American Journal of Computational Linguistics *Microfiche 26*

JUNCTION GRAMMAR

AS A BASE FOR
NATURAL LANGUAGE PROCESSING

ELDON G. LYTEL
DENNIS PACKARD
DARYL GIBB
ALAN K. MELBY
FLOYD H. BILLINGS, JR.

Brigham Young University
Provo, Utah

Copyright 1975 by the Association for Computational Linguistics

Abstract

Junction Grammar, a model of language structure developed by Eldon Lytle, is being used to define the interlingua for a machine-assisted translation project. Junction Grammar representations (called junction trees) consist of word sense information interrelated by junctions, which contribute syntactic and semantic information. The first step of the current translation system is interactive analysis. During this step, the program interacts with the human operator to resolve ambiguities and then produces a junction tree representation of the meaning of the input text. The second and third steps of the translation process are automatic transfer and synthesis into one or more target languages. For each target language the transfer step makes adjustments on each junction tree, if needed, before sending it to the synthesis program for that language. This translation system is currently under development at Brigham Young University in Provo, Utah. Present lexicons for English analysis and Spanish, German, French, and Portuguese synthesis contain about 10,000 word senses each.

TABLE OF CONTENTS

LIST OF SYMBOLS

N	Noun
PN	Predicate with a noun nucleus
SN	Predication with a noun nucleus
A	Adjective or adverb
PA	Predicate with an adjective or adverb nucleus
SA	Predication with an adjective or adverb nucleus
V	Verb
PV	Predicate with a verb nucleus
SV	Predication with a verb nucleus
P	Preposition
PP	Predicate with a preposition nucleus
SP	Predication with a preposition nucleus
E	Empty node
JG	Junction Grammar
Adj	Adjective
Adv	Adverb
PA_j	Adjective predicate
PA_v	Adverb predicate
PX	Any predicate
SX	Any predication

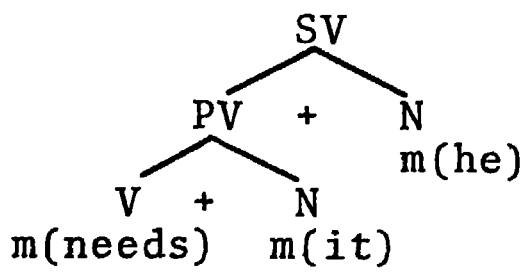
I. An overview.

Assuming that semantics must be taken into account in a translation system, there are at least two major classes of intermediate representations to choose from for use in a translation system. One class of representations is non-linguistic. In this class syntax is ignored and use is made of various conceptual units and relations (e.g. Schank's conceptual dependency). Another class of representations is linguistic. In this class syntax and semantics are combined into a syntacto-semantic representation which is connected to surface structure in a fairly direct fashion (e.g. Montague grammars [1] and Junction Grammar [2]).

We have chosen to test Junction Grammar as the interlingua for a machine-assisted translation project. The basic concepts of this model of language were developed by Eldon Lytle in the late 1960's, but, as with many theories, it has been under continual development ever since.

For subject-verb-object sentences, junction trees look much like the familiar phrase markers. Let us then preview the steps of the Junction Grammar translation process using an extremely simple sentence.

Given the English surface string, "He needs it", English analysis would produce the junction tree:



Note that the tree has an order independent of the surface word order of the input text. The function m(x) produces the word sense or sememe of its argument. The values of this function have been implemented as positive integers called semantic indices. The plus signs under the categories PV (verbal predicate) and SV (verbal sentence) stand for adjunction, one of the basic types of junction recognized by Junction Grammar.

Suppose we want to synthesize this junction tree into French. Transfer would adjust this tree because French expresses the meaning of 'need' as 'have need of'. Then French synthesis would perform lexical selection and ordering to produce the string "Il a besoin de ca", or "Il en a besoin".

The following chapters will discuss junction trees in more detail and then explain how the analysis, transfer, and synthesis steps are currently implemented.

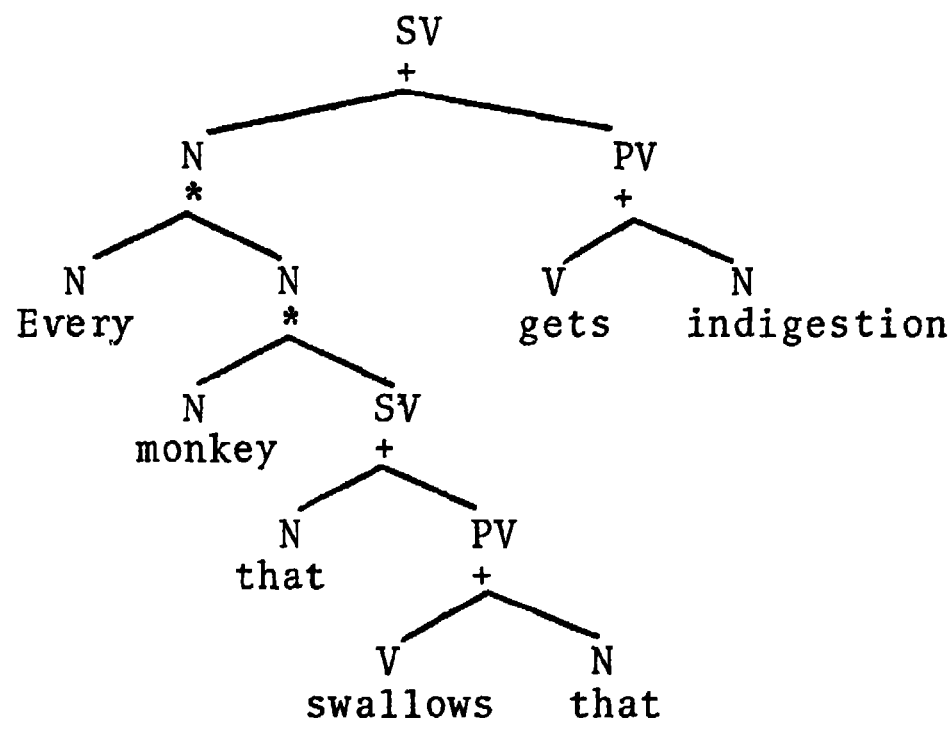
II. Introduction to Junction Grammar.

A. Basic Relative Modifiers.

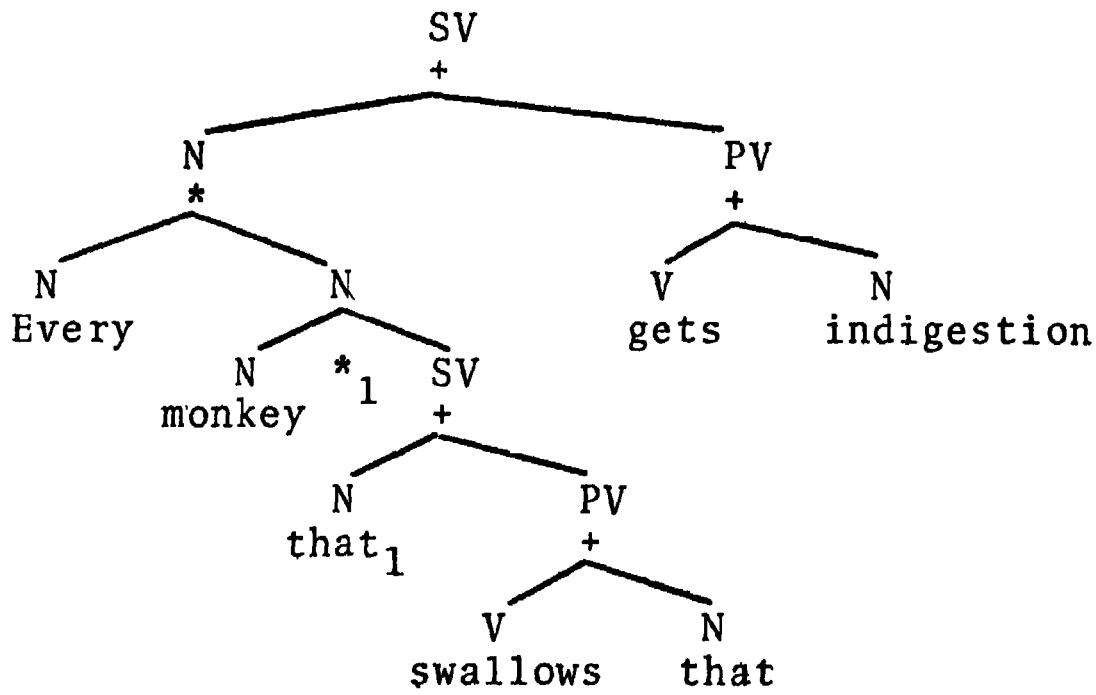
Consider the following sentence:

(1) Every monkey that swallows that, gets indigestion.

Utilizing a modification junction * called subjunction, consider the following as a junction marker for (1): (Consider the terminal words as names of sememes and consider quantifiers to be of a noun-type category).



The problem with the above junction marker is that it is not clear which <u>that</u> is the relative pronoun. Following Montague (see also Partie [3] and Gabbay [4]) we could subscript the subjunction operation and the appropriate <u>that</u> with respect to which the modification occurs:

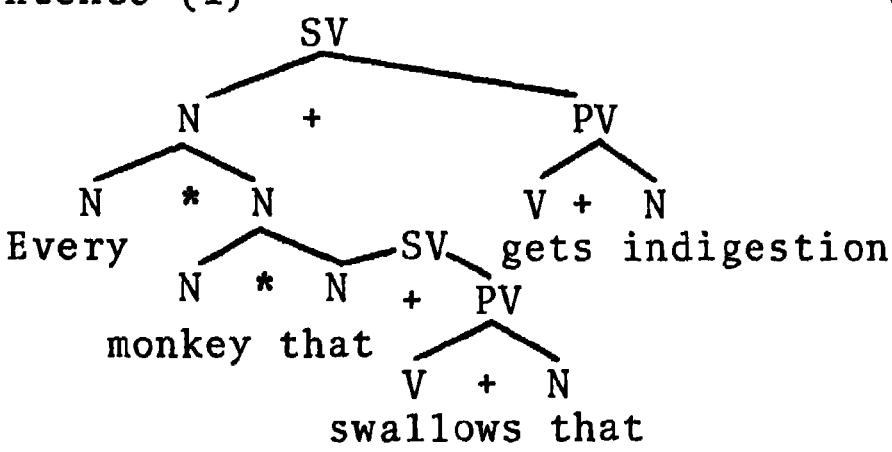


The above tree would then be lexicalized as sentenc (1). If we had subscripted just the other that then the marker would be lexicalized not as sentence (1) but rather as:

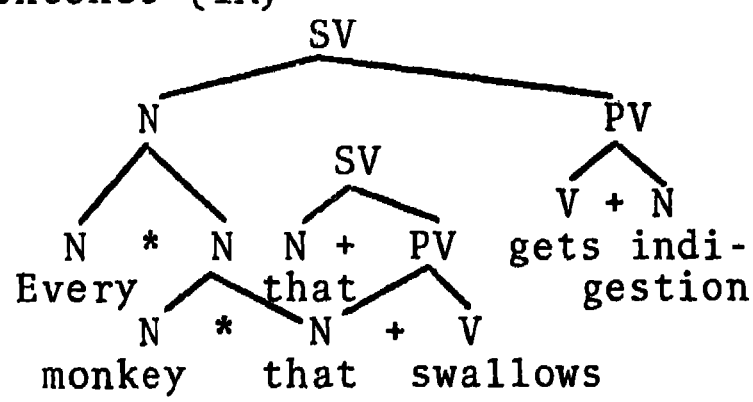
(1A) Every monkey that that swallows gets indigestion.

A readable way of handling such clauses is the method used in Junction Grammar: The proform with respect to which the modification occurs is directly joined to the noun being modified. Thus the structure takes the place of subscripts.

Sentence (1)

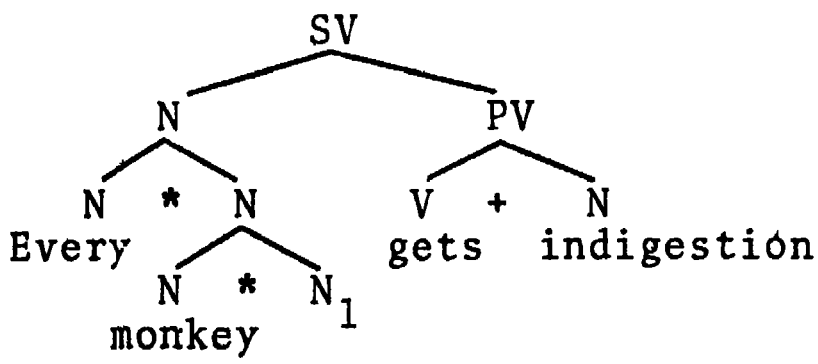


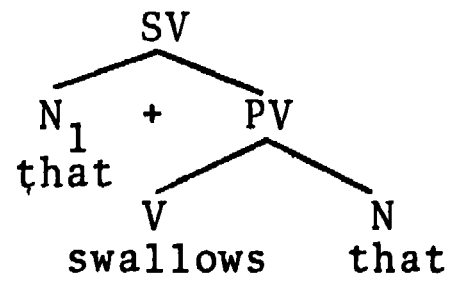
Sentence (1A)



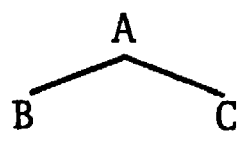
The reader will notice that the above diagrams are not normal binary trees, but we call them trees or intersecting trees because they can be drawn as sets of cross-referenced trees.

The following tree illustrates the notational variations of (a) drawing intersecting trees as cross-referenced sets of normal trees, (b) drawing the junction symbol between the brothers (subordinate nodes) instead of next to the father (superordinate) node, and (c) optionally leaving out the junction symbol when it is adjunction (+).

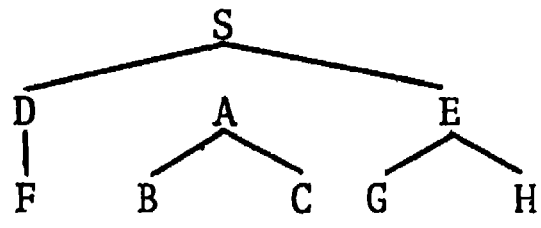




It is interesting to note that one simple method of generating in a context-free way such intersecting trees as those used in Junction Grammar is to employ context-sensitive rules as node admissability conditions. This method is essentially in harmony with McCawley's [5] proposal. For example, the rule A→BC allows the well-formed structure



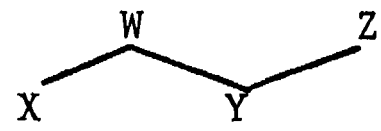
Further, the rule A→BC/F___G is construed as saying that a subtree with A immediately dominating the sequence BC is a well-formed subtree provided that elsewhere in the tree a node F immediately precedes A and A immediately precedes a node G. Thus, as an example, the following tree is well-formed from the rules indicated.



S → DAE
D → F/ __ B
E → GH/C__
A → BC/F__

Though the tree is well-formed if the rules are node admissability conditions, FBCGH cannot be derived from this grammar in a standard rewrite derivation way.

Intersecting trees are of the form:



If such production rules were to be used, their form would be:

$$W \rightarrow XY$$

$$WZ \rightarrow Y$$

Such production rules are type 0 and not type 1.

B. Noun Complements.

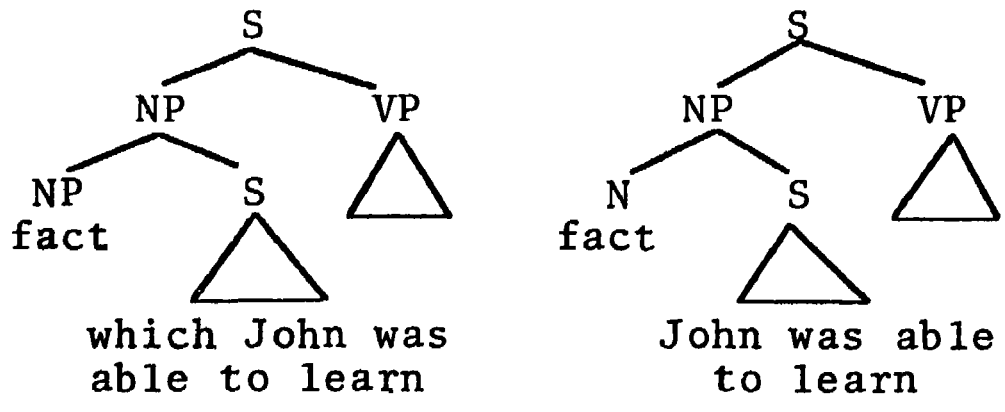
It is instructive to contrast the above Junction Grammar treatment of relative clauses with its treatment of noun complements.

It would seem that there are serious theoretical problems associated with the phrase structure rules $NP \rightarrow NP\ S$ and $NP \rightarrow N\ S$ of Transformation Grammar [6].

RELATIVE CLAUSE RULE: $NP \rightarrow NP\ S$

COMPLEMENT RULE: $NP \rightarrow N\ S$

(2) The embarrassing fact that John was able to learn frustrated his adversaries.

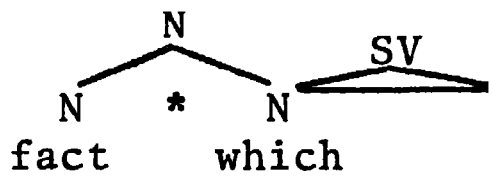


TG Treatment of Noun Complements and Relative Clauses.

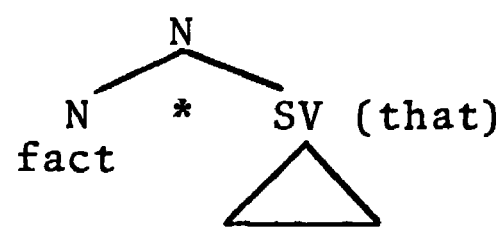
These rules propose to account for restrictive relative clauses and noun complements, respectively, both embedding an entire sentence to a nominal antecedent, encompassing both constituents with brackets labelled NP. It seems clear that the two structures in question are in fact related, i.e. in some sense similar, but not in the way suggested by the P-rules proposed to generate them. Specifically, in each case there seems to be an overlapping of constituents in the main clause with those in the subordinate clause: In the relative clause, an NP of the main clause coincides referentially with an NP of the dependent clause; in the complement, a noun, or potentially, a noun phrase, of the main clause is equated referentially with the entire dependent clause. Thus sentence (2) is ambiguous over the relative clause and the complement readings. (Notice that one can replace <u>that</u> with <u>which</u> for the relative clause reading but not for the complement reading.) There is nothing in the P-rule formulation to make this overlapping of constituents explicit, however. Hence, a mechanism for checking the coreference of NP's is required so that the relative clause transformation could apply to sentences embedded by NP$\longrightarrow$NP S and produce the appropriate relative pronoun. It is not clear, however, whether this mechanism is supposed to establish a coreference relation between the head N and the complement S in NP$\longrightarrow$N S, since no T-rule seems to depend upon such a check. Moreover, there is no clear justification for using N rather than NP to the right of the arrow in the complement rule, since the head of a complement can have articles and modifiers too.

Still more serious is what appears to be implied by the relative clause rule. Namely, the entire clause is bracketed with a nominal category (NP), suggesting that it, like the complement, is functioning in its entirety as a nominal constituent. The structural symmetry of these two rules results in a false generalization (the illusion that both clauses were nominalized) while failing to make explicit the generality which actually exists and is semantically crucial (the referential overlap between constituents in the main and dependent clauses). What is needed are structural representations which reflect the overlapping of constituents without violating what seem to be the correct categorizations.

The Junction Grammar solution is as follows:



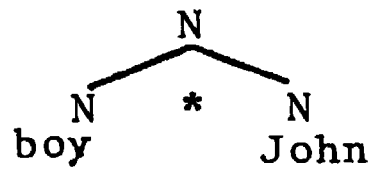
Relative Clause



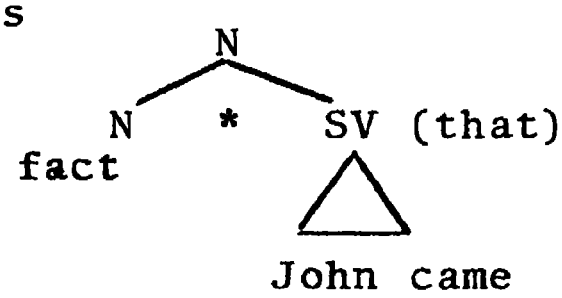
Noun Complement

In the case of the relative clause, the intersection occurred on categorially homogeneous nodes (on N's), whereas in the complement structure the intersection occurred on heterogeneous nodes (N /S) so that the entire subordinate clause intersected with an NP of the main clause.

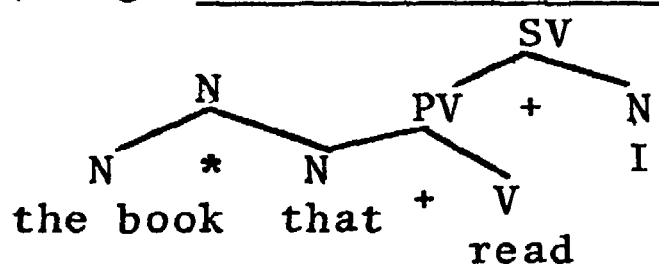
Subjunction can thus be seen to be of two basic types--full-subjunctions, as in noun complements, and interjunctions, as in relative clauses. In full-subjunction the modifying constituent node is completely subjoined, e.g. number five, or the boy John as



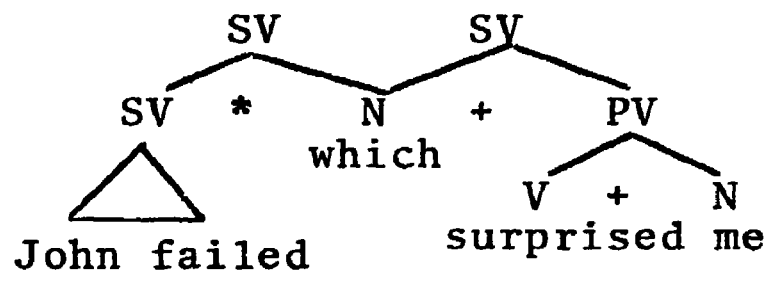
or <u>the fact John came</u> as



On the other hand, in interjunction the modifying constituent is just interjoined, e.g. <u>the book (that) I read</u> as



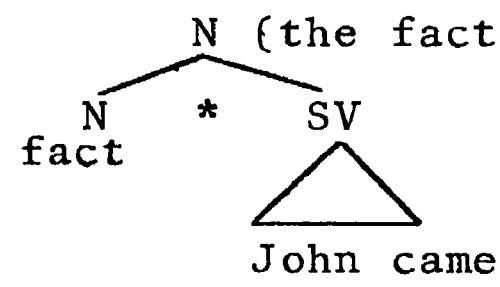
or <u>John failed, which surprised me</u> as



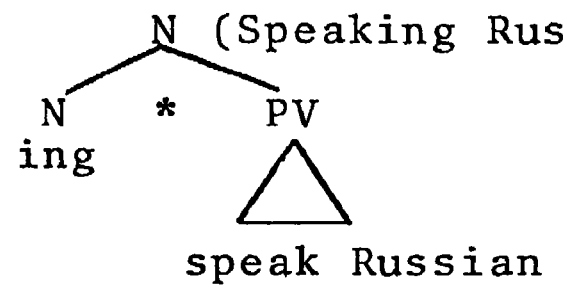
In both cases, however, the nodes which intersect referentially are directly joined to each other.

The schematic expression for full-subjunction is $Z \longrightarrow X*Y$. The following are representative members of this schema:

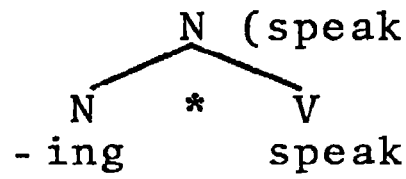
N → N * SV (the fact) John came surprised us.



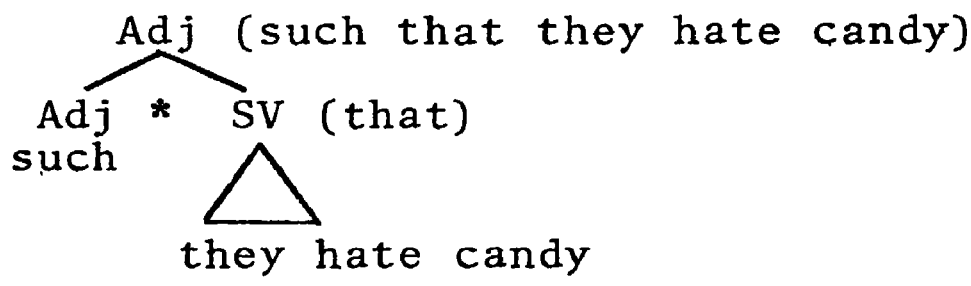
N → N * PV Speaking Russian is difficult.



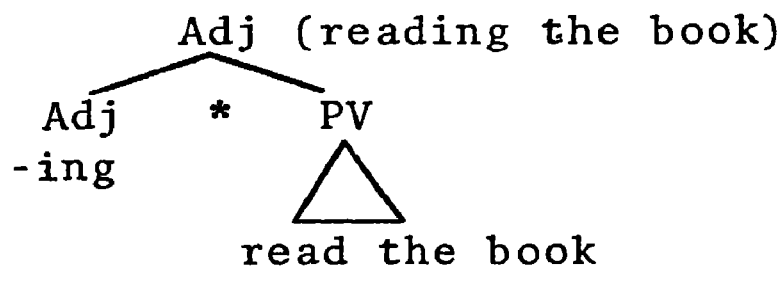
N → N * V The speaking of Russian is difficult.



Adj → Adj * SV Children such that they hate candy are rare.

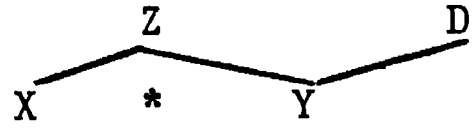


Adj → Adj * PV The boy reading the book is John.



C. Other Modifiers as Relative Statements.

Returning now to the discussion of relative modifiers, we continue our description of interjunction, which, as the reader will recall, entails intersecting trees corresponding to the following schema:

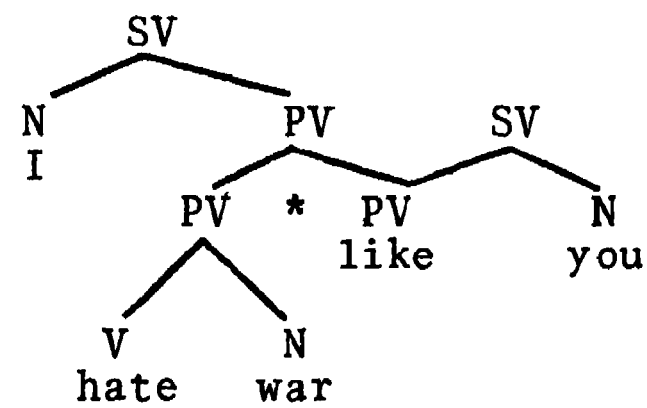


Basic relative modifiers are defined to be relative clauses of which the modifying node is of noun category, the relative marker for such clauses being in some cases null (e.g. the boy I saw was crying), or such words as <u>which</u>, <u>who</u>, and <u>that</u>. Non-basic relative modifiers are defined to be all others which entail interjunction. Some of these relatives have not in some cases been recognized in the literature as relative constructions at all, but were identified during the elaboration of junction theory by deducing from the interjunction schema specific possibilities not hitherto noted by observation of random data. This deduction was done in terms of the constituent categories extant in the junction grammar system of diagramming at that time. Sentences (3-8) below illustrate some of the non-basic relative modifiers in question. Notice that one might search available data a long time before finding an instance of some of the types exemplified, as each would require a special context. Yet, any grammar of English would be incomplete without them. The fact that some rules are used less in no way invalidates them; in a number of cases, deductions which at first seemed most ridiculous were found to have instances that were perfectly acceptable. The perspective deriving

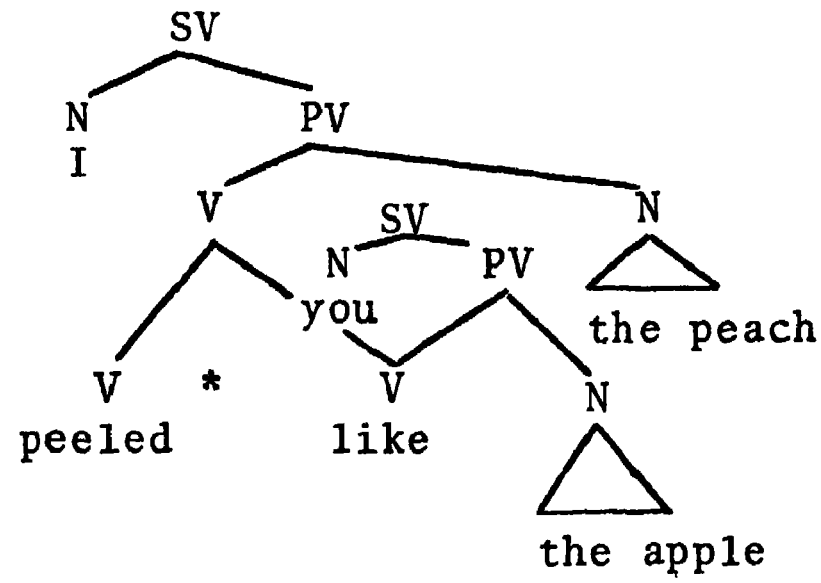
from this experience suggests that any grammar which is based on rules arrived at inductively, i.e. simply set down in a list as available data suggests them, will be incomplete at best, and most probably not motivated by any significant generalizations.

Diagrams for sentences (5) - (8) will be supplied in the discussion of non-verb cored relatives.

(3) Verb phrase. I hate war, like you.



(4) Verb. I peeled the peach, like you the apple.



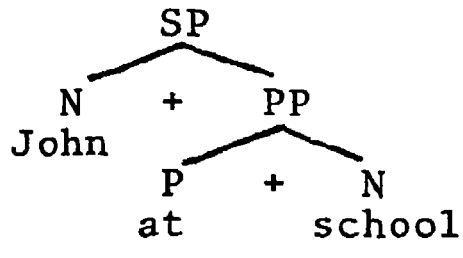
(5) Prepositional Phrase. My kite is on the roof, like your ball.

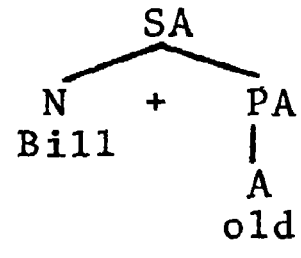
(6) Preposition. Fred is above a store, like you were a bakery.

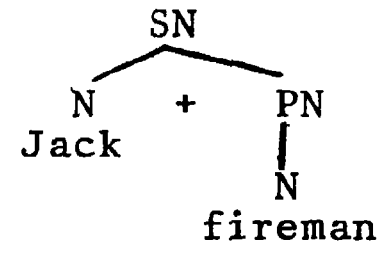
(7) Adjective. He is like (such as) Bill is.

(8) Noun. A soldier, as I am, seeks adventure.

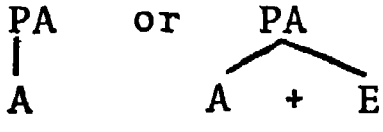
Junction Grammar claims that some statements occur without a verb in the relative modifier. One way to represent such sentences in a natural way is to allow non-verb cored statements as follows:







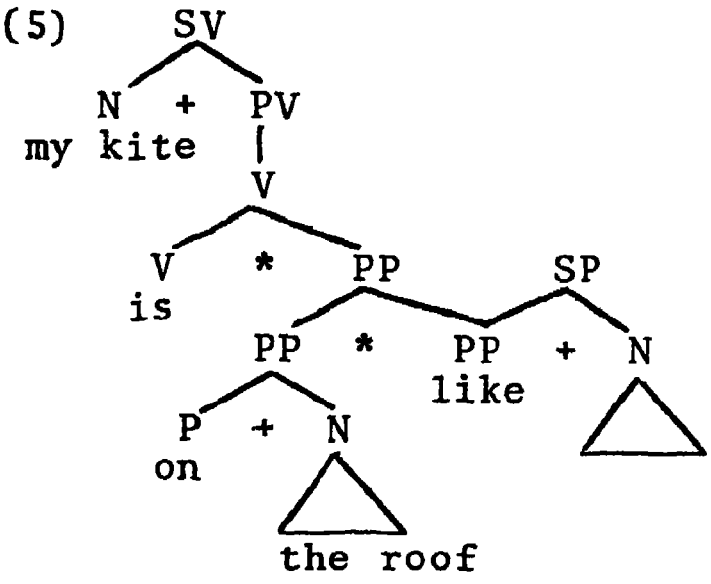
We will indicate that a predicator (e.g. adjective or adverb) becomes a predicate without a direct object either by the notation



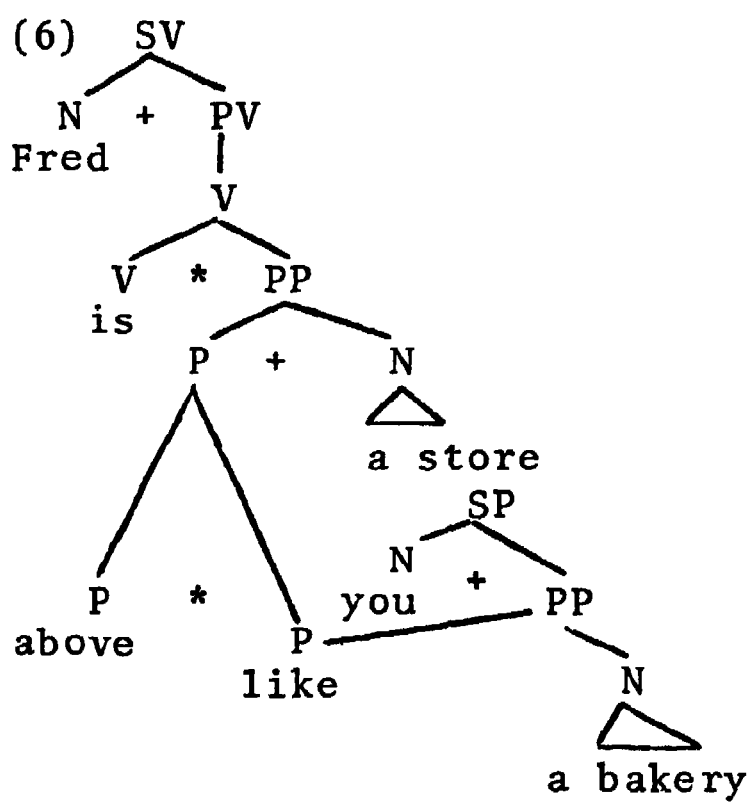
, where E means empty slot.

Preposition cored: My kite is on the roof, like your ball.
Fred is above a store, like you a bakery.

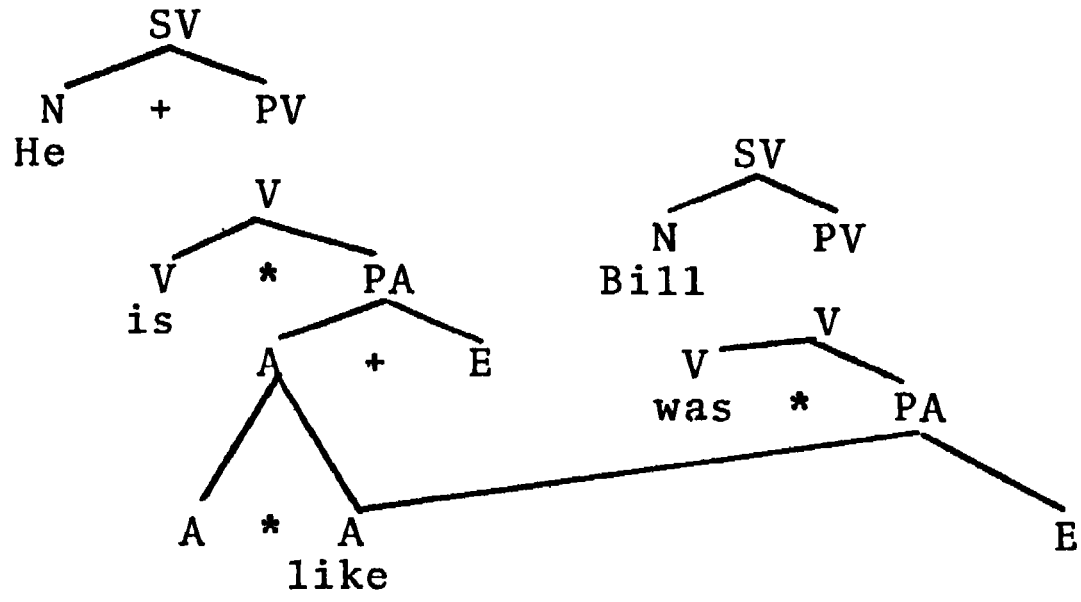
(5)



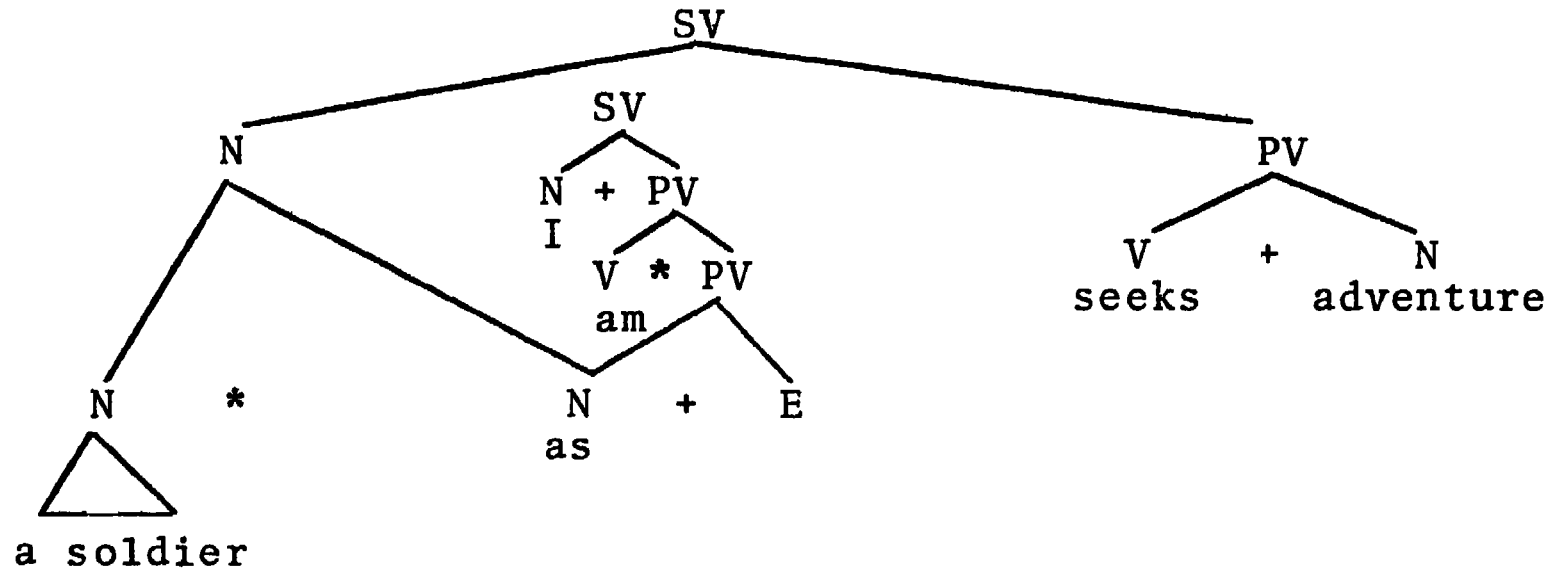
(6)



(7) Ad cored: He is like Bill was



(8) Noun cored: A soldier, as I am, seeks adventure.



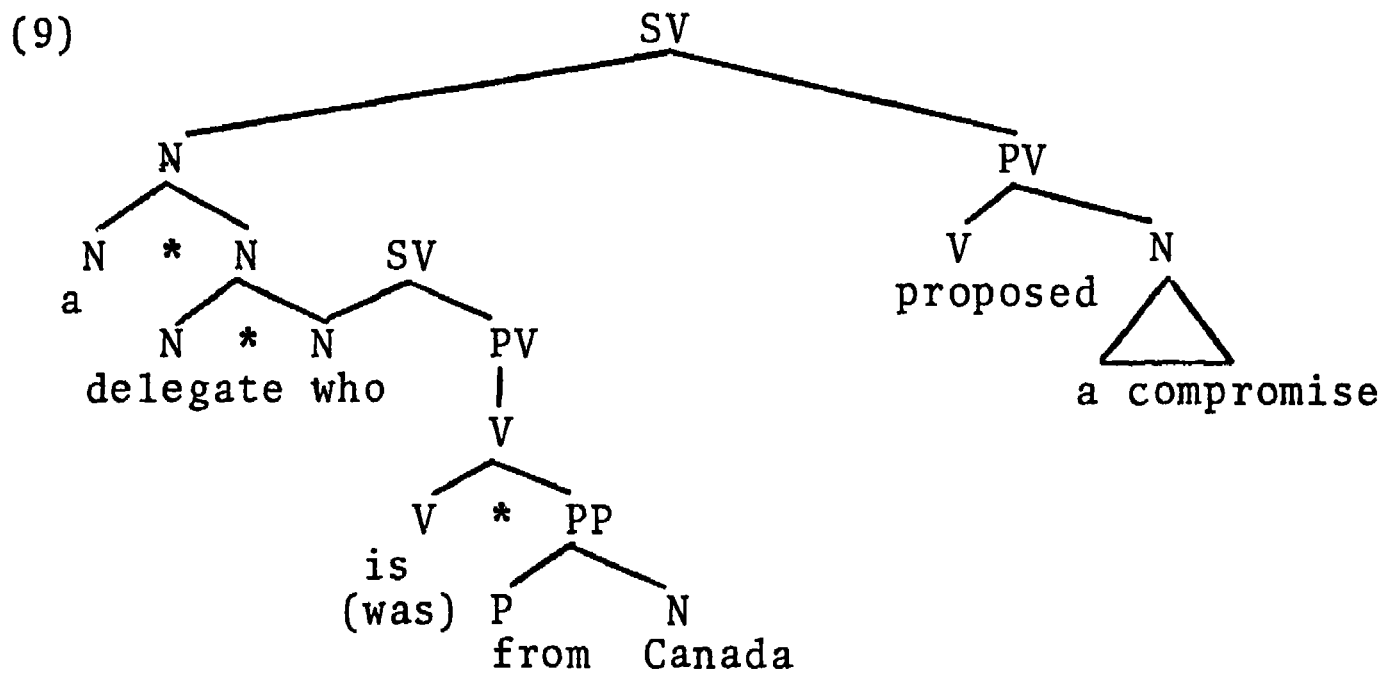
Now consider the following two sentences:

(9) A delegate from Canada proposed a compromise.

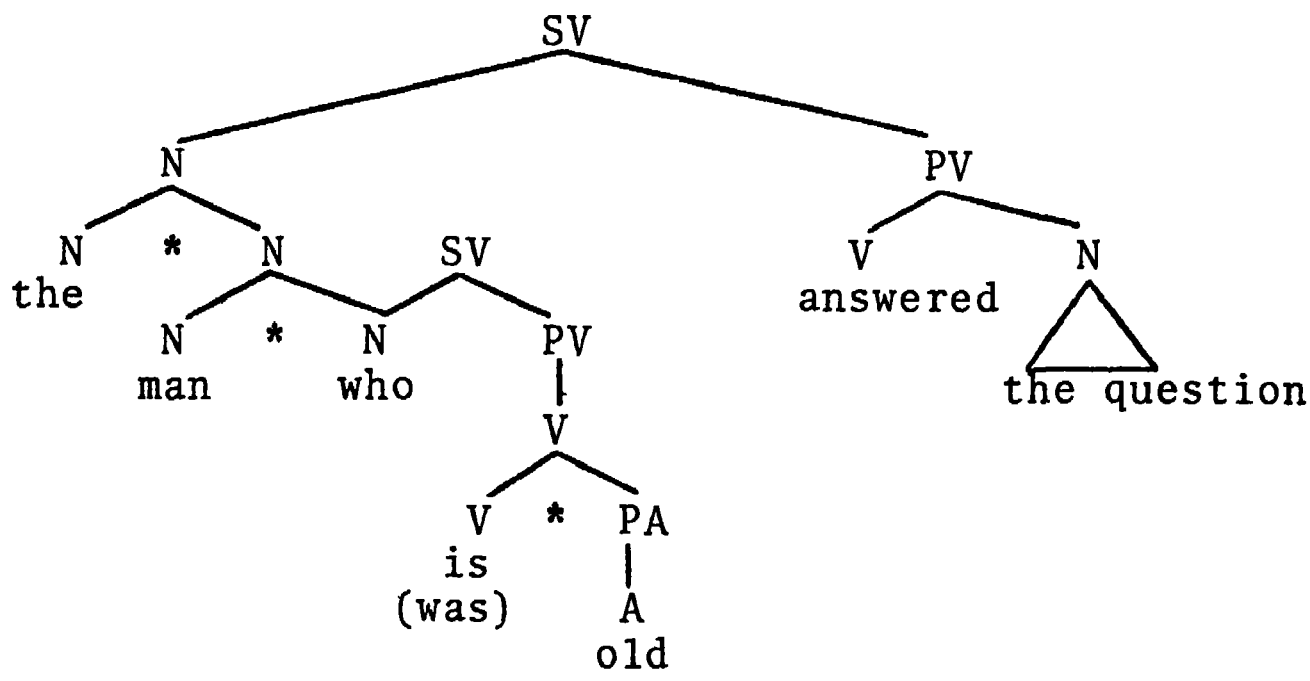
(10) The old man answered the question.

The modification occurring in these sentences can be expanded out to relative clauses and represented as follows:

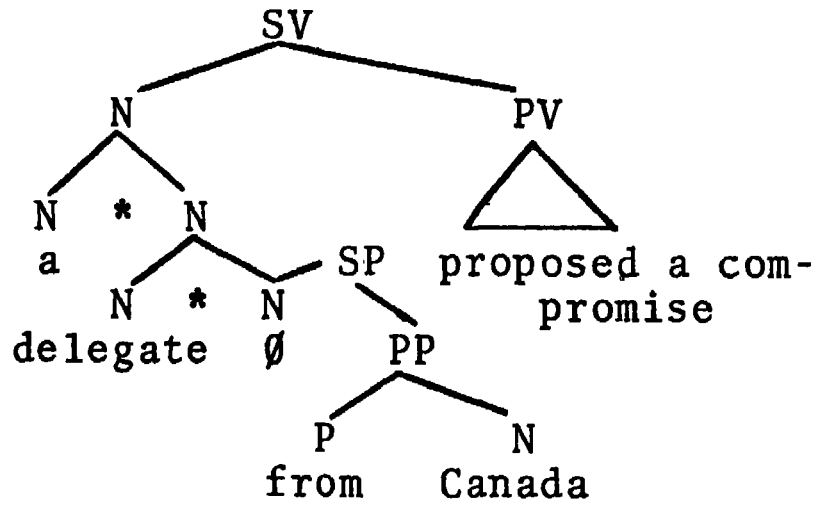
(9)

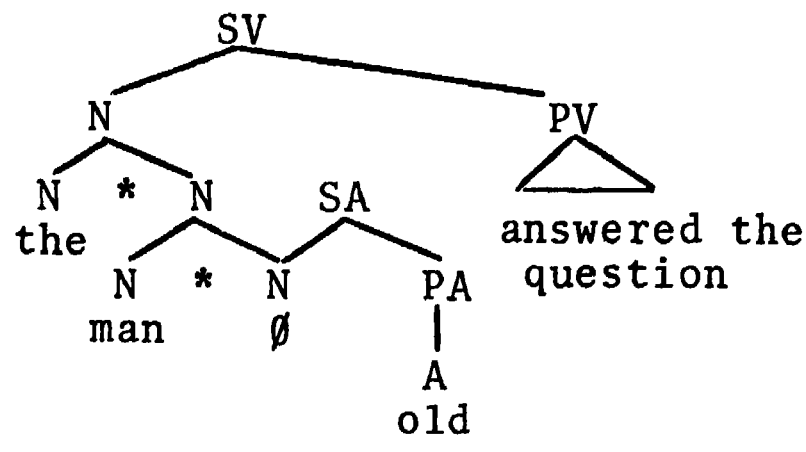


(10)



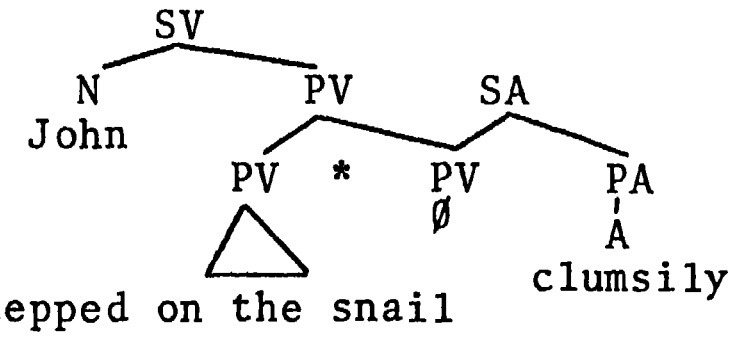
However, these sentences can also be represented with relative modifiers by employing the non-verb cored statements (below):





One benefit of considering modifiers such as these to entail entire statements is that various types of ambiguity occurring with them can be explained as types of ambiguity also characteristic of clausal statements. For instance, consider the following ambiguous sentence:

(11) John clumsily stepped on the snail.

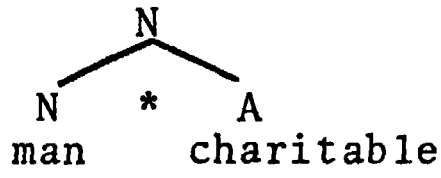


The ambiguity in question hinges upon whether or not we consider the relative modification to be restrictive or non-restrictive, a contrast which in junction grammar is expressed by subcategorizing the subjunction operation (*).

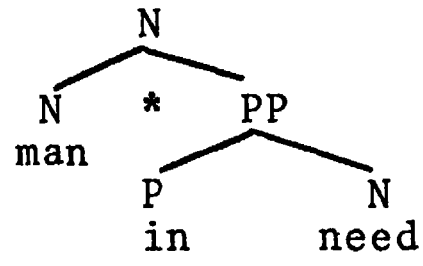
Note also that semantic complexities which occur with the non-clausal modifiers of sentences (9) and (10) can also occur with basic relative clauses. For example, if John is tall (as a person) and John is a basketball player, we cannot conclude that John is a tall basketball player. And similarly, if John is an expert and John recommends Gillette, we cannot necessarily conclude that John is an expert who recommends Gillette (the reason being that he might be a blimp expert who just recommends Gillette products to his friends, while saying that he is an expert who recommends Gillette implies that he is an expert on the types of products that Gillette produces).

Further, if modifiers such as those in (9) and (10) are con-

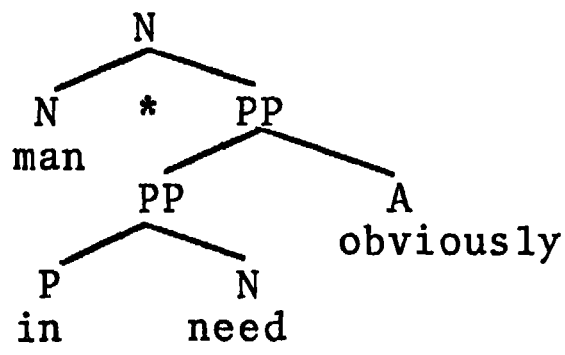
sidered to be relative statements, it explains our intuition that obviously, surprisingly, etc., are usually modifiers at statement level. For example, a natural way, it would seem, to represent charitable man is as follows:

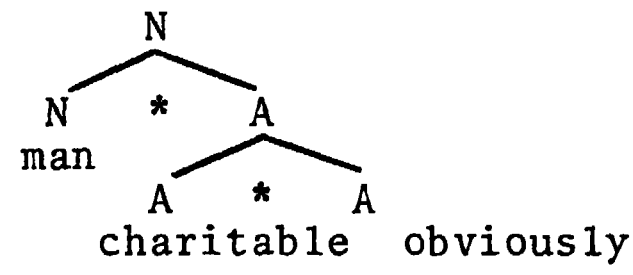


Similarly, for The man in need asked for help, man in need would be represented as follows:



But if this were done, then obviously charitable man and man obviously in need would, it seems, have to be represented as follows:

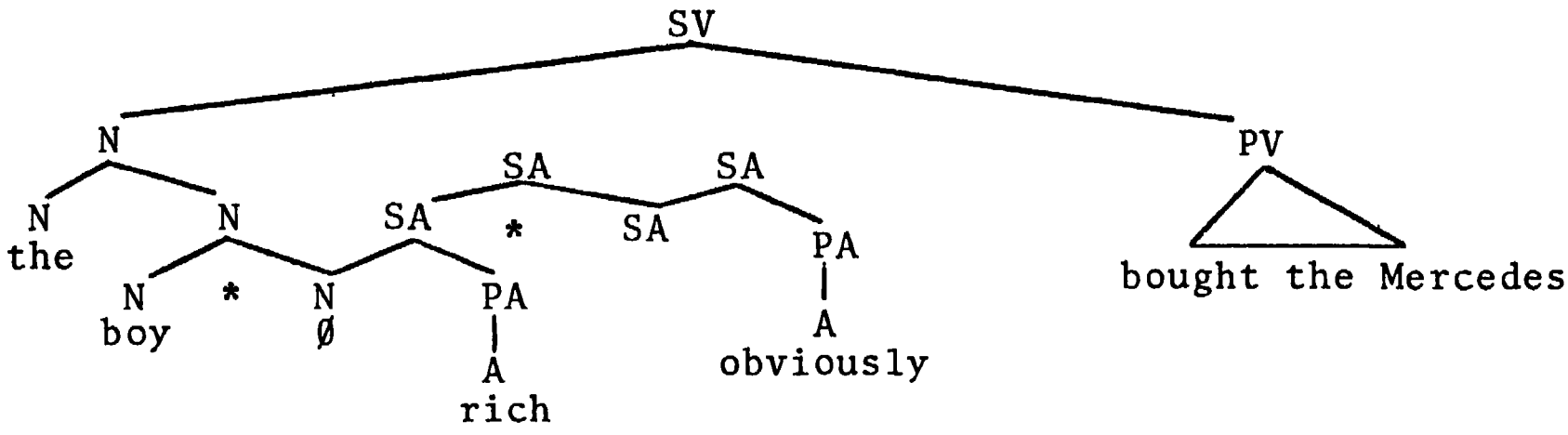




But this represents the reading in which obviously is taken as a manner adverb, and it seems that because we have not represented the modification as a relative clause, we are prevented from using obviously as a sentence level modifier.

Now consider the following sentence with its Junction Grammar diagram:

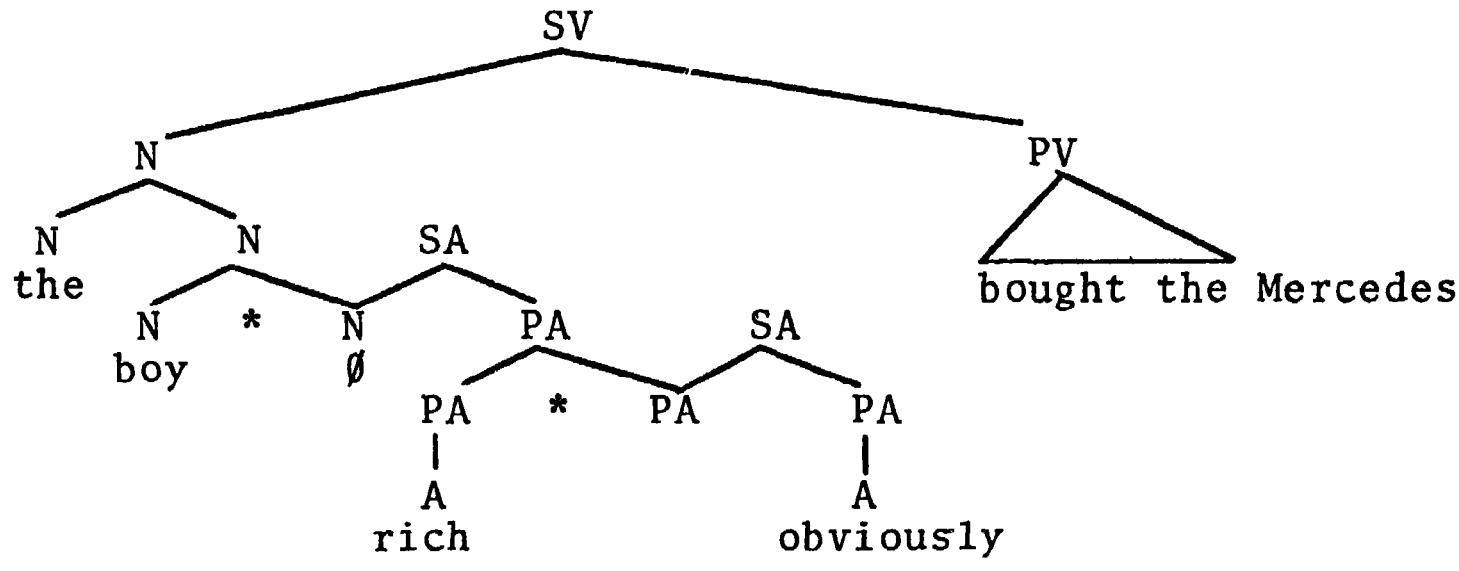
(12) The obviously rich boy bought the Mercedes.



By expanding the rich relative clause to include a verb, the sentence might be lexicalized as:

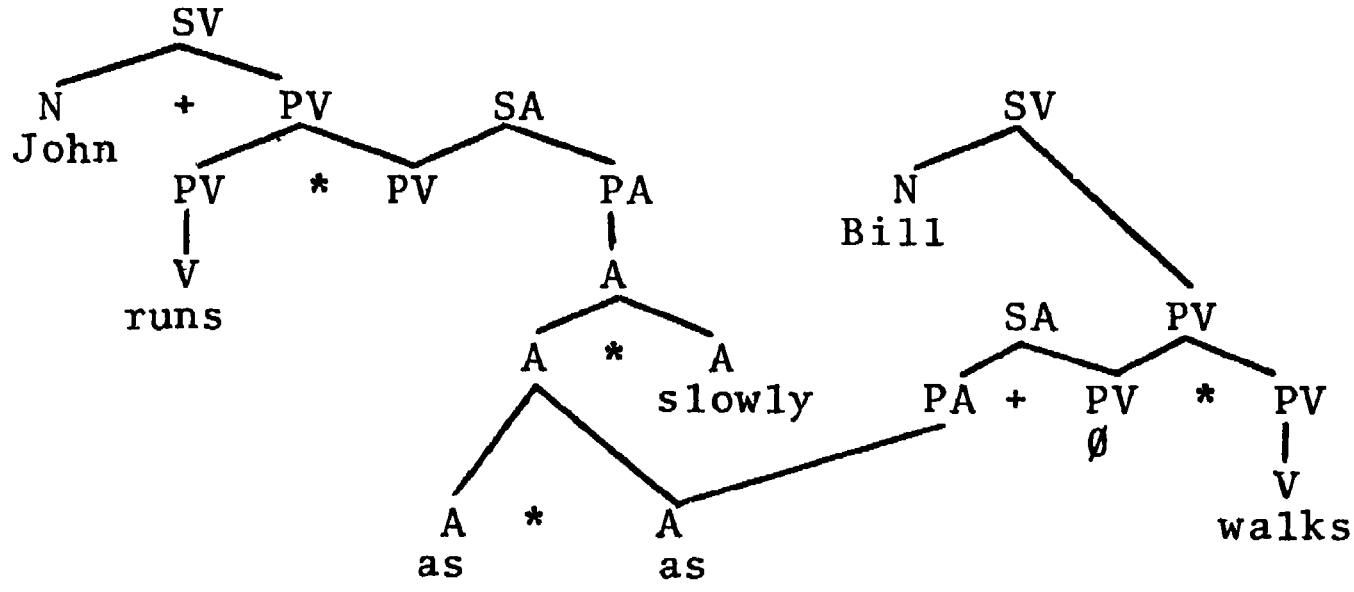
The boy who obviously was rich, bought the Mercedes.

But also, obviously can be used as an adverb of manner. Taken in this way (12) would mean that the ostentatiously rich boy bought the Mercedes. As such, it would be diagrammed as follows:

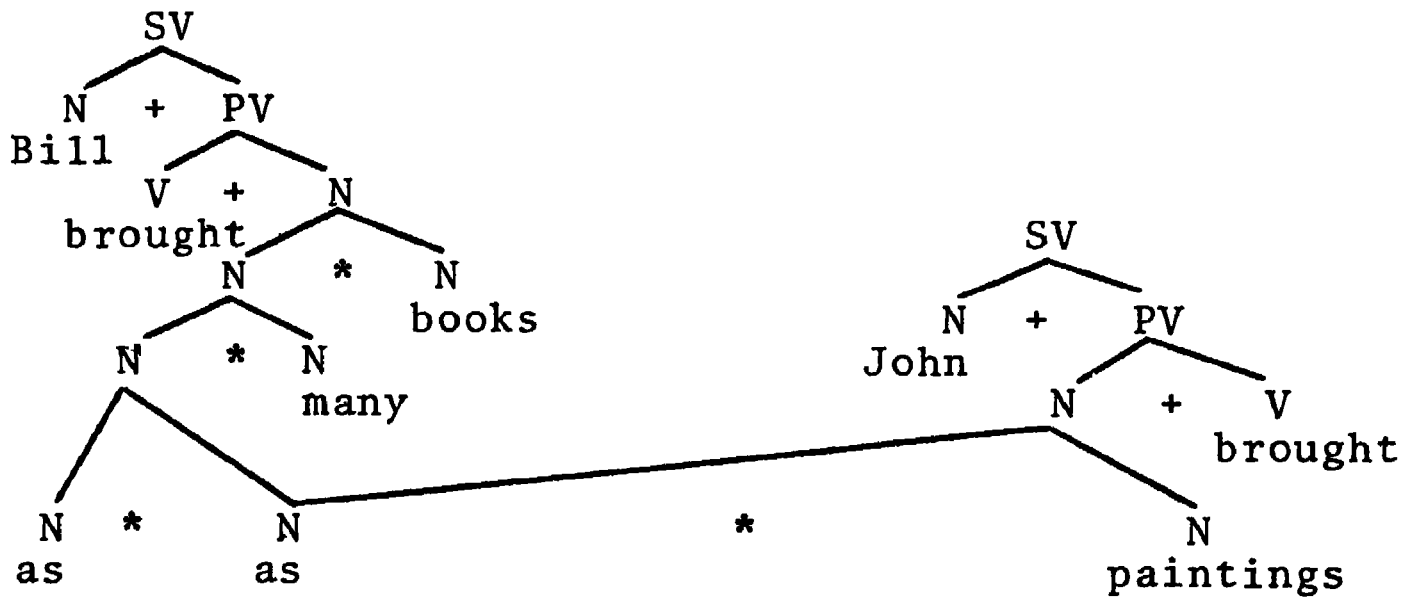


Finally, consider these two relative modifier sentences:

John runs as slowly as Bill walks.



Bill brought as many books as John bought paintings.



It should be pointed out that the above discussion has omitted major areas of Junction Grammar, such as specializations of subjunction which indicate flow of information in "compiling" a junction tree, conjunction, and the function and application of lexical rules to junction trees.

III. Implementation.

A. Analysis.

The analysis step must accept natural language input and produce the appropriate junction tree for each segment of text (normally, but not necessarily, a segment is a sentence), according to the context of that segment.

Two types of ambiguity can be distinguished: word sense ambiguity and syntactic ambiguity. Without some logical processing, an utterance such as "John bought some ink for the pen" is word sense ambiguous as to pen, between the writing utensil and the enclosure senses. Also, the sentence "The boy flipped the coin by the book," is syntactically ambiguous as to the point of intersection between the prepositional phrase and the rest of the sentence.

Several English analysis systems (e.g. W. Woods, Y. Wilks, T. Winograd, etc.) have seen considerable success in resolving word sense and syntactic ambiguities automatically on texts that stay within some restricted vocabulary or context. However, no one has attempted to apply such principles to a large scale system (10,000 to 20,000-word lexicons) which can analyze a wide variety of structures and types of texts. At present, it is not known whether any system is expandable to such a degree or, even if it were, how many years would be needed to complete such a project and how much it would cost to run. Clearly, research and development in the area of automatic analysis should be continued.

In the meantime, however, work is needed on the processing that must follow automatic analysis to form a complete translation system.

Therefore, our research group has decided to develop a machine-assisted translation system consisting of interactive analysis (where the human operator resolves difficult ambiguities via his video screen) followed by automatic transfer-synthesis into multiple target languages with junction trees as the interlingua. This configuration, which has also been proposed by Kay [7], has some attractive features. Suppose that the Junction Grammar transfer-synthesis system produces acceptable translations of real-world text from the output of the interactive analyzer. The analyzer could then be replaced by a more automatic version without disturbing the rest of the system.

On the other hand, if the whole system fails, we will not have invested nearly as much effort as if we had tried to develop a large-scale automatic analysis to junction trees.

In addition, during the period of development and testing, counters can easily be set up to keep statistics on what interactions are most common and therefore should be given priority in a more automatic version or during the testing phase of automatic routings to see if they disambiguate the same way the human does.

Finally, regardless of the particular version of analysis used, the system divides the effort expended in doing the analysis by the number of target languages being translated into from the

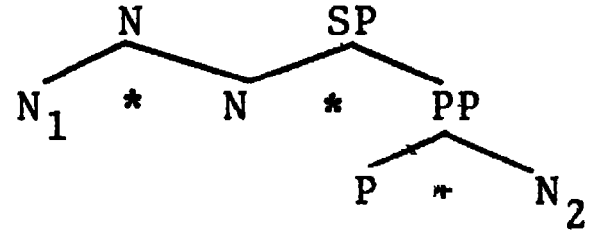
same junction tree.

Now let us describe the current five-phase implementation of interactive analysis. The first phase performs limited morpheme splitting (e.g. possessives and contractions). As each word is identified it is matched against an index-sequential dictionary stored on disk. If there is more than one word sense associated with the word then several definitions are chained to the entry. These several meanings appear on the video screen and the human is asked to choose the meaning for this particular context. When a choice is entered, the information associated with that particular use of the word (or phrase of which it is the keyword) is retrieved from the dictionary. The dictionary contains semantic indices, categories, and binary features. Information specific to the source language and information considered to be language independent are represented in these features. This information is put into a two dimensional array (Reference table) and will be available during the other phases. If a word is not in the dictionary, the person monitoring has the option of picking a synonym, or passing the word through untranslated as he would a proper noun, signaling only the category of the word (noun, verb, adjective, adverb, preposition). It is by this interactive process that we temporarily solve the word sense ambiguity problem.

The second phase of analysis, called "PHRASER", logically segments the reference table (sentence) into "phrases" containing a maximum of one noun each. The first word of the sentence starts the first phrase. Thereafter, a noun or comma ends a phrase, and

the succeeding phrase begins with the next word. The sentence "The boy in the car ate a very good hamburger while driving down the street" is symbolically segmented as (The boy) (in the car) (ate a very good hamburger) (while driving down the street). During the segmenting operation PHRASER assigns a global reference category to each element (word) of the reference table. The main purpose of PHRASER is to divide the sentence into "phrases" or segments which are individually processed by the infix generation routines as separate units, thus simplifying their task.

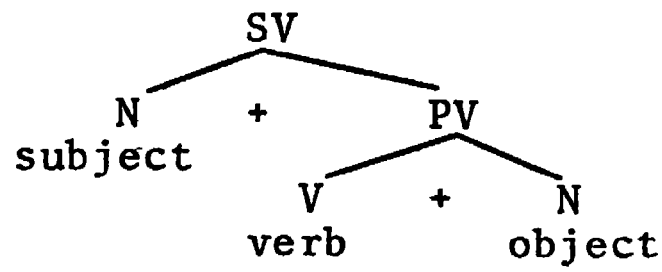
The third phase of analysis is the syntactic resolver and infix builder (Figure 1). This phase produces an infix representation equivalent to normal junction trees. For example, $N_1\$(P+N_2)$ is equivalent to



The basic philosophy of this routine is to process phrases generated in PHRASER by calculating their internal structure, then placing them in their proper structural position in the total sentence. This is facilitated by the introduction of what we call "order-rules."

The Order-Rule

As discussed earlier, a simple sentence can be represented in a tree diagram as



which can be stated in an infix notation as (N+(V+N)), which will

TYPSET	(TYPE)	INFIX ROUTINE
0000	"," ";"	INFX 00
0001	N Topic	INFX 01
0010	Nouns	INFX 02 **
0011	V Topic	INFX 03
0100	Verbs	INFX 04
0101	Adj Topic	INFX 05
0110	Adjectives	INFX 06
0111	Adverbs	INFX 07
1000	Prep Topic	INFX 08
1001	Prepositions	INFX 09
1010	? !	INFX 10
1011	Conjunction	INFX 11
1100	Articles	INFX 12
1101	Adv Topic	

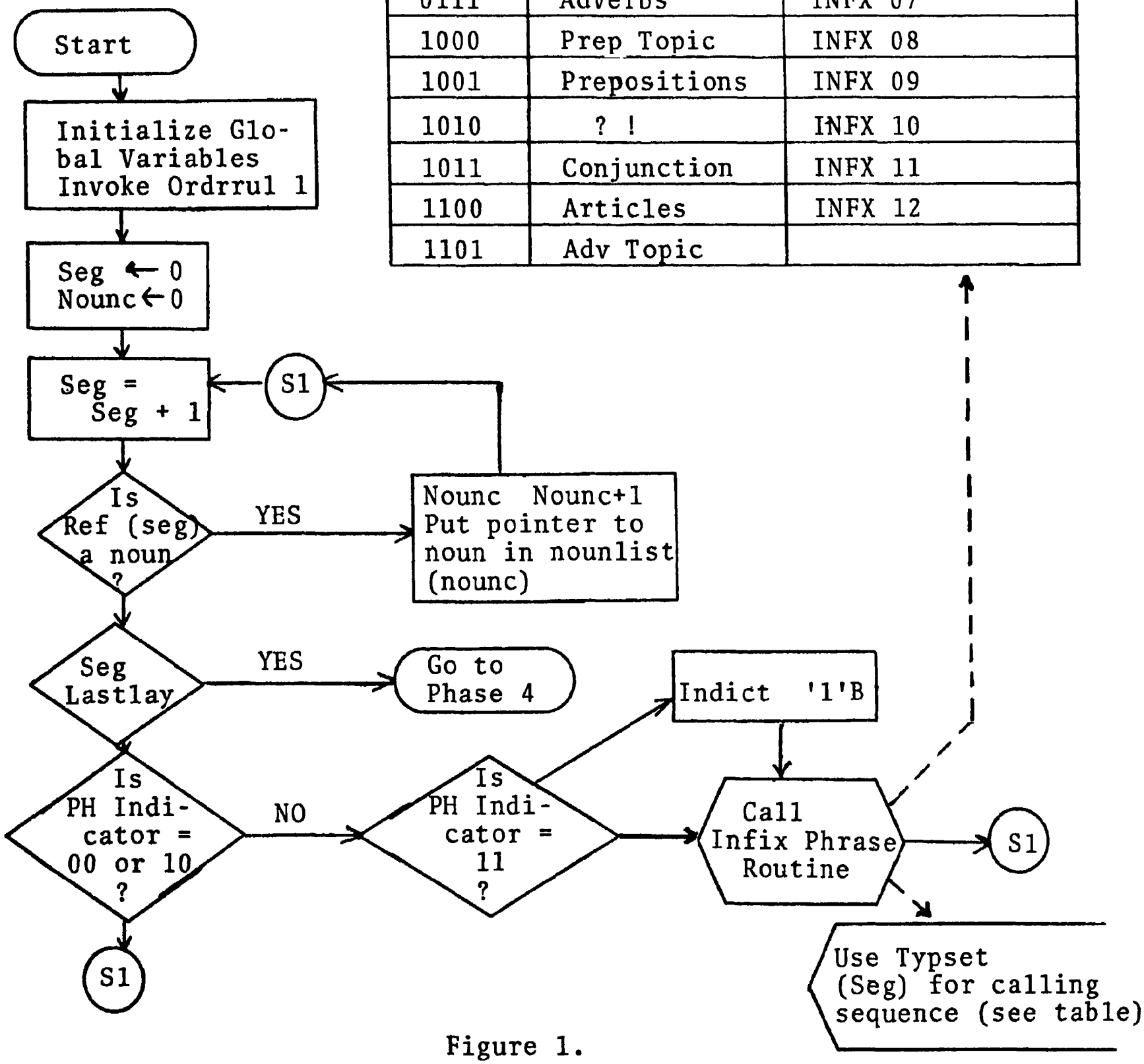


Figure 1.

PHASE III MAINLINE INFIXER.

be called a clause order-rule. Notice that the order rule has the terminal nodes (N + (V + N)) as well as the father node markings (parentheses).

(	+	(	+	)	)
pre-sentence		pre-predicate		post predicate	post sentence

This gives seven slots of layers for the actual entries of the order-rule. An order-rule has the order portion, i.e. (N+(V+N)), and the entries that fill the slots of the order:

subject
verb
object
pre-predicate
pre-sentence
post predicate
post sentence

There are three basic constraints on order-rules. First, they can never represent discontinuous word order. The discontinuous orders (V, N subject, N object and N object, N subject, verb) cannot be represented directly but must use variant order rules. The second basic constraint in the use of order rules is that when order rules are interjoined, the first terminal node element of the subordinant order-rule must be the intersect node. For example: "I saw the boy that likes you" has two order rules.

must be first terminal node of the subordinate order-rule

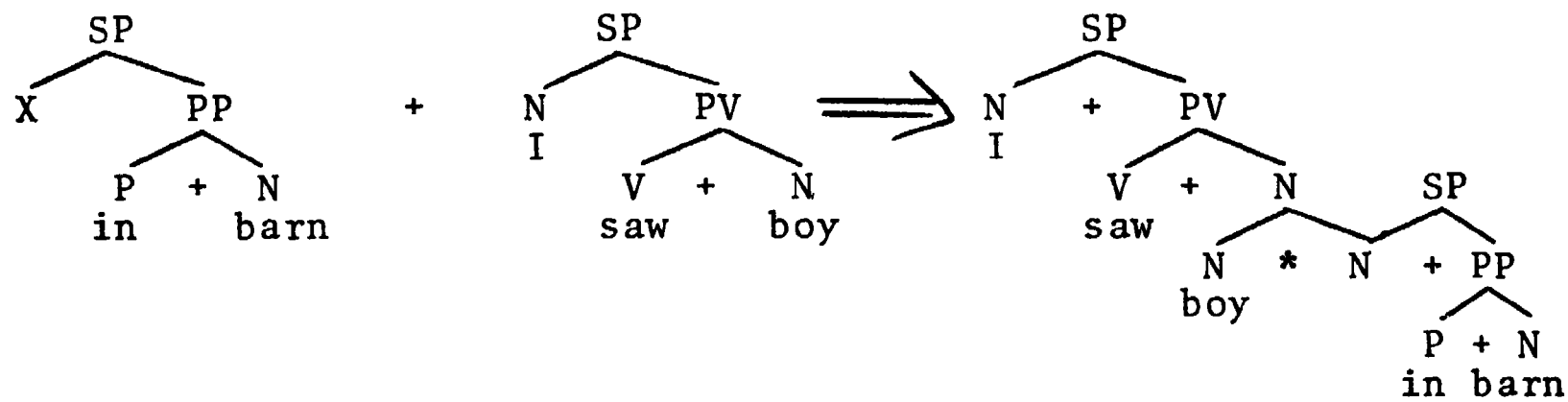
I saw boy → (N + (V + N)) (I saw boy) that likes you → (N + (V + N)) topic (that) likes you

When these are combined, the following interjunction will be created: (N + (V + (N $ (V + N)))). The third constraint is that all subordinate order rules must indicate their antecedent nodes

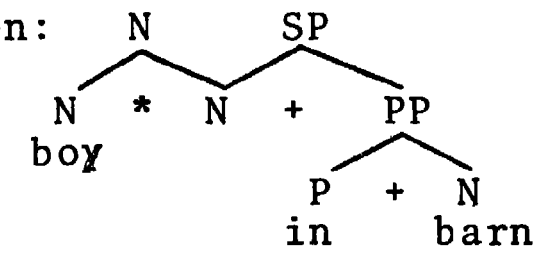
in some other order-rule.

To review, phase three routines build order-rule representations; phase four inserts the slot or layers up into the order-rule, then it conflates all the order-rules into one linear infix string. This linear infix string cross references the information table produced in phase one (cross-referencing is achieved via doubly-linked-list processing).

The reader will recall from Section I that Junction Grammar relative clause modification involves an interjunction and phrasal modifiers (prepositional phrases, adjectives, and adverbs) have the same basic form as the clause.

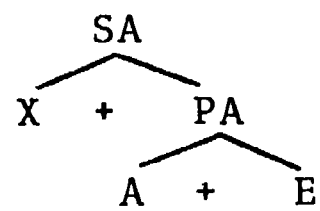


Notice the interjunction:



In infix notation this is: (N + (V + N $ (P +N)))
I saw boy in barn

providing of course that boy is what in barn modifies. An adjective or adverb has the tree structure:



Adjectives normally pre-pose their head in English, so <u>big boy</u> →

(A + E) $ N. "I saw the big boy in the unsightly barn" is
big boy

represented in infix notation as

(N + (V + (A + E) $ N $ (P + (A + E) $ N))).
 I saw big boy in unsightly barn

The order-rule with consitutent values for this is the following:

```
(N + (V + N))
     N
     V
(A+E)$N$(P+(A+E)$N)
     ø
     ø
     ø
     ø
```

Note that we create new order-rules only when we have new verb core predicates.

Phase three programs can generally identify the subject, predicate, and, if present, indirect objects of clauses, both independent and subordinate. There are, however, certain areas of processing that present difficulties for the program. A simple sentence like, "I threw the ball in the room" illustrates such problems because the program does not know without context whether <u>in the room</u> identifies which ball it was that I threw, the direction of its path (motion), or where I and the ball were during the action. As stated before, the key to this problem is the context. As a matter of fact, almost every prepositional phrase is ambiguous in the computer environment. For this reason, we have the program query the person monitoring the program and ask,

I ⟨1⟩ threw ⟨2⟩ ball ⟨3⟩ in ⟨4⟩ room ⟨5⟩

"What does 'prep-object' modify?", in this case, "What does 'in-room' modify?" If the answer is "3" (ball) then the infix would be

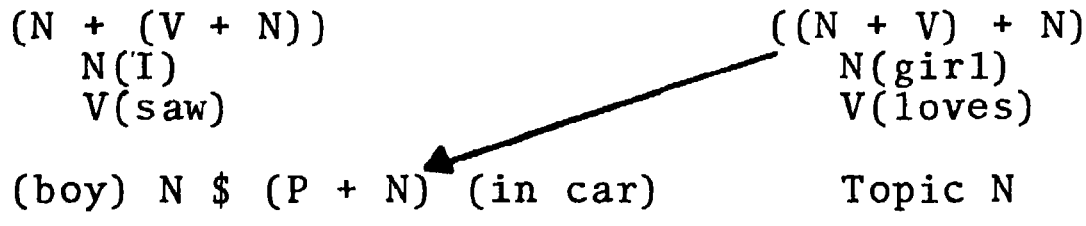
```
(N + (V + N $ (P + N)))
 I  threw ball in room
```

and it would identify which ball was thrown. If PV 2 is entered, it is interpreted to modify the predicate (throw + ball); then the infix would be

```
(N + (V + N) $ (P + N))
 I threw ball  in  room
```

and implies that the phrase tells where the action took place. If "2"(throw) is entered, the infix would be (N +(V $ (P + N) + N)) and the phrase modifies the verb. Note that in addition to identifying the ambiguity, we can signal the intended meaning and represent any of those meanings in a systematic manner.

Another "contextual" problem in English is the identification of the antecedents for relative pronouns. "I saw the boy in the car that the girl loves." The question, of course, is: "What does the girl love--the boy or the car?" This is solved by having the program ask "Does 'that' refer to X?" where X is popped off of a push-down stack of previously encountered nouns. The relatives that, who, whom, and which all invoke new order-rules. The result of this interaction is to point the subordinate order-rule to the antecedent identified by the human's response.

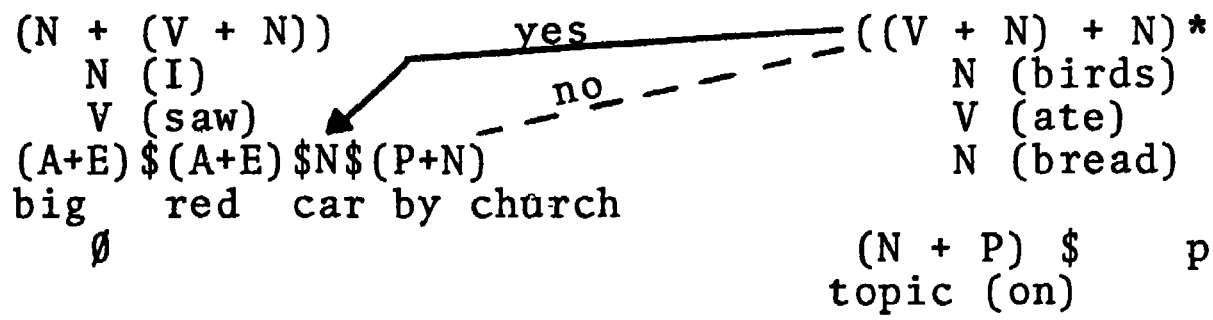


Combining the prepositional phrase and antecedent problems, we produce a sentence like this:

```
I saw the big red car by the church on which the birds ate
1  2   3   4   5   6  7   8    9    10  11   12   13   14

                                the bread.
                                15   16
```

The program will ask what by the church modifies--we will answer "6" (car). Then it will ask "Does which refer to church?" This is really asking "Did the birds eat the bread on the church?" If we say "No" then it will ask, "Does which refer to car?" "Yes." Next the program asks "What does 'on-which' modify?" This is somewhat difficult because we have a prepositional phrase whose antecedent is in one clause yet the phrase modifies something in another. In the example the prepositional phrase modifies the predicate 'birds ate the bread' by telling where the action happened.



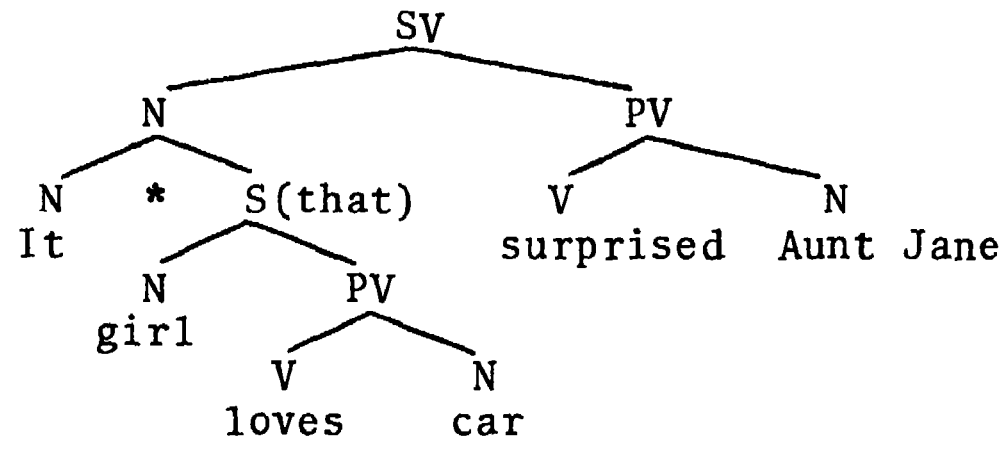
Passive constructions may have an ambiguous element in them: "The cake was eaten by the dog." The question is: did the dog do the eating or does the prepositional phrase tell where the cake was eaten. Interaction indicates which reading should be represented.

Complement constructions can be identified in this same way (through interaction). "It surprised Aunt Jane that the girl loves the car." In this instance the program asks "Does 'that' refer

---

During the layer insert phase the pre PV will be inserted to the left of the predicate parenthesis, making the topic the first node encountered.

to Aunt Jane?" Reply "No." "Does 'that' refer to it?" This time the reply is "YesC" where the "C" indicates that the subordinate clause is the complement of it. This structure is not interjoined but fully subjoined.



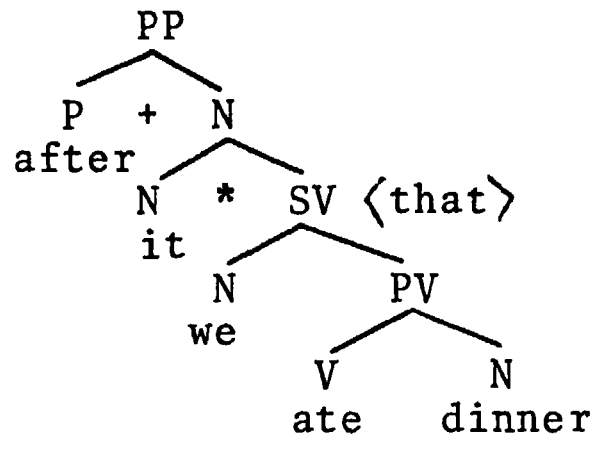
The infix notation for the above is: (N * (N + (V + N))+(V + N)).
It girl loves car surp. A.Jane

The order rule representation:

(N + (V + N))	(N + (V + N))
N (it) ←	N (girl)
V (surprised)	V (loves)
N (Aunt Jane)	N (car)
ø	ø
ø	* (The operation of full subjunction)
ø	ø
ø	ø

A variation of the complement response is: "He came after we ate dinner." The program asks "What does 'after-we' modify?" This is erroneous, and is so indicated by answering instead what the true prepositional phrase "after-it (that)" modifies and putting a "C" for complement at the end of the reply: "PV 2C." The "C" will remove the "we" as object of the preposition and replace it with a noun "it." A new order-rule is then invoked, pointing to the noun

"it." The underlying structure is represented "after it that we ate dinner"



or, in infix: (N + (V + N) $ (P + N * (N + (V + N)))).

The details of the other interactive situations will be omitted. Interaction is necessary for the adjectival like as in "I saw a man with a dog like you." The question is, "What is like you, the man or the dog?" So the program asks, "Does 'like' refer to dog?" If the response is "no," then the program asks, "Does 'like' refer to man?"

Participles and gerunds are very ambiguous. For example, "I like chewing gum." Is it "I like to chew gum" or "I like chewing-gum"? Again, "John's hobby is shocking me." Is the "shocking" a progressing verb or is it a gerund? At this point we have the program ask, "What type of participle is __X__ing"; the replies can be:

N Adj	for the chewing-gum type, use of an ing form
V	for the progressive verb
Adj X	(where X is the segment modified) for adjectival participles
Adv X	(where X is the segment modified) for adverbial participles

Of course, the infinitive form also is often ambiguous. For example, "I like to buy candy" is much different from "He sent me to buy candy." There are several answers to the program's question,

"What type of infinitive is 'to __X__'?" where X is the verb.

Other questions that the programs are designed to ask all have two points in common: first, they are stimulated when the surface string does not map into the underlying semantic string on a one-to-one basis; second, when the decision is made by the human as to its use and the reply given to the program, the processing continues in standard existing programs and no ad hoc routines are needed.

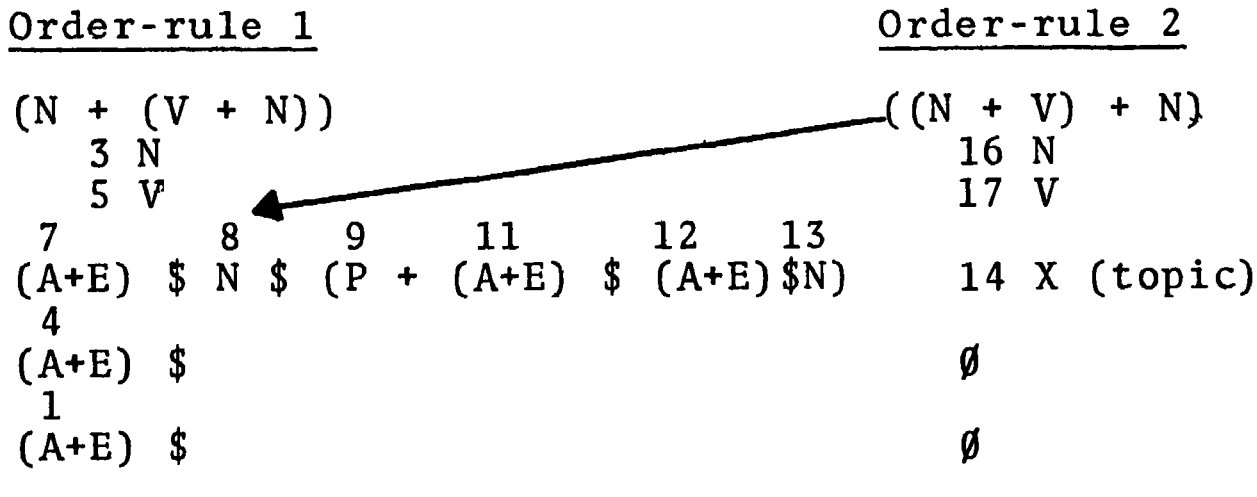
The output of phase three is a series of order-rules. Each order-rule has the form:

```
(N + (V + N))          order specifier
N (subject)
V (verb)
N (object)
Pre-predicate modifiers
Pre-sentence modifiers          layers or slots
Post predicate modifiers          for elements
Post sentence modifiers
```

If a sentence in the input stream should have more than one clause, there will be at least one order-rule for each:

```
Unfortunately, I plainly saw the little boy in the big red car that
      1       2 3    4      5   6    7      8  9  19  11  12  13   14

                    the girl loves.
                    15   16   17  18
```



Phase four will make one continuous infix string of these order-rules. First, the rules are checked for those situations where both of the father nodes of an interjunction are available for further junctions. If such a situation exists, then the operations involved are marked. Next, all the layers are inserted into the order-rule, giving the following:

```
Order-rule 1:  (A+E)$(N+(A+E)$(V+(A+E)$N$(P+(A+E)$(A+E)$N)))
                 1     3  4     5  7    8  9 11    12   13

Order-rule 2:  ((N+V)+N)
                16 17 14
```

Notice that the second rule still points to the antecedent in the first order rule. Next all order-rules are conflated into one, by performing the necessary interjunctions between order-rules.

```
(A+E)$(N+(A+E)$(V+(A+E)$((N$V)+N)$(P+(A+E)$(A+E)$N)))
 1     3  4     5  7       8 17 14  9 11    12   13
```

The above infix string is now ready for phase five.

Phase five converts the infix notation into postfix notation to be used as input to the transfer-synthesis steps.

As a summary of this section, we will follow a sample sentence through the five phases of analysis.

Figure 2 shows the processing involved to disambiguate the reference for some of the words in the input string. In this sentence there were several potentially ambiguous sequences (See Figure 3) "TPUT" identifies the computer programs question of the operator and "TGET" shows the reply. Most of these are self-explanatory. This interplay between man and program makes it possible for the computer to build a structure depicting its syntax and semantics for the given context, provided of course that the human responds

correctly to the questions. The output of phase three is several order rules which are merged into one infix structure (Figure 4).

```
 THE(1) SWIMMING(2) COACH(3) LIVING(4) IN(5) C$ALIFORNIA(6) BOUGHT(7)
THE(8) OLD(9) HOUSE(10) ON(11) THE(12) HILL(13) THAT(14) RESEMBLES(15)
THE(16) CHURCH(17) THAT(18) YOUR(19) GRANDFATHER(20) ATTENDED(21)
,(22) WHICH(23) SURPRISED(24) THE(25) PEOPLE(26) IN(27) THE(28)
TOWN(29) .(30)


ENTER MEANING FOR: COACH(3)

 1 N T$HE COACH ENCOURAGED THE TEAM TO DO BETTER.(PERSON THAT TRAINS,
     TUTORS)
 2 N H$E RODE IN THE QUAINT OLD COACH TO THE STATION. (VEHICLE)
 3 V M$Y FATHER WILL COACH THE BASEBALL TEAM. (ACT AS A COACH)


ENTER MEANING FOR: LIVING(4)

 1 N H$E EARNED HIS LIVING BY REPAIRING REFRIGERATORS. (SUSTENANCE)
 2 V I$ LIVE IN THE U$NITED S$TATES OF A$MERICA. (DWELL)
 3 V S$HE CHOOSES TO LIVE THE PRINCIPLES OF THE GOSPEL. (OBEY,KEEP)


ENTER MEANING FOR: IN(5)

 1 A ***PARTICLE OF VERB + PARTICLE COMBINATION***
 2 P T$HE SHOE WAS IN THE KITCHEN. (CONCRETE LOCATION)
 3 P T$HERE SHOULD BE LOVE IN THE FAMILY. (ABSTRACT LOCATION)
 4 P T$HE SENATOR SPOKE IN HONOR OF THE MAYOR'S ACCOMPLISHMENTS.
 5 P L$ET'S GO IN THE HOUSE. (DESTINATION; INTO)
 6 P I$ DID THE EXERCISE IN TEN MINUTES. (WITHIN OR DURING A PERIOD
     OF TIME)
 7 P W$E MUST SAY IT IN WORDS AND ACTIONS. (INDICATE MEANS)
 8 P DUMMY PREPOSITION


ENTER MEANING FOR: THAT(14)

 1 N I$ WOULD LIKE TO READ THAT BOOK. (ARTICLE--FAR DEMONSTRATIVE,
     SINGULAR)
 2 N T$HAT IS THE MOST INTERESTING MOVIE I$'VE SEEN. (PRONOUN--3P,
     S,DEM,NOM)
 3 N I$ HAVE NEVER SEEN THAT BEFORE. (PRONOUN--3P,S,DEM,OBJ)
 4 N T$HE FIRST PERSON THAT CALLS WILL WIN A PRIZE. (TOPIC)
```

Figure 2

```
ENTER MEANING FOR: CHURCH(17)

 1 N H$E HAS BEEN A MEMBER OF THAT CHURCH FOR SEVERAL YEARS. (OR-
     GANIZATION)
 2 N W$E SAW THE CHURCH THAT WAS JUST BUILT. (BUILDING)
 3 N T$HE C$HURCH IS CONTINUING TO GROW THROUGHOUT A$SIA. (L$D$S$
     C$HURCH)
```

Figure 2 (cont.).

```
PHASE 3 SENTENCE 3
*TPUT*   THE(1) SWIMMING(2) COACH(3) LIVING(4) IN(5) C$ALIFORNIA(6)
*TPUT*   BOUGHT(7) THE(8) OLD(9) HOUSE(10) ON(11) THE(12) HILL(13)
*TPUT*   THAT(14) RESEMBLES(15) THE(16) CHURCH(17) THAT(18) YOUR(19)
*TPUT*   GRANDFATHER(20) ATTENDED(21) ,(22) WHICH(23) SURPRISED(24)
*TPUT*   THE(25) PEOPLE(26) IN(27) THE(28) TOWN(29) .(30)
*TPUT*   WHAT TYPE OF PARTICIPLE IS SWIMMING(2)?
*TGET*   NADJ
*TPUT*   WHAT TYPE OF PARTICIPLE IS LIVING(4)?
*TGET*   ADJ3
*TPUT*   WHAT DOES IN(5) - C$ALIFORNIA(6) MODIFY?
*TGET*   4
*TPUT*   WHAT DOES ON(11) - HILL(12) MODIFY?
*TGET*   10
*TPUT*   DOE S THAT(14) REPRESENT HILL(13)?
*TGET*   NO
*TPUT*   DOES THAT(14) REPRESENT HOUSE(10)?
*TGET*   YES
*TPUT*   DOES THAT(18) REPRESENT CHURCH(17)?
*TGET*   YES
*TPUT*   WHAT DOES WHICH(23) REFER TO?
*TGET*   SV7
*TPUT*   WHAT DOES IN(27) - TOWN(29) MODIFY?
*TGET*   26
```

Figure 3.

```
PHASE 4 SENTENCE 3      Output of phase three  (input to phase 4)

*** LEVEL 1 ***       ANTECEDENT 0 FROM LEVEL 0   ORDER TYPE 1

ORDER  NODE   # ( ( N * ( V + N ) ) ' N $ ( A + E ) - ( V +
        REF   1    31     2  32       3    33            7

ORDER  NODE   ( A + E )   N $ ( P + N ) ) ) #
        REF     9        10    11  13

*** LEVEL 2 ***       ANTECEDENT 33  FROM LEVEL 1  ORDER TYPE 5

ORDER  NODE   # ( * ( V $ ( P + N ) + E ) ) #
        REF   2       4     5   6

*** LEVEL 3 ***       ANTECEDENT 10  FROM LEVEL 1  ORDER TYPE 1

ORDER NODE    # ( N - ( V 3 N ) ) #
       REF    3  14    15  17

*** LEVEL 4 ***       ANTECEDENT 17  FROM LEVEL 1  ORDER TYPE 2

ORDER NODE    # ( ( N - V ) 4 ( ( A * N ) + E )   N ) #
       REF    4    18  21        34  19           20

*** LEVEL 5 ***       ANTECEDENT 0   FROM LEVEL 1  ORDER TYPE 1

ORDER NODE    # ( N - ( V + N $ ( P + N ) ) ) #
       REF    5  23    24  26    27  29
```

Figure 4.

B. Transfer

Within the context of Junction Grammar, a transfer grammar from a source language S to a target language T is an algorithm which inputs a junction tree from an analysis grammar of S and outputs the junction tree, with any needed adjustments, to a synthesis grammar of T.

In the present implementation, we are developing transfer grammars from English into Spanish, German, French, and Portuguese (abbreviated SPN, GER, FRN, and POR). Figure 1 is a general diagram of the flow of information in the transfer system.

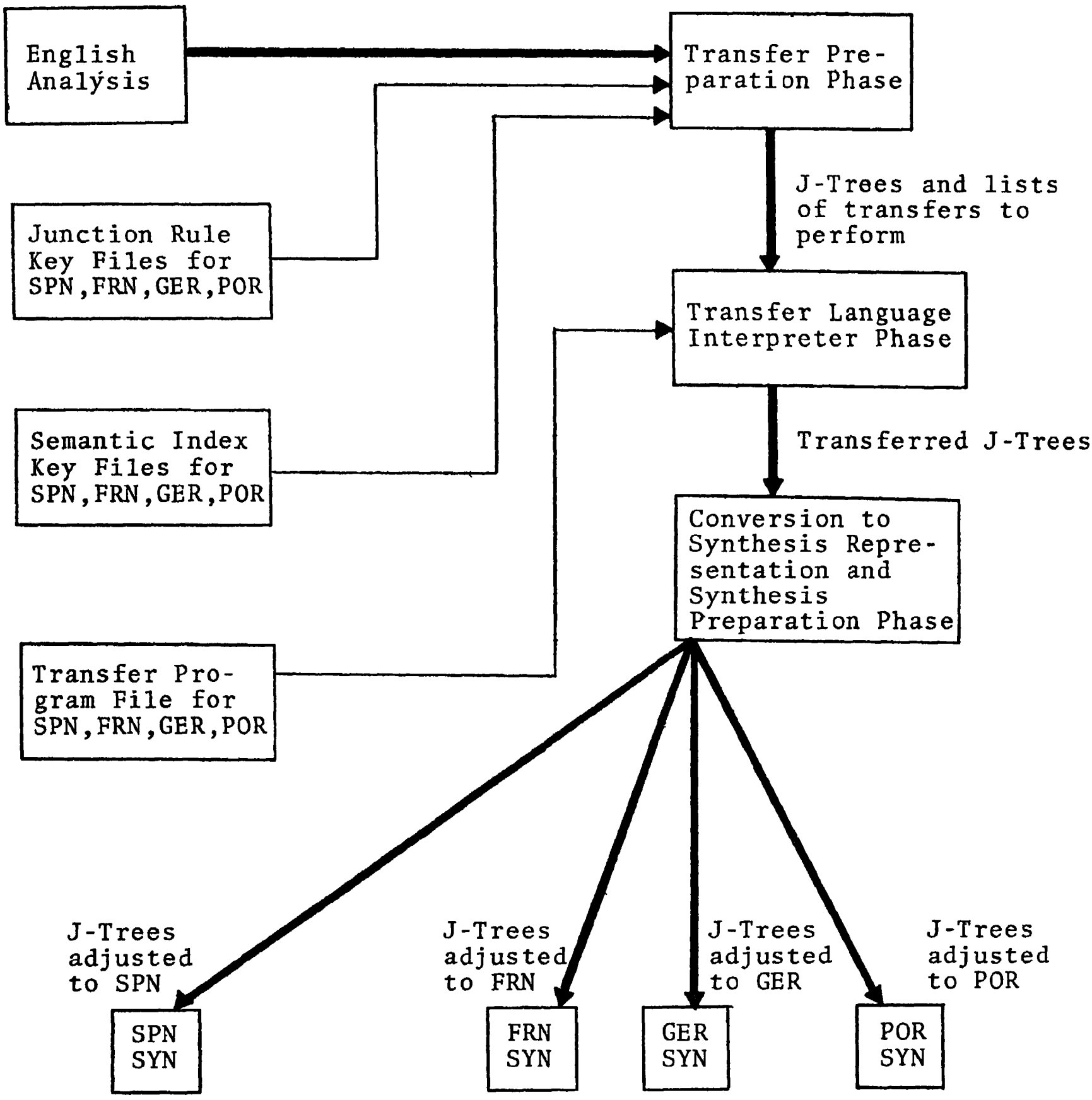

English Analysis
Junction Rule Key Files for SPN,FRN,GER,POR
Semantic Index Key Files for SPN,FRN,GER,POR
Transfer Program File for SPN,FRN,GER,POR
Transfer Preparation Phase
J-Trees and lists of transfers to perform
Transfer Language Interpreter Phase
Transferred J-Trees
Conversion to Synthesis Representation and Synthesis Preparation Phase
J-Trees adjusted to SPN
J-Trees adjusted to FRN
J-Trees adjusted to GER
J-Trees adjusted to POR
SPN SYN
FRN SYN
GER SYN
POR SYN

Figure 1.

In the preparation phase, each junction tree is scanned for junction rules and semantic indices which stimulate transfers going into a given language. This scanning is accomplished by consulting files which associate key junction rules and semantic indices with transfer numbers. The list of transfers is then associated with the tree.

After the preparation phase, the transfer language interpreter phase (Phase two) executes those transfer programs whose numbers have been recorded. Of course, as the transfer executes, it may check the context and decide not to change the junction tree. All the transfer programs are in one file so that several languages can share common algorithms. The third phase of transfer converts the adjusted junction trees to synthesis representation, adding to them features from the target language lexicons. None of the three phases of transfer contain any language-specific coding; only the files are language-specific. Thus, a transfer grammar can be made machine executable simply by loading records into the key files and the transfer program files.

Transfer programs are written in a linguistic programming language called Transfer Language (TL) (see [8]). The TL loader compiles the programs into a pseudo-machine language which is stored in the transfer program file and is interpreted at execution time in the second phase of transfer. Before describing TL in detail, we present an entry from a simple transfer grammar. Figure 2 shows the necessary input to the TL loader and the semantic key file loader. This figure shows the form of entries loaded under

the current transfer system implemented on an IBM 360.

```
//GRAMMAR JOB  (...)
//LOADPGM EXEC TRNTLSLD  *THIS PROCEDURE LOADS TRANSFER PROGRAMS*
19
*SIMPLE TRANSFER FROM "X MISSES Y" TO "Y MISSES TO X"
*E.G. "HE MISSES HER" TO ELLE LUI MANQUE"
*THIS TRANSFER INSERTS INTO THE JUNCTION TREE THE INDIRECT OBJECT
*PREPOSITION, WHICH HAS BEEN ASSIGNED SEMANTIC INDEX 1ØØ86
LET =2 BE Y(A(A(=1)))
LET =3 BE Y(A(=1))
REPLACE =2 WITH =3
REPLACE =3 WITH E
JOIN =1 $ (P1ØØ86 + =3)
*)
//LOADKEY EXEC TRNTKLD,LANG=FRN  *THIS PROCEDURE LOADS KEYS*
//*WHEN GOING INTO FRENCH, STIMULATE TRANSFER 19 FROM SEMEME Ø9172.ØØØ
//*ANY OTHER VERB WHICH CHANGES FROM "X VERB Y" TO "Y VERB TO X" COULD
//*ALSO KEY INTO TRANSFER 19
FRNØ5916.ØØØ 19
//
```

Figure 2.
A simple transfer program and its key.

The details of TL needed to fully understand Figure 2 will be explained later. For now, let us consider the simple linguistic principle behind the example. When translating "I miss my brother" into French, brother becomes the syntactic subject, giving "mon frere me manque," which, taken literally back into English, is "my brother (to) me misses." In terms of verb orientation, the English verb to miss is normally used as a "subject is interested person" verb while in French the sense wish someone were here is a "subject is interesting person" verb. Note, however, that the other sense of to not hit is a "subject is interested" verb in French, the same as English, i.e. "I threw the ball but I missed

him" is translated "J'ai lance la balle mais je l'ai manque," and receives a different semantic index in analysis. Therefore, it does not stimulate a transfer as does the first word sense.

In Figure 2, the desired adjustment from "X misses Y" to "Y misses X" is accomplished by a transfer program which has been numbered 19. In transfer 19, the first two statements (LET...; LET...) set pointers to crucial nodes of the junction tree and the next three statements (REPLACE...; REPLACE...; JOIN...) make the direct object into the subject and the subject into the indirect object. The transfer will be executed at the appropriate time because at the bottom of Figure 2 we associate the semantic index 5916 (to miss someone, i.e. wish he were there) with transfer 19 going into French (FRNØ5916.ØØØ 19). The qualifier '.ØØØ' attached to the semantic index is a sememic refinement code used to make very fine distinctions if needed. Work on this aspect of our lexicon is in its formative stages and will be reported later.

With the preliminary example of Figure 2 in mind, let us now examine transfer language as a programming language.

GENERAL STRUCTURE OF TL

TL is a free-format, block-structure, list-processing language similar in general form to PL/I or ALGOL. Internal to a transfer program are DO-END blocks and external are subroutine blocks. Computer scientists may be interested to know that TL contains no GOTO-type statement. The central data structure is a

junction tree, represented as one doubly-linked list whose internal structure is logically a set of intersecting binary trees in postfix notation. Of course, TL shields the user from the details of linked list manipulation through the use of high-level commands.

The user data types are: (1) address variables (e.g. '=4') which point to nodes of the junction tree, (2) condition variables (e.g. 'C4') which can be true-false or contain integer values (such as semantic indices), (3) parameters (e.g. 'P2') which contain optional information associated with a sememe key in addition to the number of the transfer program to invoke, and (4) integer constants.

As previously mentioned, a transfer grammar is a set of keys and transfer programs. The transfer programs are not ordered relative to each other; their execution is stimulated by the presence of a key sememe or junction rule in a junction tree. When a given top-level transfer (i.e. one that is invoked directly by sememe or rule, not invoked by another transfer) is executed, address variable one ('=1') is pointed at the sememe or junction rule which stimulated the transfer program. Normally, a top-level transfer program will begin with some statements which derive (from the address in =1) the addresses of some other crucial nodes in the junction tree. Then, except in the simplest cases, this context will be tested and appropriate manipulations, if any, will be performed on the junction tree. If errors are detected during the execution of one of the transfers on a list, control is normally returned to the transfer supervisor, which continues processing the sentence by invoking the next transfer on the list. However, the user may

choose to set up "ON UNITS" which intercept error conditions and invoke some specified transfer designed for error recovery.

TRANSFER LANGUAGE STATEMENT TYPES

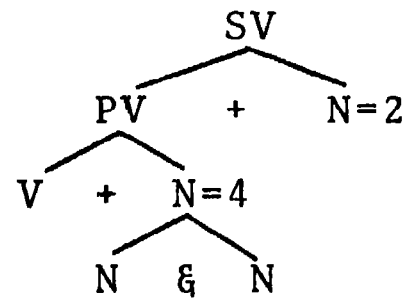
Let us now consider the most important statement types in TL, beginning with assignment, which is accomplished with the LET statement.

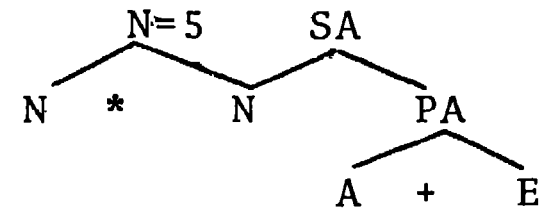
LET. One form of LET is used to move around the junction tree. For example, if =3 is set to the address of the verb of a clause, we can point =5 to the direct object, skipping over any verb-level modifiers, by the statement: LET =5 BE Y(A(=3)). The built-in function A moves to the next level of adjunction (e.g. from a V to its PV) and Y moves down the tree to the right brother.

Other built-in functions include X (which moves down to the left brother), L (which moves up to the father), B (which moves down successive sons, or left brothers, until reaching a terminal node), S (which moves from the father of a point of intersection in a ranking tree to the corresponding father node in the subordinate tree), R (which moves from subordinate trees to ranking trees), and C (which moves to the SV governing the start node). In the example LET =5 BE Y(A(=3)), the movement is relative to an initial node pointed to by =3. In this initial address slot we can also use the symbol 'H', which stands for the topmost node of the highest ranking subtree of the junction tree, or '=', which stands for the address of the most recently created node in the tree.

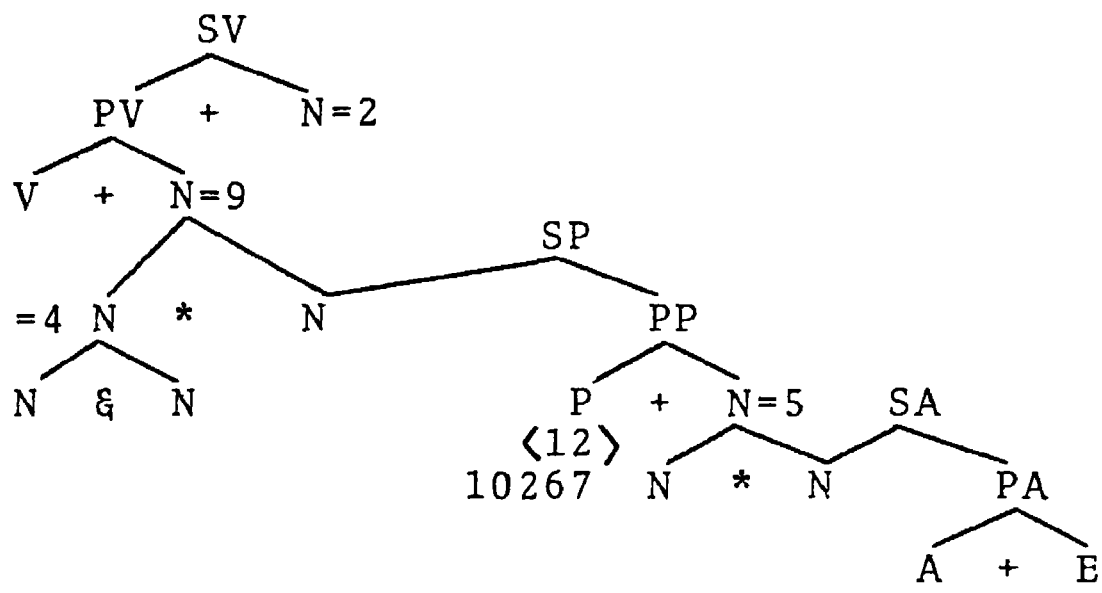
Another form of LET assigns values to condition variables. When an address variable is assigned to a condition variable (e.g. LET C7 BE =5), the condition variable receives the semantic index of the node pointed to by the address variable. Also available in TL is an easily expandable set of built-in functions to perform such tasks as indicating the category of a node, checking for the presence or absence of certain semantic features on a node, or reporting on the structural context of a node. For example, LET C7 BE INODCAT(=8) sets C7 to a number indicating the category of the node pointed to by =8. Other special forms of the LET statement set features and the special pointer 'H', which indicates the top of the junction tree.

JOIN. The JOIN statement allows the user to insert any desired structure into the junction tree. The dollar sign ($) is used to indicate an interjunction; otherwise, the symbols used in the JOIN statement are a straightforward infix representation of a segment of a tree. Thus, if we assume the fragmented tree:



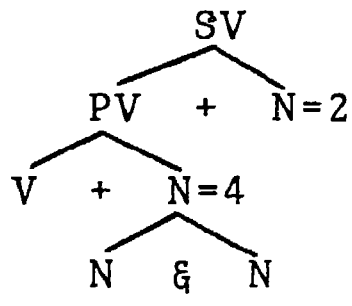


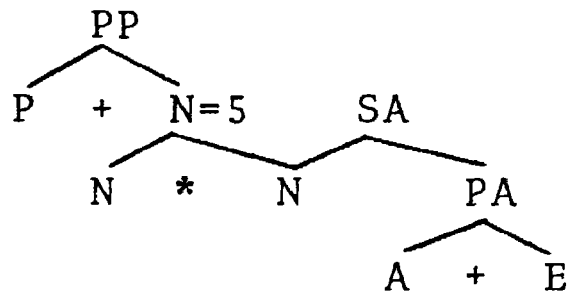
and execute the statement JOIN =4 $ (P1Ø267⟨12⟩ + =5) /=9, we obtain the tree:



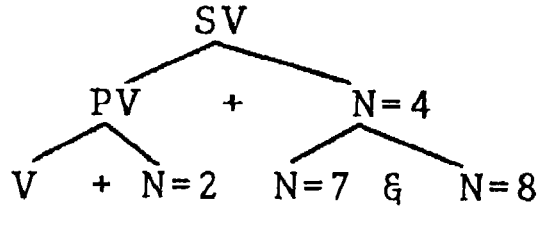
whose P node contains the semantic index 10267 and the semantic feature number 12. Also, the /=9 indicates that the primary result of the last performed operation should be pointed to by =9

UNJOIN. If we then execute UNJOIN =9 on the junction tree just obtained, we undo the effect of the interjunction, obtaining the trees:

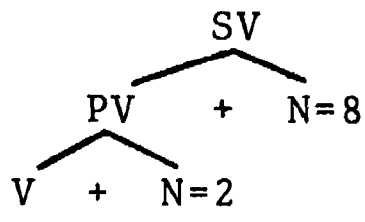




REPLACE. If, using the same tree, we execute the statements LET =7 BE X(=4), LET =8 BE Y(=4), and REPLACE =4 WITH =2, we set some pointers and switch subject and the object, obtaining:



Finally, if we replace the left conjoined part of the subject with nothing, i.e. by the statement REPLACE =7 WITH Ø, then we obtain:



Note that the unneeded operation and label node are deleted automatically.

IF. The IF statement with optional ELSE clause is simply borrowed from PL/I, but one form of the IF statement deserves comment. An expandable set of KEYWORDS is available, each of which indicates that a built-in function is to be called and its value compared against a predefined constant associated with the KEYWORD. For example, IF =14 ISA VERB--- is equivalent (using C49 as a work variable) to the two statements:

```
LET C49 BE INODCAT(=3)
IF C49 EQ 5 THEN---
```

since 5 is the number which stands for the category V.

SKIP and HALT. SKIP exits from the current transfer program, while HALT returns control to the supervisor, which moves on to the next top-level transfer.

TRANSFER. The statement TRANSFER 11 is simply a call to transfer number 11 as a subroutine, while the statement TRANSFER C4 is a call to that transfer program whose number is in C4 at the time that statement is interpreted. There seems to be no need for an algebraic do-loop in transfer but a while-loop is useful. For example, TRANSFER 12 WHILE C2 executes transfer number 12 repeatedly until some context test within transfer 12, or some other transfer called by it, sets condition variable 2 to false (i.e. to zero).

ON. The statement ON CONDITION(NO-LABEL) TRANSFER 1 sets up an ON UNIT so that if at any time during transfer on the current node an attempt is made to move up from a node without a label, transfer 1 will be executed. If an error occurs in transfer 1, control is then passed to the supervisor which processes the next top-level transfer.

A few final details should be mentioned. When a transfer is called, it executes to its end or until a SKIP statement is executed. Comments can be scattered through a transfer program by enclosing them by '/* ----- */' or beginning the record with '*'

Figure 3 summarizes the selected statement types we have discussed.

```
- LET =4 BE Y(A(=2))
- LET =9 BE H
- LET =11 BE =
- LET C7 BE 4
- LET C7 BE =5
- LET C7 BE C2
- LET C7 BE INODCAT(=8)
- LET C7 BE P4

- LET FEATURES(=9) BE <-SINGULAR,+MASS>
- LET H BE =9

- JOIN =4 $ (P1Ø267 12 + =5)  /=9
- UNJOIN =12
- REPLACE =10 WITH =2
- REPLACE =3 WITH Ø
- IF C9 IS TRUE THEN stmt
- IF C2 EQ 12 THEN DO stmt1 stmt2 . . . END
- IF =14 ISA VERB THEN stmt ELSE stmt
- TRANSFER 11
- TRANSFER C2
- TRANSFER 12 WHILE C2
- SKIP
- HALT
- ON CONDITION(14) TRANSFER 2
```

Figure 3.

Sample Transfer Language Statements.

SAMPLE TRANSFERS

Having discussed the major statement types in TL, let us consider a few specific transfers. This will be but an indication of what is done in transfer, but later reports are expected to present substantial transfer grammars between language pairs.

Languages frequently differ in their preferred conceptualization of a given action. For example, English prefers eat supper to sup, while French prefers souper to prendre le souper. Similarly, to watch television has become the single sememe fernsehen in German. Alternatively, a verb and non-verbal participle [9] may correspond to a single verb (e.g. make happy goes to alegrarse in Spanish). Several examples of such patterns are listed in Figure 4.

All these correspondences are handled in TL by writing one simple transfer for each general pattern and then loading the appropriate keys. For example, the first case (V + N → V + E) could be treated by keying on the verb in each case and loading as parameters with the key the crucial object or objects. This will cause collapse and the composite sememe to be formed. In order to make this and the subsequent examples easier to follow we will use the English words themselves instead of their semantic indices. Thus, we might load the following keys:

```
FRN EAT      12   P1 = LUNCH, JOIN DEJEUNER =11
                  P2 = SUPPER, JOIN SOUPER =12

SPN GIVE     12   P1 = THANKS, JOIN AGRADECER =11

GER TAKE     12   P1 = WALK, JOIN SPAZIERGEHEN =11
                  P2 = BATH, JOIN BADEN =12

GER WRITE    12   P1 = POEM, JOIN DICHTEN =11
```

- V + N ⟶ V + E

  to eat lunch ⟶ dejeuner (FRN)
  to eat supper ⟶ souper (FRN)
  to eat breakfast ⟶ desagunar (SPN)
  to give thanks ⟶ agradecer (SPN)
  to take a walk ⟶ spaziergehen (GER)
  to take a bath ⟶ baden (GER)
  to have an accident ⟶ verunglücken (GER)
  to write a poem ⟶ dichten (GER)

- V + E ⟶ (V + E) $ (P + N)

  to point ⟶ montrer du doigt (FRN)
  to clap ⟶ battre des mains (FRN)

- V $ (A * (A + E) + E) + E ⟶ V + E

  to make sad ⟶ attrister (FRN)
  to make angry ⟶ encolorizarse (SPN)

- V $ (P + N) ⟶ (V + N) $ (A + E)

  to stare at X ⟶ regarder X fixément (FRN)

- V + N $ (P + N) ⟶ V + N

  to turn the pages of a book ⟶ feulleter un liver (FRN)

Figure 4.

Some multisememe-single sememe associations.

and treat them all with the following transfer:

```
12
*SET =2 TO THE OBJECT
LET =2 BE Y(L(=1))
*SEE IF THE OBJECT CORRESPONDS TO ONE OF THE PARAMETERS
IF (=2,=3) ISA MATCH THEN DO
REPLACE =1 WITH =3
REPLACE =2 WITH E
END
*)
```

This transfer is straightforward except for the statement "IF (=2,=3) ISA MATCH...." Here it is assumed MATCH has been defined as a key-word so that this statement is equivalent to the pair of statements:

```
LET C49 BE SMATCH (=2,=3)
IF C49 IS TRUE . . .
```

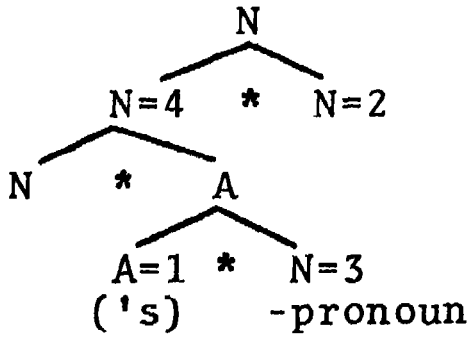
The built-in function SMATCH checks the parameters of the stimulating key against the index of the node pointed to by the first argument (=2). It either returns FALSE (no match) or TRUE and points the second argument (=3) to the proper composite node made available through the auxiliary JOIN parameters of the transfer key.

In this simple transfer, any modifiers on the object would cancel the transfer by blocking a match (e.g. eat a big supper ⟶ manger un grand souper), but if the transfer writer so desired, he could easily UNJOIN an adjective on the object and JOIN it onto the predicate as an adverb (e.g. SOUPER GRAND). This could be done by expanding the first statement to the following:

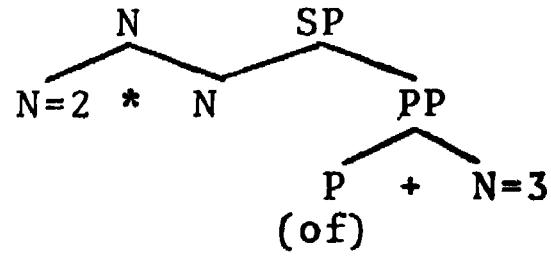
```
LET =4 BE Y(L(=1))
LET =2 BE B(=4)
IF =4 ISAN INTERJUNCTION THEN DO
LET =5 BE Y(S(=4))
LET =6 BE L(=1)
UNJOIN =4
JOIN (ADV * =4) + E
JOIN =6 $ =
END
```

Now let us consider a transfer stimulated by a difference in productivity of a junction pattern between two languages. English and French both have possessive adjectives (his, my → son, sa, ses, mon, ma, mes) but on non-pronouns the English adjectival form becomes prepositional in French, e.g. my friend's mother's kitchen → la cuisine de la mere de mon copain (the kitchen of the mother of my friend).

Fortunately, in TL we need not consider the embedded cases separately from the simple case; rather we can write one transfer which will be involved automatically by the transfer supervision for each occurrence of the possessive. The basic transfer should modify structure in the following way:



becomes:

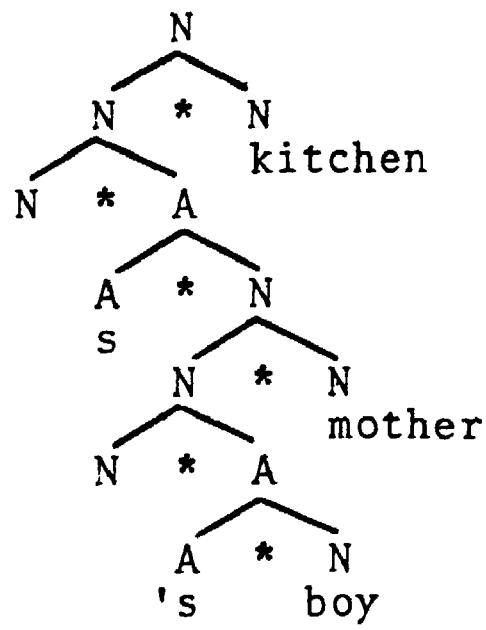


This might be done by the following transfer (stimulated by 's):

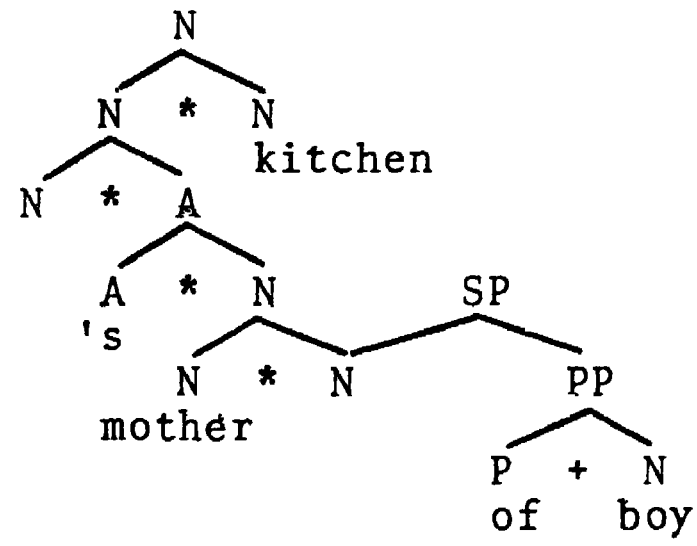
```
17
LET =3 BE Y(L(=1))
IF =3 ISMARKED -PRONOUN THEN DO
LET =4 BE L(L(=1))
REPLACE =4 WITH Ø
REPLACE =3 WITH Ø
JOIN =2 $ (OF + =3)
END
*)
```

In the case of one embedding the boy's mother's kitchen, the transfer would be executed on boy's and then on mother's, changing structure as follows:

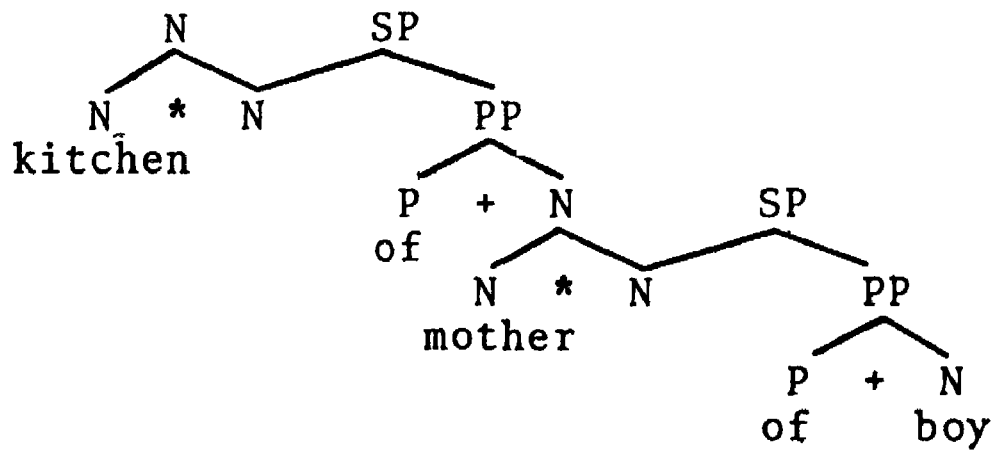
(1) before transfer:



(2) after first execution of transfer 17:



(3) finally, after second execution of transfer 17:



Before leaving this second example, it should be remarked that in this transfer, as with all transfers, we do not consider questions of differing word order or inflectional patterns between languages. In transfer we simply modify the structure, and synthesis later produces the proper word order and inflection through the lexical-rule system.

As a final example we will consider a simplified version of an extensive transfer written as a major component of a thesis which considered several forms of the English passive [10]. In our simplified version we do not make many of the needed checks to weed out unwanted modifiers, etc.

Consider the three closely related sentences:

(1) John gave him a book.
(2) He was given a book by John.
(3) A book was given to him by John.

Many grammars would consider these sentences to have the same meaning and give them the same underlying representation. Junction Grammar, taking a slightly different approach, chooses to represent them in such a way that indicates both their similarities and their differences. This attitude is based on two assumptions of Junction Grammar.

(1) The path between deep and surface structure should generally be direct, not tortuous, and

(2) Any difference in structure implies some difference, however minute, in meaning, since meaning (in Junction Grammar) is defined as the composition of structure, reference, and context.

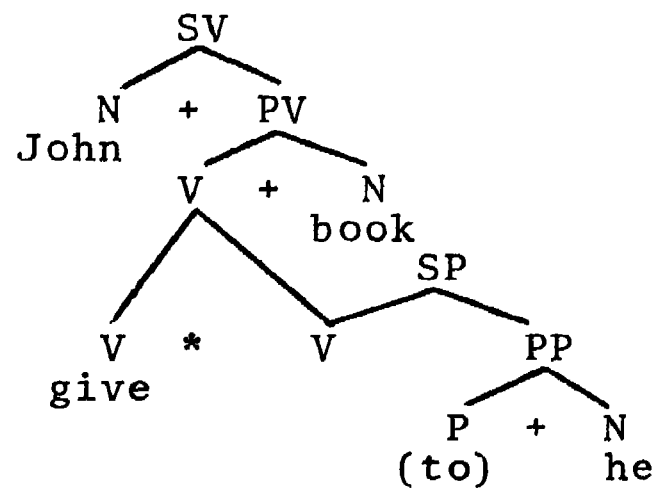
However, the philosophical gulf between Junction Grammar and other grammars is not as wide as might appear at first, for if in any particular application area it is useful to ignore such differences as those between the passive and the active, standard junction trees can easily be passed through a special normalizing transfer grammar. One transfer in such a normalizing grammar would be the following, which is stimulated by the rule A * PV and transfers sentences of type (2) and (3) into active sentences of type (1), examples of which are shown in Figure 5. (Going into this transfer, it is assumed that =1 is set to the label node of the rule A * PV.)

```
15
*PASSIVE-ACTIVE TRANSFER
*SEE SAMPLE JUNCTION TREES IN FIGURE 5
*FIRST, DETERMINE WHETHER THIS IS INDEED A PASSIVE SENTENCE.
LET =2 BE Y(=1)
IF =2 NOTMARKED PASSIVE-PARTICIPLE THEN STOP
*FIND THREE ELEMENTS--THE AGENTIVE, THE OBJECTIVE, AND THE DATIVE,
*(TO DRAW ON FILLMORE'S TERMINOLOGY).
LET =3 BE Y(X(S(L(=1))))
LET =4 BE Y(=2)
LET =5 BE Y(X(S(X(=2))))
*ALSO FIND THE SYNTACTIC SUBJECT.
LET =6 BE Y(A(A(L(T(=1)))))
*SWITCH THE PASSIVE CONSTITUENT (IT MAY BE DATIVE OR OBJECTIVE)
*WITH ITS COREFERENT, THE SYNTACTIC SUBJECT
IF =5 ISMARKED PASSIVE-CONSTITUENT THEN DO
REPLACE =5 WITH =6
LET =6 BE =5
END
ELSE DO
REPLACE =4 WITH =6
LET =6 BE =4
END
```

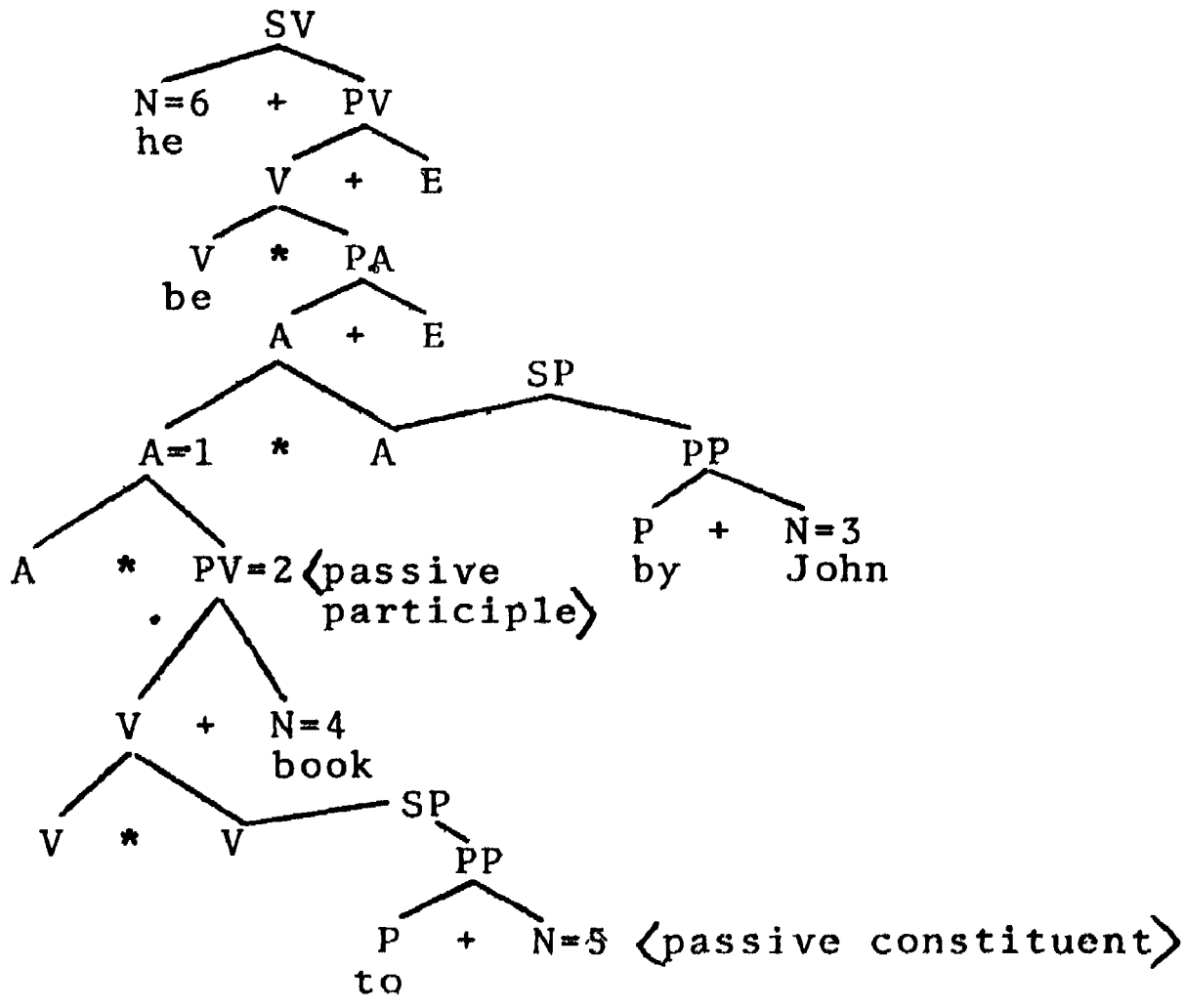
```
*NOW TRANSFER FROM PASSIVE TO ACTIVE BY PLACING THE AGENTIVE ELE-
*MENT AS THE SUBJECT AND THE PASSIVE PARTICIPLE AS THE PREDICATE.
REPLACE =6 WITH =3
LET =7 BE X(L(=6))
*REPLACE THE PREDICATE WITH A DUMMY ONE.
REPLACE =7 WITH PV=8
REPLACE =8 WITH =2
*)
```

[Figure 5 begins here and continues on the following page.]

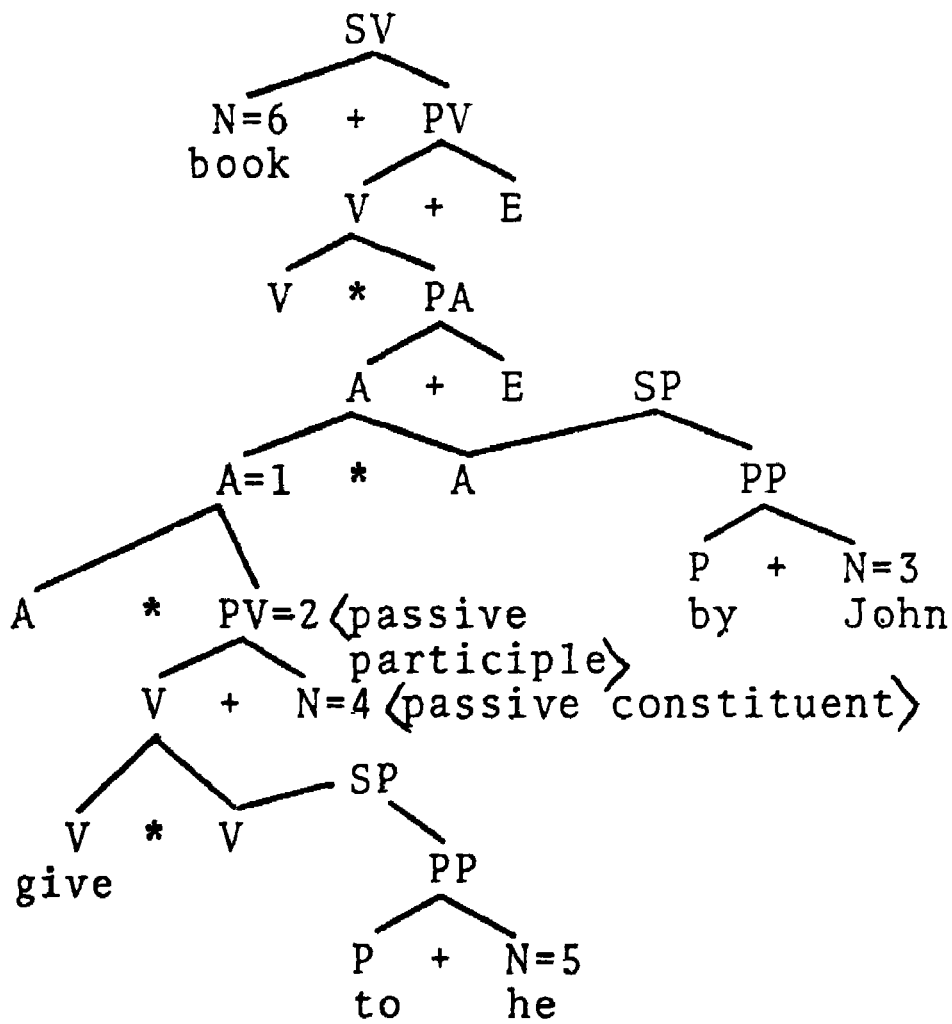
(1) John gave him a book.



(2) He was given a book by John.



(3) A book was given to him by John.



The above description of Transfer Language and the sample transfers should convey some idea of transfer within Junction Grammar, but it should also point out the difficulty of comparing Junction Grammar transfer with the intermediate adjustment phase of some other translation system (e.g. [11], [12], [13]). The reason for the difficulty is simply that a transfer phase is tightly interlaced with the analysis and synthesis phases and the theoretical base. For example, we do some adjustments in transfer which otner systems neutralize in analysis, while some other systems consider aspects of word order and word choice in transfer which we handle in analysis and synthesis.

C. Synthesis.

The input to this program is a junction tree or J-tree. The program applies lexical ordering rules, lexical hiatus rules, lexical matching rules, lexical agreement rules, and graphological and phonological adjustment rules. Figure 1 shows the synthesis mainline in a simplified form which omits some details pertaining to the processing of intersecting trees and discontinuities.

Basic processing follows the figure. First a J-tree is read in to the computer. The perspective routine recognizes some essential relationships which are implicit in the J-tree and makes them available to the synthesis program in an explicit form. Then, beginning with the topmost node of the J-tree, the cycle processes the nodes one at a time. Label nodes are processed by the hiatus and ordering routines. The LHO algorithm determines the ordering procedure which should be employed and takes the steps necessary to implement it. These steps always include designating the order in which the operands of the label node should be processed. They may include the marking of hiatus (understood) elements, discontinuous elements, and insertion points.

Processing continues through the J-tree in the sequence designated by the LHO algorithm until the entire J-tree has been interpreted. Terminal nodes are interpreted by the LMA (lexical matching and agreement) routines and appropriate lexical manifestations are generated.

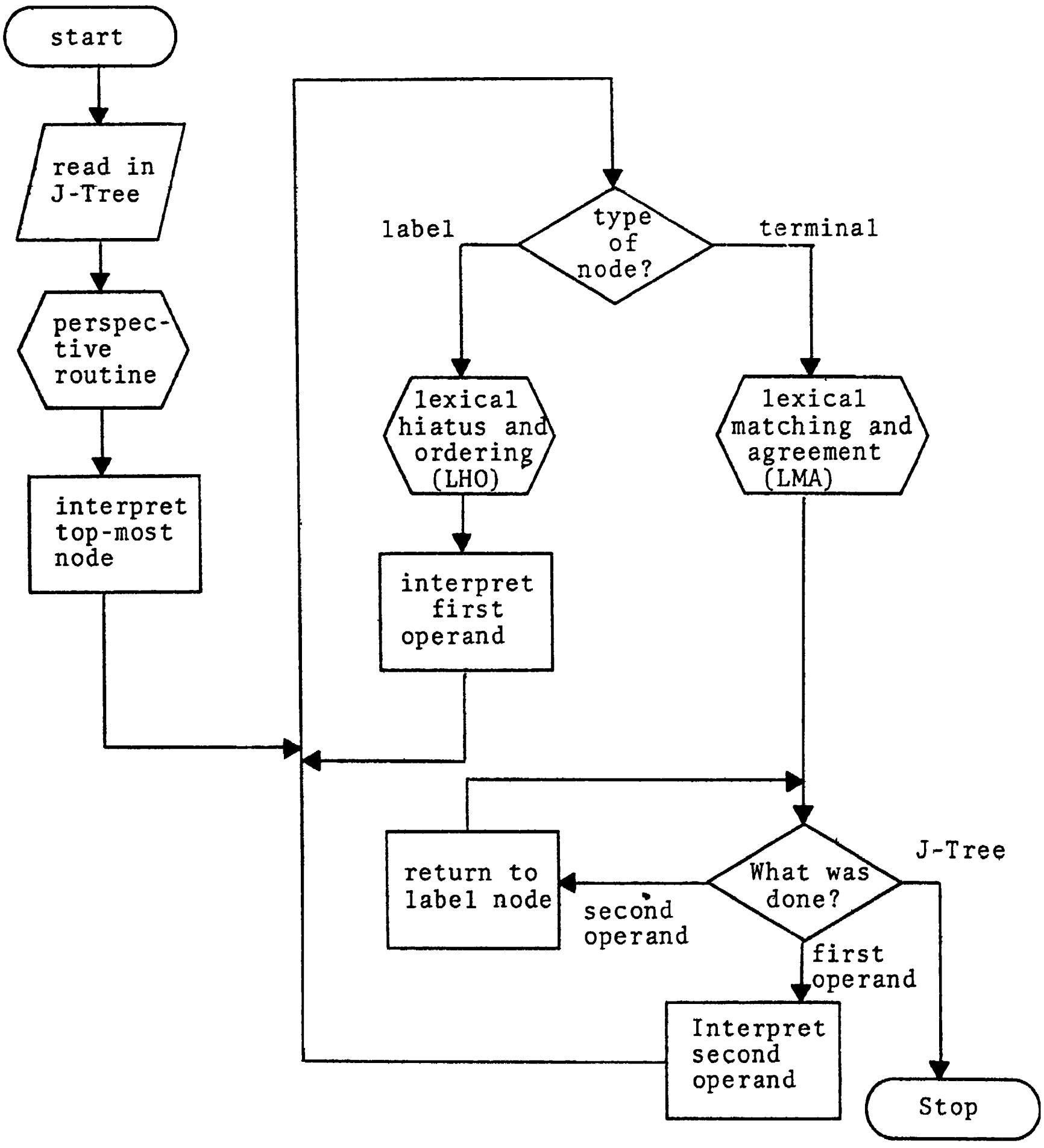


Figure 1.

The Simplified Synthesis Mainline.

Two types of lexical ordering rules are used in the synthesis programs. There are: (1) left-right and right-left ordering of operands, and (2) discontinuous ordering. These will be considered individually

(1) When the LHO (lexical hiatus and ordering) algorithm interprets a label node, it designates one of its operands as the first operand and the other as the second operand. Processing then moves to the operand designated as the first. At the top of the chart the flow of processing is controlled by whether a node is terminal or a label. At the bottom of the chart, the order of processing is determined by whether the node which has just been processed was designated as a first or a second operand. Processing moves from the first operand to the second and from the second back to the label node.

This seemingly simple ordering procedure gives the junction grammar system great flexibility and power. A single junction which has two operands has two ordering possibilities. However, if one of those operands is itself a junction with two possible orders, then the string is capable of four orders. In a similar manner, a string of three junctions can be ordered in eight distinct ways. In general, a tree composed of N junctions can be ordered in 2^n distinct patterns.

(2) Discontinuous ordering exists when sentences elements which are most closely related structurally are not contiguous. The sentences "It surprised me that he came" and "It is so big that I can't lift it" have discontinuous order. With continuous

ordering they would read "It that he came surprised me" and "It was so that I can't lift it big."

The synthesis program handles discontinuous ordering by redirecting the flow of processing in such a way that the discontinuous element is omitted at its normal position and processed instead at a predetermined insertion point. Figure 2 shows the synthesis mainline with the reset routines and their skip points. Figure 3 shows the reset routines in detail.

If the flow of processing is followed as shown in the figures, the operation of the reset routines is relatively simple. On each cycle the reset routines check the node which is being interpreted to determine if it has been designated by the LHO algorithm as an insertion point or a discontinuous element. If it has not, no action is taken. If it has, normal processing is interrupted to achieve the desired order. For instance, if the reset routine at the top of the diagram detects that a node is a discontinuous element, it causes the flow of processing to jump to the second skip point. This bypasses the LHO and LMA routines, in effect omitting the discontinuity. At the bottom of the chart processing resumes just as if the discontinuity had not been omitted. In a similar manner the reset routines direct processing from the insertion point to the discontinuity and from the discontinuity back to the insertion point.

The ordering of intersecting structures is quite similar to the ordering of discontinuous elements. When the point of intersection is reached, the flow of processing shifts to the top of

the subordinate structure and that entire structure is processed. Then processing shifts back to the main structure which is completed in the usual manner. Thus, the point of intersection has some similarity to an insertion point while an intersecting structure resembles the discontinuous element. This same procedure is followed when there are several intersections.

The lexical matching rules access the target language lexicon. The lexical agreeement rules examine the junction and feature environment in which the particular sememe appears in order to determine which inflected form should appear.

At present, most inflected forms appear in the lexicon in full. Others, however, are generated by the synthesis program from stems and endings. The two approaches are theoretically equivalent since most of the inflected forms in the lexicon are generated from stems and endings by the lexicon building routine anyway.

Thus, the synthesis program interprets a J-tree to generate an output string. When this output string has been adjusted by the graphological and phonological rules, which shape the final form of the output, the target language text appears.

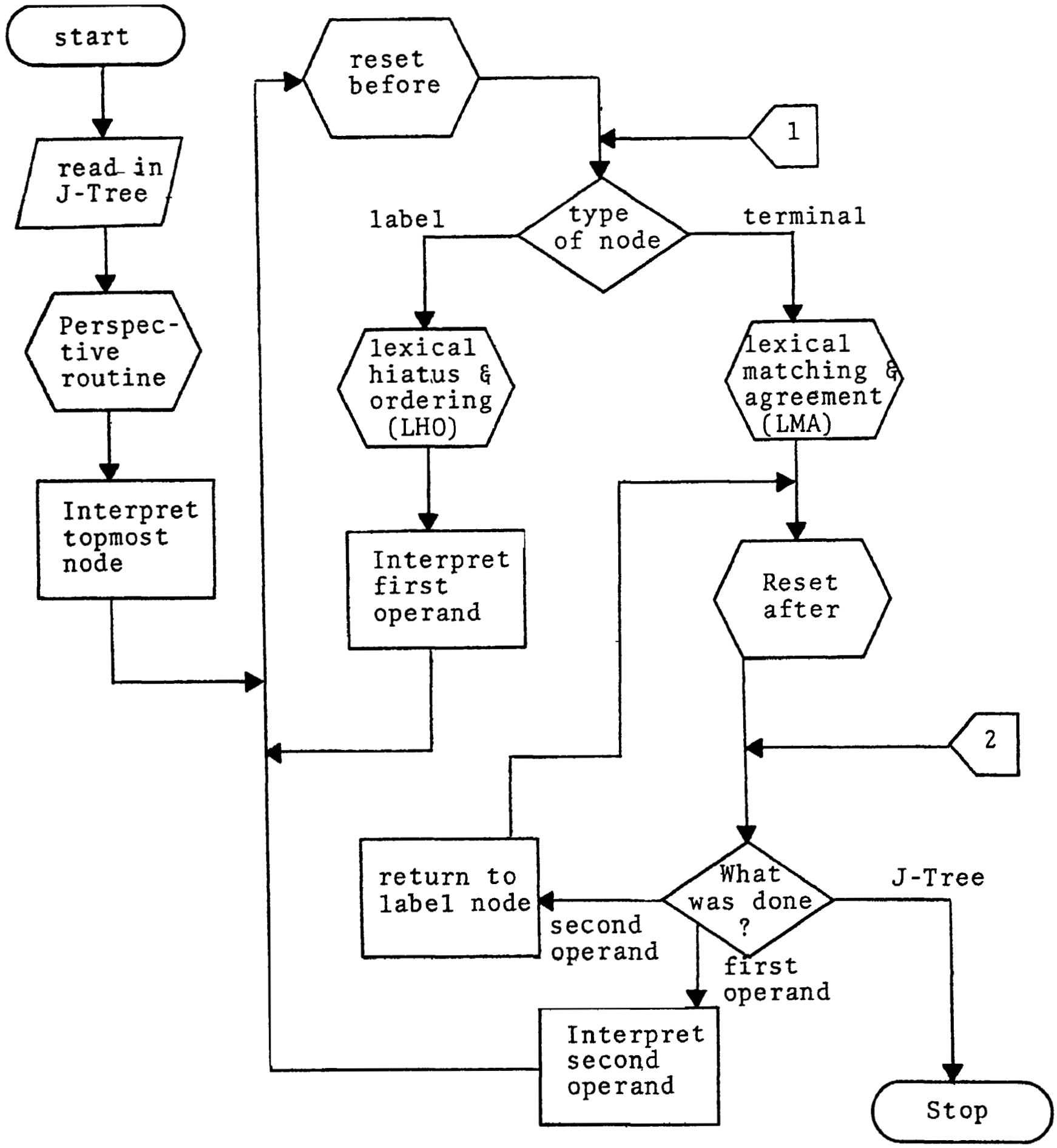
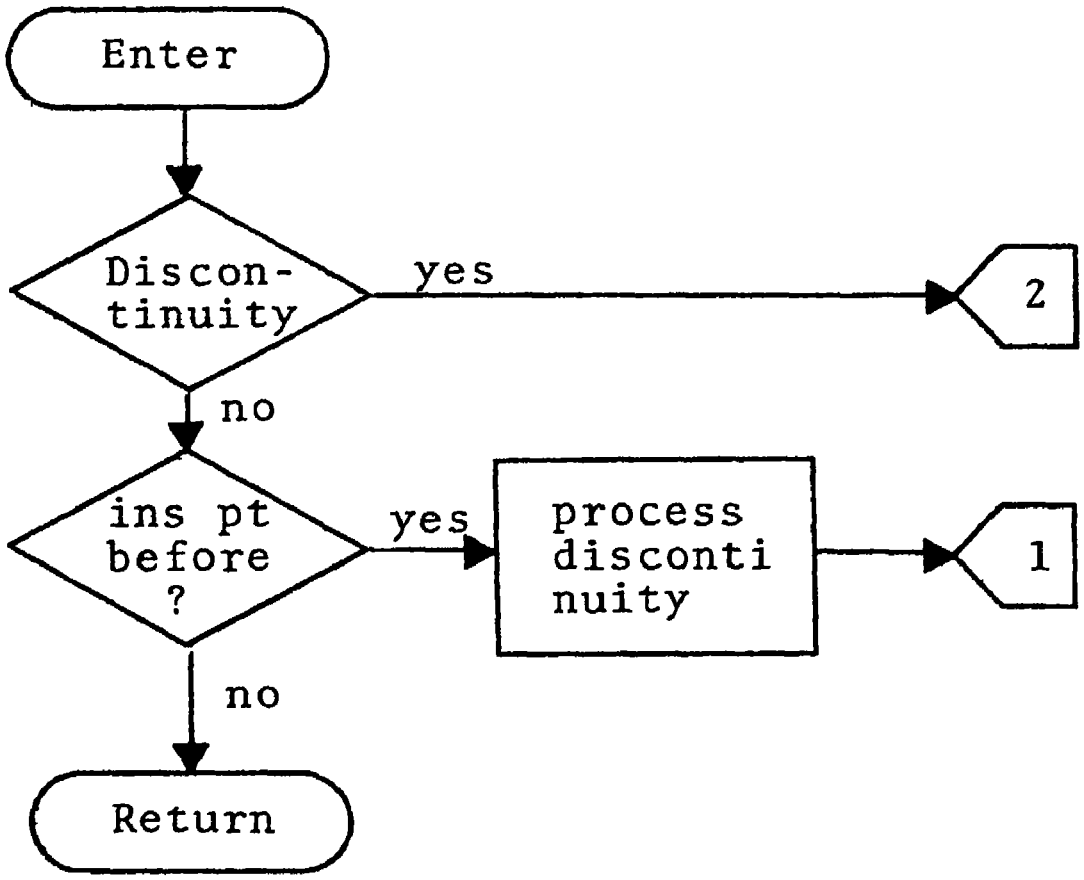
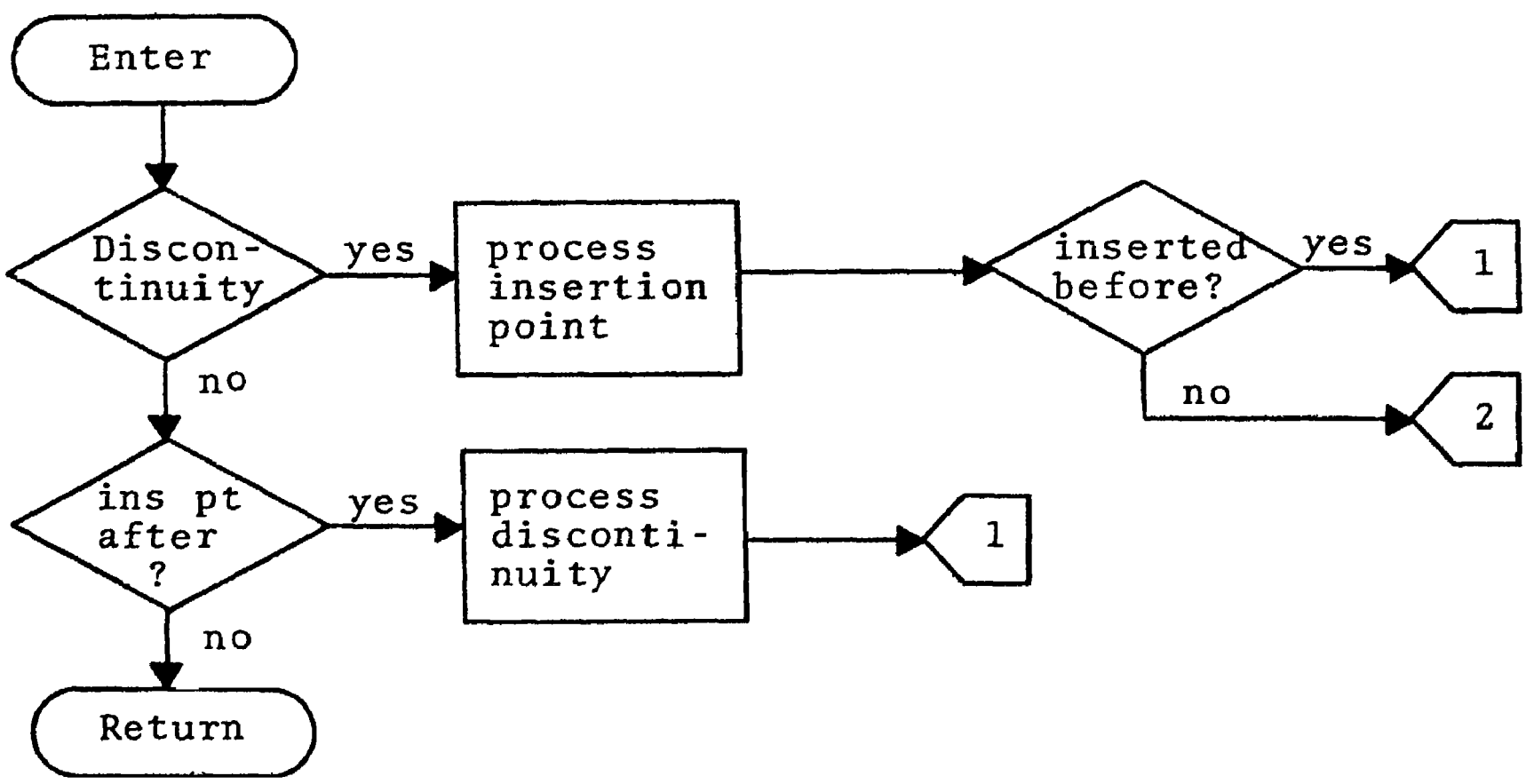


Figure 2.

The Synthesis Mainline Including the Processing of Discontinuities.



Reset Before



Reset After

Figure 3

The Reset Before and Reset After Routines.

IV. Conclusion.

It is proposed that junction schemata of the type illustrated constitute a universal pool of structural patterns available to all languages. Presumably, most languages use the majority of them but no one language need use them all. Perhaps no two languages use exactly the same subset, nor any two persons speaking the same language, for that matter. Languages can be expected to vary in their manner of realizing junction patterns as surface strings. Specifically, the lexical stock, word order, concord, inflection and conjugation paradigms, etc., are known, for the most part, to be language specific.

Our experience to date supports such expectations. For example, the basic word order for Japanese can be obtained by shifting the left-right options on subjunction used by English to right-left. While English uses heterogeneous interjunction freely ("John passed, which surprised us", etc.), Japanese does not; it either uses the corresponding full-subjunction ("That John passed surprised us"), or two conjoined sentences ("John passed, and this surprised us"). Of course, the lexical stock and morphophonemic patterns of Japanese are radically different from those of English. But the conclusion we have been forced to repeatedly is that while surface phenomena are vastly disparate from language to language, junction phenomena are more alike than different.

While we use the familiar ANALYSIS/TRANSFER/SYNTHESIS scheme as a general framework for our translation system, the design of these components is rigidly governed by the junction grammar model, with junction trees serving as the interlingua.

Analysis, operating in an interactive mode, produces junction trees from the source text. The data thus obtained is then passed to the transfer segment of the system.

Transfer from a source language A to a target language B begins with inspection of every junction rule and semantic index appearing in the junction trees produced by analysis. If some rule or semantic index of A does not belong to the subset used by language B, then a transfer subroutine adjusts the junction tree. The library of A-B transfer subroutines is a partial contrastive grammar of languages A and B. However, since word order and other lexical phenomena are abstracted away from the semantic component in our model, these do not enter into transfer at all, but are handled independently by lexical synthesis for language B.

The adjusted trees output by transfer are passed to a synthesis program. The four steps of the synthesis process correspond to four lexical rule types--ordering, matching, ellipsis (or hiatus), and agreement.

Our development group is currently engaged in developing English analysis and transfer-synthesis for translation into Spanish, French, and German, using the University's IBM 360/65. Our system's dictionaries are largely based on a vocabulary drawn from a data base of materials published for the membership of the

LDS (Mormon) Church, and hence they are not technically oriented. at present. Currently, each lexicon contains entries for over 10,000 semantic indices and we hope to double this size by the end of 1976.

We expect that this prototype will provide a gauge for the utility of various on-line interaction techniques for computer-assisted translation systems and for the feasibility of achieving commercial grade computer-assisted translation via the junction grammar model of language described herein. Work is also in progress to develop a phonological component of Junction Grammar that accounts for the connection between prosodic features (pitch, amplitude, duration, pause, etc.) and semantic considerations of sentences.

REFERENCES

1. Montague, Richard (1970). "English as a Formal Language " in B. Visentini, et al, Linguizzi nella Societa e nella Technica. Milan, 189-224.

_____ (1970). "Universal Grammar," Theoria, 36, 373-398.

_____ (1972). "The Proper Treatment of Quantification in Ordinary English," in Hintikka, et al, Approaches to Natural Language. Reidel, Dordrecht.

2. Lytle, Eldon G. A Grammar of Subordinate Structures in English. Mouton and Co., Hungary, 1974.

3. Partee, Barbara Hall. Montague Grammar and Transformational Grammar. In preparation.

_____. "Comments on Montague's Paper," in Hintikka, et al, Approaches to Natural Language. Reidel, Dordrecht.

4. Gabbay, Dov M. "Representation of the Montague Semantics as a Form of the Suppes Semantics with Application to the Problem of the Introduction of the Passive Voice, the Tenses, and Negations as Transformations," in Hintikka, et al, Approaches to Natural Language. Reidel, Dordrecht.

5. McCawley, James D. "Concerning the Base Component of a Transformational Grammar," Foundations of Language, 4 (1968), 243-269.

6. Chomsky, Noam. Aspects of the Theory of Syntax. MIT Press, Cambridge, Massachusetts, 1965.

7. Kay, Martin. "The Mind System," Natural Language Processing, ed. Rustin. Algorithmics Press, Inc., New York, 1973.

8. Melby, Alan. Forming and Testing Syntactic Transfers. Brigham Young University, M.A. Thesis, 1974.

9. Lytle, Eldon G. "An Analysis of Non-Verbal Participles," Brigham Young University Linguistics Symposium Proceedings, 1973.

10. Bush, Charles. Structural Passives and the "Massive Passive" Transfer. Brigham Young University, M.A. Thesis, 1974.

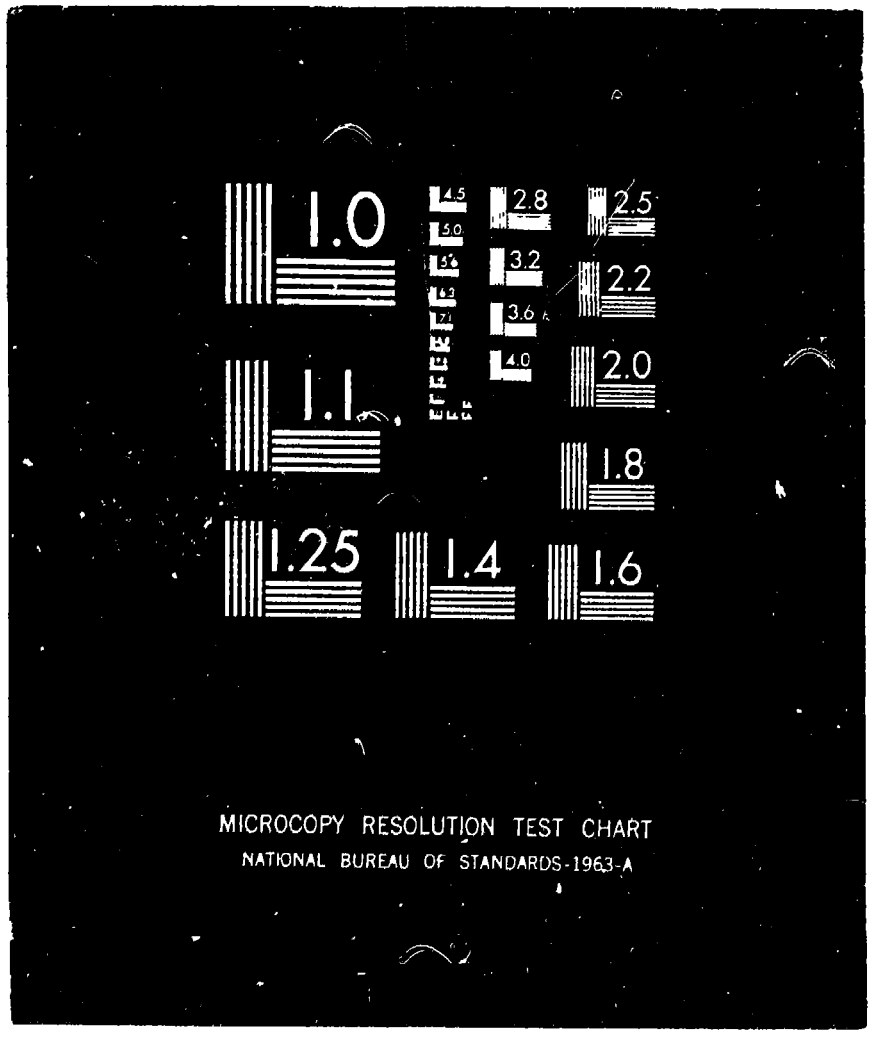
11. Wilks, Yorick. "Identification of Conceptualizations Underlying Natural Language," Computer Models of Thought and Language, ed. Roger C. Schank and Kenneth Mark Colby, San Francisco: Freeman and Company, 1973.

_____. "An Intelligent Analyzer and Understander for English," Communications of the ACM, 1975.

12. Kittredge, Richard I. Projet de Traduction Automatique de l'Universite de Montreal, TAUM73, Rapport do mois d aout, 1973.

13. Vauquois, Bernard. "Structures Profondes et Traduction Automatique le Systeme du C.E.T.A.", Revue Roumaine de Linguistique, XIII, 2, 1968.

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
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A