5. THE RESTRICTIONS: WHAT THEY LOOK AT

Each restriction states a condition on the parse tree and the words of the sentence and their attributes which must be met before the tree is accepted as an analysis of the sentence. These restrictions must be written in a special restriction language which will be described in the following sections.

The restriction language differs in several respects from most other programming languages with which the reader may be familiar. Most strikingly, the external form of this language is declarative rather than procedural. That is, the restriction states a condition, rather than explicitly indicating the sequence of operations which must be performed to determine if the condition is true. This distinction, however, is more one of form than of substance. The grammar compiler translates the declarative statement into a sequence of operations which test the parse tree and the attributes of words, and the grammar writer should in general be aware of the order in which these operations are performed. Nonetheless, we believe that the declarative format of the restrictions does make them easier to write and read.

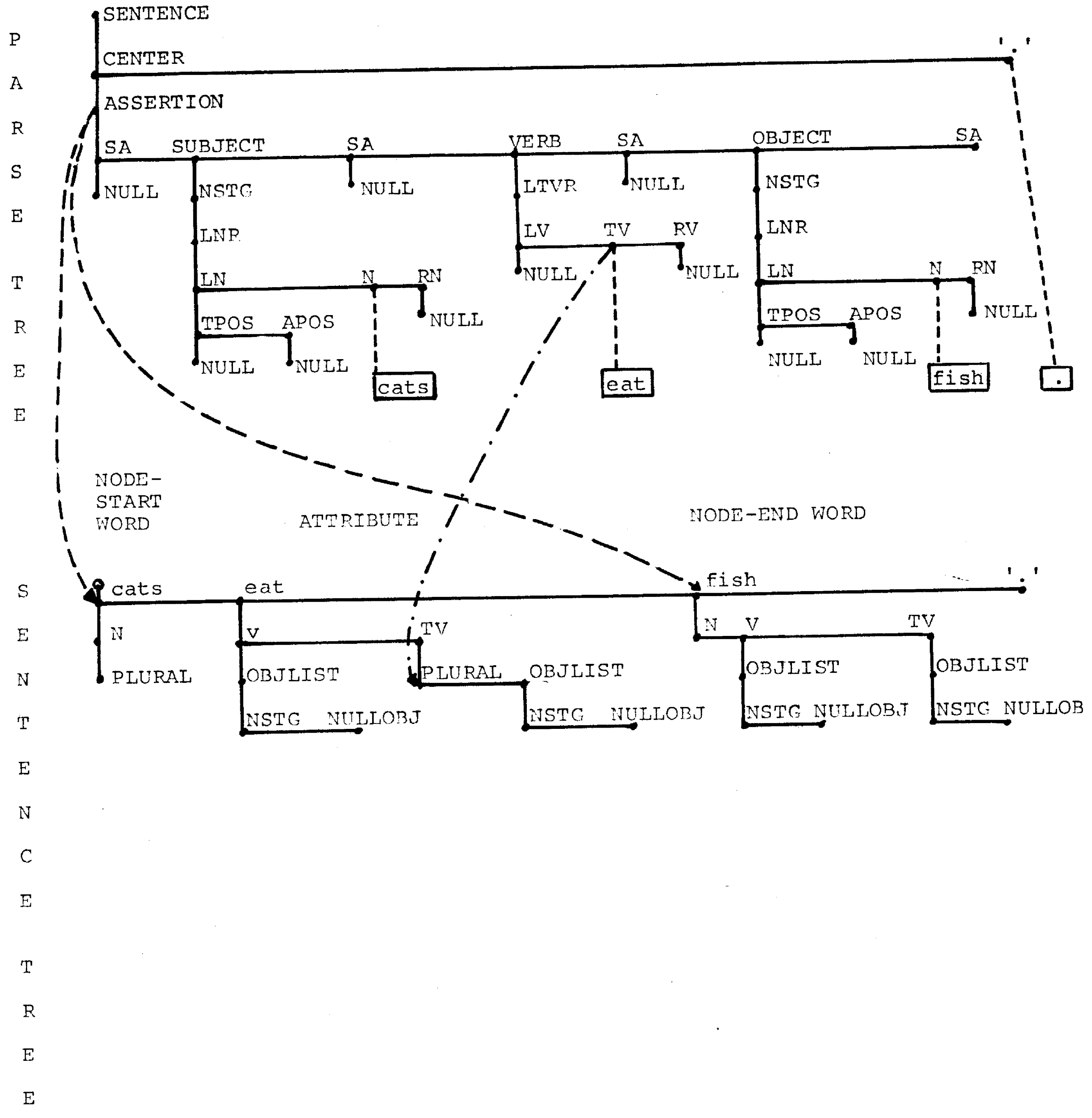
A more fundamental difference of the restriction language lies in the types of data objects it uses. Most programming languages work with numbers and character strings; programs in these languages are able to build up and manipulate arrays and lists of such items. In contrast, restrictions don't build up anything. All the data structures--the parse tree, the sentence words and their trees of categories and attributes--are already there when the restriction begins to execute. The restrictions are able only to move around and test these trees. More formally, we might say that the only objects the restriction language can manipulate are pointers to the parse tree, sentence, and attribute lists.

Most of the operations performed in the execution of a restriction operate on one distinguished pointer, moving it around the parse tree, from the parse tree to the sentence words and their attributes, and around on the attribute trees. Since we intend to become on familiar terms with the restrictions, we can't keep on saying that "the restriction pointer is now pointing to... ." We shall simply say instead that the restriction "is looking at..." or "is at..." or, when we really get into the spirit of the thing, that we are looking at... or are at... .

We may consider the forest within which the restriction moves as consisting of only two trees: a parse tree and a sentence tree (see Fig. 2). The form of the parse tree was described above (in § 3.). In the sentence tree the elements on the level immediately below the root node are the words of the sentence. Below each word is the list of its word categories (the last word, the period, has no categories). Hanging below each word category is the subtree of its attributes, whose structure was described in § 4.

The restriction language includes a number of operations for moving about the parse and sentence trees. In addition, it provides a few operations for going from points in the parse tree to points in the sentence tree. NODE-START WORD and

Fig. 2



P A R S E T R E E
SENTENCE
CENTER
ASSERTION
SA
SUBJECT
SA
VERB
SA
OBJECT
SA
NULL
NSTG
NULL
LTVR
NULL
NSTG
LNR
LV
TV
RV
LNR
LN
N
RN
NULL
NULL
LN
N
RN
TPOS
APOS
NULL
TPOS
APOS
NULL
NULL
NULL
NULL
NULL
cats
eat
fish
.
'.'
NODE-START WORD
ATTRIBUTE
NODE-END WORD
S E N T E N C E T R E E
cats
eat
fish
'.'
N
PLURAL
V
TV
OBJLIST
NSTG
NULLOBJ
PLURAL
OBJLIST
NSTG
NULLOBJ
N
V
TV
OBJLIST
NSTG
NULLOBJ
OBJLIST
NSTG
NULLOB

NODE-END WORD go from a node in the parse tree to the first and last words below that node in the parse tree, respectively. For instance, in Figure 2, starting from the node ASSERTION, these two operations would go to the words "cats" and "fish". The HAS ATTRIBUTE operation, which is valid only if the restriction is looking at a non-null atomic node, goes to an attribute of the word category matched by that node. For example, starting from the TV node on the parse tree, the phrase "HAS ATTRIBUTE OBJLIST" would cause the restriction to look at the OBJLIST below TV in the subtree for "eat". All these operations will be discussed in detail later in this volume.

In general, it would be possible to execute all restrictions after each (context-free) parse tree of the sentence had been generated by the parser (as described in § 3). It is, however, much more efficient to execute each restriction as early in the parsing process as possible, so that many fruitless parse trees need not be completed. The grammar writer is therefore allowed to specify, through a phrase called the <u>housing</u> at the beginning of the restriction, the time when a restriction is to be executed.

This phrase names the BNF definition (or definitions) in which the restriction is to be "housed". If the definition has several options, it indicates whether the restriction is to apply to one or all the options. In addition, the housing may specify the point in time during the building of the subtree for this definition that the restriction is to be executed. The restriction may be executed before any elements of the definition have been attached to the tree (this is called a "disqualify" restriction) or after some or all of the elements have been completed (this is called a "wellformedness" restriction).

For example, the restriction for subject-verb number agreement in ASSERTION could be executed as soon as the VERB position was completed, so its housing would be

IN ASSERTION AFTER VERB

In contrast, the restrictions correlating the verb and its object could not be executed until the OBJECT position was completed, so their housing would be

IN ASSERTION AFTER OBJECT

Disqualify restrictions are used most frequently to look ahead in a sentence and see whether an option is worth trying at all. For instance, there is a restriction on the QUESTION option of CENTER (which is not included in our small grammar) which checks whether the sentence endmark is indeed a question mark. To avoid needless complications at a time when we are trying to grasp the basics of the restriction language, we shall make all of our restrictions wellformedness restrictions to be executed when all the elements of the definition have been completed. The refinements of housing will be left for a later section of this volume.

We indicate by the first letter of the name of the restriction whether it is to be a disqualify or wellformedness restriction; D for disqualify and W for wellformedness. Except for this rule, the restriction name is entirely arbitrary. It may consist of a W or D followed by up to 19 letters or digits. We shall normally try to use names of some mnemonic value, but you are free to use any name that meets your fancy for the restrictions you write.

By default (i.e., unless explicitly stated otherwise) the wellformedness restriction is executed after all elements of the definition in which it is housed have been completed. If a definition has several options, one option can be specified by adding AFTER OPTION ____ to the housing. For example, a restriction beginning

```
WNULLO = IN OBJECT AFTER OPTION NULLOBJ:
```

would be executed after NULLOBJ has been attached below OBJECT. If no option were specified

```
WOBJ = IN OBJECT:
```

the restriction would be executed when OBJECT was completed, regardless of its value.

As these examples indicate, the format of a restriction is

restriction-name = housing : body-of-restriction.

We shall now turn our attention to the body of the restriction.

6. THE RESTRICTIONS: BASIC STATEMENT FORMS

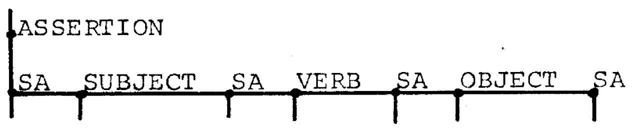
The basic restriction statement consists of a subject followed by a predicate:

WT1 = IN ASSERTION: OBJECT IS EMPTY.

WT1A = IN ASSERTION: $\underbrace{\text{OBJECT}}_{\text{subject}}$ $\underbrace{\text{HAS VALUE NULLOBJ}}_{\text{predicate}}$

When the restriction is executed, it first locates the subject and then tests whether or not the predicate is true. If it is true, the restriction succeeds and the parse continues as if the restriction had not been there. If the predicate is false, the restriction fails and the parser is forced to back up and try to find some alternate analysis for ASSERTION.

The simplest type of subject is just the name of a node. When a restriction begins, it is looking at the node in which it is housed (at ASSERTION in the above examples). The subject may name either the starting node or any node on the level immedi tely below. Thus, a restriction housed in ASSERTION may have as its



subject, either ASSERTION, SA, SUBJECT, VERB, or OBJECT (the level below the starting node is searched from left to right, so the subject SA would bring you to the leftmost SA).

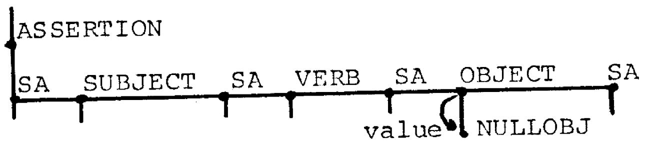
The predicate IS EMPTY checks that the current node (the node specified by the subject of the restriction) does not subsume any words of the sentence. In other words, a node IS EMPTY if and only if all the terminal nodes below it in the tree are null atomics (either NULL or NULLOBJ in our grammar).

Put a simple subject together with this predicate and we get a restriction like WT1 (above). This is clearly a rather silly restriction. Neither NSTG nor THATS can be empty, so they are in effect ruled out as options of OBJECT, and we are left with only the option NULLOBJ. If that is what we wanted, we could have written

<OBJECT> ::= <NULLOBJ>

in the first place. Until we develop a larger restriction language vocabulary, we must ask the reader to bear with us in our linguistically useless examples.

Most restrictions, and particularly useless ones, can be written in more than one way. The restriction WT1A (above) is equivalent to WT1 for our small grammar. The VALUE of a given node is the node immediately below the given node. Thus the predicate HAS VALUE NULLOBJ checks whether the node immediately below the current node is named NULLOBJ. This restriction therefore rules out options NSTG and THATS of OBJECT, just like WT1.



While we're at it, we could also rewrite WT1 as

WT1B = IN OBJECT: OBJECT IS EMPTY

or WT1C = IN OBJECT: OBJECT HAS VALUE NULLOBJ.

These restrictions would be executed somewhat earlier in the parsing process than the two mentioned before, namely, as soon as the OBJECT node was completed. They would also be slightly faster, since the node specified as the subject is the node at which the restriction begins -- the restriction does not have to search one level below the starting node. Note, however, that these two restrictions are equivalent to our first two only because OBJECT is used only in ASSERTION in our small grammar. In the full grammar OBJECT is used in several string definitions, so WT1B and WT1C would restrict OBJECT wherever it was used whereas WT1 and WT1A would only restrict OBJECT when it appears in ASSERTION.

In an effort to exhaust the subject (and perhaps the reader) we shall consider one more variant:

WT1X = IN ASSERTION: VALUE OF OBJECT IS NULLOBJ.

In terms of the operations which will be performed when the restriction is executed, this restriction is identical to WT1A. What we have done is simply to shift the task of finding the VALUE from the predicate to the subject of the restriction sentence. In doing so we have introduced a new subject form, VALUE OF node, which first finds node (which, as before, must either be the starting node or on the level immediately below) and then goes to the node immediately below, and a new predicate form, IS node, which tests whether the current node is named node.

As we already see, most restrictions are of the form 'go somewhere on the tree and make a test.' The restriction will fail not only if the test fails but also if it is not possible to go where the restriction asks. For instance,

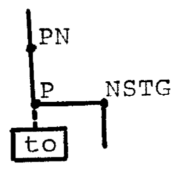
WILLFAIL1 = IN ASSERTION: NULLOBJ IS EMPTY.

will always fail, even if there is a NULLOBJ on the tree, because it will not be on the level immediately below ASSERTION. Or,

WILLFAIL2 = IN LNR: VALUE OF N IS SENTENCE.

will fail because N is an atomic node and hence has no value (there is nothing below it on the tree).

Although we can't take the VALUE of an atomic node, there are a few interesting things we can find out about it. For one thing, we can test for the actual word in the sentence which has been matched. Thus



WT2 = IN PN: P IS 'TO'.

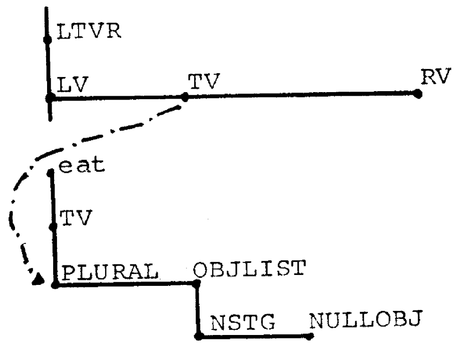
restricts us to prepositional phrases beginning with the word TO. We can also test if the matched word has a particular attribute:

WT3 = IN LTVR: TV HAS ATTRIBUTE PLURAL.

This restriction verifies that, in the sentence word matching the atomic TV, one of the attributes on the level immediately below the category TV is PLURAL. The HAS ATTRIBUTE operation will not search more than one level of the attribute tree of its own accord; for example,

WT4X = IN LTVR: TV HAS ATTRIBUTE NSTG.

will fail for the tree shown at right, even though NSTG is in the attribute subtree of TV. We are obliged to specify which attribute the restriction should look for on each level of the tree. In the example just cited, we



would have to tell the restriction first to look for the attribute OBJLIST on the level below TV and then to look for NSTG below OBJLIST. We indicate this in restriction language by writing the successive elements to be looked for one after the other, separated by colons, thus:

WT4 = IN LTVR: TV HAS ATTRIBUTE OBJLIST: NSTG.

On the principle that there should be at least two ways of saying anything, the restriction language allows "HAS ATTRIBUTE" to be replaced by "IS". Thus we could rephrase WT3 as

WT3REPHRASED = IN LTVR: TV IS PLURAL.

and, though it may sound less "natural,"

    WT4REPHRASED = IN LTVR: TV IS OBJLIST: NSTG.

One final note for this chapter: the test made by any predicate may be inverted by changing IS to IS NOT and HAS to DOES NOT HAVE; e.g.,

    WT1N = IN ASSERTION: OBJECT DOES NOT HAVE VALUE NULLOBJ.

    WT4N = IN LTVR: TV IS NOT OBJLIST: NSTG.

The last restriction will succeed if either the category TV of the word does not have the attribute OBJLIST or OBJLIST does not have the attribute NSTG. A restriction with NOT will succeed if the predicate of the restriction without NOT would fail for any reason. This leads to the seemingly odd but perfectly logical situation where

    WODD1 = IN LNR: VALUE OF N IS NOT SENTENCE.

will fail because the operation in the subject (taking the value of an atomic) fails, whereas

    WODD2 = IN LNR: N DOES NOT HAVE VALUE SUBJECT.

will succeed, because the operation in the predicate of computing the value fails. This is an indication that, while the restriction language is designed to make the meaning of "reasonable" restrictions quite transparent, the freedom of expression it provides can be perverted in the cause of deception and confusion.

Let us close this chapter by listing the subjects and predicates we have introduced so far:

$$\begin{array}{cc} \text{SUBJECT} & \text{PREDICATE} \\ \left\{\begin{array}{c}\text{node}\\ \text{VALUE OF node}\end{array}\right\} & \left(\begin{array}{ll} \left\{\begin{array}{l}\text{IS}\\ \text{IS NOT}\end{array}\right\} & \left\{\begin{array}{l}\text{EMPTY}\\ \text{node}\\ \text{text (word in quotes)}\\ \text{ATTRIBUTE [:attribute]...}\end{array}\right\} \\ \left\{\begin{array}{c}\text{HAS}\\ \text{DOES NOT HAVE}\end{array}\right\} & \left\{\begin{array}{l}\text{VALUE node}\\ \text{ATTRIBUTE [:attribute]...}\end{array}\right\}\end{array}\right\} \end{array}$$

Items written one above the other and grouped together with braces are alternative forms for a component (subject or predicate) of a restriction language statement.

7. THE RESTRICTIONS: THE STRING RELATIONS

The trouble with rewriting the string grammar in BNF is that simple relations between strings in a sentence may turn into rather complicated relationships among nodes in the parse tree. A major objective of the restriction language is to obviate this difficulty by including as a basic part of the language those operations on the parse tree which correspond to string relations. In contrast to the primitives introduced in the previous chapter, VALUE and HAS ATTRIBUTE, these operations may search through several levels of the parse tree.

Consider for example the restriction on subject-verb number agreement in its simplest form, as a check on the SINGULAR or PLURAL attributes of the noun or pronoun and the tensed verb. In string theory these are attributes of two successive categories in a center string. In our parse tree, however, these attributes