American Journal of Computational Linguistics *Microfiche 52*

A COMPUTATIONAL TREATMENT OF COORDINATE CONJUNCTIONS

CAROL RAZE

Linguistic String Project

New York University

New York 10012

SUMMARY

The present paper describes a computational solution to the problem of locating words that are zeroed under conjunction. In this solution, which is based on general properties of conjunctional constructions, a mechanism locates zeroed elements in the conjunction strings and cross-references them with respect to elements in the head construction. Constraints can then be applied to elided conjuncts as though they were expanded, and transformational expansion, which reconstructs the complete sentences underlying conjunctional occurrences, can be carried out straightforwardly by following the pointers which have been set up.

The main innovation is called "stacking". It is a non-deterministic programming device which causes restrictions (i.e., subprograms applying detailed constraints) to be re-executed on conjoined segments whenever the restriction is inovked on an element which has a conjunct.

CONTENTS

A COMPUTATIONAL TREATMENT OF COORDINATE CONJUNCTIONS

Carol Raze

One particularly intricate problem in the computer parsing of natural language texts is the complexity introduced into the parsing system by conjunctions. This complexity is due to the richness of conjunctional constructions and to the material implicit in sentences containing conjunctions. The computational problem can be divided into three parts: (I) generating parse trees which cover the occurrences of conjunction-headed strings in sentences; (II) locating words in the sentences which are repeated implicitly in a "zeroed," i.e., elided, form in particular positions in conjunction strings; and (III) reconstructing the complete sentences underlying conjunctional occurrences when the application requires such an expansion.

A generalized algorithmic solution to problem (I) was provided over a decade ago by the New York University Linguistic String Project (LSP) in the framework of linguistic string analysis (Sager et al. 1966, Sager 1967, Raze 1967), and similar devices have been described more recently, e.g., by Woods in the framework of augmented transition networks (Woods 1973). The present paper describes a computational solution to problem (II), locating the words that are zeroed under conjunction. In this solution, which is based on general properties of conjunctional constructions, a mechanism locates zeroed elements in the conjunction strings and cross-references them with respect to elements in the head construction. Constraints can then be applied to elided conjuncts as though they were expanded, and transformational expansion, which was problem (III) above, can be carried out straightforwardly by following the pointers which have been set up.

The main innovation is called "stacking." It is a nondeterministic programming device which causes restrictions (i.e., subprograms applying

detailed constraints) to be re-executed on conjoined segments whenever the restriction is invoked on an element which has a conjunct. Since the constraint usually applies to a combination of words in the main string and in the conjunct, the device must ensure that the proper grammatical relation holds between these words. That is, it treats these separated words as a single linguistic entity, obtaining the effect of a full expansion of the conjunctional occurrence without carrying out the physical rearrangement and copying or the parse tree. The strategy of treating ellipsis in two steps (locating deleted elements and later carrying out the physical expansion) is important for a solution to conjunctions for several reasons. First, it is costly and often fruitless to expand a conjunctional string until one is sure one has a good parse. Second, to get a good parse one has to execute restrictions on the generated parse tree including the conjunction subtrees, and this requires locating deleted elements. And third, the decision as to whether conjunctional constructions should be expanded or not is application specific.

This article is divided into four sections: 1. The Definition of Conjunction Strings; 2. Restrictions under Conjunctions; Stacking; 3. Less Common Deletion Forms; and 4. Implementation. An appendix describes the LSP system.

1. THE DEFINITION OF CONJUNCTION STRINGS

Constructions containing coordinate conjunctions CONJ have the overall form: A CONJ B, where A and B are structures of the same type in the grammar. In linguistic string grammar, A and B are particular elements or sequences of elements in defined strings of the grammar. An element or a sequence of elements in a string occurrence may be conjoined by a conjunctional string which consists of the conjunction followed by another occurrence of the

same type of string element (or elements) that precedes the conjunction. In Hearsay and rumors can ruin careers we have and rumors (AND + SUBJECT) conjoined to Hearsay (SUBJECT) in the ASSERTION string, which consists of the elements SUBJECT + TENSE + VERB + OBJECT. (See Appendix for string definitions.) And in Rumors can ruin careers and can cause much hardship we have and can cause much hardship (AND + TENSE + VERB + OBJECT) conjoined after Rumors can ruin careers (SUBJECT + TENSE + VERB + OBJECT).

Computationally, to include all the conjunctional combinations of a large grammar in the grammar definitions would complicate the grammar and make it immense. Instead, an interrupt mechanism can be used to achieve the same result.* An interrupt occurs when a conjunction is reached while parsing a sentence. When an interrupt occurs, a conjunctional node is attached to the part of the tree being built. To illustrate, a simplified tree of the type the LSP system would generate for the noun phrase Hearsay and rumors hastily printed is shown in Fig. 1. After Hearsay is matched as a noun, an interrupt occurs, and the special process node ANDSTG is attached to the right of Hearsay. However, a restriction of the LSP grammar limits the insertion of a special process node to occurring only in LXR type sequences or in strings,+ and therefore

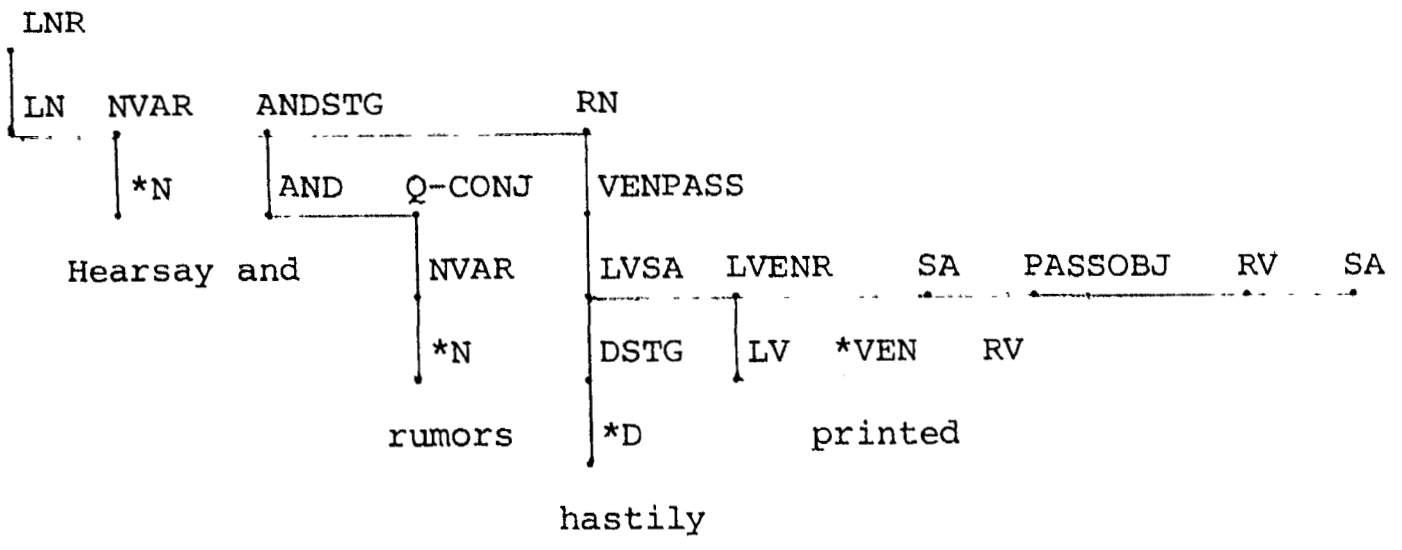
---

*The mechanism described here for generating conjunction strings was first programmed by James Morris for the 1966 IPL version of the LSP system (Sager et al. 1966). It was expanded in the FAP version of the LSP system, programmed by the author (Raze 1967), and is also part of the current FORTRAN implementation by Ralph Grishman (Grishman 1973). As noted above, the LSP system is not the only system to employ a dynamic device for generating conjunction parse trees.

+An LXR definition consists of three elements: a position for the left adjuncts of X, a core position for the word class X, and a position for the right adjuncts of X. See Appendix for a definition of the term "string" as used formally in the LSP grammar.

FIGURE 1

Parse tree of noun phrase: Hearsay and rumors hastily printed

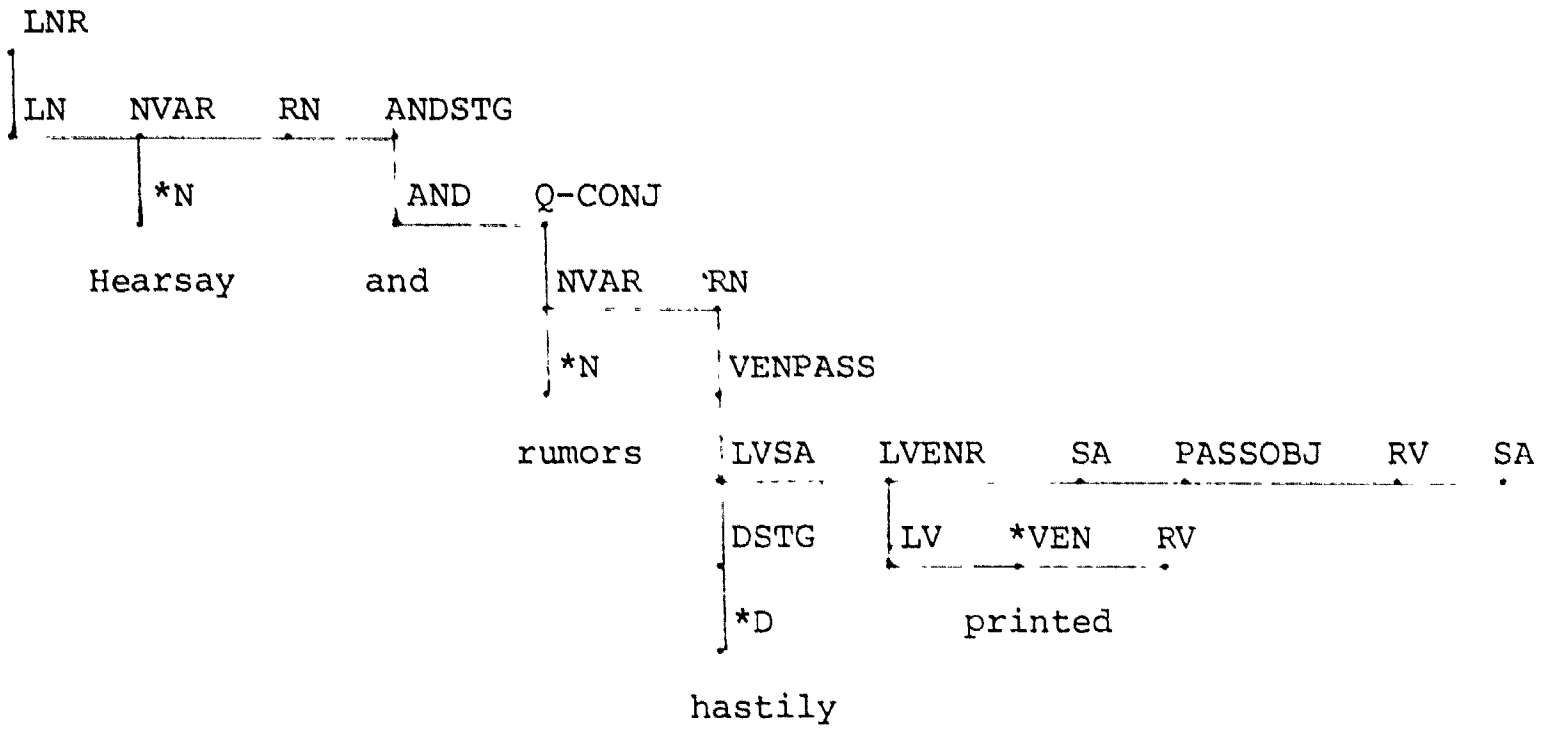


ANDSTG is rejected in the lowest level. This restriction avoids various redundancies created by conjoining at intermediate levels and regularizes the points of conjunction in the parse tree. When insertion fails, the parser detaches the special process node and continues parsing as if no interrupt had occurred. Another interrupt may occur, however, after the next element in the tree has been satisfied. Thus for the noun phrase in Fig. 1 an interrupt occurs after NVAR is satisfied and ANDSTG is attached to the right of NVAR. ANDSTG consists of and followed by the general conjunctional string Q-CONJ. Q-CONJ contains a restriction which generates a definition for Q-CONJ. Its definition consists of a set of alternate values: the first value is the element to the left of the inserted node, the second consists of the two elements to the left of the inserted node, etc. Thus, in Fig. 1 Q-CONJ is NVAR (rumors). The parser resumes by constructing element RN of LNR. Here the right adjunct hastily printed adjoins the conjunction of hearsay and rumors. Another analysis for the noun phrase hearsay and rumors hastily printed is shown in Fig. 2. In this case hearsay has no right adjunct and Q-CONJ consists of the elements NVAR and RN (rumors hastily printed). The difference in these two trees shows the ambiguity in the given sentence.

FIGURE 2

Another parse tree of noun phrase:

Hearsay and rumors hastily printed



2. RESTRICTIONS UNDER CONJUNCTIONS; STACKING.

A sentence with a conjunction presents a problem for the application of detailed constraints to the parse tree (restrictions), an essential part of the parsing process. The tree structure will be different from that assumed by the restriction; conjunctional strings will have been inserted and the conjunctional strings themselves may be truncated versions of defined strings. To appreciate what this problem means, one must keep in mind that a grammar like that of the LSP for processing English text sentences is very large by comparison with grammars used in most other natural language processing systems, which are directed to particular subsets of English. The LSP grammar consists of approximately 3500 lines. The restrictions comprise by far the largest part of the grammar, and without them, text parsing is out of the question. In addition, we have found that roughly one third of all text sentences contain coordinate or comparative conjunctions, many times in complicated

interrelation. It is therefore essential that there be a means for executing restrictions on sentences containing conjunctions.

One solution to this problem is to rewrite all the restrictions so that they test for conjunctions and accommodate truncated segments. This was done in earlier versions of the LSP parser but that involved a tremendous amount of detail and both increased and complicated the grammar enormously. As an alternative, in some cases the sentence can be expanded so that the restrictions operate on complete strings. But in other cases this is not possible because the expansion necessitates the introduction of certain transformations which should be done later. In the present system we therefore use a general solution whereby restrictions are re-executed automatically for conjunctional occurrences. Thus, in the analysis shown in Fig. 1 of Hearsay and rumors hastily printed the selectional restriction will be executed automatically for the sequences hearsay printed and rumors printed. This is equivalent to expanding the sentence into two assertions, namely, (Someone) printed hearsay and (Someone) printed rumors. The actual expansion is performed in the transformational phase (if the application requires expansion), which takes place after the surface analysis is obtained. However, for the correct surface analysis it is crucial that the restrictions operate on expanded or complete strings. Thus, the conjunction computation performs some of the function of expansion prior to the transformational phase.

To apply restrictions to sentences containing conjunctions, a non-deterministic programming mechanism, which we call the stacking mechanism, was incorporated into the parser. This mechanism saves the conjoined structure so that the restriction can be re-executed for the conjoined structure. In principle, the same method can be applied to any grammar containing restrictions as long as the structure assigned to the conjunctional occurrence is of the same type as the one assigned to the segment that precedes the conjunction.

An efficient restriction component of a grammar makes use of routines for pieces of code which are used again and again. Then the stacking mechanism can be implemented efficiently by changing the routines rather than the more numerous restrictions. In the LSP case, the routines which were modified are those which locate a basic type of structure in the parse tree, such as the routines CORE, HOST, ELEMENT, LEFT-ADJUNCT, etc.* In addition to locating structures the modified routines must also test whether or not the structures are conjoined. When a restriction calls a routine which locates a conjoined structure, that routine in turn calls an operator which saves the conjoined structure(s) along with the place of the routine within the sequence of operations being executed in the restriction. The operator puts this information on a re-execution stack. The routine returns to the original structure located and the restriction interpreter executes the rest of the restriction. At this point the restriction interpreter is "looking at" the original structure located by the routine. When a restriction is successful the restriction interpreter uses the information on the re-execution stack to resume execution of that restriction. The restriction is resumed at the point immediately after the call to the routine so that the routine itself is not called again. Instead however, the restriction interpreter will "look at" the conjoined structure previously located and saved by the routine. For example, when the verb-object selection restriction WSEL1 is executed on the sentence They printed hearsay and rumors, it calls the CORE routine to obtain the core of the object. The CORE routine will go to hearsay, locate and stack rumors, and return to hearsay. WSEL1 will be successful for printed hearsay. WSEL1 will be resumed therefore at the point after the call to the CORE routine and rumors, which was saved

---

*See Appendix for a detailed description of the routines.

by the CORE routine, will be plugged in as though it was just obtained by the CORE routine. WSEL1 will be successful for printed rumors. A trace of the execution of WSEL1 which shows the re-execution of the restriction on the conjunct appears in Fig. 3. Various cases of WSEL1 with examples are explained in detail below.

An effect of dynamic generation of conjunctional parse trees is that in addition to locating conjoined values, provision must be made for restrictions to function properly on the non-conjunctional grammar, in new situations due to conjunctions; a routine may be operating in a structure into which conjunctional strings have been inserted, or in a truncated version of a defined string or host-adjunct sequence. For example, the LSP RIGHT-ADJUNCT routine is assumed to start at a node X in an LXR type node. Without conjunctions the RIGHT-ADJUNCT routine goes one node to the right from X to arrive at the right adjunct of X. In Fig. 2 for the noun phrase rumors hastily printed, RIGHT-ADJUNCT, starting at the core position NVAR subsuming rumors, goes one node to the right to arrive at its right adjunct RN (hastily printed). However, with conjunctions the routine must go right until it lands at a non-conjunctional node. Thus, in Fig. 1, for the noun phrase hearsay and rumors hastily printed, RIGHT-ADJUNCT goes from NVAR (hearsay) past the conjunction string (and rumors) to RN. When RIGHT-ADJUNCT starts in the conjunctional string at NVAR (rumors), RN is not to its right. To go to RN, the routine locates the corresponding preconjunction element NVAR (hearsay) and goes to RN from there.

To illustrate how the stacking mechanism works we will explain in detail how restriction WSEL1 is executed for several sentences with conjunctions. WSEL1 is housed in the center string ASSERTION:*

---

*WSEL1 is also housed in other strings containing object and verb elements, but for our example we will only consider the string ASSERTION. In the statement

```
WSEL1 = IN ASSERTION:  IF ALL OF $OBJECT-NOUN,
                               $GOVERNING-VERB,
                               $FORBIDDEN-NOUN-LIST
                               ARE TRUE
                       THEN $NOCOMMON.
$OBJECT-NOUN = THE CORE X1 OF THE OBJECT
               X10 IS N OR PRO.
$GOVERNING-VERB = AT X10, COELEMENT VERB X4 EXISTS.
$FORBIDDEN-NOUN-LIST = THE CORE OF X4
                       HAS ATTRIBUTE NOTNOBJ X5.
```

The above restriction statements have the following functions: $OBJECT-NOUN checks that the core of the object position is a noun or pronoun; $GOVERNING-VERB tests that a verb coelement exists; and $FORBIDDEN-NOUN-LIST checks that the given verb has an attribute NOTNOBJ.* If all these conditions are satisfied the selection check $NOCOMMON is made.

```
$NOCOMMON = LISTS X1 AND X5 HAVE NO COMMON ATTRIBUTE.
```

If the noun does have a subcategory that is on the subcategory list of NOTNOBJ, then $NOCOMMON fails and that noun is not accepted as the object of the given verb.

Consider the parse tree for the sentence shown in Fig. 4, They spread rumors and they print hearsay. In $OBJECT-NOUN, in order to go to the core of the OBJECT, OBJECT must first be located. This is accomplished by the routine

---

(The first footnote is continued from p. 8.)

$GOVERNING-VERB we can therefore use the routine COELEMENT with argument VERB. In the actual restriction a more general routine VERB-COELEMENT is used. This restriction was described in full, without reference to its operation on conjunction sentences, in Sager and Grishman 1975.

*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).

FIGURE 3

Trace of WSEL1

```
WSEL1 BEING EXECUTED IN OBJECT WHICH SUBSUMES  HEARSAY AND RUMORS ...
    IMPLY( AND( ( STARTAT(  ( OBJECT )  ) -> OBJECT=HEARSAY AND RUMORS ... )  + STORE
          ( X10 )  + ( CORE -> N=HEARSAY  (********. STACKING N=RUMORS  ********) )
           + STORE( X3 )  + IS(  ( N, PRO )  )  + LOOK( X10 )  + ( VERB-COMPLEMENT ->
          VERB=PRINTED  )  + STORE( X4 )  + LOOK( X4 )  + ( CORE -> TV=PRINTED  )
          ATTRB(  ( NOTNOBJ )  )  + STORE( X5 )  +  )  + NOT( COMMONAT( LOOK( X3 )
           + LOOK( X5 )  +  )  -  )  +  )  +

 RESTRICTION RESTARTED DUE TO STACKING AT NODE N
 WHICH SUBSUMES RUMORS .
     )  + STORE( X3 )  + IS(  ( N, PRO )  )  + LOOK( X10 )  + ( VERB-COMPLEMENT ->
          VERB=PRINTED  )  + STORE( X4 )  + LOOK( X4 )  + ( CORE -> TV=PRINTED  )
          ATTRB(  ( NOTNOBJ )  )  + STORE( X5 )  +  )  + NOT( COMMONAT( LOOK( X3 )
           + LOOK( X5 )  +  )  -  )  +  )  +

OBJECT HAS BEEN COMPLETED SUBSUMING  HEARSAY AND RUMORS

 SUCCESSFUL PARSE NO.  1
           RUNNING TIME  =     1.318
           NODES IN TREE  =    87
```

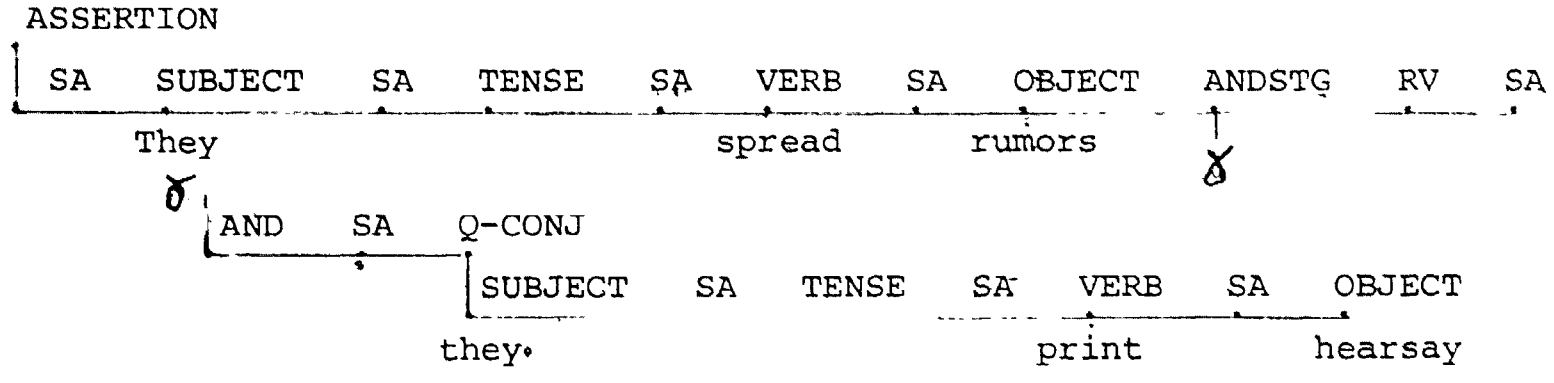
```
THEY PRINTED HEARSAY AND RUMORS .

    PARSE   1

 1. SENTENCE  = INTRODUCER  CENTER  ENDMARK
                            2.

 2. ASSERTION = SA  SUBJECT  SA  TENSE  SA  VERB       SA  OBJECT              RV  SA
                    THEY                    PRINTED      HEARSAY AND 3.

 3. Q-CONJ    = NVAR
                RUMORS
```

STARTAT(OBJECT). STARTAT both locates OBJECT and calls the stack operator for each conjoined OBJECT. Thus for the sentence in Fig. 4, STARTAT will go to the first OBJECT (subsuming rumors) and will save the second OBJECT (hearsay). When STARTAT is completed the CORE routine is called to locate the core of OBJECT. In this example it locates the noun rumors.

$GOVERNING-VERB goes to VERB, which is a coelement of OBJECT. It does this by first locating OBJECT (which was saved in register X10 by $OBJECT-NOUN) and by calling the COELEMENT routine. Thus in Fig. 4, COELEMENT(VERB) goes from the first OBJECT to its coelement VERB (spread). It locates all conjuncts of this VERB and determines whether or not to call the stack operator. VERB (spread) has a conjunct but in this case COELEMENT(VERB) will not stack the conjoined VERB (print) because this VERB has its own coelement OBJECT. When COELEMENT returns, the restriction interpreter is looking at the first VERB. WSEL1 is successful for spread rumors. Since there is something on the re-execution stack, the execution of WSEL1 is resumed. It is resumed in $OBJECT-NOUN at the point immediately after the call to STARTAT(OBJECT). However, this time the restriction interpreter is located at the second OBJECT. The core (hearsay) of the second OBJECT is obtained and the rest of the restriction is executed for the second time. In particular, COELEMENT(VERB) goes to the second VERB (print) from the second OBJECT and WSEL1 is successful for print hearsay.

FIGURE 4

Parse tree of They spread rumors and they print hearsay

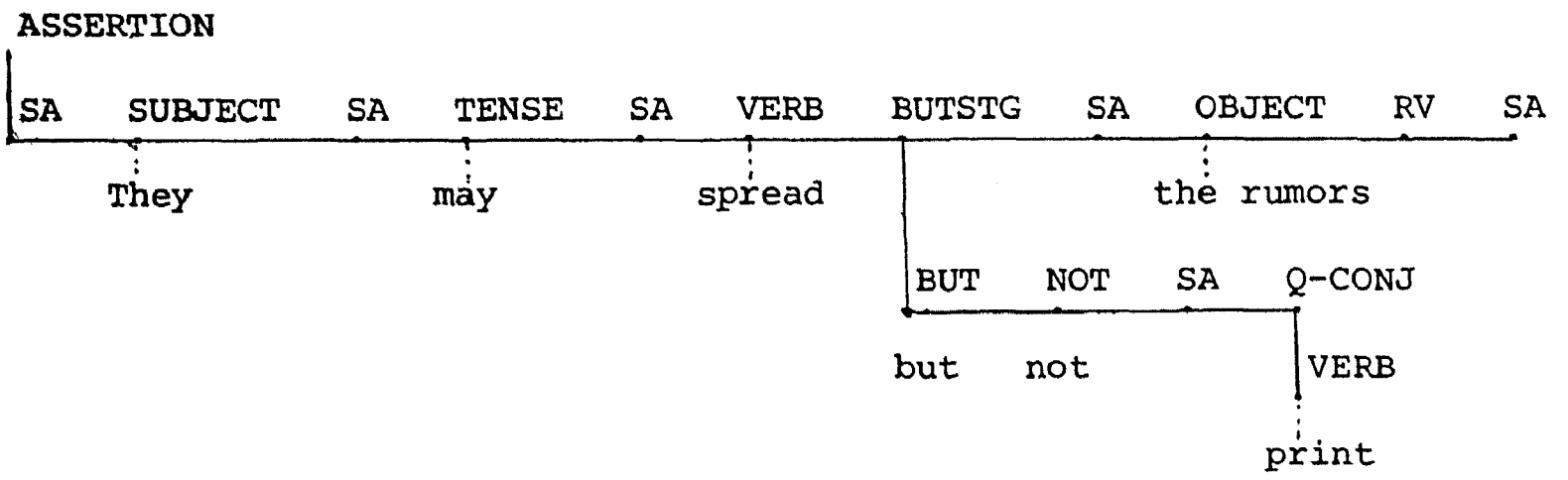


For the sentence shown in Fig. 5, They may spread but not print the rumors, the execution of WSEL1 is different. The OBJECT (the rumors) has no conjunct but the VERB does. In this case the COELEMENT(VERB) routine goes to the first VERB (spread) from OBJECT and saves the second VERB (print) on the re-execution stack. WSEL1 is successful for spread rumors. Its execution is therefore resumed in $GOVERNING-VERB at the point just after the call to COELEMENT(VERB). However, this time the restriction interpreter is looking at the second VERB (print). Therefore, the well-formedness of print rumors is also checked.

FIGURE 5

Parse tree of

They may spread but not print the rumors.



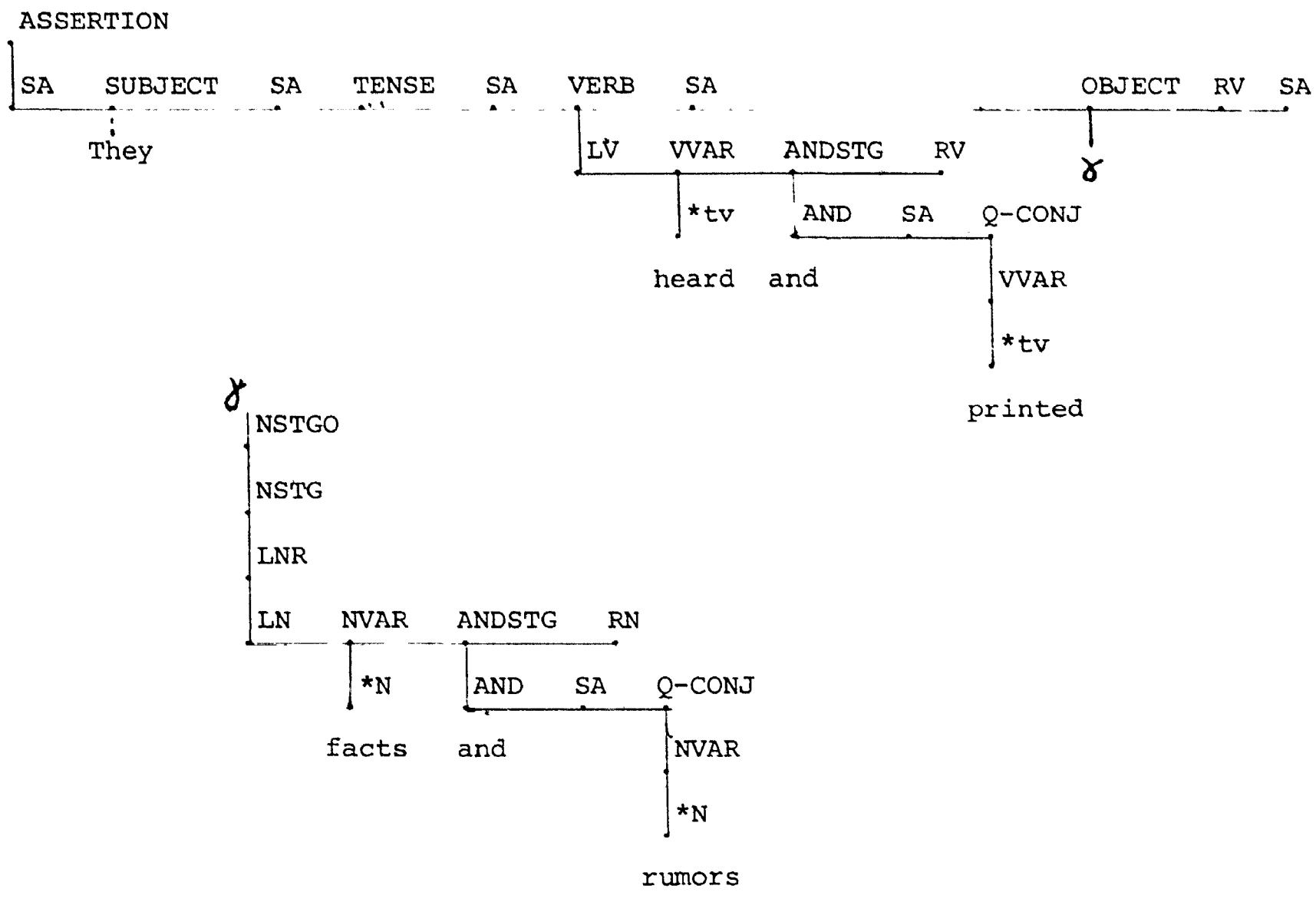
For the sentence shown in Figure 6, They heard and printed facts and rumors, the execution of WSEL1 is again different from the previous examples. OBJECT itself has no conjunct but the core of OBJECT does. Thus, when CORE is called in $OBJECT-NOUN, the CORE routine will locate N (facts) and will place the second N (rumors) on the re-execution stack. Likewise, the COELEMENT(VERB) routine will not find a conjunct for the VERB position itself However, the core of the VERB has a conjunct. When CORE is called in

$FORBIDDEN-NOUN-LIST, it will locate tv (heard) and will place the second tv (printed) on the re-execution stack. This will result in WSEL1 being executed for all four verb + object noun sequences: heard facts, heard rumors, printed facts, printed rumors.

FIGURE 6

Parse tree of

They heard and printed facts and rumors.



The following is an example of how the stacking mechanism helps to resolve syntactic ambiguity. In the sentence He printed rumors and his friend also, there are two possible parses. In one analysis the object of printed consists of the conjoined nouns rumors and friend. This analysis would be rejected by WSEL1 (assuming print to have NHUMAN as a value of NOTNORJ)

in the following manner. When the CORE routine is called in $OBJECT-NOUN, the noun rumors is located and the conjoined noun friend is placed on the re-execution stack. Printed is located by $GOVERNING-VERB. Printed rumors is successful and WSELl is re-executed for printed friend. However, printed has NHUMAN on its NOTNOBJ list and friend has a noun subcategory NHUMAN. Thus $NOCOMMON fails and this analysis is rejected.

Another analysis of this sentence contains a second (implicit) occurrence of printed rumors: He printed rumors and his friend also (printed rumors). The conjoined string consists of SUBJECT (his friend) followed by VERB and OBJECT, which both are assigned the values NULLC (Section 3).

3. LESS COMMON DELETION FORMS

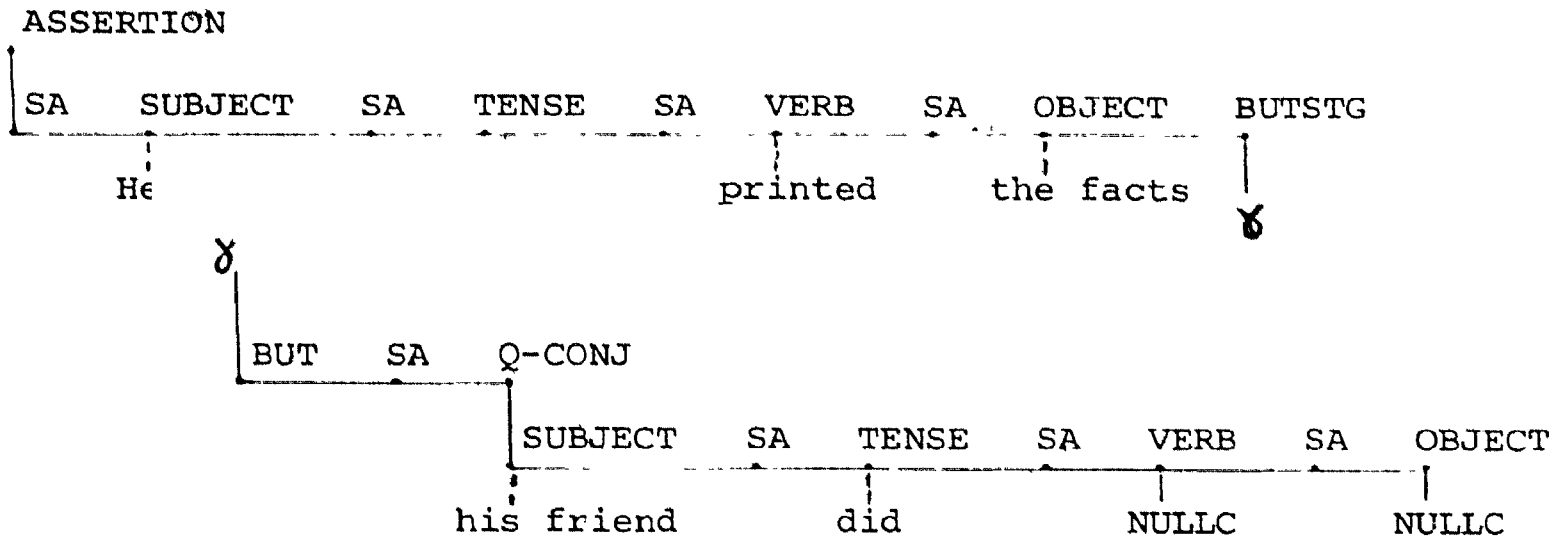
Some conjunctional strings differ from the usual form of conjunctional occurrences in that the deleted elements are not contiguous to the conjunction. For example, in He printed the facts but his friend did not, the verb and object printed the facts have been deleted in the conjunctional string; the positions of the deleted verb and object in but his friend did not do not immediately follow but, but follow the tense position (did). This type of conjunctional occurrence is covered in the LSP grammar by a node called NULLC. The NULLC is automatically satisfied without subsuming a sentence word if certain conditions are met. When it occurs in a parse, it represents the fact that a required element of a string-conjunct has been deleted in such a position that the resulting parse tree does not have similar structures on either side of the conjunction. Thus, in the parse of the preceding sentence, shown in Fig. 7, the verb and object of the assertion following but both have the value NULLC.

Before a NULLC node is accepted, several "zeroing" restrictions must be satisfied. The general conditions for acceptance of a NULLC node are that

FIGURE 7

Parse tree of

He printed the facts but his friend did not.



1) it be within a conjunctional string; 2) the pre-conjunctional element which corresponds to the deleted element subsume some sentence word(s); and 3) that the zeroing pattern be one of a few types known to be acceptable for conjunctional strings with this type of deletion (see examples below). Thus in the example sentence shown in Fig. 7, the verb of the assertion following but will have the value NULLC, and conditions 1-3 are met as follows:

1) NULLC is in a conjunctional string--the string headed by but.
2) The corresponding pre-conjunctional verb subsumes printed.
3) The NULLC verb is not contiguous to the conjunction but, but follows the tense in the conjunctional string, this construction being one of the allowed patterns of zeroing.

In addition to establishing conditions for accepting NULLC nodes, the "zeroing" restrictions have another very important function. They locate the material that would have been repeated in the conjoined phrase but which was "zeroed," that is, deleted or elided, and they save that information. When the appropriate structure is located for a NULLC node, the location is saved via a node-attribute mechanism.

The restriction assigns to the NULLC node the node attribute LINKC, whose value points to the pre-conjunctional structure that would have been repeated. Once the LINKC attribute is assigned, any constraint can locate this structure and thus identify the words that have been deleted. In the LSP system the actual physical rearrangement of the parse tree is not done until the transformational phase. However, the actual "filling in" transformation is simple and straightforward because the main bulk of the work (locating the material identical to what is zeroed) is done by the restrictions in the parsing phase. For conjunction strings with the NULLC type of deletion, restrictions can be executed only after the filling in of deleted elements has been done.

Some examples of the NULLC deletion types which are covered by the LSP grammar appear in parentheses in the following sentences:

He printed facts but his friend did not (print facts).

He washed the dishes but she did not want to (wash the dishes).

He should have tried to wash the dishes but he did not (try to wash the dishes).*

He ate supper and she (ate supper) too.

He played the played the piano and she (played) the drums.

He left and (he left) fast.

He tried to wash the dishes but she was not willing to (try to wash the dishes).*

---

*This sentence contains an ambiguity as to how much material was deleted: try to wash the dishes vs. wash the dishes.

4. IMPLEMENTATION

The basic routines* of the LSP grammar were modified to handle conjunctions. They were modified to locate the appropriate conjoined structure(s) and call an operator which saves those structure(s). In addition, the routines were modified to function properly in the non-conjunctional grammar for the new situations which occur when conjunctions are present. Although these changes involve some of the basic routines only, the restrictions use these routines so often so that the changes have to be as efficient as possible. Otherwise the execution time of a restriction would be greatly increased.

To save much repetitious moving around the tree, pointers are attached to the appropriate nodes of the tree via a node-attribute mechanism which is described in Sager and Grishman 1975. Each element E_2 in a conjunctional string of the form E_1 CONJ E_2 is assigned a node attribute called PRECONJELEM which points to the corresponding element E_1 in the string prior to conjunction. Likewise each E_1 is assigned a node attribute called POSTCONJELEM which points to the corresponding element E_2 in the post-conjunctional string. The node attribute assignments are done by a well-formedness restriction housed on the conjunctional string. Once the node attributes are assigned, the routines can quickly obtain (or check for) conjoined values of a node by using its node attribute POSTCONJELEM. And when a routine is called from inside a truncated string segment, it can quickly move to the corresponding pre-conjunction element by obtaining the node attribute PRECONJELEM. From that point the routine can then locate the appropriate element. For example in Fig. 1, the second NVAR (rumors) has been assigned node attribute PRECONJELEM pointing to a NVAR (hearsay). Using the node attribute PRECONJELEM

---

*See Appendix for a detailed explanation of the basic routines.

of the second NVAR, RIGHT-ADJUNCT goes to the first NVAR and then goes two nodes to the right to RN.

Not all restrictions may be re-executed for sentences containing conjunctional occurrences. For example, those restrictions testing number agreement have to be changed to explicitly test for the occurrence of a conjunction. Therefore, those restrictions must use routines that do not stack. Each routine that calls the stack operator has a counterpart which does not. The routines are written so that in each pair, the one which stacks calls on the nonstacking version as a subpart.

We will now go into the details of those routines that were modified for conjunctions. The explanations here will be concerned with the modifications only.

Most of the basic routines fall into three categories: 1) those which begin at X or go down to X. STARTAT(X), ELEMENT(X), LAST-ELEMENT and NELEMRT* are in this category. These routines concern only one element of a string or sequence. 2) The routines which go right or left to X. HOST, RIGHT-ADJUNCT, LEFT-ADJUNCT, NEXT-ELEMENT, PREVIOUS-ELEMENT and COELEMENT(X) are in this category. These routines involve two elements of a string or sequence. 3) The routines which start at or go up to X. IMMEDIATE(X), STARTAT(X), PRESENT-ELEMENT, IT, PRESENT-STRING, IMMEDIATE-NODE are in this category. Because the conjunction modification can be generalized depending on which category a routine is in, only several routines are needed to handle various conjunction operations. The actual modification to many of the routines therefore consists of adding a call to one of the few routines which handle conjunctional operations.

---

*Core is also in this category; however, this routine needs to perform some extra operations to get to the conjoined core word.

The routine that handles stacking for routines of category (1) above is called $STACK-TEST:*

```
$STACK-TEST = IF $POSTCONJ THEN $STACK-CONJUNCTS.
$POSTCONJ = THE PRESENT-ELEMENT HAS NODE ATTRIBUTE POSTCONJELEM.
                                                        (GLOBAL)
```

It is assumed that the restriction interpreter is "looking at" X when $STACK-TEST is called. If an element E_1 has a corresponding element E_2 in a conjunctional string E_1 will have the node attribute POSTCONJELEM. This provides a quick test to determine whether or not a node has a conjunct. If E_1 does not have the node attribute POSTCONJELEM, $POSTCONJ fails and $STACK-TEST is finished; if E_1 has the node attribute POSTCONJELEM, the attribute POSTCONJELEM has a value, namely E_2. When $POSTCONJ is finished the restriction interpreter will be "looking at" E_2.

```
$STACK-CONJUNCTS = VERIFY ITERATE $STACK-X.
$STACK-X = DO $POSTCONJ; STACK.
```

$STACK-CONJUNCTS locates all the conjuncts of the node by iterating $STACK-X. It then returns to the starting node. $STACK-X goes to each conjunct by first executing $POSTCONJ and then calling STACK, the operator which puts the conjunct on the re-execution stack. In Fig. 4, starting at the first OBJECT, $STACK-TEST will call STACK for the second OBJECT. It will return to the first OBJECT, before exiting, as a result of the VERIFY command.

The routines that handle stacking for those routines in category 2 above are called $STACK-FOR-LEFT-TO-X and $STACK-FOR-RGHT-TO-X respectively. We will only go into the details of $STACK-FOR-LEFT-TO-X since $STACK-FOR-RGHT-TO-X is similar. When $STACK-FOR-LEFT-TO-X is called, the restriction interpreter

---

*In our system $STACK-TEST is actually a global address instead of a routine. This was done because it is faster to execute an address than a routine.

is assumed to be at X. It is also assumed that the routine which called $STACK-FOR-LEFT-TO-X started at some node Y, saved Y in register X200 and went from Y left one or more nodes to arrive at X. For instance, this occurs when the routine COELEMENT(VERB) is called from OBJECT in Fig. 4.

```
$STACK-FOR-LEFT-TO-X = IF $POSTCONJ
                       THEN VERIFY $STACK-IF-NO-Y-RGHT.      (GLOBAL)
```

If X has a corresponding element in a conjunctional string it will have the node attribute POSTCONJELEM. If X does not have the node attribute POSTCONJELEM, $STACK-FOR-LEFT-TO-X is finished. If it does have node attribute POSTCONJELEM, $STACK-IF-NO-Y-LEFT is executed to determine whether or not to stack the conjunct(s):

```
$STACK-IF-NO-Y-RGHT = IF $POSTCONJ
                      THEN EITHER ALL OF $NO-Y-TO-RIGHT,
                                         $DO-STACK,
                                         $STACK-IF-NO-Y-RGHT
                           OR TRUE.
$NO-Y-TO-RIGHT = NOT ITERATE GO RIGHT UNTIL TEST FOR X200 SUCCEEDS.
$DO-STACK = STACK.
```

Looking at Fig. 4 assume COELEMENT(VERB) is called when the restriction interpreter is at OBJECT (rumors). The COELEMENT routine locates the first VERB (spread). The premise of $STACK-FOR-LEFT-TO-X is successful because this VERB has a conjunct and $STACK-IF-NO-Y-RGHT is executed. The premise of $STACK-IF-NO-Y-RGHT is successful: a conjunction is found, in this case the second VERB (print). The restriction interpreter remains at the second VERB while the rest of the implication is executed. In this example, there is OBJECT, (hearsay) to the right of the second VERB; therefore $NO-Y-TO-RGHT fails. As a result VERB (print) is not stacked. In Fig. 5, if COELEMENT(VERB) is called when the restriction interpreter is at OBJECT (the rumors) the VERB

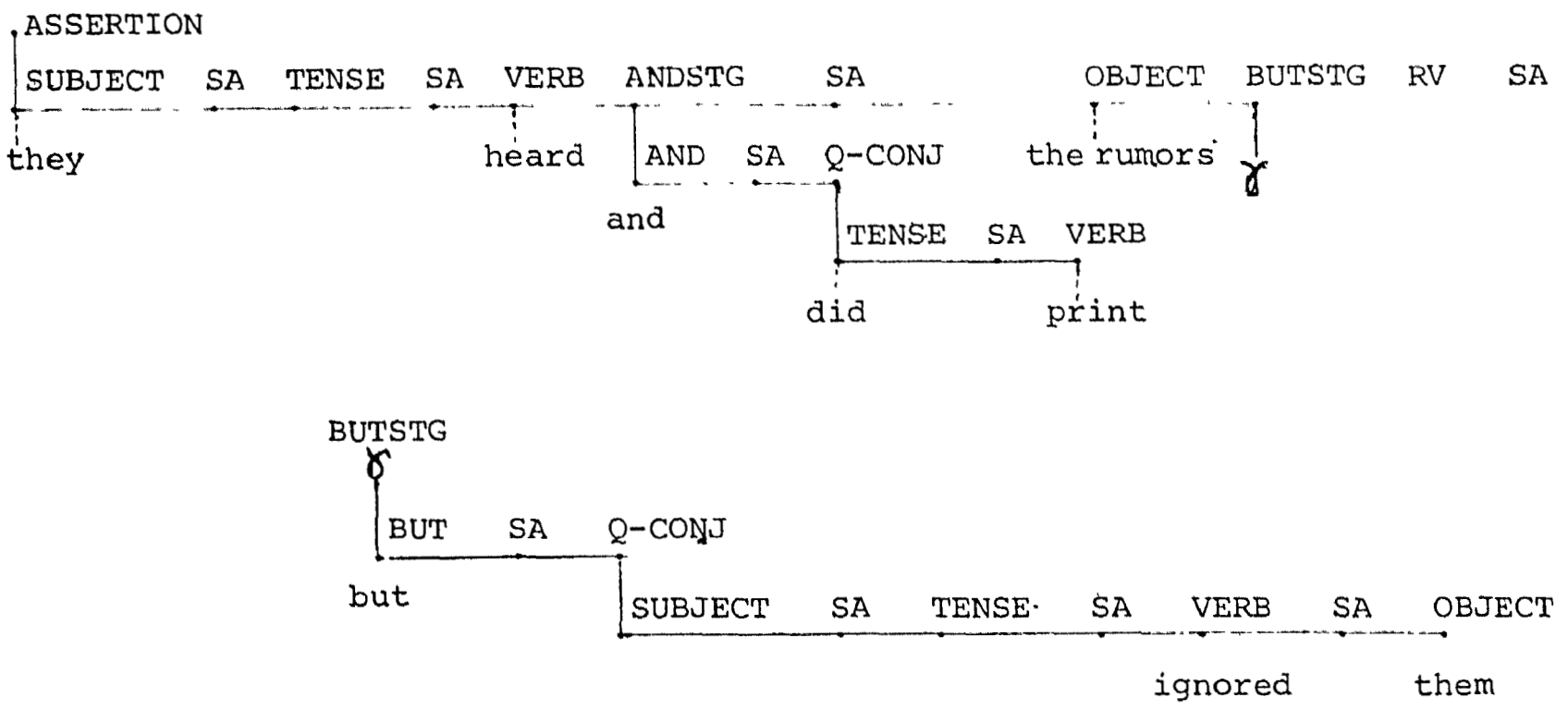
(spread) is located. It has a conjunct, which is stacked. In this case $NO-Y-TO-RGHT is successful since there is no OBJECT to the right of the second VERB.

$STACK-IF-NO-Y-RGHT is recursive so that all the conjoined structures are located and tested. In the above examples if $POSTCONJ is true starting at the second VERB then $STACK-IF-NO-Y-RGHT goes to the next corresponding post-conjunctional VERB and determines whether or not to stack it. In Fig. 8, after executing the premise of $STACK-IF-NO-Y-RGHT the restriction interpreter is at the second VERB. $NO-Y-TO-RIGHT is true and $DO-STACK is executed. It will stack the second VERB. $STACK-IF-NO-Y-RGHT is called recursively and the restriction interpreter is at the third VERB. However, OBJECT (them) is present; therefore $NO-Y-TO-RIGHT will fail and the third VERB (ignored) will not be stacked. Routines COELEMENT, HOST, LEFT-ADJUNCT and PREVIOUS-ELEMENT use $STACK-FOR-LEFT-TO-X. Routines COELEMENT, HOST, RIGHT-ADJUNCT and FOLLOWING-ELEMENT use $STACK-FOR-RGHT-TO-X.

FIGURE 8

Parse tree of

They heard and did print the rumors but we ignored them.



The routines in category 2 were also modified to operate properly if they start in a truncated segment of a defined string or host adjunct sequence. For example, if COELEMENT(OBJECT) is called from the second VERB in Figure 5, the COELEMENT routine will not be able to go left or right to OBJECT. It must first go to the corresponding pre-conjunctional element and then try to go left or right to X from there. This is accomplished by $TO-PRECONJUNCTION-Y.

```
$TO-PRECONJUNCTION-Y = EITHER $PRECONJ OR
                       $ASSIGN-PRECONJELEM                    (GLOBAL)
$PRECONJ = THE PRESENT-ELEMENT- HAS NODE ATTRIBUTE PRECONJELEM.
```

If the starting node has node attribute PRECONJELEM, $PRECONJ will go to the corresponding pre-conjunctional node; otherwise the node attributes PRECONJELEM and POSTCONJELEM have to be assigned. This is accomplished by $ASSIGN-PRECONJELEM:

```
$ASSIGN-PRECONJELEM = VERIFY $LOCATE-CONJNODE;
                      VERIFY $ASSIGN-PRE-AND-POST;
                      DO $PRECONJ.
```

$LOCATE-CONJNODE = ASCEND TO Q-CONJ; GO UP; STORE IN X100.

$ASSIGN-PRE-AND-POST assigns the node attribute PRECONJELEM to the current node. $ASSIGN-PRE-AND-POST is defined in routine PRE-POST-CONJELEM which will be described later. After the node attribute PRECONJELEM is assigned, $ASSIGN-PRECONJELEM goes to the corresponding pre-conjunctional node by executing $PRECONJ. For example in Fig. 5, if COELEMENT(OBJECT) is called from the second VERB, COELEMENT will call $TO-PRECONJUNCTION-Y to go to the first VERB; then it will try to go left or right to locate OBJECT.

Another type of adjustment is needed for restriction routines in category 3 above. The problem occurs when a restriction is executed starting at the conjunctional string Q-CONJ. When the definition for Q-CONJ is generated from the elements of a string, the well-formedness restrictions housed in the elements are transmitted along with the elements. The restrictions on those elements, therefore, were written with the assumption that the starting point is the string that the restrictions were originally housed in--i.e., two nodes up from Q-CONJ. For example in Fig. 5, all restrictions in ASSERTION assume to start at ASSERTION. Thus, the same restriction, starting at Q-CONJ would fail if, for example, we were to test whether the immediate-node of the second VERB is ASSERTION. Therefore, the routines in category 3 execute $UP-THROUGH-Q initially:

$UP-THROUGH-Q = ITERATET $GO-UP-TWICE UNTIL TEST FOR Q-CONJ
FAILS.

$GO-UP-TWICE = GO UP; GO UP.

$UP-THROUGH-Q goes to the node which is two nodes up from the top of a nest of Q-CONJ's.

Routine PRE-POST-CONJELEM assigns the node attribute PRECONJELEM to the elements of Q-CONJ. It is assumed that the starting node is the node above

Q-CONJ. To each element of Q-CONJ that is not on the C-NODE list (ANDSTG, ORSTG, BUTSTG, etc.), it assigns the node attribute PRECONJELEM. Likewise the corresponding pre-conjunction elements will be assigned the node attribute POSTCONJELEM.

```
ROUTINE PRE-POST-CONJELEM = STORE IN X100;
                            DO ELEMENT- (Q-CONJ);
                            DO LAST-ELEMENT-;
                            ITERATE VERIFY $ASSIGN-TEST UNTIL GO LEFT FAILS.
$ASSIGN-TEST = EITHER TEST FOR C-NODE OR EITHER $PRECONJ
                    [COELl-] OR $ASSIGN-PRE-AND-POST.
```

The routine PRE-POST-CONJELEM saves the starting point in register X100. It then uses the nonstacking routines* to go to the last element of Q-CONJ. The node attribute assignments start from the rightmost node of Q-CONJ and proceed left. $ASSIGN-TEST determines whether or not a node attribute should be assigned to a particular node. An assignment is not necessary if the node is on the C-NODE list or if the assignment was already made for the node. If an assignment does not have to be made, $ASSIGN-TEST is finished; if one has to be made $ASSIGN-PRE-AND-POST is executed:

```
$ASSIGN- PRE-AND-POST = STORE IN X500;
                        STORE IN X0;
                        GO TO X100;
                        ITERATE $GO-LEFT UNTIL TEST FOR X500 SUCCEEDS;
                        EITHER ITERATE $POSTCONJ [STARTAT]
                        OR TRUE;
                        DO $ASSIGN-POSTCONJELEM;
                        STORE IN X0;
                        GO TO X500;
                        DO $ASSIGN-PRECONJELEM.
```

---

* For each routine that stacks, there is a nonstacking counterpart, The nonstacking routines are used for restrictions where stacking is not desired--such as the number agreement restrictions.

```
$GO-LEFT = ITERATET $UPCONJ UNTIL GO LEFT SUCCEEDS; STORE IN X100.
$UPCONJ = GO UP; TEST FOR Q-CONJ; GO UP.
```

$ASSIGN-PRE-AND-POST saves the node to be assigned in registers X500 and X0. It then goes to the node saved in X100 (which is initially the starting C-NODE) and locates the corresponding pre-conjunctional element by executing $GO-LEFT until it finds a node which has the same name as that in register X500. That node is saved in X100 so that the search starts there for the next node to be assigned. $ASSIGN-POSTCONJELEM and $ASSIGN-PRECONJELEM assign the node attributes.

We will consider the case where the second OBJECT in Fig. 8 is being assigned node attribute PRECONJELEM. BUTSTG is saved in register X100. $ASSIGN-PRE-AND-POST saves the second OBJECT in registers X0 and X500. It then searches for the corresponding pre-conjunctional element by going left from BUTSTG. The first OBJECT is found. $POSTCONJ fails at the first OBJECT and $ASSIGN-PRE-AND-POST remains there. Node attribute POSTCONJELEM is assigned to the first OBJECT by $ASSIGN-POSTCONJELEM:

```
$ASSIGN-POSTCONJELEM = ASSIGN THE PRESENT ELEMENT NODE
                       ATTRIBUTE POSTCONJELEM.
$ASSIGN-PRECONJELEM = ASSIGN THE PRESENT ELEMENT NODE
                      ATTRIBUTE PRECONJELEM.
```

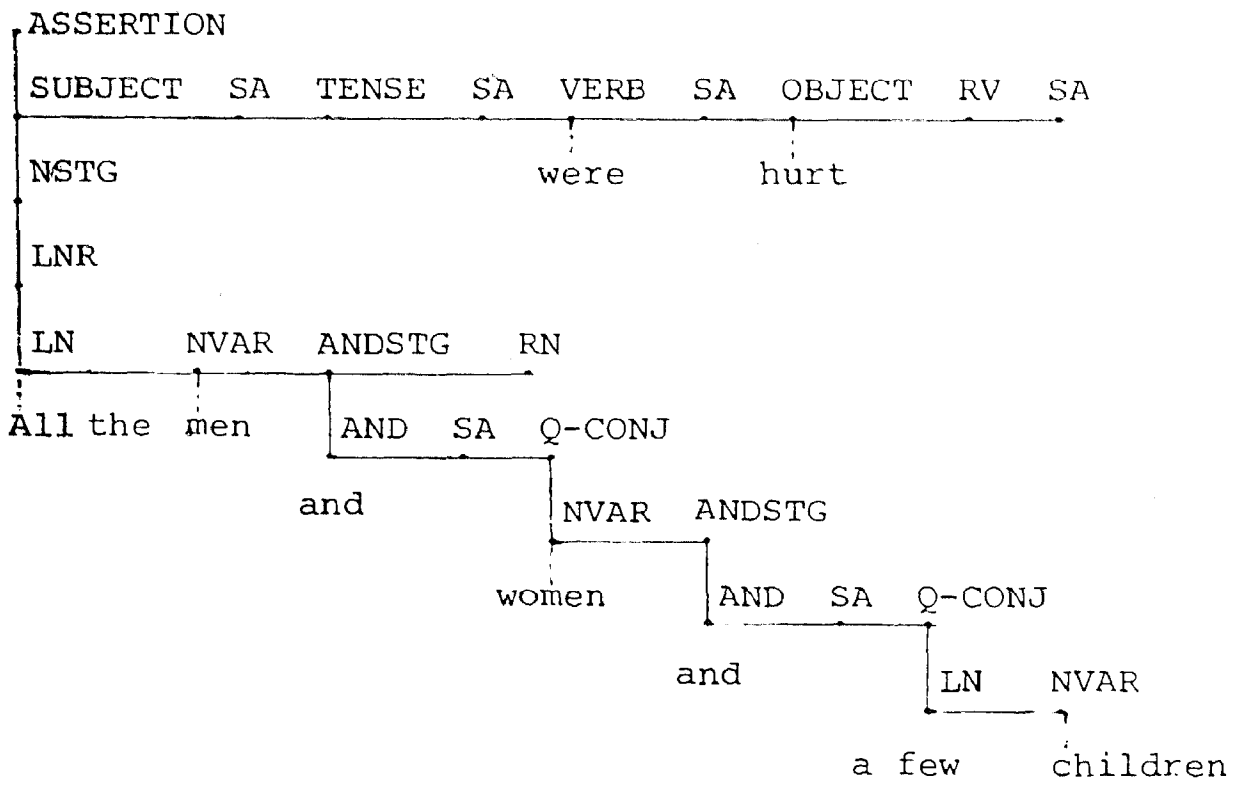
When node attribute POSTCONJELEM is assigned, if there is a node saved in register X0, that node will automatically be assigned as the value of attribute POSTCONJELEM by the node attribute assignment operator. In this case the second OBJECT is in register X0. Therefore the first OBJECT is assigned node attribute POSTCONJELEM with the second OBJECT as its value. After a node attribute assignment is made, register X0 is automatically cleared by the program. This prevents accidental value assignments from occurring in case the grammar writer forgets to clear the register. In the above example, after node

attribute POSTCONJELEM is assigned to the first OBJECT, the first OBJECT is saved in register X0 by $ASSIGN-PRE-AND-POST and the second OBJECT saved in X500 is assigned the node attribute PRECONJELEM with the first OBJECT as its value.

Sometimes a routine starts in a nest of Q-CONJ nodes; the corresponding pre-conjunctional element is not necessarily located on the next higher level. In Fig. 9 when the second LN (a few) is being assigned node attribute PRECONJELEM, PRE-POST-CONJELEM has to go up two Q-CONJ levels to find the corresponding LN. In $GO-LEFT, if the corresponding node is not on the level being searched, $UPCONJ is executed to locate the next level. In the above example, when $GO-LEFT cannot go left from the second NVAR (women), $UP-CONJ goes up to the next higher Q-CONJ and then goes up to the next C-NODE where the search for a corresponding node resumes. Thus LN (a few) is assigned node attribute PRECONJELEM with LN (all the) as its value.

FIGURE 9

Parse tree of All the men and women and a few children were hurt.



In Fig. 8, when the third verb (ignored) is being assigned node attributes by $ASSIGN-PRE-AND-POST, the restriction interpreter goes left from BUTSTG and arrives at the first VERB. The node attributes are chained. Therefore, node attribute POSTCONJELEM of the first VERB should have the second VERB as its value and node attribute POSTCONJELEM of the second verb should have the third VERB as its value. $ASSIGN-PRE-AND-POST gets the last node of the chain after it arrives at the first VERB. This is done by the section of code . . . EITHER ITERATE $POSTCONJ [STARTAT] OR TRUE; . . . . In this case VERB (ignored) is assigned node attribute PRECONJELEM with value VERB (print).

REFERENCES

1. Fitzpatrick, E. and N. Sager, The Lexical Subclasses of the Linguistic String Parser, American Journal of Computational Linguistics; microfiche 2, 1974.
2. Grishman, R., The Implementation of the String Parser of English. In Natural Language Processing, R. Rustin, ed., Algorithmics Press, New York, 1973.
3. Grishman, R., N. Sager, C. Raze, and B. Bookchin, The Linguistic String Parser. Proceedings of the 1973 National Computer Conference, 427-434, AFIPS Press, 1973.
4. Harris, Z.S., String Analysis of Sentence Structure, Mouton & Co., The Hague, 1962.
5. Raze, C., The FAP Program for String Decomposition of Scientific Texts. String Program Reports (S.P.R.) No. 2, Linguistic String Project, New York University, 1967.
6. Sager, N., Syntactic Analysis of Natural Language. Advances in Computers, vol. 8, 153-158, Academic Press, Inc., New York, 1967.
7. Sager, N., A Computer String Grammar of English, S.P.R. No. 4, Linguistic String Project, New York University, 1968.

8. Sager, N., A Computer Grammar of English and Its Applications, to be published by Gordon & Breach in the series Mathematics and Its Applications. Revised from SPR 4 (1968).

9. Sager, N., The String Parser for Scientific Literature. In Natural Language Processing, R. Rustin, ed., Algorithmics Press, New York, 1973.

10. Sager, N. and Ralph Grishman, The Restriction Language for Computer Grammars of Natural Language. Communications of the ACM, 18, 390-400, 1975.

11. Sager, N., Salkoff, M., Morris, J., and Raze, C., Report on the String Analysis Programs, Introductory Volume. String Program Reports No. 1, Linguistic String Project, New York University and University of Pennsylvania, March 1966.

12. Woods, Wm., in Natural Language Processing, R. Rustin, ed., Algorithmics Press, New York, 1973.

ACKNOWLEDGEMENT

This work was supported in part by Research Grants GS2462 and GS27925 from the National Science Foundation, Division of Social Sciences, and in part by Research Grant GN39879 from the Office of Science Information Service of the National Science Foundation.

APPENDIX - The LSP Parsing System

The LSP system obtains a surface structure analysis in the form of a string decomposition of a sentence. In accordance with linguistic string theory (Harris 1962), each sentence is composed of elementary word sequences of a few given types, statable as word class sequences (called linguistic strings). Each sentence contains one center string (an elementary sentence) and zero or more adjunct strings, adjoined to the left or right of elements of the center string or of other adjuncts. In addition, string occurrences may be restricted with regard to the subclasses of words that can co-occur in the same string or in adjoined strings. A string as a whole may also have adjunct strings (called sentence adjuncts) which occur at stated points in the string.

Figure A1 is an example of the computer output of the string decomposition of One rumor hastily printed can ruin careers. Line 2 in Fig. A1 shows that rumors can ruin careers is the center string which has the form of an assertion. Rumor has a left adjunct string LN whose decomposition is shown on line 3 and a right adjunct string RN whose decomposition is shown on line 4. LN consists of the quantifier one. RN is the passive string called VENPASS which consists of the past participle printed preceded by the adjunct hastily. In this example the object position (PASSOBJ) after the verb is null.

To produce a syntactic analysis of natural language sentences the computer program uses two components: a word dictionary (Fitzpatrick and Sager 1974) and an English grammar (Sager 1968), both of which are geared to handle English scientific texts. The word dictionary assigns to each word its major syntactic categories, e.g., noun, verb, adjective, etc., which may in turn have subcategories. The grammar consists

FIGURE A1

Computer Output of the String Decomposition of

One rumor hastily printed can ruin careers.

```
1.  SENTENCE = INTRODUCER CENTER ENDMARK
                                2.

2.  ASSERTION = SA SUBJECT SA TENSE SA VERB SA OBJECT RV SA
                      3. rumor 4.  can    ruin    careers

3.  LN = TPOS QPOS APOS NSPOS NPOS

               one

4.  VENPASS = LVSA VENPASS SA PASSOBJ RV SA
                   4.  printed

5.  DSTG = D
          hastily
```

of two parts: a context-free component and a set of restrictions. The context-free component defines the sets of center and adjunct strings of the grammar. The definitions are written in Backus Normal Form. An example of a string definition is:

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

Each of the elements of ASSERTION is also defined in the grammar. In the above example, SA (sentence adjunct) and RV (post-object right adjunct of verb) are adjunct sets; therefore their occurrence in a sentence is optional. SUBJECT, TENSE, VERB, and OBJECT are positions corresponding to required elements of the string. Each position may have alternate values in different sentences.

The parser analyzes a sentence by building a parse tree for the sentence. The tree represents the particular combination of strings and adjuncts whose terminal nodes combine to produce a well-formed sentence, or more exactly a

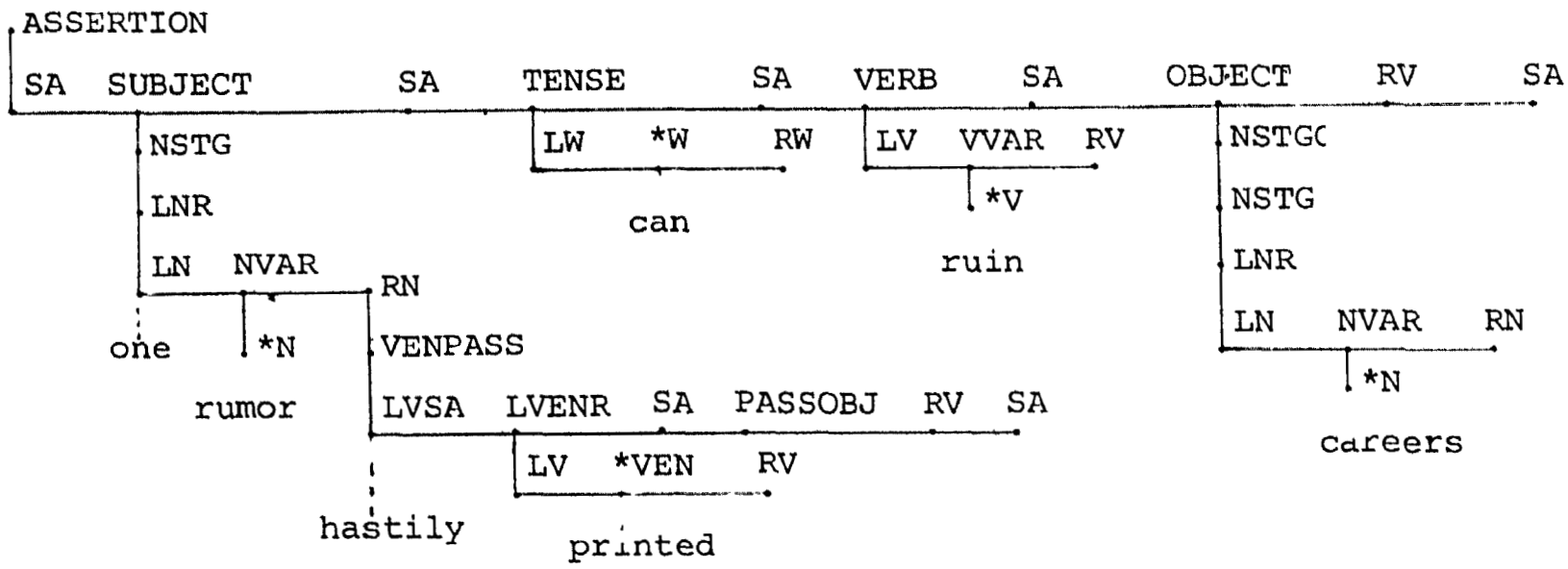
well-formed sequence of word categories which match those of the sentence words. A parse tree of ASSERTION for One rumor hastily printed can ruin careers is shown in Fig. A2. The computer output in Fig. A1 is a compressed version of the parse tree in Fig. A2. In this parse tree the elements of a string are shown as a sequence of connected sibling nodes. Thus the elements of ASSERTION are shown one level below the ASSERTION node, in the order in which they appear in the definition of ASSERTION. The terminal nodes of the tree are either word class symbols which correspond to sentence words (*N = rumors) or are null nodes. The null nodes are automatically satisfied without subsuming sentence words. In adjunct set positions they represent the fact that adjunct occurrences are optional. (In the parse tree diagrams the null nodes are omitted.)

A standard type of structure that is frequently seen in the tree is called the "LXR" node. An example is the LNR node in Fig. A2. An LXR definition consists of three elements: a position for the left adjuncts of X, a core position for the word class X, and a position for the right adjuncts of X. The core position as a rule subsumes a sentence word of class X. For example, in Fig. A2, in the SUBJECT of ASSERTION, NVAR is the core position of LNR and has the value N corresponding to rumor. NVAR (Noun Variants) will have one of several alternate values, namely noun, pronoun, Ving, etc. The "LXR" type structure is important in that both the restrictions and the conjunction mechanism depend on this regularized representation of an element and its adjuncts.

The restrictions are a set of detailed well-formedness rules which must be satisfied before an analysis is accepted. The restrictions may be strictly grammatical, such as one governing the case of a pronoun. Such a restriction will succeed for They ruined careers but not for Them ruined careers. Or the

FIGURE A2

Parse tree* for One rumor hastily printed can ruin careers.



*The terminal nodes which are null are not shown in the parse tree diagrams. In order to keep the tree diagrams as uncluttered as possible the details of certain substructures have been omitted but the sentence word(s) subsumed by those substructures are shown. Three vertical dots below which are sentence words signifies such an omission.

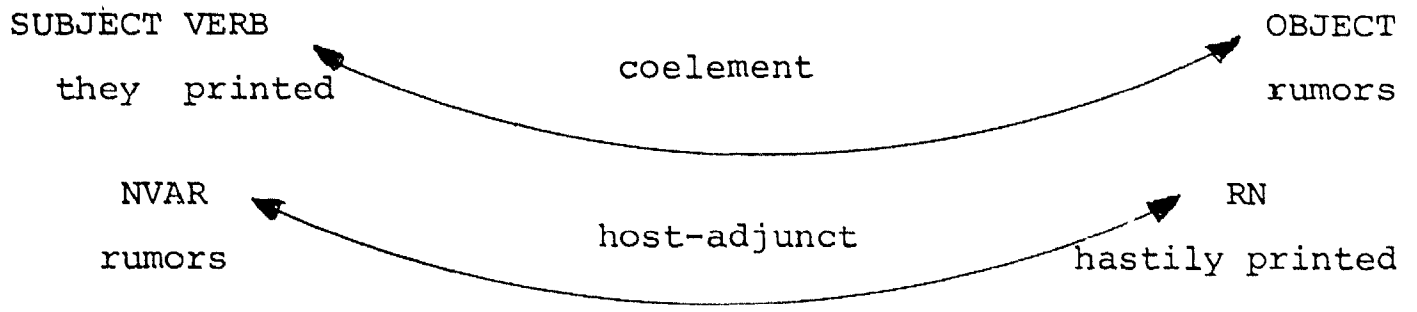
restrictions may express selectional constraints; these will succeed for sequences that are considered possible within a given area of discourse. E.g., in normal discourse one can say They printed rumors but not They printed critics. Each restriction is compiled into a series of basic operations and tests which are performed on the parse tree while parsing a sentence. The restrictions mainly test conditions between two elements of a string or between a string element and its adjunct.

Figure A3 illustrates two of the relations used by restrictions. For the Verb-Object selectional restriction of the grammar to operate on They printed rumors, the coelement relation is used to test whether the co-occurrence of the object (rumors) and the verb (printed) is well-formed. On the other hand, in Rumors hastily printed can ruin careers the host-adjunct relation is used to test the noun rumors and the verb printed since the verb here appears in the

right adjunct string of rumors. Because the restrictions use these basic string relations (coelement, left-adjunct, right-adjunct) so often to test for well formed sequences, these relations are encoded into basic routines (COELEMENT, LEFT-ADJUNCT, RIGHT-ADJUNCT, etc.), which in turn use basic tree operations (up, down, left, right, test for x, etc.). For example, in the LNR sequence subsuming one rumor hastily printed, shown in Fig. A2, the routine RIGHT-ADJUNCT goes from the core noun rumor to its right-adjunct string hastily printed.

FIGURE A3

Restrictions Use String Relations



The use of the basic routines by the restrictions greatly facilitates the formulation of the restrictions. One routine contains many tree operations. The use of these routines also facilitates modification of the grammar. If there is a basic change in the grammar, the affected routines are changed but the restrictions themselves do not have to be. The extensive use of routines by the restrictions plays a significant role in the treatment of conjunctions, as will be described later.

Both the restrictions and the routines are written in a programming language developed for the LSP (Sager and Grishman 1975). The syntax of the restriction language includes three main statement types. One part of the language is similar to a subset of English in that the statements consist of a subject followed by a predicate. For example, THE CORE OF THE SUBJECT IS PRONOUN. Another part of the restriction language consists of logical connectives, such as

IF___THEN___, EITHER___OR___, BOTH___AND___, which permit the logical combination of restriction statements. For example,

IF THE CORE OF THE SUBJECT IS PRONOUN X1
THEN X1 IS NOT ACCUSATIVE.

Another type of restriction language statement consists of a series of commands which are used mainly for writing the routines. A command may consist of a basic tree operation such as GO UP, or a call to execute a restriction routine, such as DO ELEMENT(X), or a call to execute another restriction statement, such as DO $1. There are provisions for saving a node in a register and for restoring a node from a register. For example, STORE IN X1, GO TO X1. The commands may also be logically combined.

BASIC ROUTINES OF THE GRAMMAR

There are about thirty basic routines in the grammar. Described here in detail are the ones which represent the major grammatical relations among words in a sentence and are important in the treatment of conjunctions.

CORE ROUTINE

```
ROUTINE CORE        = DO $CORE-PATH.
$CORE-PATH          = ONE OF $AT-ATOM, $DESCEND-TO-ATOM, $DESCEND-TO-STRING.
$AT-ATOM            = TEST FOR ATOM.
$DESCEND-TO-ATOM    = DESCEND TO ATOM NOT PASSING THROUGH ADJSET1.
$DESCEND-TO-STRING = DESCEND TO STRING NOT PASSING THROUGH ADJSET1.
```

The CORE routine locates the sentence word corresponding to a higher level grammatical element E by descending to a terminal node ("atom") from E. This is done by $DESCEND-TO-ATOM. When CORE descends from E it does not look at structures which are adjuncts, i.e., on list ADJSET1. Thus for One rumor hastily printed can ruin careers shown in Fig. A2, above, the routine CORE,

starting at SUBJECT, will not search below the left-adjunct node LN (arriving mistakenly at one) and will arrive at N (the noun rumor). Sometimes the starting location of CORE will be an atomic node. This is provided for by $AT-ATOM, which tests whether the current node is an atomic node, i.e., on list ATOM. Sometimes a string occurs in a particular sentence in place of a noun. In His printing rumors ruined careers, shown in Fig. 4, the string NSVINGO satisfies the SUBJECT OF ASSERTION. This situation is provided for by $DESCEND-TO-STRING. Thus CORE starting at SUBJECT in Fig. 4, will locate the string NSVINGO.

ELEMENT ROUTINE

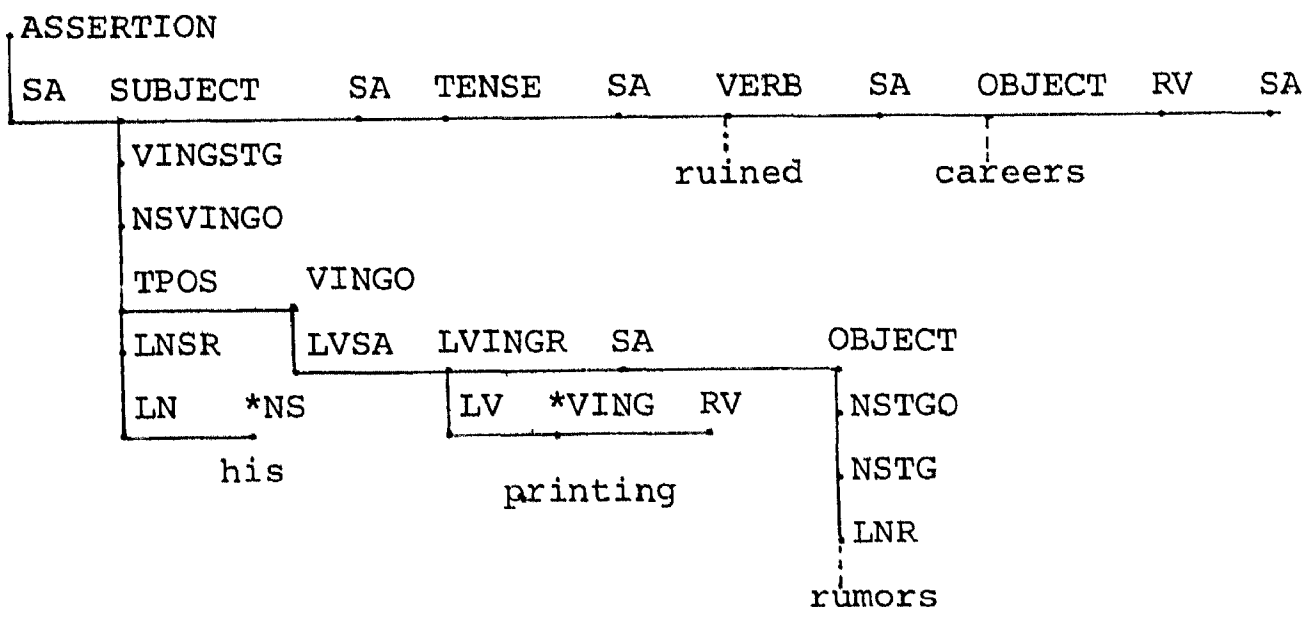
```
ROUTINE ELEMENT(X) = EITHER DO DOWN1(X)
                     OR $STRING-SEGMENT.
$STRING-SEGMENT    = DO DOWN1(STGSEG);
                     DO DOWN1(X).
ROUTINE DOWN1(X)   = GO DOWN; ITERATET GO RIGHT
                     UNTIL TEST FOR X SUCCEEDS.
```

It is assumed that ELEMENT starts at node Y and that X is an element of the string corresponding to Y. Thus ELEMENT locates X by searching the level below Y. This is done by routine DOWN1(X) which first goes to the level below Y by executing the command GO DOWN and then searches the nodes on that level until it finds X. The latter step is accomplished by an iterate command: ITERATET GO RIGHT UNTIL TEST FOR X SUCCEEDS.* In Fig. A2 the execution of ELEMENT(SUBJECT), starting at the string node ASSERTION, will locate the node SUBJECT. Sometimes it is more convenient to define a string Y by using the name of another string X in the definition of Y instead of naming all the elements of Y. This is the case in Fig. A4 where VINGO, a defined string of the grammar appears as a string segment of NSVINGO. In this situation not all the elements of NSVINGO are on one level below NSVINGO but some are one level

*ITERATET is an ITERATE command in which the exit test (the part written after UNTIL) is the first operation performed.

FIGURE A4

Parse tree of His printing rumors ruined careers.



below the string segment VINGO of NSVINGO. $STRING-SEGMENT, therefore, searches one level below Y for a node on the string segment list STGSEG and if it finds one, it searches for X one level below the string segment. In Fig. A4, ELEMENT(OBJECT), starting at NSVINGO, first locates VINGO by executing DOWN1(STGSEG) and then locates OBJECT by executing DOWN1(OBJECT).

COELEMENT ROUTINE

```
ROUTINE COELEMENT(X) = ONE OF $SAME-LEVEL, $X-IN-SEGMENT, $Y-IN-SEGMENT.
$SAME-LEVEL          = DO COEL1(X).
$X-IN-SEGMENT        = DO COEL1(STGSEG);
                       DO ELEMENT(X).
$Y-IN-SEGMENT        = GO UP;
                       TEST FOR STGSEG;
                       DO COEL1(X).
ROUTINE COEL1(X)     = EITHER DO LEFTR(X)
                       OR DO RIGHTR(X).
ROUTINE LEFTR(X)     = ITERATE GO LEFT UNTIL TEST FOR X SUCCEEDS.
ROUTINE RIGHTR(X)    = ITERATE GO RIGHT UNTIL TEST FOR X SUCCEEDS.
```

Given that X and Y are elements of some string, COELEMENT starts at Y and goes to X. COELEMENT uses several other basic routines: ROUTINE LEFTR(X) goes.

left from Y until it locates X; ROUTINE RIGHTR(X) searches to the right of Y to locate X; and combining the two, ROUTINE COELl(X) searches both sides of Y to find X. In Fig. A4 COELEMENT(SUBJECT), starting at VERB in ASSERTION, locates SUBJECT by executing $SAME-LEVEL. ROUTINE LEFTR(SUBJECT) successfully locates SUBJECT, which is to the left of VERB; this satisfies COELl(SUBJECT), which satisfies $SAME-LEVEL. If X is in a string segment, COELEMENT will locate it by executing $X-IN-SEGMENT. In Fig. A4, COELEMENT(OBJECT), starting at TPOS, first locates VINGO by executing COELl(STGSEG). It then locates OBJECT by calling routine ELEMENT(OBJECT). A different situation occurs when COELEMENT(TPOS) starts at OBJECT. This situation is handled by $Y-IN-SEGMENT. First, the string segment VINGO is located by the sequence GO UP; TEST FOR STGSEG. Then TPOS is located by routine COELl(TPOS), which searches the same level as VINGO to find TPOS.

RIGHT-ADJUNCT ROUTINE

```
ROUTINE RIGHT-ADJUNCT     = DO RIGHT-ADJUNCT-POS;
                            DO CORE.
ROUTINE RIGHT-ADJUNCT-POS = EITHER $ASCNT OR TRUE;
                            DO RIGHTR(RADJSET).
$ASCNT                    = ASCEND TO AVAR OR NVAR OR QVAR OR VVAR.
```

It is assumed that RIGHT-ADJUNCT starts at the core of an LXR type node. It goes to the core of the right-adjunct position in the LXR sequence. For example, in the LXR sequence LN NVAR RN (one rumor hastily printed), shown in Fig. A4 the routine RIGHT-ADJUNCT-POS, starting at N (rumors) ascends to NVAR by executing $ASCNT and then goes to RN by executing the routine RIGHTR(RADJSET). RIGHT-ADJUNCT goes to VENPASS by executing the CORE routine. Thus the passive string VENPASS (hastily printed) is located as the right adjunct of the noun rumor.

LEFT-ADJUNCT ROUTINE

```
ROUTINE LEFT-ADJUNCT = DO LEFT-ADJUNCT-POS;
                       EITHER TEST FOR LN
                       OR DO CORE.
ROUTINE LEFT-ADJUNCT-POS = EITHER $ASCNT OR TRUE;
                           DO LEFTR(LADJSET).
```

LEFT-ADJUNCT is similar to RIGHT-ADJUNCT except it goes to the core of the left-adjunct position of an LXR sequence. If the left-adjunct position is LN, however, the routine stops there since it is assumed that further operations will be specified to locate a particular left adjunct of the noun, e.g. a quantifier or an adjective.

HOST ROUTINE

```
ROUTINE HOST = CORE OF THE HOST-ELEMENT EXISTS.
ROUTINE HOST-ELEMENT = ONE OF $AT-LADJ, $AT-RADJ, $AT-RNSUBJ IS TRUE.
$AT-LADJ = EITHER TEST FOR LADJSET
           OR ASCEND TO LADJSET;
           GO RIGHT.
$AT-RADJ = EITHER TEST FOR RADJSET
           OR ASCEND TO RADJSET;
           EITHER $RV-IN-STRING
           OR GO LEFT.
$RV-IN-STRING = TEST FOR RV;
                STORE IN X100;
                GO UP;
                TEST FOR TYPE STRING:
                EITHER $RV-IN-OBJECT OR $RV-IN-CENTER.
$RV-IN-OBJECT = TEST FOR NTOVO OR NTHATS OR NSNWH OR PNTHATS OR PNTHATSVO
                OR PNSNWH;
                ASCEND TO OBJECT;
                DO VERB-COELEMENT.
$RV-IN-CENTER = GO TO X100;
                DO VERB-COELEMENT.
```

```
$AT-RNSUBJ = EITHER TEST FOR RNSUBJ
             OR ASCEND TO RNSUBJ;
             ASCEND TO SA;
             DO COELEMENT(SUBJECT).
```

It is assumed that the routine HOST-ELEMENT starts at node Y, which is in or at an adjunct position in an LXR structure. It goes from the adjunct position to the core position of the LXR structure. HOST then goes to the CORE of the node located by HOST-ELEMENT. For example, consider the operation of the HOST routine on the parse tree shown in Fig.A2 where HOST starts at the right-adjunct string VENPASS = hastily printed. $AT-LADJ fails but $AT-RADJ ascends to RN by executing ASCEND TO RADJSET and goes left to NVAR. The CORE routine then locates N (rumors). A similar situation occurs when HOST starts at LN. TEST FOR LADJSET succeeds and HOST-ELEMENT goes one node to the right to NVAR. HOST calls the CORE routine which locates N (rumors).

If the HOST routine starts in or at RV (right adjuncts of verb), extra maneuvering is necessary to locate the verb. There are three possibilities. RV may immediately follow the verb as in He ran quickly, shown in Fig. A5. In this case, $RV-IN-STRING fails because the node above RV (i.e., VERB) is not on the string list. HOST-ELEMENT therefore goes left to VVAR, whose core is ran. In some cases, RV follows the object of the verb as in He ran to school quickly, shown in Fig. A6. In this case, $RV-IN-STRING succeeds. The node above RV is the string ASSERTION. $RV-IN-CENTER locates the verbal element VERB (ran) of ASSERTION by calling routine VERB-COELEMENT (a generalized routine for finding a verbal coelement). In other sentences, RV is situated in the middle of an object string. In She told him quickly that he had to leave, shown in Fig. A7 , RV is situated after NSTGO (him) in the object NTHATS of the verb told. In this situation $RV-IN-OBJECT locates the verbal element of ASSERTION by ascending to OBJECT and calling routine VERB-COELEMENT.

FIGURE A5

ASSERTION parse tree of He ran quickly.

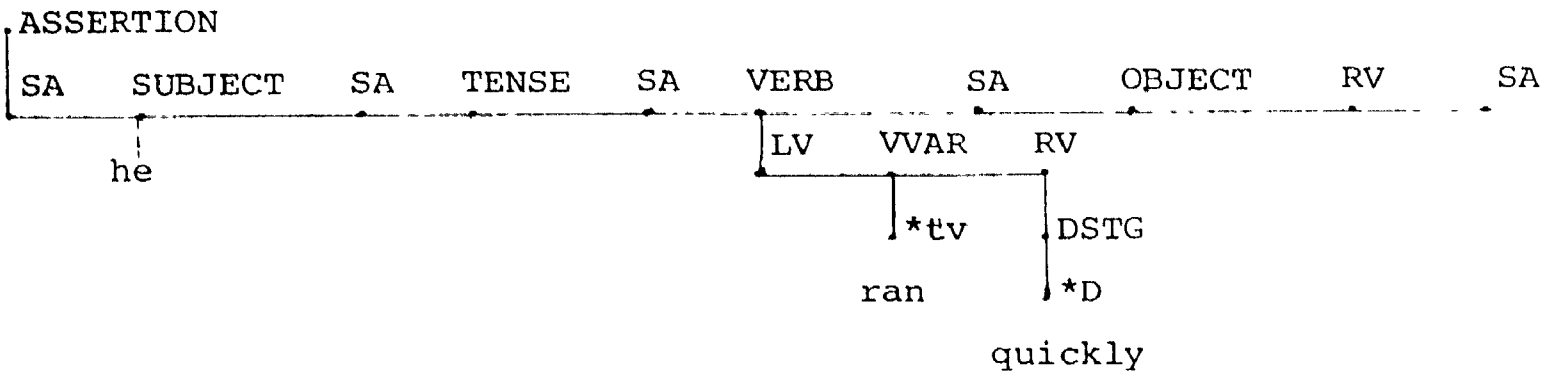


FIGURE A6.

ASSERTION parse tree of He ran to school quickly

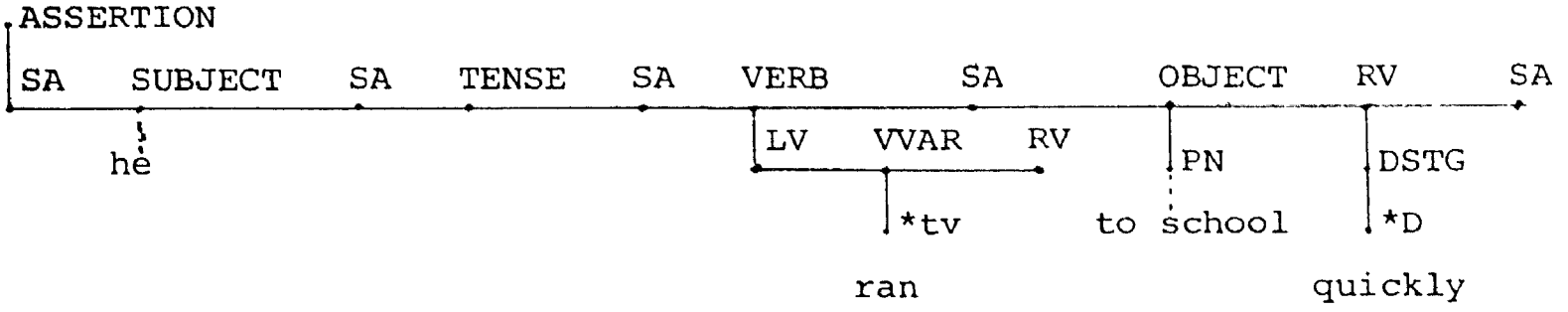
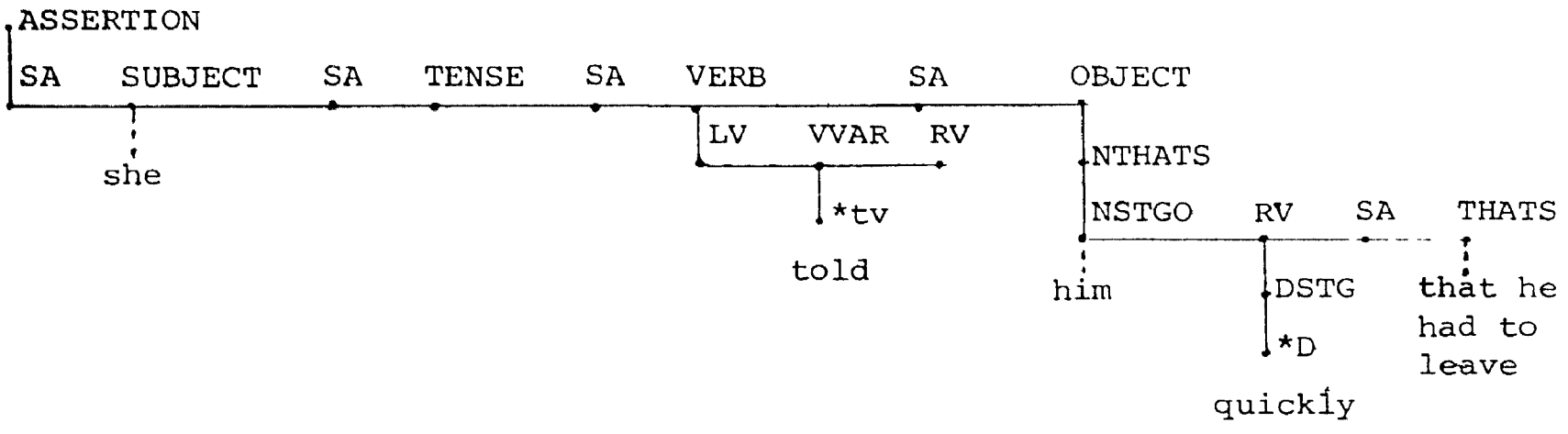


FIGURE A7.

ASSERTION parse tree of She told him quickly that he had to leave.

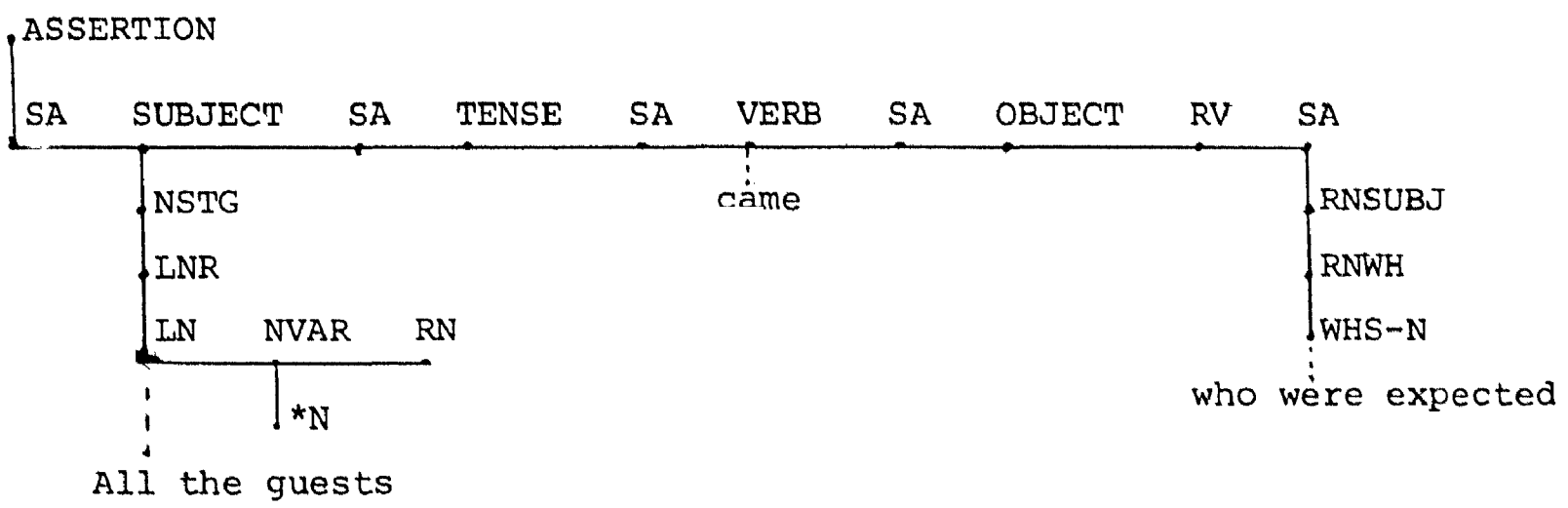


A similar situation arises in sentences where the right adjunct of the subject noun does not immediately follow the noun. In the grammar these occurrences are covered by RNSUBJ in the post-OBJECT sentence adjunct position SA, as shown in Fig. A8. In All the guests came who were expected (Fig. A8),

FIGURE A8

ASSERTION parse tree for

All the guests came who were expected.



the right adjunct of guests is WHS-N (who were expected). $AT-RNSUBJ . locates the position SUBJECT by ascending to SA from RNSUBJ and by calling COELEMENT(SUBJECT) from SA. The HOST routine uses CORE to locate N (guests).

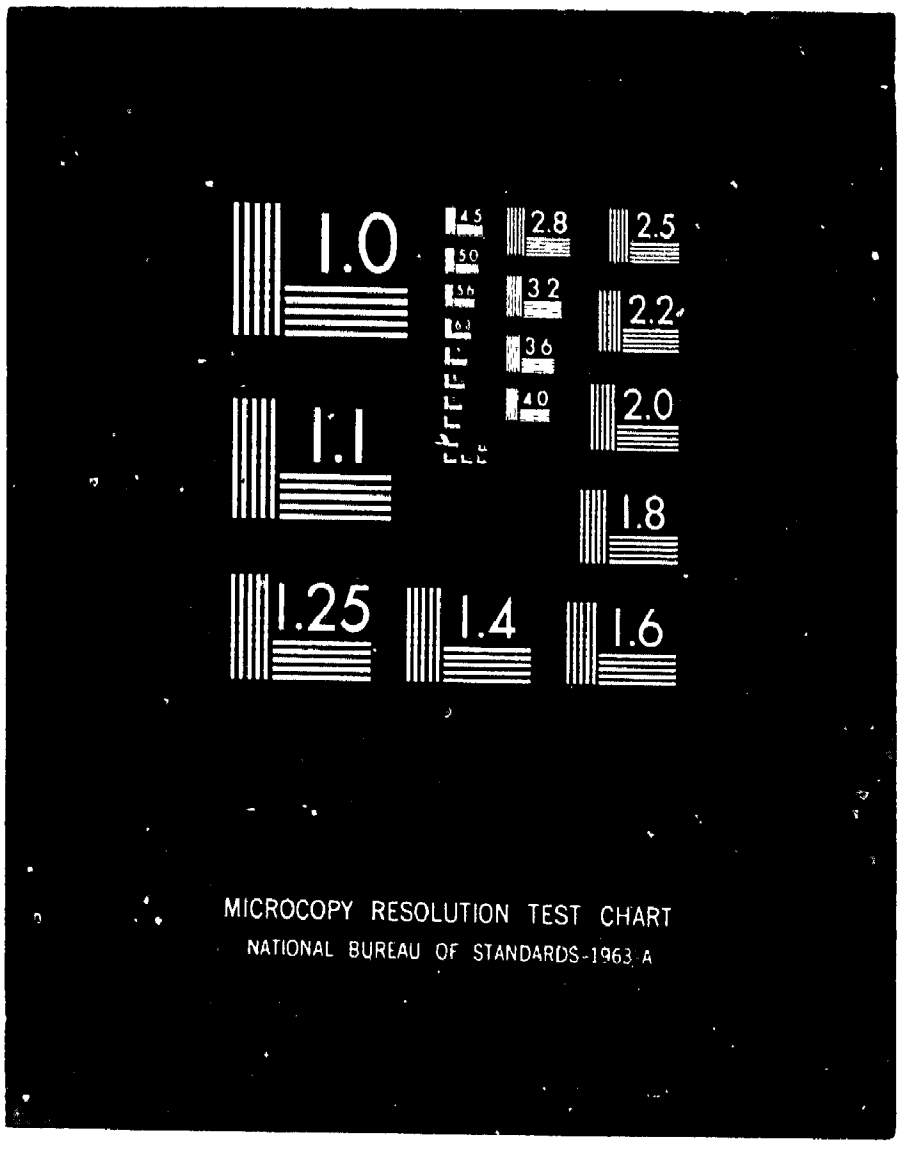
STARTAT ROUTINE

When a node name appears alone (without any routine name) as the subject of a restriction statement, e.g., OBJECT in the restriction statement OBJECT IS EMPTY, the routine STARTAT is invoked, with the node name as argument, e.g., STARTAT(OBJECT).

ROUTINE STARTAT(X) = EITHER DO DOWN1(X) OR TEST FOR X.

It is assumed that X is one level below the current node in which case DOWN1(X) locates X, or that X is the current node in which case TEST FOR X is, successful.

END


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963 A