American Journal of Computational Linguistics Microfiche 8

STRING TRANSFORMATIONS

IN THE

REQUEST SYSTEM

Warren J. Plath
IBM Thomas J. Watson Research Center
Yorktown Heights

ABSTRACT

The REQUEST System is an experimental natural language query system based on a large transformational grammar of English. In the original implementation of the system the process of computing the underlying structures of input queries involved a sequence of three steps: (1) preprocessing (including dictionary lookup), (2) surface phrase structure parsing, and (3) transformational parsing. This scheme has since been modified to permit transformational operations not only on the full trees available after completion of surface parsing, but also on the strings of lexical trees which are the output of the preprocessing phase. Transformational rules of this latter type which are invoded prior to surface parsing, are known as string transformations.

Since they must be defined in the absence of such structural markers as the location of clause boundaries, string transformations are necessarily relatively local in scope. Despite this inherent limitation, they have so far proved to be an extremely useful and surprisingly versatile addition to the REQUEST System. Applications to date have included homograph resolution, analysis of classifier constructions, idiom handling, and the suppression of large numbers of unwanted surface parses. While by no means a panacea for transformational parsing, the use of string transformations in REQUEST has permitted relatively rapid and painless extension of the English subset in a number of important areas without corresponding adverse impact on the size of the lexicon, the complexity of the surface grammar, and the number of surface parses produced.

TABLE OF CONTENTS

String Transformations in the REQUEST System

1. INTRODUCTION

The REQUEST (Restricted English Question-answering) System [1, 2] is an experimental natural language query system which is being developed at the IBM Thomas J. Watson Research Center. The system includes a large transformational grammar, a transformational parser, and a Knuth-style semantic interpreter. The grammar and its associated lexicon are broadly oriented towards question-answering on periodic numerical data, they also include material specific to natural English interaction with collections of business statistics, as exemplified by the Fortune 500

The long-range objective of the work on REQUEST is to determine the extent to which machine-understandable subsets of English can be developed to provide non-programmers with a convenient and powerful tool for accessing information in formatted data bases without having to learn a formal query language. In the interest of facilitating effective "understanding" on the part of the system, the semantic scope of the English subset we are currently dealing with is largely restricted to the world of business statistics. Within that narrow domain of discourse, however, we are attempting to cover a relatively broad range of syntactic and lexical alternatives, in the hope of permitting future users to employ their normal patterns of written expression without major adjustment. The current REQUEST

grammar covers a variety of basic English constructions in some depth, including wh- and yes-no questions, relative clauses and clausal negation It is now being extended into such areas as comparison, conjunction and quantification which, while complex, appear to be of central importance in providing a semantically powerful subset of English.

2. REQUEST System Organization

The REQUEST System consists of a set of programs written in LISP 1.5 together with an associated set of data files containing the lexicon, grammar, semantic interpretation rules and data base. The system runs interactively on a System/370 Model 158 under VM/370 in 768k bytes of virtual core. As can be observed from Figure 1, the system contains two major components, one transformational, the other interpretive.

The transformational component, which serves to analyze input word strings and compute their underlying structures, consists of two main parts: a preprocessor and a parser. The interpretive component also has two major subcomponents: (i) a semantic interpreter*, which translates each underlying structure into a logical form, i.e., a formal expression specifying the configuration of executable functions required to access the data base and compute the answer to the corresponding question and (ii) a retrieval component** which contains the various data-accessing testing, and output formatting functions needed to evaluate the logical form and complete the question-answering process.

Looking at the transformational component in somewhat greater detail, the role of the preprocessor is to partition the input string into words

---

*Implementation of the semantic interpreter, which operates according to a scheme originally proposed by D. E. Knuth [3], is due to S. R. Petrick [1, 4, 5], who has also devised the specific semantic interpretation rules employed in REQUEST.

**F. J. Damerau is responsible for the design and implementation of the current retrieval component.

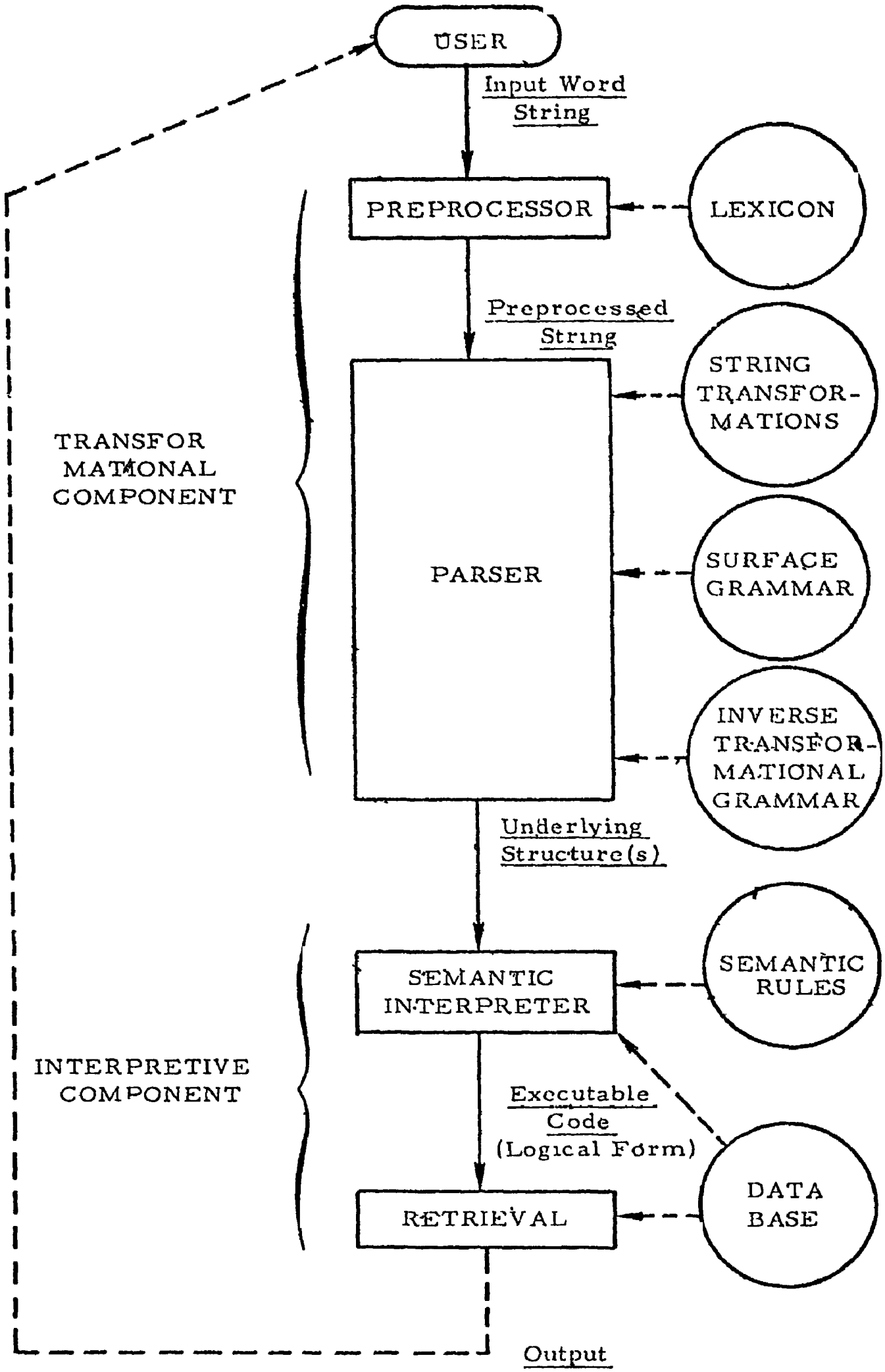


Figure 1 Overall System Organization

and punctuation marks and then look up each segment in the lexicon, yielding a preprocessed string of lexical trees which serves as input to the parser. Multi-word strings that function as lexical units are identified by a "longest match" lookup in a special phrase lexicon; while the lexical trees corresponding to arabic numerals (which may variously represent cardinals, ordinals, or year names) are supplied algorithmically rather than by matching against the lexicon. In cases where there are gaps in the preprocessed string, due to the presence in the input of misspellings, unknown words, ambiguous pronoun references, and the like, the prepro cessor prompts the user to supply the required information.

*

Operation of the transformational parser proceeds in three stages:

(1) The preprocessed string is successively analyzed with respect to the structural description of each rule in a linearly ordered list of string transformations. Each successful match against a string transformation leads to modification of one or more of the trees in the preprocessed string through application of the operations specified in the structural change of the rule in question -- operations which are drawn from precisely the

---

*The original design and implementation of the parser are due to Petrick [6]. The version currently being used in REQUEST is the result of significant revisions and extensions by M. Pivovonsky, who (with the aid of E. O. Lippmann) has also been chiefly responsible for implementing the preprocessor.

same inventory of elementary transformations that the system makes available for the processing of full trees by conventional cyclic and postcyclic transformations, namely: deletion, replacement of a tree by a list of trees. Chomsky adjunction, feature insertion, and feature deletion. (A more detailed account of the nature of string transformations and the motivation for their use in a transformational parser will be presented in the remaining sections of the paper.)

(2) Upon completion of the string transformation phase, the resulting transformed preprocessed string---still in the form of a list of trees -- is passed to a context-free parser in order to compute the surface structure(s) of the sentence. (Although one major effect of the employment of string transformations has been a substantial reduction in the number of unwanted surface parses, cases still occur with some frequency where more than one surface parse is produced.)

(3) Finally, the transformational parser processes each surface structure in turn, attempting to map it step by step into a corresponding underlying structure

according to the rules of a transformational grammar. In this process transformational inverses are applied in an order precisely opposite to that in which their "forward" counterparts would be invoked in sentence generation: inverses of the postcyclic transformations are applied first, starting with the "latest" and ending with the "earliest"; then the inverses of the cyclic transformations are applied (also in last-to-first order) working down the tree from the main clause to those that are most deeply embedded.

To help ensure validity of its final output, the parser checks each intermediate output produced by successful application of an inverse transformation to determine whether or not its constituent structure conforms fully with the set of branching patterns that can be generated by the current grammar in underlying or intermediate structures. At the end of each inverse cycle, a similar check is performed to determine whether all structure above the next (lower) level of embedded S s is consistent with the inventory of allowable underlying structure patterns alone. Failure of either test results in immediate abandonment of the current analysis path. (As described in [2], other, more stringent tests involving the

application of corresponding forward transformations can optionally be invoked in order to provide a more definitive validation of inverse transformational derivations.

3. Motivation for the Introduction of String Transformations

Within the series of major processing steps described in the preceding section, the application of string transformations occurs at a point midway between preprocessing (including lexical lookup) and surface phrase structure parsing. Taken in sequence, these three steps have the cumulative effect of shifting the locus of analysis operations from the domain of word strings to that of full sentence trees, where conventional transformations (and their inverses) can meaningfully be invoked. Unlike the balance of the transformational parsing process, these three preliminary steps do not seem to bear a direct correspondence to familiar generative operations. Nevertheless, their combined effect is to produce the tree or trees which exist at that stage of the "forward" generation where the last postcyclic transformation has applied. Accordingly, it seems reasonable to view them initially as constituting a kind of "bootstrap" whose function is to set the stage for "true" transformational parsing.

Prior to the introduction of string transformations in the REQUEST System, the entire burden of the "bootstrap" role just outlined necessarily fell on the preprocessor and the surface parser. Moreover, as will be

explained below, certain basic principles concerning the nature of the system's transformational component -- relating to the range of inputs to be accepted and the criteria for satisfactory outputs -- had the effect of ensuring that the burden would be a large one. The full dimensions of the situation began to emerge once extensive testing of the first sizeable transformational grammar was underway. There followed a series of corrective actions, the last and most far-reaching of which was the introduction of string transformations.

3.1 Some Relevant Design Principles

In the early design phases of what subsequently became the REQUEST System's transformational grammar, it was decided to adopt a level of underlying structure considerably more abstract than the deep structure of Chomsky's Aspects [7] -- a level which, somewhat in the spirit of generative semantics [8, 9], would go a long way towards direct representation of the meanings of sentences. Eschewing irrelevant details, the essentials of the representation adopted (which bears certain strong resemblances to the predicate calculus) are as follows: Each underlying structure tree represents a proposition (category S1) consisting of an underlying predicate (V) and its associated arguments (NP's) in that order. Argument slots are filled either by embedded propositions (complement S1's) or by nominal expressions (NOM's). A nominal expression directly dominates

either a NOUN, or a NOM and an S1 (the relative clause construction). Each NOUN dominates an INDEX node which is specified as a constant (+ CONST) in the case of proper nouns and as a variable (- CONST) other wise. The INDEXes and the terminal nodes they dominate play an impor tant role in the grammar, including the representation of coreference*.

One major impact which this view of underlying structure had on what the "bootstrap" had to accomplish involved the connection of prepositional phrases to the balance of the surface structure tree. In underlying structure, the noun phrase corresponding to each surface prepositional phrase would appear as a specific argument in a specific proposition, following the application of the generative transformations whose inverses the parser would employ, the resulting prepositional phrase would in most cases still be explicitly linked to the clause or clausal remnant derived from that underlying proposition. Thus, in order to make possible a correct inverse transformational derivation, the surface parser would have to make all such linkages explicit. This requirement represented a significant departure from earlier practice in a number of phrase structure parsing systems, notably those employing predictive analysis [10, 11], where the problem of connecting prepositional phrases to the correct level of structure was simply ducked by making an arbitrary linkage to the nearest available

---

*Much of the early work on the grammar, in particular the system of variables and constants, reflects suggestions by Paul Postal.

candidate, thereby avoiding what would inevitably have been a large increase in the number of unwanted analyses. (A similar approach has recently been followed in the ATN parser of Woods Lunar Sciences Natural Language Information System [12], but there the semantic interpreter is made to pick up the slack.)

A second design principle which had a major impact on the mechanisms for computing surface structures from input strings was the already-mentioned goal of providing broad coverage of syntactic alternatives to promote ease of use. (As should be fairly obvious, expansion of grammatical coverage -- even in a restricted domain of discourse -- in general entails not only an increase in the size and complexity of lexicon and surface grammar but also an increase in the potential for lexical and syntactic ambiguity.)

Two classes of syntactic alternatives whose coverage at the surface syntax level led to specific problems, ultimately resolved by the use of string transformations were stranded prepositions and classifier constructions. In both cases the problems stemmed from the introduction of new possibilities for incorrectly connecting a preposition or prepositional phrase to the balance of the surface structure. Stranded prepositions occur with some frequency in wh-questions and relative clauses in English often yielding results whose naturalness compares favorably with that of the corresponding non-stranded versions, as in (1) (3) below. Because

of these circumstances, we felt obliged to provide for such constructions

(1) a. What companies did XYZ sell oil to?

b. To what companies did XYZ sell oil?

(2) a. What was the city which ABC's headquarters was located in in 1969?

b. What was the city in which ABC's headquarters was located in 1969?

(3) a. What company was Mr. Jones the president of in 1972

b. ? Of what company was Mr. Jones the president in 1972?

even in early versions of our grammar. The case for including classifier constructions, in which proper nouns are optionally accompanied by a common noun designating their semantic class (cf. the (a) versions of (4) - (7)), did not seem quite as compelling as that for stranded prepositions, since

(4) a. the City of Sheboygan

b. Sheboygan

(5) a. the {State / Commonwealth} of Massachusett

b. Massachusetts

(6) a. (the) Tentacle {Company / Corporation}

b. Tentacle

(7) a. the year (of) 1965

b. 1965

the versions with classifiers have a formal, slightly pedantic quality that is absent from their classifier-less counterparts. Nevertheless, there appeared to be no reasonable grounds (such as obscurity, doubtful grammaticality, and the like) for excluding them from the subset.

A third factor affecting the performance of the "bootstrap" was the conscious decision to try to get along initially with a surface parser which would be maximally simple with respect to both its computational mechanism and its surface phrase structure grammar. In particular, this meant employment of a context-free parser without either the complications or the benefits of sensitivity to syntactic and semantic features [11, 13]. The hope was that any additional surface parses which resulted from this approach would be effectively filtered out during transformational parsing by the various well-formedness checks on inverse derivations discussed at the end of Section 2.

3.2 Early Experience with the Parser

Starting in late 1971, tests began on an inverse transformational grammar whose generative counterpart had been developed with the aid of Joyce Friedman's transformational grammar tester [14]. In the interest of debugging the system with as few unnecessary complications as possible, the initial examples were "spoon fed" to the parser using a minimal lexicon

and surface grammar. While revealing no critical problems with the bootstrap, these first trial runs indicated that incorrect surface structures were indeed produced along with the correct ones and tended to give rise to analysis paths which continued for some time before being aborted by well-formedness tests. Sentences with ambiguous verb forms were a case in point. Thus, in the question "What companies are computers made by?" the surface parser produced two almost identical structures -- the first with "made" taken as a finite verb in the past tense, the second with it taken (correctly) as a past participle. The first analysis initiated a lengthy inverse derivation that was terminated as ill-formed only after the entire postcycle and the first inverse cycle had been traversed, meaning that nearly as much time was spent in pursuing this incorrect path as was required to follow the correct one. In this and a number of similar cases, however, it was observed that ill-formedness of the surface structure could have readily been detected at or near the outset of the transformational parsing process by performing tests employing the pattern-matchin power of transformational rules. This observation led to the introduction of so-called blocking rules in the transformational grammar, rules which proved to be quite effective in detecting and filtering out ill-formed structures such as the incompatible auxiliary/finite verb combination in the example just considered.

In the spring of 1972, the surface grammar was greatly expanded in an attempt to cover the full range of structures that could be produced by the set of transformational rules then in use. At that point, the combined effect of the various design decisions affecting surface structures and surface parsing became immediately and painfully evident in the form of a combinatorial explosion: The brief and apparently innocuous question (8)

(8) "Is the headquarters of XYZ in Buffalo?"

produced no less than 19 surface parses, a figure which soared to 147 when a third prepositional phrase was introduced by replacing "Buffalo" with the classifier construction "the city of Buffalo" Although a blocking rule for detecting spurious stranded prepositions rather quickly killed off 16 of the 19 analyses in the former case, thereby reducing the analysis problem to tractable size, the system was unable to cope with the latter situation at all, due to problems of storage overflow.

Thoughts of what would inevitably happen if we added yet another prepositional phrase (as in "Was the headquarters of XYZ in the city of Buffalo in 1971?") made it clear that killing off unwanted surface parses after the fact by means of blocking rules was not enough, measures would have to be adopted which would prevent formation of most such analyses in the first place. Two corrective steps were taken almost immediately:

(a) coverage of classifier constructions was temporarily dropped, and

(b) it was decided to explore what could be done towards elimination of

spurious surface parses through selective refinement of category distinctions in the surface grammar.

In the latter area, it was discovered (not surprisingly) that differences in the surface structure distribution of prepositional phrases, genitive noun phrases, and other types of noun phrases could be effectively exploited to suppress incorrect parses, as could distributional contrasts between proper nouns and common nouns, finite verbs and participles, etc. (In the case of (8) above, 13 of the original 19 parses were ruled out on the ground that proper nouns cannot take modifiers, while 3 more analyses (plus 4 of the 13 already eliminated) were excluded on the basis of distributional distinctions between prepositional phrases and other noun phrases.)

Implementation of the refinements in the surface grammar required numerous part-of-speech code changes in the lexicon and a substantial increase in the number of rules in the surface grammar. Beyond this, the central problem was that the transformational grammar defines a specific class of surface structures -- employing only elements from a fixed set of intermediate symbols -- as the parses which must be found. In order to meet this requirement, the by now considerably expanded set of intermediate symbols employed in the surface grammar had to be mapped onto the smaller set compatible with the transformations. Thus, for example, PP (prepositional phrase) and NPG (genitive noun phrase) nodes in each surface structure would be replaced by NP nodes before transformational

parsing began -- fortunately an extremely simple and rapid operation. (In the most recent version of REQUEST, the surface grammar employs a total of 32 temporary node names for this purpose; they are subsequently mapped onto a set of only 9 nodes for purposes of transformational parsing.

3.3 Problems of Growth of Coverage

The various measures just described had the effect of stabilizing the incidence of artificial surface structure ambiguities at a tolerably low level for a period of about a year, during which the transformational grammar roughly doubled in size from about 35 rules to over 70 as coverage was extended to include such structures as numerical quantifiers, time compounds, and various expressions involving rank and ordinality. The principal costs of ambiguity suppression were felt not in the analysis programs, which required only negligible modification for that purpose, but rather in the surface grammar, which grew much larger and more complex to the point where it became rather difficult to work with. Since a number of additional extensions of grammatical coverage were under active consideration -- among them the restoration of classifier constructions to the subset -- it seemed desirable to seek out some new approach to ambiguity suppression which would not further overburden the surface grammar.

The alternatives originally considered were uniformly unattractive. In the case of the classifier constructions, one could have achieved the

immediate objective by simply loading up the phrase lexicon with an entry for each legitimate pairing of a classifier with a proper noun, thereby achieving a minor gain in grammatical coverage at the price of more than doubling the size of the lexicon. Another approach would have involved creating phrasal entries only for the classifiers themselves -- e.g., "the city of", the state of", etc. -- leaving it to special ad hoc routines at the end of the preprocessor first to check the preprocessed string for the presence of immediately following proper nouns of the corresponding semantic class and then to effect the appropriate amalgamations or deletions.

The second alternative was quickly rejected as even more distasteful than the first, since despite its relatively small initial cost, it would, if used at all extensively, have meant abandonment of an otherwise orderly and perspicuous analysis procedure. This train of thought, however, eventually led to the idea of modifying the preprocessed string not by ad hoc subroutines requiring accretions to the program, but by means of locally defined transformational rules employing the same computational apparatus and notational conventions as the existing forward and inverse transformations. Within a week of its conception, the idea of a string transformation facility became a reality through some minor modifications to the flow of control of the parser.

---

*Much of the ease of this transition stemmed from the generality of the original properanalysis mechanism, which was designed to accept a list of trees, rather than a single tree, as its input.

4. The Use of String Transformations in the REQUEST System

Because they apply to strings of unconnected* lexical trees, rather than to full surface trees with their representation of the structure of phrases and clauses, string transformations tend to be relatively local in scope, typically being restricted to constructions with contiguous elements. Despite this inherent limitation, such rules rapidly found a wide variety of uses within the REQUEST System: Classifier constructions were readily identified and transformed into classifierless counterparts by a handful of string transformations. Rules were also written for suppressing incorrect stranded prepositions, resolving homography, and translating certain idioms into a form more manageable for the surface parser. Finally, experiments were undertaken to explore the possibility of employing string transformations to deal with a limited but potentially useful range of conjunction constructions.

A common thread running through several of these apparently diverse applications of string transformations is the application of what would otherwise have been treated as the inverse of a late postcyclic transformation at a point preceding surface structure parsing in order to achieve a

*At least initially. Some string transformations currently in use produce what are in effect partial surface structures as output. In fact, it is quite possible that an appropriately chosen cyclically ordered set of string transformations could supplant the surface grammar entirely, however, such a development appears unattractive at this time due to efficiency considerations.

simplification of the surface grammar, a reduction in the number of spurious surface parses,, or both. (The benefits of such a reordering stem in large part from the fact that derived constituent structure patterns provided for at the string transformation level need not be dealt with in the surface grammar, thereby reducing its size, its scope, and its potential for producing incorrect surface parses.) In the case of classifier constructions (Section 4.1) and of certain idioms involving notions of rank (Section 4.4), existing postcyclic transformations were actually replaced by string transformations; while in the case of stranded preposition prevention (Section 4.2), a string transformation was made to assume much of the load of an existing postcyclic blocking rule, resulting in a highly beneficial elimination of unwanted surface parses in both instances. In other situations, such as those involving homograph resolution (Section 4.3) and the treatment of the first group of idiom-processing rules discussed in Section 4.4, a correspondence of string transformations to locally-defined postcyclic transformations, while potentially possible, did not actually exist, since no attempt had been made to cover the constructions in question prior to the introduction of string transformations.

4.1 Classifier Constructions

The string transformations relating to classifier constructions are exemplified by the rule "City, State, Year Classifier", whose statement

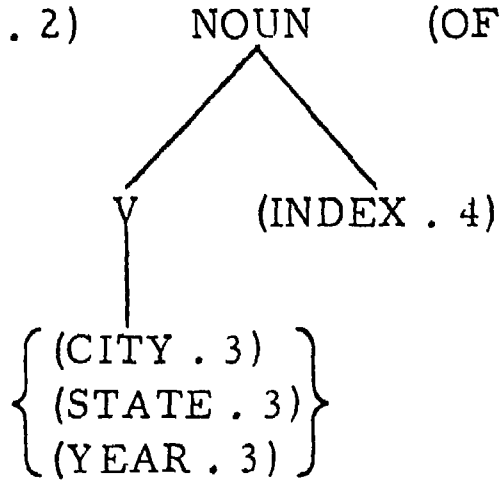
is displayed in Figure 2 using a hybrid tree/list notation in order to enhance legibility. Like all transformations in the REQUEST System, this rule consists of a list with five main sections: header, structural pattern, condition, structural change, and feature change. The header, which serves to identify the rule and a number of basic attributes governing its application, is in the form of a list comprising the name, type (FORW, INVDIR, INVINDIR, STRING, or BLOCK), optionality (OB or OP), and mode (ALL, ANY, ONE, NA, or REANALYZE) of the transformation. Thus the rule CSYCLSFR is labeled as a string transformation whose execution is obligatory for all matches that may occur in the list of trees being processed.

The structural pattern (possibly qualified by further constraints expressed in the condition section) defines the domain of applicability of the transformation in the form of a list of pattern elements, each specifying a tree or class of trees. For a match to occur, it must be possible to partition the input tree (or list of trees) into a list of non-overlapping, adjacent trees each of which matches the corresponding pattern element. Thus, the structural pattern in Figure 2 indicates that the rule CSYCLSFR requires that the preprocessed string be partitionable into the following six-segment sequence: (1) an arbitrary initial segment (possibly null) designated (X . 1), (2) an occurrence of the definite article THE, (3) a common NOUN (already represented in our surface structure as dominating

Header: (CSYCLSFR STRING OB ALL)

Structural Pattern:

((X . 1) (THE . 2) NOUN (OF . 5) ((INDEX . 6) (X . 7))
(ORX(+ CITY
+ STATE + YEAR)))



Condition:

(EQUAL ORX (QUOTE (+ (NODENAMEOF 3))))

Structural Change

(1 2 3 4 5 6 7)

(1 0 0 0 0 6 7)

Feature Change:

NIL

Figure 2: The String Transformation
"City, State, Year Classifier"

an underlying predicate (V) and an INDEX) which happens to be one of the three classifiers CITY, STATE, or YEAR, (4) an occurrence of the preposition OF; (5) an INDEX bearing one of the feature pairs (+ CITY), (+ STATE), or (+ YEAR) (the absence of a preceding V node here is sufficient to guarantee that any matching item will necessarily be an INDEX (+ CONST) -- i, e., a proper noun); and (6) an arbitrary (possibly null) final segment, designated by (X . 7). The condition adds the further stipulation that the value of the variable ORX* be compatible with node 3 in the pattern -- i. e., the proper noun must belong to the semantic class designated by the classifier.

The structural change of a transformational rule may be stated in one of two ways:

(1) If the change is relatively simple (as here) it may conveniently be stated in the form of two lists of numerals referring to the correspondingly labelled elements of the structural pattern. The first list identifies the elements under consideration; the second list (which must contain the same number of elements as the first) specifies what (if anything) happens to each of them -- replacement, deletion, sister adjunction to another element, etc. In the case of CSYCLSER, the change specified is the

*In addition to providing variables ALPHA, BETA, and GAMMA, which range over the set of feature values {+, -}, the notational system of the REQUEST transformational component includes the variables ORX, ORY and ORZ, which range over sets of (feature value, feature name) pairs.

deletion of the trees whose top nodes are labelled 2, 3 4, and 5 (including, by convention, any higher nodes which dominate only deleted nodes). Thus the effect of the rule is to eliminate all classifiers of the designated type from the preprocessed string.

(2) Alternatively the structural change may be expressed as a list of elementary operations, drawn from the set REPLACE, DELETE, LCHOMADJ, RCHOMADJ}, and their arguments. This notation is typically employed when fixed trees are inserted (although the first option may still be taken in such cases) and is obligatory whenever a choice is made among alternative structural changes by evaluating one or more conditional expressions. Had this second option been taken in the case of the present rule, its structural change would have read: ((DELETE 2) (DELETE 3) (DELETE 4) (DELETE 5)).

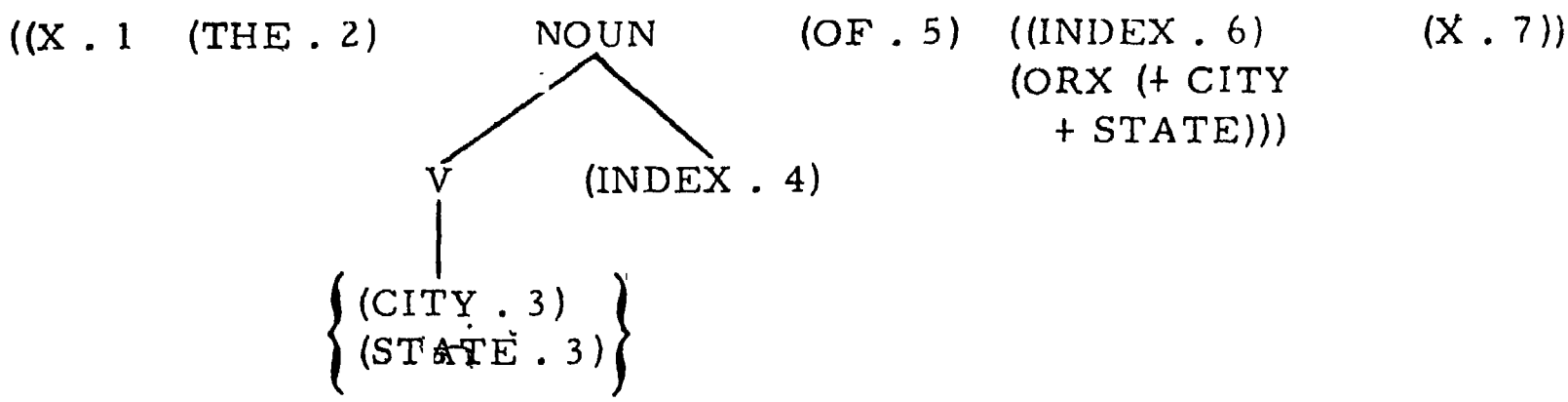
The feature change section of each transformation is always expressed as a list of elementary operations which are members of the set {INSERT, DELETE}, together with their associated arguments. Where no feature change is associated with a rule, as is the case for CSYCLSFR, this final section of the rule statement is specified as NIL, the empty list. (The structural change and condition sections of transformations can similarly be defined as NIL, denoting that the tree structure remains unchanged and that there are no extra conditions on applicability, respectively.)

Two other classifier-deleting string transformations which are very similar to "City, State, Year Classifier" are the rules "Year Classifier" (YRCLASFR) and "Company Classifier" (COCLASFR) The former deletes the lexical trees corresponding to the underlined material in examples like "...the year 1968...", while the latter does the same thing in examples such as "...(the) American Can Company... . Although the underlying predicate COMPANY is the only one specified in the structural pattern of COCLASFR, the rule actually applies to instances where a form of either of the words "company" or "corporation" has been used in the input string, owing to the fact that the lexicon assigns the same underlying predicate to both in recognition of their synonymity

"City State Block" (CSBLOCK) and "City State" (CITYSTAT) are two rules, related to the preceding ones, which illustrate additional aspects of the system. Both of these rules follow CSYCLSFR in the list of string transformations. As indicated by its header information, (Figure 3(a)), CSBLOCK is a blocking rule (BLOCK), which entails that it is obligatory (OB) and will result in termination of the current analysis path if the structural pattern matches the preprocessed string at least once. The structural pattern is identical to that for CSYCLSFR save for the omission of the alternatives relating to the predicate YEAR and the feature (+ YEAR) Due to the parallelism of the structural patterns and the relative ordering of the two rules, it is necessarily the case that CSBLOCK

Header: (CSBLOCK BLOCK OB ONE)

Structural Pattern:



Condition: NIL

Structural Change: NIL

Feature Change: NIL

(a)

Header: (CITYSTAT STRING OB ALL)

Structural Pattern:

Condition: NIL

Structural Change:

(1 2 3 4 5)

(1 (2 4) 0 0 5)

Feature Change:

((INSERT 6 ((+ CITYSTATE))))

(b)

Figure 3: The Rules CSBLOCK and CITYSTAT

will apply if and only if the classifier and the following proper noun do not correspond (any corresponding classifiers having already been deleted by CSYCLSFR). Thus CSBLOCK has the effect of aborting analyses where a proper name known to the system as designating a state has been classified as denoting a city, or vice-versa

The rule CITYSTAT does not refer to classifiers as such, but it does deal with a proper noun construction even more important for our particular subset: the precise identification of a specific city by appending the appropriate state name to the city name. This construction is essential in distinguishing among such cities as Portland, Maine and Portland, Oregon, not to mention the eighteen varieties of Springfield in the continental United States** The structural pattern of the rule (Figure 3(b)) specifies a domain consisting of a city name ((INDEX . 6) (+ CONST + CITY)) followed by an optional comma, followed by a state name (INDEX (+ CONST + STATE)), where the actual city name is a single tree*** (W . 2) and the

---

*Such a situation would always arise in processing such inputs as "the $\left\{\begin{matrix}\text{City}\\\text{State}\end{matrix}\right\}$ of New York", effectively resolving the ambiguity of the proper noun, if the user were not previously asked by the system to resolve it, as is our current practice.

**Cf. reference 15.

***The structural variable W is employed in structural patterns in place of the more usual X whenever one wishes to specify the occurrence of precisely one unknown tree.

state name a single tree (W . 4). As indicated by the structural change, each match results in the replacement of the tree labelled 2 by a list of trees consisting of itself and the tree labelled 4, thereby pairing the state name with the city name by what amounts to right sister adjunction. The optional comma (COMMA 3) and the state name (W . 4) -- plus, by the convention cited earlier, the structure dominating it -- are deleted. Finally, the feature (+ CITYSTATE) is added to the feature list of the node (INDEX . 6), where its presence will eventually be noted by the semantic interpreter as requiring a match on both elements of a (cityname, statename) pair in the data base. As far as the transformational component is concerned, the net effect of the rule is to make "city, state" constructions pass through both the surface parser and the inverse transformations as though they were simple city names.

4.2 Stranded Prepositions

"Stranded Preposition Prevention" (Figure 4) is a string transformation designed to prevent surface structure parses in which non-stranded prepositions are erroneously analyzed as stranded ones. Since most prepositions, whether stranded or not, are obligatorily present in surface structures, this rule necessarily reflects an approach very different from the "recognize and delete" strategy employed in the string transformations involving classifiers. What is done here is to assign new word

class codes to those prepositions determined to be non-strandable, and to write the surface structure rules for the new codes in such a way that they are only allowed to combine with a following noun phrase.

Expressed in ordinary English, the statement of the rule reads about as follows: "Replace the word class code of each preposition by the corresponding code for non-strandable prepositions except where the preposition immediately precedes an auxiliary, a punctuation mark, a verb form, or another preposition, assign any locative feature associated with the original word class code to the new word class code". As stated -- and as currently implemented -- the rule may well be at once both too weak and too strong, at least in an absolute sense. It is probably too weak in that it will fail to label as non-strandable any preposition which immediately precedes a noun phrase beginning with an adjective (VADJ), as, for example, in the sequence "to large companies". This sort of deficiency is of little consequence, however, since the rule will serve its purpose well if it fails to catch an occasional non-strandable preposition, leaving things as ambiguous as before in those cases.

Excessive strength, in the sense of marking some stranded preposition as non-strandable, is potentially a much more serious flaw, since it precludes obtaining a correct analysis in such instances. Examples such as (9), where SPRPPREV would fail in just this way by applying

Header: (SPRPPREV STRING OB ALL)

Structural Pattern:

```
((X . 1) { (PREP . 2)  } (X  4))
         { (W  5)      }
         { (PREPOF  3) }
         { (W  5)      }
```

Condition: (NOT (ANALYSIS 4 NIL (QUOTE({ ((BAUX)) } ((X))))))
{ ((COMMA)) }
{ ((DAUX)) }
{ ((PREP)) }
{ ((PUNCT)) }
{ ((V)) }
{ ((VADJ)) }
{ ((VING)) }

Structural Change

```
((COND (2
        (COND((ANALYSIS 2 NIL (QUOTE ( ((PREP((+ LOC2))) ))) )
              (REPLACE ( (NSPREP((+ LOC2))) ) 2))
                                  5
              (T (REPLACE (NSPREP) 2))) )
                                  5
        (3  (REPLACE (NSPREPOF) 3))) )
                                  5
```

Feature Change: NIL

Figure 4: The String Transformation "Stranded Preposition Prevention

incorrectly, are not particularly difficult to think up.. However, the

(9) Was the company XYZ bought ballbearings from a subsidiary of Universal Nut & Bolt?

great majority of such examples -- including (9) -- seem to be irrelevant to the present REQUEST data base. Thus, while it is clear that our initial rule for stranded preposition prevention does not provide anything approaching a general solution to the problem, it does appear to be working satisfactorily for the moment in eliminating artificial surface ambiguities within a narrow domain of discourse.

4.3 Homograph Resolution

One of the simplest and yet most useful of the 33 string transformations in the current version of REQUEST is the rule "Ordinal Formation" (ORDFORM). Its function is to match on each string consisting of an arabic numeral immediately followed by any member of the set of English ordinal-forming suffixes {d, nd, rd, st, th} and mark the sequence as an ordinal numeral. The operation of ORDFORM (Figure 5) is entirely straightforward. By this point in the analysis process, all arabic numerals have already been assigned lexical trees dominated by the node (VADJ (+ CARD)) -- the combination denoting a cardinal numeral -- during the input scanning phase of the preprocessor; while the ordinal-forming suffixes have been assigned trees dominated by the category ORD during

Header: (ORDFORM STRING OB ALL)

Structural Pattern:

((X . 1) ((VADJ . 2) (+ CARD)) (ORD 3) (X . 4))

Condition: NIL

Structural Change:

((DELETE 3))

Feature Change:

((DELETE 2 (CARD)) (INSERT 2 ((+ ORD))))

Figure 5: The String Transformation "Ordinal Formation"

the lexical lookup phase. ORDFORM simply finds each instance in the preprocessed string where a (VADJ (+ CARD)) immediately precedes an ORD, deletes the ORD tree, and changes the feature on the VADJ from (+ CARD) to (+ ORD), thereby identifying that item as an ordinal numeral rather than a cardinal.

The approach just described has the advantage of putting an unlimited set of ordinals at the disposal of the user at negligible cost, involving a few very minor additions to the lexicon and none at all to either the surface grammar or the preprocessor. The alternate of using a postcyclic transformation instead of a string transformation to achieve the same coverage was avoided because it would have imposed the additional requirement that the surface grammar be significantly enlarged through the inclusion of at least three new category symbols (for cardinals, ordinals, and ordinal suffixes) along with a set of context-free rules describing their distribution. Although identification of ordinal numerals of this type could also have been effected by building the appropriate tests directly into the preprocessor, the latter alternative would have been much less attractive than the string transformation approach for at least two reasons: First, it is inherently messier to bury such operations in a special program subroutine than to deal with them as just another transformational rule. Second, and more important, is the fact that the latter approach makes the system less

general and flexible, since material specific to English is directly reflected in the structure of the program itself, rather than being confined to the grammar, where it is readily accessible to the linguist who may wish to modify it or replace it by material describing some other natural language.

Another string transformation currently employed to resolve word class homography on the basis of local context is the rule "Cardinal Noun" (CARDNOUN), which will be discussed only briefly here. The rule distinguishes instances where a cardinal numeral functions as a proper noun (10) from those in which it serves as a numerical quantifier of a following nominal expression (11). It does so by checking the immediate right-hand context of each (VADJ (+ CARD)) for the presence of

(10) Is the number of companies in Chicago greater than 16?

(11) What companies employed at least 200,000 people in 1973?

items (such as articles, auxiliaries, punctuation, and verbs) which are incompatible with the latter possibility, replacing the VADJ structure by a corresponding proper noun structure whenever a match occurs. (CARDNOUN follows ORDFORM in the list of string transformations in order to take advantage of the latter's replacement of certain cardinals by corresponding ordinals.)

4.4 Idiom Processing

By their very definition, idiomatic expressions are items which present problems in grammatical analysis, semantic interpretation, or both. Although it would be very tempting to exclude all constructions of this sort from the English subset of REQUEST, the currency and naturalness of many idioms is so great that such a prohibition would entail abandonment of our goal of permitting future users to employ their normal patterns of expression.

For idioms such as "make money", (in the sense of "be profitable"), where the components are adjacent and the number of paradigmatic variants are few, one possible approach is to deal with the problem by putting appropriate entries in the phrase lexicon. For example, the entry for "makes money" in our present lexicon treats that combination as an intransitive verb in the present tense and singular number which dominates the same underlying predicate and has the same selectional features as the adjective "profitable". Even in such a relatively straightforward case, however, it is not difficult to think of minor extensions, such as the inclusion of negatives ("make no money"), which will at least require another set of phrasal entries. Moreover, the phrase lexicon approach breaks down completely as soon as one deals with an idiomatic construction that includes an open class as one of its components, producing a situation parallel to that encountered earlier for classifier constructions.

The attempt to provide broad coverage of constructions involving notions of rank and ordinality led to the consideration of a number of common idiomatic patterns including arbitrary cardinal or ordinal numerals. These patterns, three of which are illustrated in (12), were eventually dealt with successfully by the development of string transformations designed not only to cope with their syntactic peculiarities but to

(12) (a) What company { was / ranked } number 18 in 1972 sales?

(b) What were the 25 { top / highest } -ranking companies with respect to earnings in 1969

(c) List the top 20 companies in 1973 growth rate!

set the stage for correct semantic processing as well.

The nature of these idiom-processing transformations is perhaps best illustrated by considering the rule "Top n" (TOPN), whose statement appears in Figure 6. The structural pattern of TOPN specifies a sequence of elements consisting of an initial arbitrary string of trees (X . 1) followed in order by an occurrence of the definite article "the" (THE . 2), the word "top" (TOP 3), a cardinal numeral ((VADJ . 4) (+ CARD)), a nominal expression (NOM . 5), either of the prepositions "in" (IN . 6) or "with respect to". (WITH_RESPECT_TO . 6), and a

Header: (TOPN STRING OB ALL)

Structural Pattern:

((X . 1) (THE . 2) (TOP . 3) ((VADJ . 4) (+ CARD)) (W . 8)) (NOM . 5) { (IN . 6) / (WITH_RESPECT_TO . 6) } (X . 7

Condition: NIL

Structural Change:

((REPLACE (5 (VING (+ ADJ + ING)) RANK) (VADJ (+ ADJ + ORD)) (NQUOTE 1)) PREP (VADJ (+ ADJ + ORD)) 8) 5)

(DELETE 3) (DELETE 4))

Feature Change: NIL

Figure 6: The Rule "Top n"

final arbitrary string of trees (X . 7). The structural change includes a replacement and two deletions.

The syntax of a replacement operation is of the form (REPLACE <list of trees> <tree>); its execution results in the replacement of the item corresponding to tree by the items corresponding to list of trees. The replacement operation in TOPN is therefore to be understood as follows: The nominal expression tree in the input which matches the pattern element (NOM . 5) is replaced by a list of elements consisting of itself, followed by lexical trees corresponding to (i) the -ing form of the verb "rank", (ii) the ordinal numeral "first" (where the (NQUOTE 1) notation causes the "1" to be interpreted as a literal, rather than as a reference to the pattern element (X . 1)), (iii) the preposition "through", and (iv) the ordinal numeral corresponding to the cardinal which matched ((VADJ . 4) (+ CARD)) in the structural pattern. The two deletion operations remove the lexical trees for the cardinal numeral and the adjective 'top" from the preprocessed string.

In the case of (12c), the overall effect of this structural change is to replace the string of lexical trees corresponding to "the top 20 companies" by the string of trees corresponding to "the companies ranking (1st through 20th". A subsequent string transformation called "Rank Interval" (RNKINTVL), operating in a fashion similar to that of "City State" (cf. Section 4.1), then transforms the trees corresponding to

"1st through 20th" into a single ordinal numeral tree (bearing the feature (+ INTERVAL)) which dominates the numerals "1" and "20" As a result of these operations both surface and transformational parsing of such examples has become completely routine; while their semantic interpretation has required only the addition of a simple mechanism -- triggered by the feature (+ INTERVAL) -- for generating a dense set of integers from its endpoints.

Another group of string transformations involving rank are derived from what were originally late postcyclic transformations. The three rules in question -- "First Superlative" (FIRSTSUP) "Nth Superlative" (NTHSUPER), and "Nth Place" (NTHPLACE) -- collectively serve to restore the various deletions illustrated in (13).

(13) a. --- $\left\{\begin{matrix}\text{ranked}\\\text{was}\end{matrix}\right\}$ (the) first highest --- $\overset{OB}{\Longrightarrow}$

--- $\left\{\begin{matrix}\text{ranked}\\\text{was}\end{matrix}\right\}$ (the) highest -

b. --- $\left\{\begin{matrix}\text{ranked}\\\text{was}\end{matrix}\right\}$ (in) (the) $\left\{\begin{matrix}\text{first}\\\text{second}\\\text{nth}\end{matrix}\right\}$ highest --- $\overset{OP}{\Longrightarrow}$

--- $\left\{\begin{matrix}\text{ranked}\\\text{was}\end{matrix}\right\}$ (in) (the) $\left\{\begin{matrix}\text{first}\\\text{second}\\\vdots\\\text{nth}\end{matrix}\right\}$

c. --- $\left\{\begin{matrix}\text{ranked}\\\text{was}\end{matrix}\right\}$ in (the) nth highest place --- $\overset{OP}{\Longrightarrow}$>

--- $\left\{\begin{matrix}\text{ranked}\\\text{was}\end{matrix}\right\}$ (the nth highest ---

The prime motivation for shifting these rules from the postcycle to a point preceding surface parsing was that the structure and distribution of the various phrase remnants resulting from the deletions are at best difficult to describe within the framework of a context-free phrase structure grammar. A variety of ad hoc apparatus, including special word class codes for the verb "rank" and for superlative adjectives, as well as special phrase names for such sequences as "the + superlative" and "ordinal numeral + superlative", would have to be introduced in order to provide broad coverage without an accompanying combinatorial explosion. By restoring the deletions before surface parsing, however, such distasteful and complicated measures are entirely avoided, since lexical categories are left unchanged and the surface parser has to do no more than parse an ordinary prepositional phrase in the position following the verb.

4.5 Experiments in Limited Conjunction Processing

As was mentioned in the introduction to this paper, one of the principal directions in which we are currently seeking to extend the English

subset accepted by the REQUEST System is in the coverage of (coordinate) conjunction constructions. The fact that the underlying variety and complexity of these constructions tends to be masked by superficial similarities makes a selective, piecemeal approach to their coverage a generally dubious move in a system such as REQUEST, whose eventual users can hardly be expected to make distinctions that may not be immediately obvious even to a trained linguist. Despite strong reservations on this point, it was decided to employ the string transformation mechanism to deal with an extremely limited range of conjunction constructions on an experimental basis.

The range of constructions chosen was confined to conjoined proper nouns exclusively, subject to the further constraint that all terms of a given conjunction be members of the same semantic class - i.e., for the current data base, either company names, city names, state names or year names. While undeniably highly limited in scope, this particular incremental increase in grammatical coverage (if successful) had three distinct merits: (1) it appeared to be compatible with the adjacency constraints of string transformations, owing to the tendency of proper nouns to take no modifiers, (2) it seemed potentially explainable to a naive user in simple terms, and (3) it could provide a natural language interface to an existing, but as yet largely unused, capability of the output formatting routines to generate and display tables of values containing

such information as the earnings of each of a set of companies over a period of years.

The approach employed in the string transformations for processing conjoined proper nouns is exemplified by the rule "City, State, Year, Company Conjunction" (CSYCOCNJ), whose statement is displayed in Figure 7. The second and third elements of the structural pattern form a subpattern that is preceded by an asterisk and surrounded by a pair of parentheses. This notation identifies the occurrence of a so-called "Kleene star expression", which is interpreted by the transformational parser as a pattern element that is to be matched by zero or more consecutive occurrences of tree sequences matching components. The particular Kleene star expression used here will match a string of any length* which consists entirely of an alternating sequence of proper nouns and commas, provided that all the proper nouns are members of the same semantic class** The pattern elements following the Kleene star expression specify that it must be followed by: (i) another instance of a proper noun of the appropriate class (this will be the initial instance if the null value of the Kleene star expression is the only one that matches);

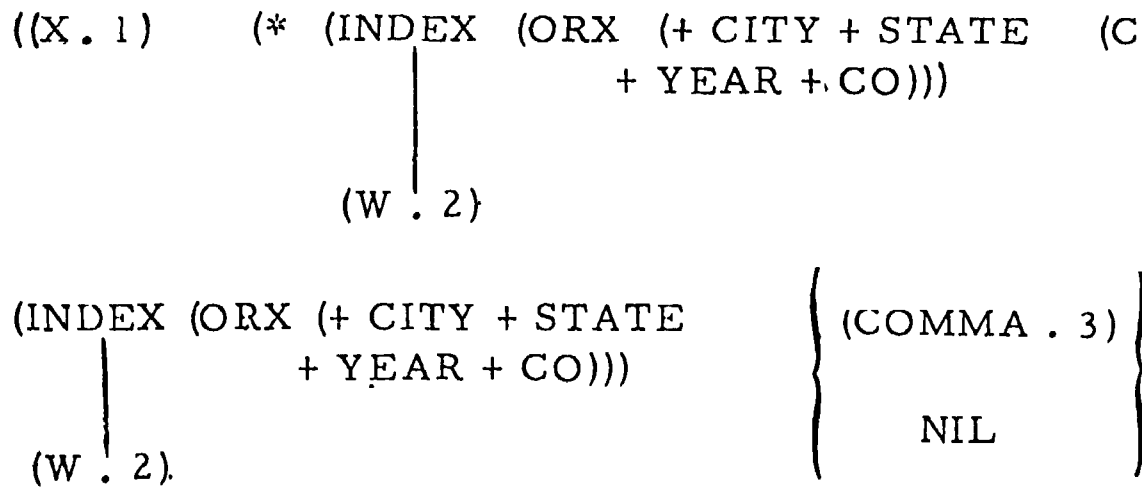
---

*The effect of the condition, which precludes any match where the left-hand structural variable (X . 1) ends in a sequence of trees satisfying the pattern of the Kleene star expression, is to force a (unique) match of maximum length.

**Repeated occurrences of ORX in a structural pattern, whether implicit or explicit, are required to match the same feature pair.

Header: (CSYCOCNJ STRING OB ALL)

Structural Pattern:



Condition:

(NOT (ANALYSIS 1 T (QUOTE (((X))((INDEX (ORX))) ((COMMA)))))

Structural Change:

(1 2 3 4 5 6 7)

(1 0 0 0 0 (2 6) 7)

Feature Change:

((COND (4 ((INSERT 9 ((+ ANDSET))) (INSERT 8 ((-SG)))))

(5 (INSERT 9 ((+ ORSET))))))

Figure 7: The Rule "City, State, Year, Company Conjunction"

(ii) an optional comma; (iii) an instance of either of the coordinating conjunctions "and" or "or", represented internally as ORR, since OR is already used to signal the presence of a disjunctive pattern element to the rule-processing routine); (iv) the final instance of a semantically compatible proper noun, and (v) the usual end variable.

The structural change specifies (1) that the terminal elements of all but the rightmost conjunct (which are collectively associated with the pattern element (W . 2) during the pattern matching phase) are to be sister adjoined to the terminal element of that rightmost conjunct and (2) that the original occurrences of all trees but those corresponding to the end variables and the final conjunct are to be deleted. Conditional on the presence of the conjunction "and" (AND . 4), the feature change adds the feature (+ ANDSET) to the feature list of the surviving INDEX and the feature (- SG) to that of the NOUN node immediately above. (The latter operation automatically results in replacement of the original (+ SG)). If the conjunction is an "or" (ORR . 5) instead, the feature change merely adds the feature (+ ORSET) to the feature list of the INDEX, leaving the number of the NOUN unchanged.

The overall effect of the rule reflects the by now familiar strategy of mapping a structure which would otherwise pose severe problems in surface parsing into a significantly simpler one which will be processed without difficulty by both the surface parser and the transformational

parser. As in the case of CITYSTATE and RNKINTVL, a special feature is attached to the node in the output structure that directly dominates two or more terminal symbols as a result of the structural change of the rule. In each case, the purpose of the feature is to communicate to the semantic interpreter how the elements of the set of terminal symbols are to be treated -- as a (city, state) pair, as the endpoints of a dense set of integers, or as the elements of a conjoined set of proper nouns.

The experimental approach to proper noun conjunction just described appeared initially to be a rather effective one. Examples such as (14) went through the transformational component as smoothly as ones like (15),

(14) How much did GM, Ford, and Chrysler earn in the years from 1967 through 1972?

whereupon the interpretive component produced what appeared to be an appropriate answer -- in the case of (14), an earnings table with 18 entries

(15) How much did Ford earn in 1969?

listed by company and by year. It was not long, however, before consideration of examples such as (16) and (17) revealed that the initial appearance of an adequate solution had been highly misleading.

(16) Was GM or Ford unprofitable in 1970?

(17) What were the earnings of the Big Three auto companies for the 1966-1968 period?

For the former example, at least two readings seem possible: one as a selectional question, paraphrased in (18a) (which would preclude a

(18) a. Which auto company was unprofitable in 1970 -- GM or Ford?

b. Was either GM or Ford unprofitable in 1970?

yes or no answer), the other as a yes-no question (18b), where the conditions for giving a positive answer depend upon the interpretation of the "or" as inclusive or exclusive. In the case of (17), there seems to be a series of possible readings, roughly paraphrased by (19a-d), reflecting ambiguity as to whether what has been requested is earnings information

(19) a. What were the earnings of each of the Big Three auto companies for each of the years 1966-1968?

b. What were the combined earnings of the Big Three auto companies for each of the years 1966-1968?

c. What did the earnings of each of the Big Three auto companies total for the 1966-1968 period?

d What did the combined earnings of the Big Three auto companies total for the 1966-1968 period?

(a) presented individually by company and by year, (b) summed over companies but not over years, (c) summed over years but not over companies, or (d) summed over both companies and years.

Ambiguities of the types exemplified by (16) and (17) were found to be quite widespread in the sort of material we are dealing with, occurring in a number of examples such as (14) where their presence was not initially perceived. Moreover, it was soon realized that such ambiguities were totally different in character from the types we had previously been most concerned with, since they involved instances of genuine multiple meaning in the language, rather than ambiguities artificially introduced by the inadequacies of a grammatical description or a parsing mechanism. It was also clear that the underlying structures assigned to these ambiguous examples were seriously deficient, in that they did not indicate the presence of an ambiguous situation, much less what the ambiguous alternatives were.

Further investigation indicated that the ambiguities encountered were not restricted to conjoined proper nouns, but could also occur in the case of plural noun phrases. For example, (20) is ambiguous between a reading requesting earnings listed individually by company and a reading

(20) What were the 1972 earnings of the companies in Chicago?

requesting a combined earnings figure -- exactly the same readings which would exist if the phrase "the companies in Chicago" were replaced by the conjoined names of all companies satisfying that description. Thus,

it appeared that the ambiguities we wished to understand and cope with were related not to conjunction per se, but to semantic properties of sets and relations on sets.

This view was reinforced by the discovery of syntactically parallel examples with sharply contrasting ambiguity patterns, as in (21). While both (21a) and (21b) share a reading where what is desired is a production (employment) figure for each year in the period, only (21a) has a

(21) a. How many cars were produced by Chrysler in the 1969-1972 period?

b. How many people were employed by Chrysler in the 1969-1972 period?

sensible reading where the annual figures are to be totalled up arithmetically. The reason lies in the distinction between quantities like earnings, auto production, and rainfall, -- which are inherently additive and are measured on a cumulative basis -- and quantities like employment, assets and temperature, which are measured on an instantaneous basis and are not additive over time in a meaningful sense*. On the other hand, (21b) seems to have two other possible readings (22a) and (22b), reflecting questions about the size of a set union and of a set intersection, respectively. Although neither version of (22) could be answered with

*Although it is meaningful to add them on the way to computing an average over a period of time.

(22) a. How many different people were employed by Chrysler in the 1969-1972 period?

b. How many people were employed by Chrysler during the entire 1969-1972 period?

respect to a Fortune-500-type data base, where people are countable but indistinguishable, both are questions which it would be quite reasonable to try to deal with in a data base environment that included personnel files.

At present, we are continuing to work on problems of conjunction-handling both by pursuing the line of investigation just touched upon and by studying patterns of disambiguation suggested by such examples as (18), (19), and (22). The richness and subtlety of the material we have encountered -- scarcely hinted at here -- is particularly remarkable in the light of the severe limitations placed on the types of conjunction constructions to be considered. While the use of string transformations has not provided us with a satisfactory solution for even a small part of the domain of conjunction constructions, it has had the highly beneficial effect of bringing us face-to-face with a range of significant problems of which we had previously been almost totally unaware.

5. Summary and Conclusions

In the REQUEST System, string transformations are transformational rules of relatively local scope which are applied to strings of lexical trees at a point midway between lexical lookup and surface phrase structure parsing. From the standpoint of linguistic theory, the status of the string transformation facility is unclear, since it is a component that seems to have no direct generative counterpart. The fact that a number of existing string transformations are in effect inverses of late postcyclic transformations suggests that there may be some value in viewing the facility in terms of such relationships. However, the rule writer is entirely free to ignore linguistic considerations of this sort and define any of a wide range of tree manipulations as string transformations. Accordingly, the string transformation facility can, with some justification, simply be viewed as a convenient mechanism whereby the tree processing power inherent in grammatical transformations is made available for purposes of implementing a wide variety of parsing heuristics.

In contrast to the obscurity of its theoretical role, the string transformation facility of REQUEST has had a clear and decidedly favorable impact on the practical development of the system. The facility was originally added in order to provide a more satisfactory input interface to the transformational parser -- an interface which would be considerably

less vulnerable to the undesirable side-effects of expanding grammatical coverage than one consisting solely of a preprocessor and a surface parser. More specifically, this innovation was aimed at preventing the proliferation of unwanted surface parses in a way which would be at once less costly and more perspicuous than alternatives requiring extension of the preprocessor or of the surface grammar.

Based on approximately one year's experience in the use of the string transformation facility, it appears to have fulfilled these original objectives. During this period, the grammatical coverage of REQUEST has been significantly expanded, but the lexicon and the surface grammar have undergone only very modest growth as a result, and there has been no accompanying upsurge in the number of spurious surface parses. The strategy of reordering the inverses of certain late postcyclic rules within the parsing system by placing them before, rather than after, the surface structure rules has proved to be effective both in reducing the number of unwanted surface analyses and in simplifying the surface grammar (and hence the structures that it produces). Moreover, string transformations have also shown an unexpected versatility in such areas as idiom processing and homograph resolution.

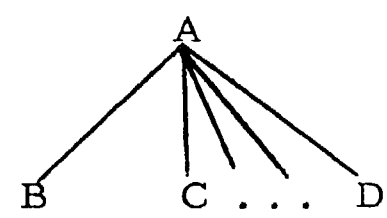
In contrast to these favorable results, our attempt to employ string transformations in dealing with conjunction constructions -- while of great

indirect benefit -- can hardly be viewed as an unqualified success. What the latter experience has clearly demonstrated is the fact that string transformations are a tool, not a panacea, and cannot be expected to yield satisfactory results in areas where the necessary linguistic groundwork is lacking. Despite its limitations, we expect to make continued heavy use of this tool in our ongoing work on extending the grammatical coverage of the REQUEST System.

Appendix: Listing of String Transformations

The following is a complete computer listing of all 33 string transformations in the REQUEST System grammar as of October 1974. The fully-parenthesized list notation employed in the computer file has been "pretty printed" i.e., printed with indentations) in order to make the internal structure of the rules more legible. Each list is surrounded by a pair of parentheses, with its main components (if any) in general printed starting two spaces to the right of the beginning of their "parent" list. Thus, for example, the left parenthesis of the pair surrounding each rule is indented two spaces to the right of the left parenthesis that initiates the entire list of rules. Similarly, with the exception of the header list (which is indented only one space to make it stand out), the main components of each rule -- the header, structural pattern, condition, structural change and feature change -- are indented two spaces with respect to the rule, and so forth.

In contrast to the two-dimensional graphical representation employed for trees in the figures in Section 4 of the text, trees in the listing are rep resented in a linear, parenthesized notation with the following essential characteristics:

1. Within a structural pattern, an expression of the form (A B C . . . D) stands for a tree of the form  where

B, C . . . D themselves may be replaced by parenthesized expressions

that stand for subtrees, etc. As in the figures of Section 4, association of a feature list with a node is denoted by enclosing the list in one pair of parentheses and then surrounding the node and the list with a second pair of parentheses, e.g. (A (+ FEAT1 - FEAT2)). In place of the curly bracket notation used in the figures to denote mutually exclusive sequences of trees, the listing employs expressions of the form (OR (list of trees1) (list of trees 2) ... (list of treesn)), where the arguments of the OR stand in one-to-one correspondence with the sequences of trees. Thus, for example, the expression

```
(OR
   (( PREP . 2 ) (W . 5 ))
   (( PREPOF . 3 ) (W . 5 )))
```

in the structural pattern of the rule SPREPREV (p.77) corresponds to the curly bracket expression that appears near the top of Figure 4 (p. 33).

2. Within a condition, structural change, or feature change, trees are represented in a fully parenthesized 'dressed" notation which contrasts as follows with the "peeled" notation just described for trees in structural patterns: Each node in a tree always has two pairs of associated parentheses -- an inner pair surrounding the node and its feature list (if any) and an outer pair enclosing the node, the feature list, and any subtrees dominated by the node. Each feature list contains at least two pairs of parentheses -- one surrounding the entire list, and one for each (feature value, feature name) pair. Thus in "dressed" notation the "peeled'expressions (A B C ... D) and (A (+ FEAT1 - FEAT2)) become ((A) ((B)) ((C))... ((D))) and ((A((+ FEAT1) (- FEAT2)))), respectively.

```
((WHATBLOC BLOCK OB NA)
  ( (OR
      ( ((VADJ . 1) WH SOME)
        (OR ((BE . 2)) ((DO . 2))) )
      (PREP
        ((NOM . 1) (V WH SOME) NOM)
        (* VADJ)
        (OR ((NOUN . 2)) ((NMNL . 2))) )
      ( ((NOM . 1) (V WH SOME) NOM)
        (* VADJ)
        (OR ((NOUN . 2)) ((NMNL . 2))) ) )
    (X . 3) )
  NIL
  NIL
  NIL )
-----------------------------------

((ORDFORM STRING OB ALL)
  ( (X . 1)
    ((VADJ . 2) (+ CARD))
    (ORD . 3)
    (X . 4) )
  NIL
  ((DELETE 3))
  ( (DELETE 2 (CARD))
    (INSERT 2 ((+ ORD))) ) )
-----------------------------------

((TOPN STRING OB ALL)
  ( (X . 1)
    (THE . 2)
    (TOP . 3)
    (((VADJ . 4) (+ CARD)) (W . 8))
    (NOM . 5)
    (OR ((IN . 6)) ((WITH_RESPECT_TO . 6)))
    (X . 7) )
  NIL
  ( (REPLACE ( 5
               ((VING ((+ ADJ) (+ ING)))
                 ((RANK)) )
               ((VADJ ((+ ADJ) (+ ORD)))
                 (((NQUOTE 1))) )
               ((PREP) ((THROUGH)))
               ((VADJ ((+ ADJ) (+ ORD))) 8)
             5 )
    (DELETE 3)
    (DELETE 4) )
  NIL )
-----------------------------------
```

```
((HYPHNRNK STRING OB ALL)
  ( (X . 1)
    (THE . 2)
    (OR
      (((VADJ . 3) (+ ORD)))
      ( (((VADJ . 4) (+ CARD)) (W . 10))
        (OR
          ((((VADJ . 5) (+ EST + POL)) HIGH))
          (((VADJ . 5) TOP)) ) )
      ((((VADJ . 5) (+ EST + POL)) HIGH))
      (((VADJ . 5) TOP)) )
    (HYPHEN . 6)
    ((VING . 7) RANK)
    ((NOM . 8) (NOUN (V COMPANY) INDEX))
    (X . 9) )
  NIL
  ( (COND ( 3  (REPLACE (8 7 3) 8) )
          ( 4
            (REPLACE ( 8
                       7
                       ((VADJ ((+ ADJ) (+ ORD)))
                         (((NQUOTE 1))) )
                       ((PREP) ((THROUGH)))
                       ((VADJ ((+ ADJ) (+ ORD))) 10) )
                     8 ) )
          ( T
            (REPLACE ( 8
                       7
                       ((VADJ ((+ ADJ) (+ ORD)))
                         (((NQUOTE 1))) ) )
                     8 ) ) )
    (DELETE 3)
    (DELETE 4)
    (DELETE 5)
    (DELETE 6)
    (DELETE 7) )
  NIL )
```

```
((NUMBRNCO STRING OB ALL)
  ( (X . 1)
    (THE . 2)
    ((NOUN . 3) (V NUMBER) INDEX)
    ((VADJ . 4) (+ CARD))
    ((NOM . 5) (NOUN (V COMPANY) INDEX))
    (X . 6) )
  NIL
  ( (REPLACE ( 5
               ((VING ((+ LOC2) (+ ADJ) (+ ING)))
                 ((RANK)) )
               3
               4 )
             5 )
    (DELETE 3)
    (DELETE 4) )
  NIL )
```

```
((NUMBERN STRING OB ALL)
     (X . 1)
     (OR ((RANK . 2)) ((BE . 2)))
     ((NOMQ . 3) (NOUN (V NUMBER) (INDEX . 7)))
     (((VADJ . 4) (+ CARD)) (W . 8))
     (OR ((IN . 5)) ((WITH_RESPECT_TO . 5)))
     (X . 6) )
   NIL
   ( (REPLACE ( ((PREP ((+ LOC2)))
                  ((IN)) )
                ((VADJ ((+ ADJ) (+ ORD))) 8)
                ((NOM)
                  ((NOUN ((+ SG) (- HUMAN) (+ PLACE)))
                    ((V) ((PLACE)))
                    7 ) ) )
              3 )
     (DELETE 4) )
   NIL )
```

```
((RNKINTVL STRING OB ALL)
   ( (X . 1)
     (OR
       ((BETWEEN . 2)
         ((VADJ (+ ORD)) (W . 3))
         (AND . 4) )
       ((FROM . 2)
         ((VADJ (+ ORD)) (W . 3))
         (OR ((TO . 4)) ((THROUGH . 4))) )
       ( ((VADJ (+ ORD)) (W . 3))
         (OR
           ((TO . 4))
           ((THROUGH . 4))
           ((HYPHEN . 4)) ) ) )
     (((VADJ . 7) (+ ORD)) (W . 5))
     (X . 6) )
   (NOT
     (ANALYSIS
       1
       T
       (QUOTE
         (((X)) ((FROM))) ) ) )
   (1 2 3 4 5 6)
   (1 0 0 0 (3 5) 6)
   ((INSERT 7 ((+ INTERVAL)))) )
```

```
((FIRSTSUP STRING OB ALL)
  ( (X . 1)
    (OR ((RANK . 2)) ((BE . 2)))
    (OR ((THE . 3)) NIL)
    ((VADJ . 4) (+ EST))
    (OR ((IN . 5)) ((WITH_RESPECT_TO . 5)))
    (X . 6) )
  NIL
  ( (REPLACE ( 2
               ((PREP ((+ LOC2)))
                 ((IN)) ) )
             2 )
    (REPLACE ( ((VADJ ((+ ADJ) (+ ORD)))
                 ((NQUOTE 1))) )
               4
               ((NOM)
                 ((NOUN ((+ SG) (- HUMAN) (+ PLACE)))
                   ((V) ((PLACE)))
                   ((INDEX ((- CONST)))
                     ((XN)) ) ) ) )
             4 ) )
  NIL )
```

```
((NTHSUPER STRING OB ALL)
  ( (X . 1)
    (OR ((RANK . 2)) ((BE . 2)))
    (OR ((THE . 3)) NIL)
    ((VADJ . 4) (+ ORD))
    (OR ((VADJ . 5) (+ EST))) NIL)
    (OR ((IN . 6)) ((WITH_RESPECT_TO . 6)))
    (X . 7) )
  NIL
  ( (REPLACE ( 2
               ((PREP ((+ LOC2)))
                 ((IN)) ) )
             2 )
    (REPLACE ( ((NOM)
                 ((NOUN ((+ SG) (- HUMAN) (+ PLACE)))
                   ((V) (PLACE)))
                   ((INDEX ((- CONST)))
                     ((XN)) ) ) )
               6
             6 )
  NIL )
```

```
((NTHPLACE STRING OB ALL)
  ( (X . 1)
    (OR ((RANK . 2)) ((BE . 2)))
    ((PREP . 3) IN)
    (OR ((THE . 4)) NIL)
    ((VADJ . 5) (+ ORD))
    (OR
      ((((VADJ . 6) (+ EST)) HIGH))
      NIL )
    ((NOM . 7) (NOUN (V PLACE) INDEX))
    (OR ((IN . 8)) ((WITH_RESPECT_TO . 8)))
    (X . 9) )
  NIL
  ( (COND ( 6
            (REPLACE ( ((PP)
                        3
                        ((NP)
                          ((THE))
                          ((NOM)
                            ((V)
                              ((ADV ((+ EXT))) 5)
                              6 )
                            7 ) ) ) )
                     5 ) )
          ( T
            (REPLACE ( ((PP)
                        3
                        ((NP)
                          ((THE))
                          ((NOM)
                            ((V)
                              ((ADV ((+ EXT))) 5)
                              ((V ((+ ADJ) (+ POL) (+ EST)))
                                ((HIGH)) ) )
                            7 ) ) ) )
                     5 ) ) )
    (DELETE 3)
    (DELETE 4)
    (DELETE 6)
    (DELETE 7) )
  NIL )
```

```
((YRINTRVL STRING OB ALL)
  ( (X . 1)
    (OR
      ( ((PREP . 2) BETWEEN)
        ((INDEX (+ YEAR)) (W . 3))
        (AND . 4) )
      ( ((PREP . 2) FROM)
        ((INDEX (+ YEAR)) (W . 3))
        (OR ((TO . 4)) ((THROUGH . 4))) )
      ( ((INDEX (+ YEAR)) (W . 3))
        (OR
          ((TO . 4))
          ((THROUGH . 4))
          ((HYPHEN . 4)) ) ) )
    (((INDEX . 7) (+ YEAR)) (W . 5))
    (X . 6) )
  (NOT
    (ANALYSIS
      1
      T
      (QUOTE
        (((X)) ((FROM))) ) ) )
  ( (COND ( (AND
              2
              (NOT
                (ANALYSIS
                  1
                  T
                  (QUOTE
                    ( ((X))
                      ((PREP))
                      ((THE))
                      (OR
                        ( ((NOUN ((- SG)))
                           ((V) ((YEAR)))
                           ((INDEX)) ) )
                        ( ((NOUN ((+ SG)))
                           ((V) ((PERIOD)))
                           ((INDEX)) ) ) ) ) ) ) )
            (REPLACE ( ((PREP ((+ LOC2)))
                        ((IN)) ) )
                     2 ) )
          ( T  (DELETE 2) ) )
    (DELETE 3)
    (DELETE 4)
    (REPLACE (3 5) 5) )
  ((INSERT 7 ((+ INTERVAL)))) )
```

```
((CSYCLSFR STRING OB ALL)
  ( (X . 1)
    (THE . 2)
    (NOUN
      (V
        (OR
          ((CITY . 3))
          ((STATE . 3))
          ((YEAR . 3)) ) )
      (INDEX . 4) )
    (OF . 5)
    ((INDEX . 6) (ORX (+ CITY + STATE + YEAR)))
    (X . 7) )
  (EQUAL ORX (QUOTE (+ (NODENAMEOF 3))))
  (1 2 3 4 5 6 7
  (1 0 0 0 0 6 7
  NIL )
```

```
((CSBLOCK BLOCK OB ONE)
  ( (X . 1)
    (THE . 2)
    (NOUN
      (V
        (OR ((CITY . 3)) ((STATE . 3))) )
      (INDEX . 4) )
    (OF . 5)
    ((INDEX . 6) (ORX (+ CITY + STATE)))
    (X . 7) )
  NIL
  NIL
  NIL )
```

```
((PRDCLSFR STRING OB ALL)
  ( (X . 1)
    (THE . 2)
    (OR
      ( ((NOUN . 3) (V PERIOD) INDEX)
        ((INDEX . 4) (+ INTERVAL)) )
      (((INDEX . 4) (+ INTERVAL))
        ((NOUN . 3) (V PERIOD) INDEX) ) )
    (X . 5) )
  NIL
  ((DELETE 2) (DELETE 3))
  NIL )
```

```
((YRCLASFR STRING OB ALL)
  ( (X . 1)
    (THE . 2)
    ((NOUN . 3) (V YEAR) INDEX)
    ((INDEX . 4) (+ YEAR))
    (X . 5) )
  NIL
  (1 2 3 4 5)
  (1 0 0 4 5)
  NIL )
```

```
((COCLASFR STRING OB ALL)
  ( (X . 1)
    (OR ((THE . 2)) NIL)
    ((INDEX . 3) (+ CO))
    ((NOUN . 4) (V COMPANY) INDEX)
    (X . 5) )
  (NOT
    (ANALYSIS
      1
      NIL
      (QUOTE
        (((X)) ((THE))) ) ) )
  (1 2 3 4 5)
  (1 0 3 0 5)
  NIL )
```

```
((CITYSTAT STRING OB ALL)
  ( (X . 1)
    (((INDEX . 6) (+ CONST + CITY)) (W . 2))
    (OR ((COMMA . 3)) NIL)
    ((INDEX (+ CONST + STATE)) (W . 4))
    (X . 5) )
  NIL
  (1 2 3 4 5)
  (1 (2 4) 0 0 5)
  ((INSERT 6 ((+ CITYSTATE)))) )
```

```
((CSYCOCNJ STRING OB ALL)
  ( (X   1)
    (*
      ((INDEX (ORX (+ CITY + STATE + YEAR + CO))) (W . 2))
      (COMMA . 3) )
    ((INDEX (ORX (+ CITY + STATE + YEAR + CO))) (W . 2))
    (OR ((COMMA . 3)) NIL)
    (OR ((AND . 4)) ((ORR . 5)))
    (((NOUN . 8) (+ SG))
      (((INDEX . 9) (ORX)) (W . 6)) )
    (X . 7) )
  (NOT
    (ANALYSIS
      1
      T
      (QUOTE
        ( ((X))
          ((INDEX (ORX)))
          ((COMMA)) ) ) ) )
  (1 2 3 4 5 6 7)
  (1 0 0 0 0 (2 6) 7)
  ( (COND ( 4
            ( (INSERT 9 ((+ ANDSET)))
              (INSERT 8 ((- SG))) ) )
          ( 5  (INSERT 9 ((+ ORSET))) ) ) ) )
```

```
((GENAFCNJ STRING OB ALL)
  ( (X . 1)
    (*
      ((INDEX (ORX (+ CITY + STATE + YEAR + CO))) (W . 2))
      (GENAF . 3)
      (COMMA . 4) )
    ((INDEX (ORX (+ CITY + STATE + YEAR + CO))) (W . 2))
    (GENAF . 3)
    (OR ((COMMA . 4)) NIL)
    (OR ((AND . 5)) ((ORR . 6)))
    (((NOUN . 10) (+ SG))
      (((INDEX . 11) (ORX)) (W . 7)) )
    (GENAF . 8)
    (X . 9) )
  (NOT
    (ANALYSIS
      1
      T
      (QUOTE
        ( ((X))
          ((INDEX (ORX)))
          ((GENAF))
          ((COMMA)) ) ) ) )
  (1 2 3 4 5 6 7 8 9)
  (1 0 0 0 0 0 (2 7) 8 9)
  ( (COND ( 5
            ( (INSERT 11 ((+ ANDSET)))
              (INSERT 10 ((- SG))) ) )
          ( 6  (INSERT 11 ((+ ORSET))) ) ) ) )
```

```
((PPCONJ STRING OB ALL)
  ( (X . 1)
    (*
      (OR
        ((PREP (W . 2)))
        ((PREPOF (W . 2))) )
      ((INDEX (ORX (+ CITY + STATE + YEAR + CO))) (W . 3))
      (COMMA . 4) )
    (OR
      ((PREP (W . 2)))
      ((PREPOF (W . 2))) )
    ((INDEX (ORX (+ CITY + STATE + YEAR + CO))) (W . 3))
    (OR ((COMMA . 4)) NIL)
    (OR ((AND . 5)) ((ORR . 6)))
    (OR
      ((PREP (W . 7)))
      ((PREPOF (W . 7))) )
    (((NOUN . 10) (+ SG))
      (((INDEX . 11) (ORX)) (W . 8)) )
    (X . 9) )
  (AND
    (NOT
      (ANALYSIS
        1
        T
        (QUOTE
          ( ((X))
            (OR
              (((PREP)))
              (((PREPOF))) )
            ((INDEX (ORX)))
            ((COMMA)) ) ) ) )
    (COMPARELISTITEM 2 7) )
  (1 2 3 4 5 6 7 8 9)
  (1 0 0 0 0 0 7 (3 8) 9)
  ( (COND ( 5
            ( (INSERT 11 ((+ ANDSET)))
              (INSERT 10 ((- SG))) )
          ( 6  (INSERT 11 ((+ ORSET))) ) ) ) )
```

```
((RTIMEDST STRING OB ALL)
  ( (X . 1)
    ((PROPNOM . 2) (NOUN (INDEX (+ YEAR))))
    (NMNL (+ PERIODIC))
    (OR
      ( (OR (PREP) (PREPOF))
        (INDEX (+ CO)) )
      NIL )
    (*
      COMMA
      ((NMNL . 3) (+ PERIODIC))
      (OR
        ( (OR ((PREP)) ((PREPOF)))
          (INDEX (+ CO)) )
        NIL ) )
    (OR
      ( (OR ((COMMA . 5)) NIL)
        AND
        ((NMNL . 4) (+ PERIODIC)) )
      (COMMA
        ((NMNL . 4) (+ PERIODI
        (OR
          ( (OR ((PREP)) ((PRE
            (INDEX (+ CO)) )
          NIL )
        (*
          COMMA
          ((NMNL . 6) (+ PERIO
          (OR
            ( (OR ((PREP)) ((P
              (INDEX (+ CO)) )
            NIL )
          (OR ((COMMA . 7)) NIL)
          AND
          (NMNL (+ PERIODIC)) ) )
    (X   8) )
  (AND
    (NOT
      (ANALYSIS
        8
        T
        (QUOTE
          ( (OR
              ( ((PREP))
                ((INDEX ((+ YEAR))))
                ((X)) )
              ( (OR
                  (((PREP)))
                  (((PREPOF))) )
                ((INDEX ((+ CO))))
                ((PREP))
                ((INDEX ((+ YEAR))))
                ((X)) ) ) ) ) ) )
    (COND ( 5  3 )
          ( T  T ) ) )
  ((REPLACE (2 4) 4))
  NIL )
```

```
((RGENDIST STRING OB ALL)
  ( (X . 1)
    ((NPROPNOM . 2) (NOUN (INDEX (+ CO))))
    (GENAF . 3)
    (OR ((INDEX (+ YEAR))) NIL)
    (NMNL (+ PERIODIC))
    (OR (PREP (INDEX (+ YEAR))) NIL)
    (*
      COMMA
      (OR ((INDEX (+ YEAR))) NIL)
      ((NMNL . 4) (+ PERIODIC))
      (OR (PREP (INDEX (+ YEAR))) NIL) )
    (OR
      ( (OR ((COMMA . 7)) NIL)
        (AND . 5)
        (OR ((INDEX (+ YEAR))) NIL)
        ((NMNL . 6) (+ PERIODIC)) )
      ((COMMA . 5)
        (OR ((INDEX (+ YEAR))) NIL)
        ((NMNL . 6) (+ PERIODIC))
        (OR (PREP (INDEX (+ YEAR))) NIL)
        (*
          COMMA
          (OR ((INDEX (+ YEAR))) NIL)
          ((NMNL . 8) (+ PERIODIC))
          (OR (PREP (INDEX (+ YEAR))) NIL) )
         OR ((COMMA . 9)) NIL)
        AND
        (OR ((INDEX (+ YEAR))) NIL)
        (NMNL (+ PERIODIC)) ) )
    (X . 10) )
   AND
    (NOT
      (ANALYSIS
        10
        T
        (QUOTE
          ( (OR
              ( (OR
                  (((PREP)))
                  (((PREPOF))) )
                ((INDEX ((+ CO))))
                ((X)) )
              ( ((PREP))
                ((INDEX ((+ YEAR))))
                (OR
                  (((PREP)))
                  (((PREPOF))) )
                ((INDEX ((+ CO))))
                ((X)) ) ) ) ) ) ) )
    (COND ( 7  4 )
          ( T  T ) ) )
  ((REPLACE (5 2 3) 5))
  NIL )
```

```
((LTIMEDST STRING OB ALL)
  ( (X . 1)
    (OR
      ( (OR
          ((INDEX (+ CO))
            GENAF
            ((NMNL . 2) (+ PERIODIC)) )
          ( (OR (THE) NIL)
            ((NMNL . 2) (+ PERIODIC))
            (OR
              ( (OR (PREP) (PREPOF))
                ((NPROPNOM . 4) (NOUN (INDEX (+ CO)))) )
              NIL ) ) ), )
        ( (OR
            ((INDEX (+ CO))
              GENAF
              ((NMNL . 3) (+ PERIODIC)) )
            ( (OR (THE) NIL)
              ((NMNL . 3) (+ PERIODIC))
              (OR
                ( (OR (PREP) (PREPOF))
                  (INDEX (+ CO)) )
                NIL ) ) )
          (*
            COMMA
            (OR
              ((INDEX (+ CO)) GENAF (NMNL (+ PERIODIC)))
              ( (OR (THE) NIL)
                (NMNL (+ PERIODIC))
                (OR
                  ( (OR (PREP) (PREPOF))
                    (INDEX (+ CO)) )
                  NIL ) ) ) )
          COMMA
          (OR
            ((INDEX (+ CO))
              GENAF
              ((NMNL . 2) (+ PERIODIC)) )
            ( (OR (THE) NIL)
              ((NMNL . 2) (+ PERIODIC))
              (OR
                ( (OR (PREP) (PREPOF))
                  ((NPROPNOM . 4) (NOUN (INDEX (+ CO)))) )
                NIL ) ) ) ) )
    (*
      COMMA
      (OR
        ((INDEX (+ CO))
          GENAF
          ((NMNL . 3) (+ PERIODIC)) )
        ( (OR (THE) NIL)
          ((NMNL . 3) (+ PERIODIC))
          (OR
            ( (OR (PREP) (PREPOF))
              (INDEX (+ CO)) )
            NIL ) ) ) )
    (OR ((COMMA . 5)) NIL)
    AND
    (OR
```

```
    ((INDEX (+ CO))
      GENAF
      (NMNL (+ PERIODIC))
      (PREP . 6)
      ((PROPNOM . 7) (NOUN (INDEX (+ YEAR)))) )
    ( (OR |THE) NIL)
      (NMNL (+ PERIODIC))
      (OR
        ( (OR (PREP) (PREPOF))
          (INDEX (+ CO))
          (PREP . 6)
          ((PROPNOM . 7) (NOUN (INDEX (+ YEAR)))) )
        ((PREP . 6)
          ((PROPNOM . 7) (NOUN (INDEX (+ YEAR))))
          (OR (PREP) (PREPOF))
          (INDEX (+ CO)) ) ) ) )
  (X . 8) )
(AND
  (NOT
    (ANALYSIS
      1
      T
      (QUOTE
        ( (OR
            ( ((X))
              ((INDEX ((+ CO))))
              ((GENAF)) )
            (((X)) ((THE)))
            ( |((X))
              ((NMNL ((+ PERIODIC))))
              (OR
                ( (OR
                    (((PREP)))
                    (((PREPOF))) )
                  ((INDEX ((+ CO)))) )
                NIL )
              ((COMMA)) ) ) ) ) ) )
    (COND ( 5  3 )
          ( T  T ) ) )
  ( (COND ( 4  (REPLACE (4 6 7) 4) )
          ( T  (REPLACE (2 6 7) 2) ) ) )
  NIL )
```

```
((LGENDIST STRING OB ALL)
    (X . 1)
    (OR (THE) NIL)
    (OR
      ( (OR
          ((INDEX (+ YEAR))
            ((NMNL . 2) (+ PERIODIC)) )
          (((NMNL . 2) (+ PERIODIC))
            (OR (PREP (INDEX (+ YEAR))) NIL) ) ) )
        (OR
          ((INDEX (+ YEAR))
            ((NMNL . 3) (+ PERIODIC)) )
          (((NMNL . 3) (+ PERIODIC))
            (OR (PREP (INDEX (+ YEAR))) NIL) ) )
        (*
          COMMA
          (OR (THE) NIL)
          (OR
            ((INDEX (+ YEAR)) (NMNL (+ PERIODIC)))
            ((NMNL (+ PERIODIC))
              (OR (PREP (INDEX (+ YEAR))) NIL) ) ) )
        COMMA
        (OR (THE) NIL)
        (OR
          ((INDEX (+ YEAR))
            ((NMNL . 2) (+ PERIODIC)) )
          (((NMNL . 2) (+ PERIODIC))
            (OR (PREP (INDEX (+ YEAR))) NIL) ) ) ) )
    (*
      COMMA
      (OR (THE) NIL)
      (OR
        ((INDEX (+ YEAR))
          ((NMNL . 3) (+ PERIODIC)) )
        (((NMNL . 3) (+ PERIODIC))
          (OR (PREP (INDEX (+ YEAR))) NIL) ) ) )
    (OR ((COMMA . 4)) NIL)
    AND
    (OR (THE) NIL)
    (OR
      ((INDEX (+ YEAR)) (NMNL (+ PERIODIC)))
      ((NMNL (+ PERIODIC))
        (OR (PREP (INDEX (+ YEAR))) NIL) ) )
    (OR ((PREP . 5)) ((PREPOF . 5)))
    ((NPROPNOM . 6) (NOUN (INDEX (+ CO))))
    (X . 7) )
  (AND
    (NOT
      (ANALYSIS
        1
        T
        (QUOTE
          ( (OR
              (((X)) ((THE)))
              ( ((X))
                ((NMNL ((+ PERIODIC))))
                (OR
                  ((PREP))
                  ((INDEX ((+ YEAR)))) )
```

```
                    NIL )
                  ((COMMA)) )
                ( ((X))
                  ((INDEX ((+ YEAR)))) ) ) ) ) ) )
    (COND ( 4  3 )
          ( T  T ) ) )
  ((REPLACE (2 5 6) 2))
  NIL )
```

```
((CARDNOUN STRING OB ALL)
  ( (X . 1)
    (((VADJ . 2) (+ CARD)) (W . 5))
    (OR
      ((A . 3))
      ((BAUX . 3))
      ((COMMA . 3))
      ((DAUX . 3))
      ((PREP . 3))
      ((PUNCT . 3))
      ((THE . 3))
      ((V . 3))
      (((VADJ . 3) (+ CARD)))
      ((VADJ . 3) PREP)
      ((VPART . 3)) )
    (X . 4) )
  (NOT
     AND
      (ANALYSIS 3 NIL (QUOTE (((V)))))
      (ANALYSIS
       4
       T
       (QUOTE
         ( ((INDEX ((- CONST))))
           ((X)) ) ) ) ) )
( (REPLACE ( ((NP)
               ((NOM)
                 ((NOUN ((+ SG) (- HUMAN)))
                   ((INDEX ((+ CONST) (+ CARD))) 5) ) ) ) )
           2 ) )
NIL )
```

```
((ABTAPPRX STRING OB ALL)
  ( (X . 1)
    ((PREP . 2) ABOUT)
    (OR
      (((VADJ . 3) (+ CARD)))
      ((EQUAL . 3))
      ((WH . 4)
        SOME
        (OR (LARGE) (MANY) (MUCH)) ) )
    (X . 5) )
  (COND ( 4  (NULL 1) )
        ( T  T ) )
  ( (REPLACE ( ((ADV)
                  ((V ((+ ADJ)))
                    ((APPROX)) ) ) )
               2 ) )
  NIL )
```

```
((COMPNQFR STRING OB ALL)
  ( (X . 1)
    (OR
      ( ((PREP . 2) A1)
        (OR ((LEAST . 3)) ((MOST . 4))) )
      ( (OR ((NO . 5)) ((NOT . 5)) NIL)
        (OR
          (((VCOMP . 7) LESSTHAN))
          ( (OR ((MORE . 6)) ((FEWER . 7)))
            ((THAN . 8)) ) ) ) )
    ((VADJ . 9) (+ CARD))
    (X . 10) )
  (NOT
    (ANALYSIS
      1
      NIL
      (QUOTE
        ( ((X1))
          (OR
            (((NO)))
            (((NOT))) ) ) ) ) )
  ( (COND ( (AND 6 (NOT 5))
            (REPLACE ( ((ADV)
                        ((V ((+ ADJ) (+ COMP) (+ TWOARGS) (+ NMA3X)))
                          ((GREATERTHAN)) ) )
                       9 )
                     9 ) )
          ( (AND 7 (NOT 5))
            (REPLACE ( ((ADV)
                        ((V ((+ ADJ) (+ COMP) (+ TWOARGS) (+ NMA3X)))
                          ((LESSTHAN)) ) )
                       9 )
                     9 ) )
          ( (OR 3 (AND 5 7))
            (REPLACE ( ((ADV)
                        ((V ((+ ADJ) (+ COMP) (+ TWOARGS) (+ NMA2X)))
                          ((GRTRTHANEQ))
                       9 )
                     9 ) )
          ( (OR 4* (AND 5 6))
            (REPLACE ( ((ADV)
                        ((V ((+ ADJ) (+ COMP) (+ TWOARGS) (+ NMA2X)))
                          ((LESSTHANEQ)) ) )
                       9 )
                     9 ) ) )
    (DELETE 2)
    (DELETE 3)
    (DELETE 4)
    (DELETE 5)
    (DELETE 6)
    (DELETE 7)
    (DELETE 8) )
  NIL )
```

```
((WHNUMAMT STRING OB ALL)
  ( (X . 1)
    (OR
      (((VADJ . 2) WH SOME))
      ( ((WHADV . 2) (V WH SOME))
        ((VADJ . 2) LARGE)
        (A . 2) ) )
    ((NOMQ . 3)
      (NOUN
        (V
          (OR ((NUMBER . 7)) ((AMOUNT . 8))) )
        (INDEX . 9) ) )
    (OR
      ((OF . 4) (THE . 5))
      ((OF . 4))
      NIL )
    (X . 6) )
  (AND
    (NOT
      (ANALYSIS
        6
        /S1
        (QUOTE
          ( (OR
              (((OF)) ((X)))
              (((THE)) ((X1))) ) ) ) ) )
    (COND ( (NULL 4)  8 )
          ( T  T ) ) )
  ( (COND ( 4
            (COND ( 7
                    (REPLACE ( ((WHADJ)
                                ((ADV ((+ EXT)))
                                  ((V ((+ ADJ) (+ QUANT)))
                                    ((WH))
                                    ((SOME)) ) )
                                ((V ((+ ADJ) (+ QUANT) (+ POL)))
                                  ((MANY)) ) ) )
                               3 ) )
                  ( 8
                    (REPLACE ( ((WHADJ)
                                ((ADV ((+ EXT)))
                                  ((V ((+ ADJ) (+ QUANT)))
                                    ((WH))
                                    ((SOME)) ) )
                                ((V ((+ ADJ) (+ QUANT) (+ POL)))
                                  ((MUCH)) ) ) )
                               3 ) ) ) )
          ( T
            (REPLACE ( ((QNOM)
                        ((ADV ((+ EXT)))
                          ((V ((+ ADJ) (+ QUANT)))
                            ((WH))
                            ((SOME)) ) )
                        ((NOM)
                          ((NOUN ((- HUMAN) (+ SG)))
                            ((V ((+ ADJ) (+ QUANT) (+ POL)))
                              ((MUCH)) )
                            9 ) ) ) )
                       3 ) ) ) )
```

```
    (COND ( (AND 4 (NOT 5))  (DELETE 4) ) )
    (DELETE 2) )
  NIL )
-----------------------------------------

((SPRPPREV STRING OB ALL)
  ( (X . 1)
    (OR
      (((PREP . 2) (W . 5)))
      (((PREPOF . 3) (W . 5))) )
    (X . 4) )
  (NOT
    (ANALYSIS
      4
      NIL
      (QUOTE
        ( (OR
            (((BAUX)))
            (((COMMA)))
            (((DAUX)))
            (((PREP)))
            (((PUNCT)))
            (((V)))
            (((VADJ)))
            (((VING))) )
          ((X)) ) ) ) )
  ( (COND ( 2
            (COND ( (ANALYSIS
                      2
                      NIL
                      (QUOTE
                        (((PREP ((+ LOC2))))) ) )
                    (REPLACE ( ((NSPREP ((+ LOC2))) 5) )
                             2 ) )
                  ( T  (REPLACE (((NSPREP) 5)) 2) ) ) )
          ( 3  (REPLACE (((NSPREPOF) 5)) 3) ) ) )
  NIL )
-----------------------------------------

((TIMECMPD STRING OB ALL)
  ( (X . 1)
    ((PROPNOM . 2) (NOUN (INDEX (+ YEAR))))
    ((NMNL . 3) (+ PERIODIC))
    (X . 4) )
  NIL
  ( (REPLACE (((NMNL) 2 3)) 3)
    (DELETE 2) )
  NIL )
-----------------------------------------
```

```
((RANKCMPD STRING OB ALL)
  ( (X . 1)
    (NMNL . 2)
    ((NMNL . 3) (V PLACE RANK) INDEX)
    (X . 4) )
  (NOT
    (ANALYSIS
      1
      /S1
      (QUOTE
        ( ((X)N
          ((INDEX ((+ YEAR)))) ) ) ) )
  ( (REPLACE ( ((NMNL) ((NOM) 2) 3) )
               3 )
    (DELETE 2) )
  NIL )
```

```
((PROPERPP STRING OB ALL)
  ( (X . 1)
    (OR ((NSPREP . 2)) ((NSPREPOF . 3)))
    (OR ((PROPNOM . 4)) ((NPROPNOM . 4)))
    (X . 5) )
  (NOT
    (OR
      (ANALYSIS
        5
        T
        (QUOTE
          (((GENAF)) ((X))) ) )
      (AND
        (ANALYSIS
          5
          /S1
          (QUOTE
            ( ((NMNL ((+ PERIODIC))))
              ((X)) ) ) )
        (ANALYSIS
          4
          NIL
          (QUOTE
            ( ((PROPNOM)
                ((NOUN)
                  ((INDEX ((+ YEAR)))) ) ) ) ) ) ) ) )
  ( (COND ( 2
            (REPLACE ( ((PP) 2 ((NP) 4)) )
                     2 ) )
          ( 3
            (REPLACE ( ((PPOF) 3 ((NP) 4)) )
                     3 ) ) )
    (DELETE 4) )
  NIL )
```

```
((TICOPPCD STRING OB ALL)
  ( (X . 1)
    (OR
      (((NOM . 2) (NOUN (+ PERIODIC))))
      (((NMNL . 3) (+ PERIODIC)))
      (((NMNL . 4) NOM (NMNL (V PLACE RANK) INDEX)))
    (OR
      ( (OR
          ( ((PPOF . 5)
              NSPREPOF
              (NP (NPROPNOM (NOUN (INDEX (+ CO))))) )
          ( ((PP . 5)
              NSPREP
              (NP (NPROPNOM (NOUN (INDEX (+ CO))))) ) )
        (OR
          ( ((PP . 6)
              NSPREP
              (NP (PROPNOM (NOUN (INDEX (+ YEAR)))))) )
          NIL ) )
      ( ((PP . 6)
          NSPREP
          (NP (PROPNOM (NOUN (INDEX (+ YEAR))))) )
        (OR
          ( ((PPOF . 5)
              NSPREPOF
              (NP (NPROPNOM (NOUN (INDEX (+ CO))))) )
          ( ((PP . 5)
              NSPREP
              (NP (NPROPNOM (NOUN (INDEX (+ CO))))) )
          NIL ) ) )
    (X . 7) )
  (AND
    (NOT
      (ANALYSIS
        1
        T
        (QUOTE
          ( ((X))
            (OR
              (((INDEX ((+ YEAR))))
              (((VCOMP)))
              (((VC)))
              ( ((VCPART))
                (OR
                  ( ((NSPREP))
                    ((NPROPNOM))
                    ((GENAF)) )
                  NIL ) )
              (((VCING))) ) ) ) ) ) )
    (COND ( (NOT (AND 5 6))
            (NOT
              (ANALYSIS
                7
                T
                (QUOTE
                    (OR
                      (((NSPREP)))
                      (((NSPREPOF)))
                      (((PREP))) )
```

```
                    ((X)) ) ) ) ) )
          ( T  T ) )
  (COND ( (NOT 5)
          (ANALYSIS
            1
            T
            (QUOTE
              (((X)) ((GENAF))) ) ) )
          ( T  T ) ) )
( (COND ( 2
          (REPLACE ( ((NOM) 2 ((Z1) 5 6)) )
                   2 ) )
          ( 3
            REPLACE ( ((NOMN)
                       ((NOMN) 3)
                       ((Z1) 5 6) ) )
                   3 ) )
          ( 4
            (REPLACE ( ((NOMN)
                       ((NOMN) 4)
                       ((Z1) 5 6) ) )
                   4 ) ) )
  (DELETE 5)
  (DELETE 6) )
NIL )
```

References

1. Petrick, S. R., "Semantic Interpretation in the REQUEST System", in A. Zampolli (ed.) Computational and Mathematical Linguistics. Proceedings of the International Conference on Computational Linguistics. Pisa, 27 VIII - 1/IX 1973, Casa Editrice Olschki, Firenze (1974), Vol. I.

2. Plath, W. J., "Transformational Grammar and Transformational Parsing in the REQUEST System", in A. Zampolli (ed.), Computational and Mathematical Linguistics. Proceedings of the International Conference on Computational Linguistics. Pisa, 27 VIII - 1/IX 1973, Casa Editrice Olschki, Firenze (1974), Vol. II.

3. Knuth, D. E., "Semantics of Context-free Languages" Mathematical Systems Theory, Vol. 2 (1968), pp. 127-145.

4. Petrick, S. R. "On the Use of Syntax-based Translators for Symbolic and Algebraic Manipulation", in S. R. Petrick (ed.), Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, ACM, New York (1971), pp. 224-237

5. Petrick, S. R., Mapping of Linguistic Structures into Computer-Interpretable Form, AFCRL-TR-73-0055, Final Report, Contract No. F19628-72-C-0129 (December 1972).

6. Petrick, S. R., A Recognition Procedure for Transformational Grammars, MIT Doctoral Dissertation (1965)

7. Chomsky, N., Aspects of the Theory of Syntax, MIT Press, Cambridge, Mass. (1965).

8. McCawley; J. D., "Meaning and the Description of Languages" Kotoba No Uchu, Vol. 2, Nos. 9-11 (1967).

9. Lakoff, G., "Linguistics and Natural Logic", Synthese, Vol. 22, Nos. 1-2 (1970).

10. Kuno, S. and Oettinger, A. G., "Multiple-path Syntactic Analyzer", Mathematical Linguistics and Automatic Translation, Report No. NSF-8, Sec. I, The Computation Laboratory of Harvard University (1963).

11. Plath, W. J., "Multiple-path Syntactic Analysis of Russian", Mathematical Linguistics and Automatic Translation, Report No. NSF-12, The Computation Laboratory of Harvard University (1963)

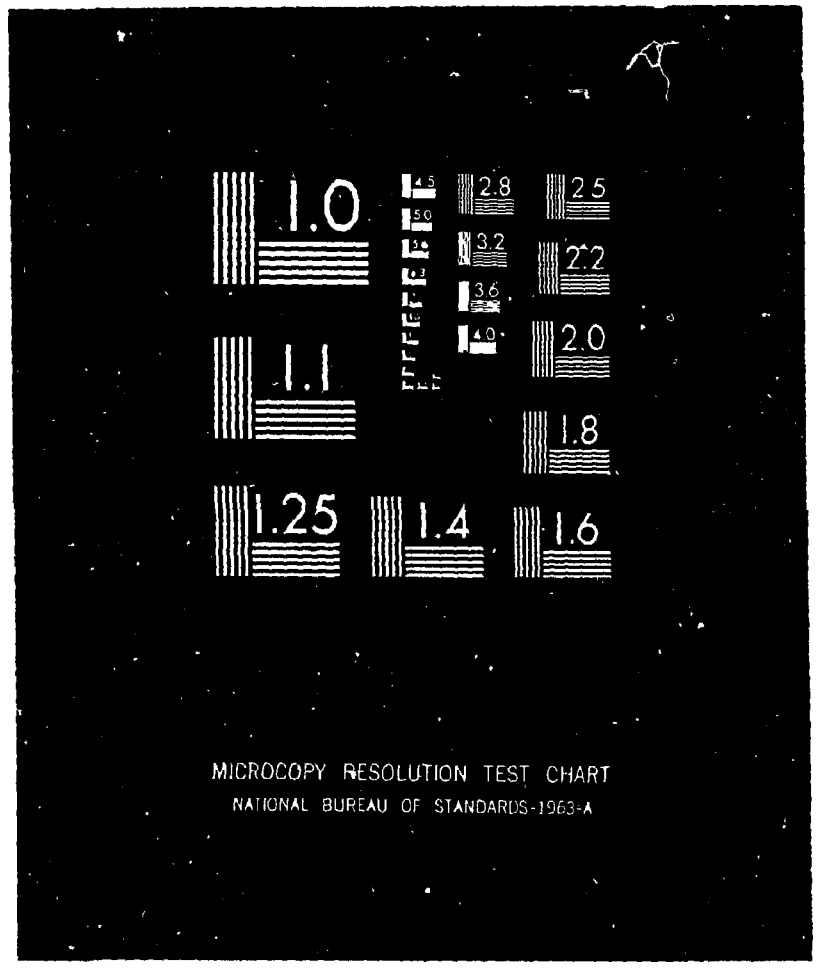
12. Woods, W. A., Kaplan, R. M. and Nash-Webber, B., The Lunar Sciences Natural Language Information System, Final Report, BBN Report No. 2378, Cambridge, Mass. (1972)

13. Plath, W. J., "A Tag Language for Syntactic and Semantic Analysis" in H. H. Josselson (ed.), Proceedings of the 1965 Conference on Computer-Related Semantic Analysis, Wayne State University, Detroit, Mich. (1966).

14. Friedman, J., Bredt, T. H. et al., A Computer Model of Transformational Grammar, American Elsevier, New York (1971)

15. Selzer, L. E. (ed.), The Columbia Lippincott Gazetteer of the World, Columbia University Press, New York (1961).

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