parse tree, and other operations, such as testing the name of the node. The basic tree operations are not used directly by the restrictions. The routines correspond roughly to the low-level assembly language routines in a large programming system. The routines not only simplify the task of writing the restrictions, but also permit fundamental modifications to be made to the grammar by changing the routines rather than having to individually change each of the restrictions. One example of this, the modification for conjunctions, will be described later.

The restriction language contains the full range of logical connections required for combining basic subject-predicate sentences into larger restrictions: BOTH____ AND____, EITHER____OR____, NOT____, IF____ THEN____, etc. For example, a very simple restriction for subject-verb agreement in number is

IF THE CORE OF THE VERB IS SINGULAR
THEN THE CORE OF THE SUBJECT IS NOT PLURAL.

Registers (i.e., variables) of the form Xn, n an integer, may be used to temporarily save points in the tree. For example, in

BOTH THE CORE X1 OF LNR IS NAME OR NSYMBOL
AND IN THE LEFT-ADJUNCT OF X1, NPOS IS NOT EMPTY.

$X1$ is used to save the point in the tree corresponding to "CORE OF LNR" so that it need not be recomputed for the second half of the restriction.

* Informally speaking, the noun subclass NHUMAN refers to *human* nouns. Grammatically it is any word which can be adjoined by a relative clause beginning with "who."

The syntax-directed compiler has the job of translating the restriction language statements into lists composed of the basic operations recognized by the restriction interpreter. For instance, the restriction sentence given above, "THE CORE OF THE SUBJECT IS NHUMAN." would be compiled into*

```
(EXECUTE [(STARTAT [(SUBJECT)])],
EXECUTE [(CORE)],
ATTRB [(NHUMAN)])
```

The EXECUTE operator is used to invoke routines. The first operation, a call on the routine STARTAT with argument SUBJECT, searches the level below ASSERTION for the node SUBJECT. This is followed by a call on the routine CORE and then by the operation ATTRB, which checks if the word associated with the node reached has the attribute NHUMAN.

The restriction language syntax (RLS) which guides the compilation process is a set of BNF productions into which have been incorporated directives for generating the list structures. Each directive is a list of code (list structure) generators which are to be executed when the production is used in the analysis of the restriction language statement.

Our first version of the RLS followed a format described by Cocke and Schwartz.[10] The generators to be invoked were simply listed between the elements of the BNF definitions; the parser called on these generators during the parsing of the statement. This arrangement is described in detail in Reference 5. It is quite efficient but has several drawbacks. First, the parser cannot back up (since it cannot undo the action of a generator) so the RLS must be written to allow analysis in a single left-to-right pass. Second, if the list structure to be generated is at all complicated, the task of creating the generator sequences is error-prone and the resulting BNF productions do not clearly indicate what code will be generated.

We have circumvented the first problem by first parsing the statement and then scanning the parse tree and executing the generators. We have attacked the second problem by allowing the user to write a list structure expression as part of each production and having the system compile this to the appropriate sequence of generator calls. In our new version of the compiler, after the restriction statement is parsed its parse tree is scanned from the bottom up. At each node, the sequence of generators in the corresponding production of the compiled RLS is executed. These generators should re-create the list structure which the user wrote as part of the (source) RLS production. This list structure is then assigned as the "value" of that node; generators on the node one level up may use this value in building a larger list structure. In this way, the small list structures at the bottom of the tree are gradually combined into larger structures. The structure assigned to the root node of the tree is the generated code for the statement; this code is written out as part of the compiled grammar of English. This procedure is similar to the compiler described by Ingermann[11] and the formalism of Lewis and Stearns.[12]

A simple example of an RLS statement is

```
<REGST>: : = <*REG>
                →(STORE[<REG>]) | <*NULL>.
```

This says that the symbol REGST may be either a token of lexical type REG (a register, *Xn*) or the null string. If REGST matches a register, the code which will be generated and assigned to REGST is the operation STORE with an argument equal to the name of the register matched. If the null option is used in the parse tree, no code is generated.

As a more complicated example we consider two options of NSUBJ, the subject of a restriction statement:

```
<NSUBJ>: : = <*NODE>
          →(EXECUTE[(STARTAT[(<NODE>)])] |
                    CORE<REGST>OF<NSUBJ>
→<NSUBJ>:(EXECUTE[(CORE)]):<REGST> | · · · .
```

The first type of subject is simply the name of a node of the tree; the generated code is a call on the routine STARTAT with that node as argument. The second type is CORE OF followed by some other valid subject, with a register name following the word CORE if the position of the core is to be saved. The generated code here is the concatenation of three list structures (":" indicates concatenation). The first of these is the code generated to locate the NSUBJ following CORE OF; the second is a call on the routine CORE; the third is the code generated to save the present position in a register (this last structure will be empty if no register is present).

To illustrate the combined effect of these RLS statements, we present, in somewhat simplified form, the parse tree for our restriction sentence, "THE CORE OF THE SUBJECT IS NHUMAN." in Figure 2. To the left of each node appears its name; to the right, the list structure assigned to that node by the generating process.

To compile the RLS into lists of syntactic categories and generator calls in the proper internal format, we require one further application of the syntax-directed compiler. This makes for a total of three uses of the parser, two as a compiler and one as an analyzer of English. The entire process which results finally in parses of Eng-

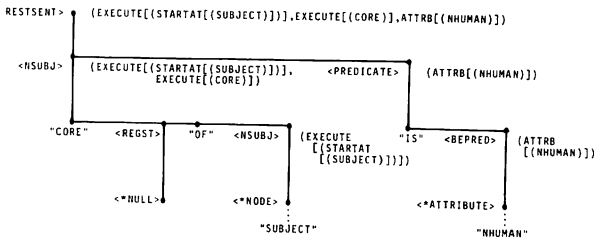

Figure 2—Parse tree for CORE OF SUBJECT IS NHUMAN

* In our list notation, square brackets enclose arguments of operators and routines.

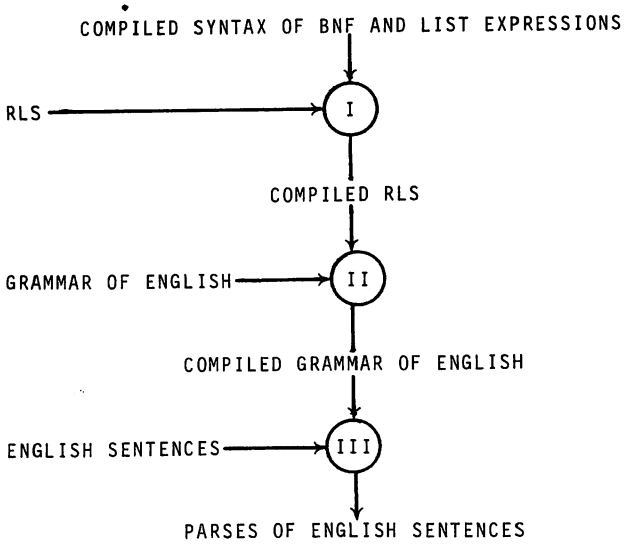

Figure 3—The three uses of the parser

lish sentences is diagrammed in Figure 3. To avoid an infinite regression, the first grammar, the compiled syntax of BNF and list expressions, must be produced by hand; fortunately, it is quite short.

CONJUNCTIONS

The problem in computing conjunctions is threefold. (1) To provide for the richness of conjunctional constructions in full-bodied English texts. This is best handled by locally conjoining the syntactic categories as they appear in the sentence (*noun* and *noun, tensed-verb* and *tensed-verb*, etc.) (2) To recover hidden or "zeroed" material from the sentence so that semantic and selectional constraints can be applied to all relevant parts of the sentence. This implies an expansion of the sentence. For example, the sentence

(a) They program and debug systems

has the expansion

(a′) They program (systems) and (they) debug systems.

(3) To meet our first two objectives without greatly complicating and enlarging the grammar. This necessitated an addition to the parser and modifications to the restriction language routines.

According to linguistic string theory an element or sequence of elements of a string may be conjoined in the following manner: If the successive syntactic elements of the string are $E_1E_2 \cdots E_i \cdots E_n$ then conjunction may occur after E_i, and the conjoined sequence consists of a conjunction followed by E_i or $E_{i-1}E_i$ or $\cdots$ or $E_iE_2 \cdots E_i$. In sentence (a), above, the syntactic categories are *pronoun tensed-verb conjunction tensed-verb noun.* In the sentence

(b) They program and we debug systems.

the elements are *pronoun tensed-verb conjunction pronoun tensed-verb noun.* In addition, if E_i is a positional variant (such as OBJECT) representing a set of alternative values for a particular position in the string, one value of E_i may be conjoined to another. In the sentence

(c) I don't like him and what he stands for.

two different values of object are conjoined, namely *pronoun* and the *N*-replacement string *what* ASSERTION.*

To include the conjunctional definitions explicitly in the BNF grammar would cause the grammar to increase enormously. Instead conjunctional strings are inserted dynamically by a special process mechanism when a conjunction is encountered in parsing the sentence. This is equivalent in effect to having all possible conjunctional definitions in the grammar before parsing begins.

The special process mechanism interrupts the parser when an element of a definition has been completely satisfied and the current sentence word to be analyzed is a conjunction. An interrupt results in the insertion in the tree of a special process node. For each special word there is a corresponding definition in the grammar. This definition consists of the conjunctional word and a string which defines the general feature of the type of special process to be computed (conjunctional, comparative). For example, <SP-AND>=AND<Q-CONJ>. The general conjunctional string Q-CONJ contains a restriction which generates a definition for Q-CONJ depending on where the interrupt takes place. If it occurs after E_i the following definition will be generated for Q-CONJ: $<E_i>|<E_{i-1}><E_i>|\cdots\cdots|<E_1><E_2>\cdots\cdots<E_i>$. Consider the sentence

(d) He and she like computers.

A tree of the analysis of the sentence is shown in Figure 4. After *he* has been matched as a pronoun an interrupt occurs and SP-AND is inserted after NVAR in LNR.

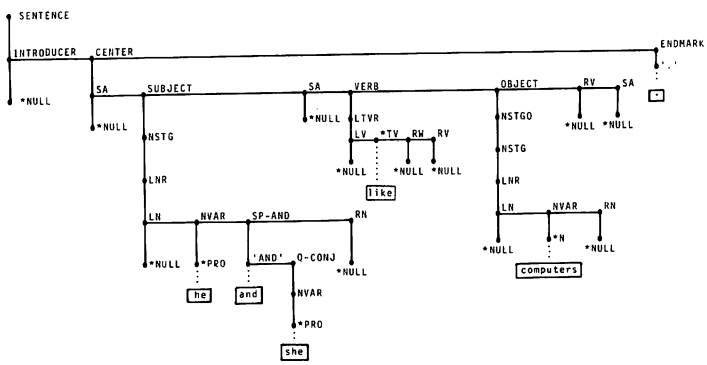

Figure 4—Parse tree for "He and she like computers"

* The ASSERTION string after words like "what," "who," etc., is expected to occur with one noun position empty, here the object of "stands for."

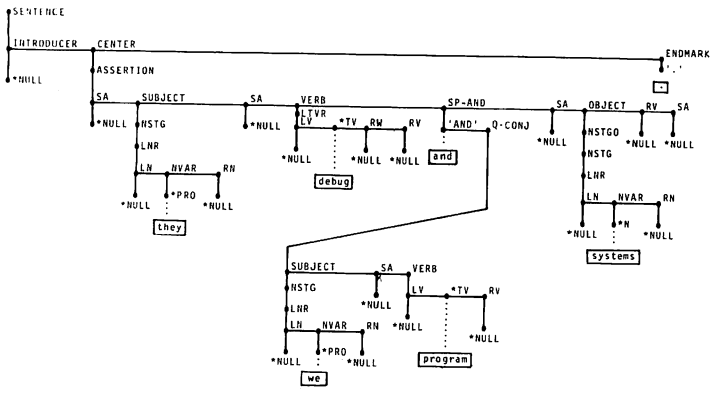

Figure 5—Parse tree for "They debug and we program systems."

Local conjoining (i.e. conjoining within a single string-element position such as SUBJECT) will fail however for

(e) They debug and we program systems.

The tree for this sentence is shown in Figure 5. Here, the conjunctional string covers two string-element positions (SUBJECT, VERB) and must be conjoined at a higher level in the analysis tree.

When local conjoining fails, the special process node is detached and normal parsing continues. But an interrupt will occur again if the conjunctional word has not alternatively been analyzed as a non-special word (such as "but" in "I will be but a minute."). A second interrupt occurs when the next element satisfied is NULL or the parse moves to a higher level of the analysis. For example, in sentence (e), "and" is still current when the VERB position of ASSERTION is completed. Therefore an interrupt will occur again at this higher level and the proper conjunctional string will be inserted. After the special process node is completed normal parsing resumes.

If the sentence contains a special process scope marker such as *both* as part of a *both . . . and* sequence an additional check is made in the restriction that defines Q-CONJ. It removes any option that contains an element whose corresponding element in the pre-conjunctional string precedes the scope marker. Thus we accept "He both debugs and programs systems," but not "He both debugs and we program systems."

Although the special process mechanism can interrupt the parser at many different points in the analysis tree, the regularity of conjunctional strings with respect to the rest of the string grammar enables us to confine the conjunctional interrupt process to just two types of definitions: to host-adjunct sequences (i.e., after each element of LXR type definitions) and to linguistic strings (i.e., after each required element in definitions of the type "linguistic string"). In this way numerous redundancies caused by conjoining at intermediate nodes are eliminated. In addition, confining the conjoining to these two types of definitions simplifies the modification to the restriction language routines which is needed for conjunctions.

Since restrictions are the key to obtaining correct parses, it is essential that they operate on conjunctional strings as well as on the ordinary strings of the grammar. However, conjunctional strings very often contain only a portion of the sequence to which a restriction applies, necessitating corrections to each restriction of the grammar to take account of conjunctional strings, or else a general solution to the problem of truncated conjunctional strings. The general solution employed by the current string parser is to modify, not each restriction, but the small number of basic routines used by all restrictions.*

Our solution is based on the following: If a restriction requires a special condition for element E_i of a string S or of a host-adjunct sequence, then that condition should also be true for E_i in the segment conjoined to S. Similarly, if a restriction requires a wellformedness condition between two elements E_i and E_j of S (or between host and adjunct), then the same condition should be true for E_i and E_j in the conjoined segment. If one of the elements E_i is not present in the conjoined segment, the restriction applies between E_j in the conjoined segment and E_i in S.

Certain of the basic restriction routines were modified in accord with the above, by introducing a "stack" operation. If in the execution of a restriction a routine is called to locate an element and this element has conjoined elements then the stack operation is executed for each of the conjoined elements. If the restriction is successful, it is resumed at the point in the restriction immediately following the routine which executed the stack operation. But when the restriction is resumed, it is looking not at the original element located by the routine but rather at its conjoined element. The restriction is successful only if it succeeds for all the conjoined elements.

Consider the operation of the selectional restriction WSEL2. This restriction states: If the core of the subject is a noun or pronoun and if the core C of the coelement verb (of the subject) has a subclassification NOTNSUBJ which consists of a list of forbidden noun or pronoun subclasses for the given verb, then the sentence word corresponding to C should not have any of those subclassifications. The verb *occurs* forbids as its subject a noun having a human subclassification. Thus the sequence *programmers occur* is not well formed whereas *problems occur* is. For WSEL2 to test the core of the subject position the routine CORE is called. In the sentence *Problems and difficulties occurred later* the CORE routine will stack *difficulties* so that the wellformedness of both *problems occurred* and *difficulties occurred* will be tested. WSEL2 will therefore fail for the sequence *Problems and programmers occurred later.* In the sentence *Difficulties and problems occurred but later disappeared* WSEL2 will be executed four times (the core value of the verb position is conjoined also) testing the wellformedness of the sequences *difficulties occurred, problems occurred, diffi-*

* In the previous implementations the restrictions were individually modified so as to operate correctly under conjunctions. This demanding task was carried out by Morris Salkoff.

culties disappeared, problems disappeared. In this way stacking has the same effect as expanding the sentence. There are some restrictions, however, such as the ones testing number agreement, which cannot use stacking and which must be changed to test specifically for conjoining. Therefore each routine that stacks has a nonstacking counterpart which these restrictions use.

Stacking is sufficient for the bulk of conjunctional occurrences, those whose form is described above. However, there are other types. In

(f) He debugged the parser and she the grammar.

there is zeroing in the middle position (i.e., in the verb position between the subject and object positions) of the assertion beginning with "she." In

(g) He heard of and liked the new system.

the noun element following *of* in the prepositional string has been zeroed. It is possible and highly desirable for the zeroed element to be filled in during the analysis. The program has a mechanism for these cases which is activated by a restriction routine. It temporarily delays the execution of all those restrictions that may refer to the zeroed element. Further on in the analysis the zeroed slot is filled in by linking it to the subtree corresponding to another part of the sentence. For example in sentence (g), above, the zeroed slot for the noun-sequence after the preposition "of" will be linked to the subtree corresponding to *the new system* so that in effect the zeroed information has been restored. The sentence thus becomes

(g′) He heard of (the new system) and he liked the new system.

After linking occurs, the delayed restrictions are executed. From that point on any restriction that refers to the zeroed element will automatically be switched to its linked subtree. While backing up, a restriction may obtain an alternative link to another subtree. In this way all analyses via different subtree links are arrived at. In "We chose and started to implement another algorithm," one analysis is "We chose (another algorithm) and started to implement another algorithm." Another analysis is "We chose (to implement another algorithm) and started to implement another algorithm."

THE USE OF THE STRING PARSER

A practical goal for the parser is to aid in the processing of scientific information. It is conceivable, for example, that answers to highly specific informational queries could be extracted from large stores of scientific literature with the aid of natural language processing techniques. Such an application requires that there be a close correlation between the computer outputs for a text and the information carried by the text.

An examination of the string output parses for texts in various fields of science[13] shows that the decomposition of a sentence into its component elementary strings constitutes a first breakdown of the sentence into its elementary informational components. The correlation would be much improved if we could (1) reduce the redundancy of grammatical forms (redundant from the point of view of the information carried by the sentence), i.e., arrive at a single grammatical form for sentences or sentence parts carrying the same information; (2) reduce ambiguity, i.e., arrive at the author's intended reading from among the syntactically valid analyses produced by the parser.

Fortunately, methods are available for attacking both problems. Transformational refinement of the grammar leads precisely to determining a single underlying sentence in semantically related forms, such as the active and passive, and numerous nominalization strings, e.g. "We should reduce ambiguity," "ambiguity should be reduced," "the reducing of ambiguity," "the reduction of ambiguity," etc.

With regard to syntactic ambiguity, the largest number of cases have their source in different possible groupings of the same string components of the sentence, the decisive criterion being which of the resulting word-associations is the correct one for the given area of discourse. For example, in "changes in cells produced by digitalis," only one of the possible groupings (that in which digitalis produces changes, not cells) is correct within a pharmacology text dealing with cellular effects of digitalis. Recently it has been shown that it is possible to incorporate into the grammar which is used to analyze texts in a given science subfield additional grammatical constraints governing the wellformedness of certain word combinations when they are used within that subfield.[14] These constraints have the force of grammatical rules for discourse within the subfield (not for English as a whole), and have a very strong semantic effect in further informationally structuring the language material in the subfield, and in pointing to the correct word associations in syntactically ambiguous sentences.

REFERENCES

1. Sager, N., "Procedure for Left-to-Right Recognition of Sentence Structure," *Transformations and Discourse Analysis Papers*, No. 27, Department of Linguistics, University of Pennsylvania, 1960.
2. Harris, Z. S., et al., *Transformations and Discourse Analysis Papers*, Nos. 15-19, University of Pennsylvania, Department of Linguistics, 1959.
3. Sager, N., Morris, J., Salkoff, M., *First Report on the String Analysis Programs*, Department of Linguistics, University of Pennsylvania, 1965. Expanded and reissued as *String Program Reports* (henceforth SPR), No. 1, Linguistic String Project, New York University, 1966.
4. Raze, C., "The FAP Program for String Decomposition of Sentences," *SPR*, No. 2, 1967.
5. Grishman, R., "Implementation of the String Parser of English," Presented at the *Eighth Courant Computer Science Symposium*, December 20, 1971. To be published in *Natural Language Processing*, Rustin, R., ed., Algorithmics Press, New York (in press).

6. Sager, N., "Syntactic Analysis of Natural Language," *Advances in Computers*, No. 8, Alt, F., and Rubinoff, M., (eds), Academic Press, New York, 1967.
7. Harris, Z. S., *String Analysis of Sentence Structure*, Mouton and Co., The Hague, 1962.
8. Joshi, A. K., Kosaraju, S. R., Yamada, H., "String Adjunct Grammars," Parts I and II, *Information and Control*, No. 21, September 2, 1972, pp. 93-116 and No. 3, October 3, 1972, pp. 235-260.
9. Sager, N., "A Two-Stage BNF Specification of Natural Language," *Information Sciences*, (in press).
10. Cocke, J., Schwartz, J., *Programming Languages and Their Compilers*, Courant Institute of Mathematical Science, New York University, 1969.
11. Ingerman, P. Z., *A Syntax-Oriented Translator*, Academic Press, New York, 1966.
12. Lewis, P. M., II, Stearns, R. E., "Syntax-Directed Transduction" *JACM*, No. 15, p. 465, 1968.
13. Bookchin, B., "Computer Outputs for Sentence Decomposition of Scientific Texts," *SPR*, No. 3, 1968.
14. Sager, N., "Syntactic Formatting of Science Information," *Proceedings of the 1972 Fall Joint Computer Conference*, pp. 791-800.