HOW FEATURES RESOLVE SYNTACTIC AMBIGUITY

Bozena H. Dostert and Frederick B. Thompson
California Institute of Technology, Pasadena, California

ABSTRACT

Ambiguity is a pervasive and important aspect of natural language. Ambiguities, which are disambiguated by context, contribute powerfully to the expressiveness of natural language as compared to formal languages. In computational systems using natural language, problems of properly controlling ambiguity are particularly large, partially because of the necessity to circumvent parsings due to multiple orderings in the application of rules.

Features, that is, subcategorizations of parts-of-speech, constitute an effective means for controlling syntactic ambiguity through ordering the hierarchical organization of syntactic constituents. This is the solution adopted for controlling ambiguity in REL English, which is part of the REL (Rapidly Extensible Language) System. REL is a total software system for facilitating man/machine communications. The efficiency of processing natural language in REL English is achieved both by the detailed syntactic aspects which are incorporated into the REL English grammar, and by means of the particular implementation for processing features in the parsing algorithm.

KEY WORDS AND PHRASES

REL System, natural language, syntax, grammar, semantics, features, parsing, computational linguistics, man/machine communication, ambiguity

HOW FEATURES RESOLVE SYNTACTIC AMBIGUITY

I. Introduction

Natural language, e.g. English, is highly ambiguous. Computational analysis of natural language has added a new dimension to the discovery of language ambiguity, and, not infrequently, created problems in ambiguity which are not transparent to the ordinary linguist, and even less to the ordinary language user.

Many sentences may have more than one interpretation, for example: "The students suggested that they might stop smoking in the meetings." Even more commonly, phrases that occur as parts of sentences may be highly ambiguous. For example: "German books" may mean (a) books in German, (b) books on German, (c) books made in Germany, and so on. Ambiguity, either at the phrase level or at the sentence level, is a powerful mechanism in language. Ambiguity inherent in syntax allows for fewer syntactic constructions to be used in expressing meanings. If every phrase and sentence of the language had to be unambiguous in isolation as well as context, the amount of syntactic structure to render different meanings would have to be considerably increased. Ambiguities, which are disambiguated by context, contribute powerfully to the expressiveness of natural language as compared to formal languages.

This paper is concerned with ambiguity problems as they occur in the environment of a computational system which allows the use of natural language in man/machine interaction. Syntactic ambiguity inherent in natural language is only part of the problem that faces a linguist in this environment. He must also deal with ambiguity which is, in a sense, his own creation, or more precisely, the result of the simple-mindedness of the algorithms that he is forced to impose on the analysis of language. Thus:

Where was the brother of John's wife in 1966?

might be intuitively unambiguous. Yet from the point of view of the parsing algorithm these interpretations are possible:

Where was the (brother of ((John's wife) in 1966)?

Where was the (brother of (John's wife)) in 1966)?

Where was the ((brother of John)'s wife) in 1966)?

All overt clues present in a language must be utilized for disambiguation, and ways must be found to circumvent multiple parsings where no inherent ambiguity exists.

In the case of systems working with a data base (some well defined universe of discourse) ambiguity can often be resolved by reference to the data base -- as indeed it is commonly resolved in natural language. It has been found however that it is often time-saving to complete syntactic analysis of a sentence before proceeding to establishing its semantic interpretation. All possible syntactic analyses of a sentence must therefore be considered. Further, where only one interpretation is possible, and only the order of constituent analysis is to be determined, the choice of parsing can greatly influence the subsequent semantic computations. Thus, in the case of:

John's (son who lives in Boston)

(John's son) who lives in Boston

the former parsing will require far more accesses to the data base, since all ambiguous readings of "son who lives in Boston" have to be found. To put it another way, John is likely to have fewer sons than there are sons who live in Boston.

Resolution and control of syntactic ambiguity requires attention to the overt syntactic clues present in the input sentence. These may be morphological, such as inflectional suffixes, agreement in gender, number and case, clues provided by the syntactic environment such as determiners and auxiliary verbs, and positional clues such as inversion of subject and verb in questions.

One way of exploiting some of these clues is to use multiple parts-of-speech. Thus one might have both <possessive noun>and <nominative noun>, instead of just <noun>, to distinguish between "boy's" and "boy." This, however, leads to further subcategorizations, and thus rapid proliferation of parts-of-speech: <possessive plural noun>and <possessive singular noun> for "boys'" and "boy's." An immediate undesirable result of this solution is the proliferation of the rules of grammar. Thus, a simple rule such as:

<noun> → <article> <noun>

must be replaced by several, e.g.:

<determiner modified nominative plural noun>→

<article> <non-determiner modified nominative plural noun>.

A second device, one that is often resorted to, is to adopt special parts-of-speech whose only function is to make explicit such syntactic clues. Thus one might have the context-sensitive rule:

<noun> <possessive> → <noun>'s

There again problems arise. To parse "boys' location" and "boy's location," we would need such rules as

<noun> → <noun> <plural> <possessive> <noun>

<noun> → <noun> <singular> <possessive> <noun>

The result is again a great multiplying effect on the rules of grammar and on parsing time.

A major defect of both of the above solutions is that they increase markedly, rather than decrease, the syntactic ambiguity they seek to control.

The device which we have employed to control syntactic ambiguity is features. The role of features is dual: subcategorization of parts-of-speech and the determination of the order of constituent analysis. As an example of the former, the word "boys'" is assigned the simple part-of-speech <noun> and the plural and possessive features. As an example of the latter, in the case of "(brother of ((John's wife) in 1966," parsing proceeds in two steps: (1) the normal matching of a grammar rule to a segment of the parsing graph; (2) a check to determine whether the features assigned to the constituent phrases match those prescribed by the rule. The explicit mechanisms are discussed in Section IV. As will be seen, the algorithm we have developed is indeed efficient. In Section III, we present the features that we are now using and examples of their application in syntactic disambiguation. First, however, we need to characterize the particular computational linguistic system we are using, namely REL.

II. The REL Environment

The REL (Rapidly Extensible Language) System is a complete software system for communicating with the computer for the purpose of data analysis, and construction and application of conceptual models. It includes a natural component, REL-English, which enables the user to work with the computer in a subset of natural English. [1, 2, 3]

The REL system language processor makes use of the parsing algorithm due to Martin Kay. [4] This is a bottom-to-top, right to left, single pass parser that can handle any general rewrite rule grammar. REL does not make use of a lexicon but rather takes as its terminal vocabulary all of the printable characters: upper and lower case alphabetics, digits, punctuation marks, and blank. Thus it would recognize a word such

as "John" in terms of the grammar rule:

<name> → John

The REL English parts-of-speech strike an ordinary linguist as highly unconventional. They are:

<name>	e.g. boy, tall, John
<relation>	e.g. location, sister, author
<number>	e.g. 523
<verb>	e.g. arrive, marry
<time modifier>	e.g. January, before 1960
<relative clause>	e.g. who lived in Boston

Function words (in the sense of Fries) [5] e.g. "have," "of," "and," "all," are included in rules in their literal form, for instance, "of" in the rule:

<name> → <relation> of <name>

which applies to, e.g.:

Boston → location of John

The reasons for using these rather than conventional parts-of-speech stem from the way the surface sentence is related to the corresponding data structures used for the various data bases (universes of discourse). We are aware of the fact that our parts-of-speech may make our grammar less transparent. We, in turn, find the separation between the syntactic and semantic levels of language to be an unnatural one. Our parts-of-speech reflect our view of language as a functionally integrated phenomenon. Since we are leaving the semantic part of our analysis out of account, it would not be difficult to conventionalize our parts-of-speech for the purposes of this presentation. We certainly feel that the use of features that our system exemplifies can be carried over to other systems, using conventional parts-of-speech or otherwise. Secondly, we feel it particularly important that we can cite actual operating experience, and this can be done without distortion only if we adhere to the actual form of our grammar. The REL system became operational in February 1970, and has amply demonstrated its efficiency as a time-shared relational data base question-answering system.

III. Features and Their Functions

The general notion of features as subcategorizations of parts-of-speech is certainly not new. They are implicit in a paper of G. N. Harman. [6] Chomsky discusses features at length in Aspects of the Theory of Syntax. [7] However, these and other references deal almost exclusively with linguistic issues of a somewhat different sort than we are interested in here. In particular, Chomsky's features characterize lexical items in terms of such categories, for instance, as Count, Common, Animate, Human for the entry "boy"; they also characterize the syntactic environment in which an item can occur, e.g. for "grow": + verb, which can occur with a noun phrase (grow flowers), by itself (flowers grew), and with an adjective (grow old). The primary role of our features is the ordering of the hierarchical organization of syntactic constituents with the aim of controlling syntactic ambiguity. This constitutes an interesting extension of the use of features, both from strictly linguistic and computational points of view.

Each part of speech is assigned a class of features. A particular word or phrase bearing that part of speech is checked for the presence or absence of these features.

A. Features that Structure Nominal Phrases

In nominal phrases, features are used to indicate inflectional structure (plural and possessive) and constituent structure. They mark the presence of determiners and quantifiers and enforce the desired grouping of modifiers.

The plural (PLF) and possessive (POF) features result from the following rules:

$$\text{<name>}_{1+\text{PLF}} \rightarrow \text{<name>}_{-\text{PLF}}\ s$$

e.g. boys → boy s

This rule allows the plural "s" to go on a name which is not plural (−PLF) and results in a name that is marked as plural (+PLF). The "1" means that the features of the first constituent on the right-hand side are also carried over and assigned to the resulting left-hand side.

The rules for possessive, singular and plural, are:

$$\text{<name>}_{1+\text{POF}} \rightarrow \text{<name>}_{-\text{PLF}-\text{POF}}\ 's$$

e.g. boy's

$$\text{<name>}_{1+\text{POF}} \rightarrow \text{<name>}_{+\text{PLF}-\text{POF}}\ '$$

e.g. boys'

The possessive is set last, after all premodification, as controlled by other nominal features

e.g. (big dog)'s master.

The determiner (DTF) and quantifier (QNF) features result from these rules:

$$\text{<name>}_{1+\text{DTF}} \rightarrow \begin{bmatrix}\text{the}\\ \text{a}\end{bmatrix}\ \text{<name>}_{-\text{DTF}}$$

e.g. the boy

By requiring that the right-hand side <name> must not be modified by a determiner (i.e. the DTF feature must be off), the rule prevents such ungrammatical sequences as

* the the boy

Examples of quantifier rules are:

<name>$_{1+QNF-PLF}$
→ some <name>$_{-QNF-DTF}$
→ all <name>$_{-QNF+PLF}$
→ all of <name>$_{-QNF+PLF+DTF}$

These rules allow such sequences as exemplified above plus such as "some boys" (the plural feature is not marked in the rule), "all the boys" (DTF is not marked in the rule) and rule out such ungrammatical sequences as * some the boy (DTF must be off) and * all boy (PLF must be on).

The setting of determiner and quantifier features prevents further adjectival or possessive modification. The remaining features govern the order of constituent analyses (in conjunction, of course, with the ones above). This is necessitated by the fact that noun phrases are hierarchically structured, i.e. some constituents serve as modifiers of others, and the ordering of the sequence of modification determines the meaning of a given phrase. Some phrases are genuinely ambiguous, e.g. "Jane's children's book" can mean either 'the book of Jane's children' or 'a children's book which is in some relationship with Jane, e.g. owned by her.' Other examples are the familiar "stout major's wife" or "little boys' school." But in computational analysis, phrases which are normally unambiguous also turn out to have alternate analysis, thus "wealthy benefactor's statue" parses both ways, i.e. "(wealthy benefactor)'s statue" and "wealthy (benefactor's statue)." The second is obviously incorrect, and our features are directed at discarding such analyses.

We distinguish between (1) adjectival (APF), (2) possessive (PSF), (3) prepositional ("of"-phrase) (PMF), and (4) relative clause modification (RCF), e.g. (1) old uncle, (2) John's uncle, (3) uncle of John, (4) uncle who attended Harvard.

APF marks phrases resulting from modification by nouns other than possessive and adjectives.

PSF marks phrases resulting from modification by a possessive noun.

PMF marks phrases containing "of."

RCF is discussed in greater detail below.

The following five rules illustrate rules used to control groupings.

(1) <name>$_{2+APF}$ → <name>$_{-APF-DTF-PMF-POF-PSF-RCF-QNF}$ <name>$_{-DTF-PMF-POF-PSF-RCF-QNF}$

Yale women

(2) <name>$_{2+PSF}$ → <name>$_{-DTF-PMF+POF-RCF}$ <name>$_{-DTF-PMF-POF-PSF-RCF-QNF}$

Scott's Ivanhoe

(3) <name>$_{2+PSF}$ → <name>$_{-DTF-PMF+POF-RCF}$ <relation>$_{-DTF-PMF-POF-PSF-RCF}$

John's sister

(4) <name>$_{1+PMF}$ → <relation>$_{-DTF-POF-PSF-RCF-PMF}$ of <name>$_{-POF}$

location of John

(5) <name>$_{1+RCF}$ → <name>$_{-DTF-PMF-POF-RCF}$ who <verb>

women who left Yale

Adjectival structures are grouped first, i.e. before possessive and "of"-phrases,

e.g.: John's (old uncle)

: (crowded New York)'s subways

: dormitories of (Yale women)

: sister of (uncle John)

If there is more than one adjective, they group to the right,

e.g. good (old uncle).

Possessive structures are formed after adjectival structures and before "of"-phrases are put on,

e.g. friend of (Chicago's mayor).

If there are more than one possessive, they group to the left,

e.g. (John's son)'s teacher.

If there are more than one "of"-phrases, they group to the right,

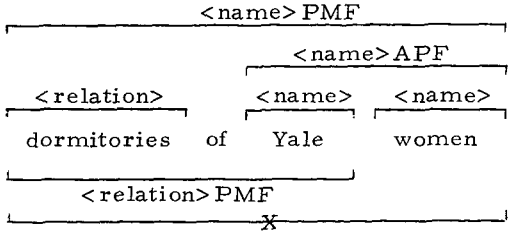
e.g. sister of (the father of John).

Modification by a pre-posed possessive and a post-posed "of"-phrase is prevented,

e.g. * New York's governor of New York

* John's sister of Boston

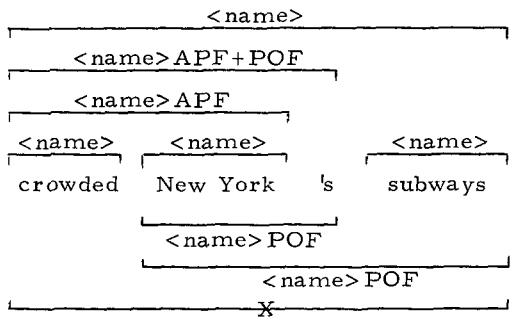
How are these results achieved and is the control of ambiguity according to our solutions desirable in all cases? Let us consider some examples of our analyses.



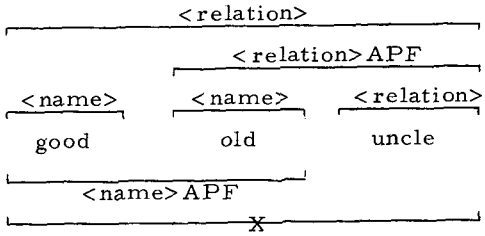
The bottom parsing is prevented by PMF since the rule

<name> → <relation> <name>

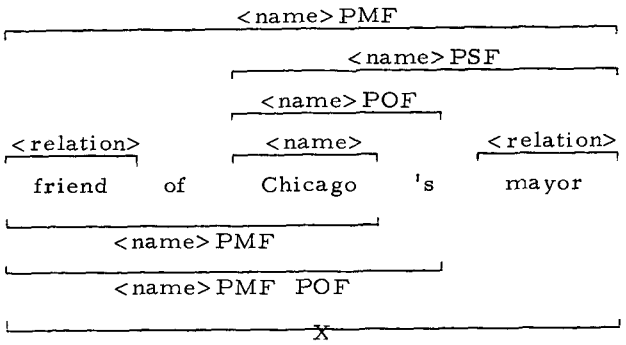
requires the PMF (prepositional modification) to be off on the <relation>. The correct analysis is achieved here.



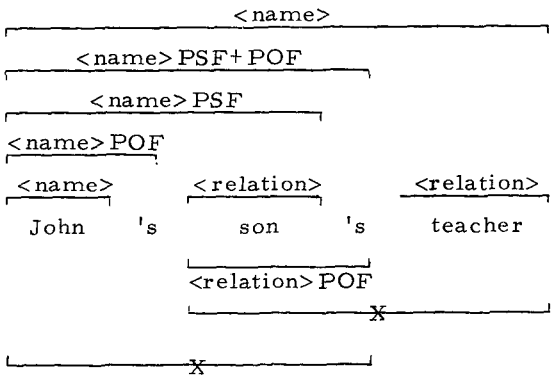
The bottom parsing is prevented, since rule (1) combining <name> and <name> calls for the possessive PSF to be off on the second name. But this is clearly a case of over-disambiguation, since both readings are possible. Correct disambiguation is obtained in "(wealthy benefactor)'s statue," but incorrect in "(wealthy building)'s proprietor."



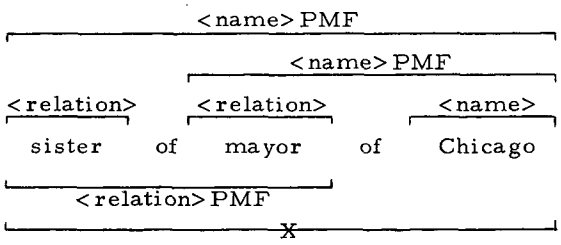
The rule combining <name> and <relation> requires that the <name> be not modified by an adjective (APF) and thus the rule does not apply. But in the case of "Vermont (marble statues)" or "light (grey suit)" we would have over disambiguation.



The rule combing <name> with <relation> requires that the PMF must be off on the <name>. However, we also exclude "(mayor of Chicago)'s sister."



The rule combining <relation> with <relation> calls for the POF to be off on the first <relation> and the rule combining <name> and <relation> calls for the POF to be off on the <relation>. These rules exclude, however, * "Jill's (children's book)" which is legitimately ambiguous.



The rule combining <relation> of <name> requires the PMF to be off on the <relation>.

The remaining feature, RCF, marks phrases consisting of nouns modified by relative clauses. Such modification is subsequent to adjectival and possessive modification, e.g. "(old man) who left Boston," "(John's friend) who left Boston," and precedes "of"-phrase modification, e.g. sister of (the boy who left Boston).

An interesting constraint on constructions involving relative clauses is that the noun on the left-hand side of "of" cannot be modified by a relative clause unless the noun on the right-hand side is also, thus

* the sister who left Boston of John

is ungrammatical, while

the sister who attends Yale of the boy who attends Harvard

is well-formed. The ungrammatical construction is prevented by rules which require parallel occurrence of the RCF feature on both nouns.

Another interesting use of features and the comma is the resolution of ambiguity in constructions involving relative clauses such as

sister of the boy who left Harvard

in which the relative clause may refer to either of the nouns. There are two sets of rules, with and without the comma, e.g.

<name> → <name> who <verb>

<name> → <name>, who <verb>

and together with the function of features they result in different groupings:

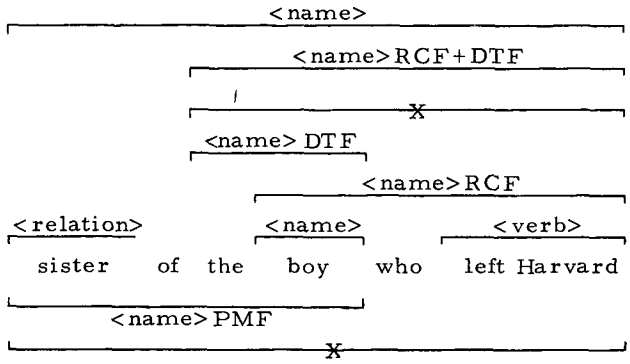
sister of (the boy who left Harvard)

(sister of the boy), who left Harvard

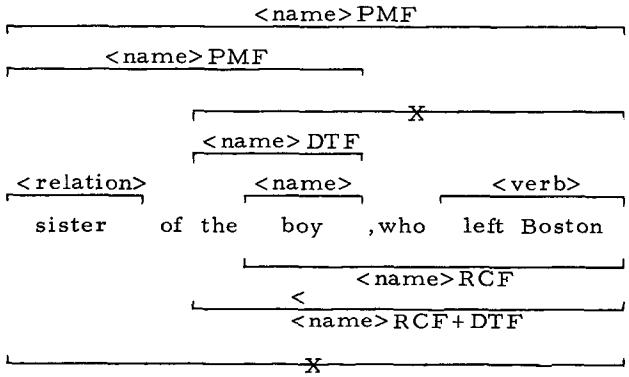
boys who knew (girls who left Boston)

(boys who knew girls), who left Boston.

How this is achieved can be observed in the following examples:



The bottom parsing is discarded since the rule which puts the relative clause on the <name> requires that the <name> must not be modified by an "of"-phrase. Also, the parsing combining "the boy" with "who <verb>" is prevented by requiring that the <name> must not be determiner modified.



The bottom parsing is excluded since the rule combining <relation> and <name> prohibits the name to be modified by a relative clause.

As we have seen, features are a powerful mechanism in controlling undesirable ambiguity. In some cases, their use leads to over-disambiguation. This could be avoided by relaxing their functions, and allowing ambiguous parsings to show up. Unfortunately, undesirable and illegitimate ambiguities would also show up. Our purpose is to facilitate man/machine communication rather than exhaustive linguistic analysis, and thus we have strived to find a balance between computing efficiency and freedom of language use. We have not reached a final conclusion in some of these matters and may relax certain of our feature rules to allow more ambiguous parsings when their disambiguation is likely to result from semantic checks. But final decisions must await more experience with the system.

B. Features that Structure Verb Phrases

One essential departure in our terminology, which reflects our grammatical analysis, is that a verb phrase is not limited to verb + noun phrase (+ modifiers), but includes, in the final stages of analysis, also the subject.

Limitations of space preclude detailed discussion of verb phrase features, and only a few interesting examples are considered.

Verb phrase features mark:

(1) Inflectional shape of verbs:

FSI - singular, which restricts the verb to singular subjects. The rules which call for it involve: "does, doesn't, begins, ceases," "is, was, has," and the third person singular marker "s," e.g. "arrives."

FPP - past participle, in the usual sense. Modification by "be" and "have" auxiliaries is permitted, but no other. This form is an allowed verbal modifier of nouns, e.g. "married man." The feature is set by "d" (or "ed") morphemes and "had," "been" and "begun to."

(2) Negative modification:

FNG - set by "not" and auxiliaries containing "not," e.g. "won't."

(3) Copula verbs:

FCV and FHV mark, respectively, "be" and "have." FCV is the encompassing feature to deal with copula verbs in general, which results in fewer rules.

(4) The following features determine the order of parsing:

FPH - phrasal, FLC - left collecting, and FTM - tense modifier. Examples are:

for FPH:

(John (enters)) vs. * (John (enter)s)

i.e. when a phrase is built, the FPH prevents putting the morphemic modification "s" on the phrase rather than on the verb itself.

for FLC:

(John enters) Harvard vs. *John (enters Harvard)

((Is (John seen)) by Mary?

* Is ((John seen) by Mary)?

* ((Is John) seen) by Mary?

i.e. the subject and auxiliaries are combined with the verb before the object is put on.

for FTM:

(lived in Boston) (from 1960 to 1965)

* (lived in Boston from 1960) to 1965

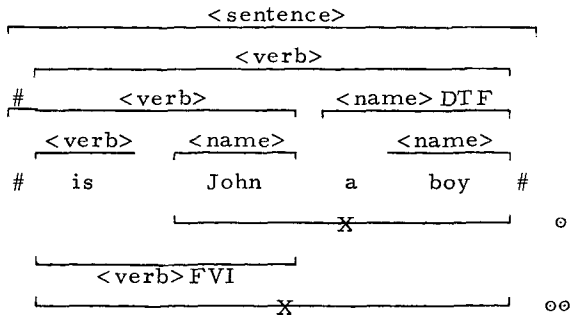
(5) The remaining three features are used to deal with deep structural relations in verb phrases.

(a) FPA, passive, is required by the rule which puts the surface object, preceded by "by," as deep structure subject. The feature is turned on by the "be" auxiliary on transitive verbs, which are distinguished from intransitive verbs by the feature FVI. An example is "the dog is owned by John," where "John" is the deep subject and "dog" is the deep object. Ungrammatical strings such as * "the dog owns by John" and * "the dog is owned John" are not permitted by the rules. The specific rules are (only relevant features are shown):

$$\langle \text{verb} \rangle \rightarrow \langle \text{verb} \rangle_{-\text{FPA}-\text{FVI}-\text{FHV}} \quad \langle \text{name} \rangle_{-\text{POF}}$$

$$\langle \text{verb} \rangle \rightarrow \langle \text{verb} \rangle_{+\text{FPA}} \text{ by } \langle \text{name} \rangle_{-\text{POF}}$$

(b) FVI, intransitive verb. One of its important roles is to allow the "be" auxiliary without triggering the passive transformation, e.g. "born" can be defined as "begun to be alive" and such phrases as "was born" are not treated as passives. Another function is to reject an incorrect parsing in questions beginning with "be" or "have."



o (fails because of determiner)
oo (fails because of FVI)

The final rules are (features omitted)

<verb> → # <verb> <name>

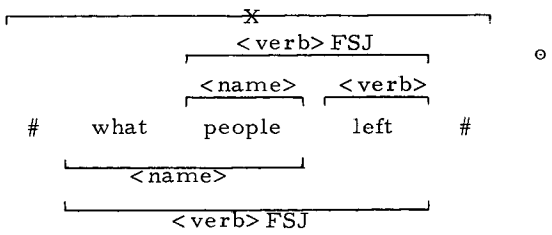
<sentence> → # <verb> #

(# is the boundary symbol). By marking "is John" as intransitive, the feature does not allow "a boy" to go on as object, forces the recognition of "John" as subject, and forces the rule which utilizes the boundary symbol to give the correct analysis.

(c) The last feature, FSJ, marks the presence of the syntactic subject. It disambiguates such sentences as

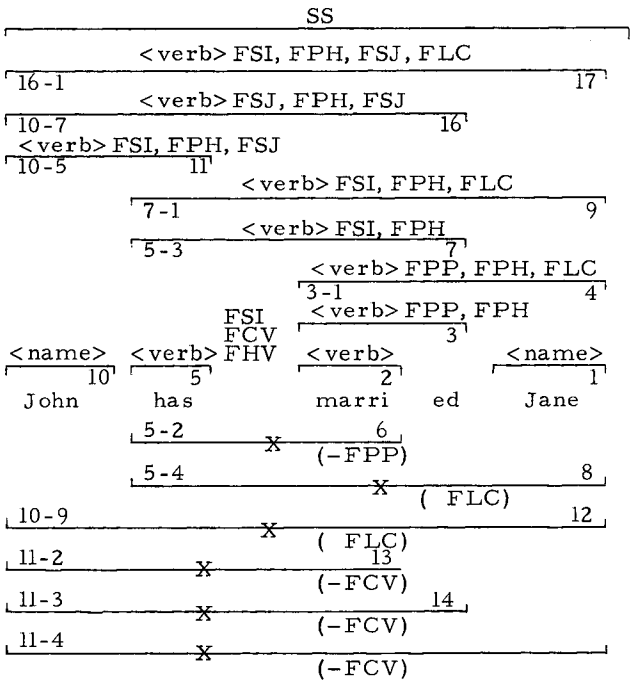
What people left?

(the second interpretation being "What did people leave?"). Only the first interpretation is allowed. The analysis is as follows (as usual, only relevant features are shown).



o (fails because of FSJ)

The function of features can best be observed by following the parsing of a sentence:



Let us follow the top parsings:

Parsing procedes from right to left. Thus, "Jane" is parsed into a <name> first. Next, "marri" parses into a <verb>. Irregular verbs, like this one, are treated in a simple manner: they are regularized by defining, i.e. "marri" is defined in terms of (as equal to) "marry." Then the past participle morpheme "ed" goes on in the usual manner. Next, "married" is combined with the object "Jane" by the rule that puts objects on verbs. The feature of 'left collecting' (FLC) is then set to indicate that the object has been put on. The next item available for parsing is "has" and it parses into a singular copula verb. It then combines with "married" into another <verb>. That, in turn, combines with the object "Jane," again setting the 'left collecting' feature.

Next, "John" is parsed into a <name>, and combines, as subject, with "has," thus setting the subject feature (FSJ). Parsings which fail due to feature checks are discussed below. The next good parsing to take place is "John" and "has married." It carries on the appropriate features. Finally, that verb phrase is combined with the object "Jane" to give the correct analysis of the sentence.

The exclusion of incorrect parsings is a more interesting process: combining "has" with "marri" is prevented by lack of the 'participle feature' (-FPP) on "marri." Combination of "has" with "married Jane" is prevented by the 'left collecting' (+FLC) feature, which does not allow left modification of the verb once the right modification (in this case the object "Jane") has been put on. The combination of the subject "John" with "has married Jane" is prevented by the same rule. "John has" "marri" is excluded by the fact that the verb phrase "John has" is marked as not including a copula verb (i.e. "has" is treated here as no longer being a copula verb, but as a full verb, as it would be in, e.g. "John has books"). "John has" + "married" is prevented for the same reason, and so is "John has" + "married Jane." This analysis is of particular interest. It expressly resolves the ambiguity of such sentences as "John has stolen watches." In our analysis, only the reading "(John has stolen) watches" is allowed, thus ruling out

* (John has) (stolen watches)

* (John has) (married sons)

The latter two readings are allowed, though, by different rules, which limited space did not permit discussing here.

As stated at the outset, ambiguity is a basic aspect of language. However, no sentence is an

island entire in itself, to paraphrase John Donne. [8] A sentence, or a phrase, lives in its context and derives its meaning therefrom. Computational analysis is deprived of many clues, such as intonation, that resolve much of phrasal or sentential ambiguity. It must, therefore, resort to other means, and, as a payoff, it externalizes linguistic structure, which is otherwise not easily seen. It thus gives us clues to the ways language works.

The question might arise as to why we seek to control ambiguity. The answer is that computing time has to be cut down: a researcher working with the computer to obtain information would not be happy if he had to sit at the console for three minutes after having asked a question and wait while all the ambiguous parsings are being ground out. Besides, who except linguists would be interested in having several ambiguous analyses of their sentences printed out. The detailed analysis of the following sentence supplies rather interesting statistics. In the analysis of the sentence:

"What was the college that was entered by John? "

the statistics are the following: 176 rules would have applied had no feature mechanism been used; 31 rules actually applied, of which 12 were aborted by features. Of the remaining 19 parsings, only 12 were actually needed to obtain the correct analysis of the sentence. These figures speak for themselves, or, rather for the need of a controlling mechanism on ambiguities.

The computer analysis of the last sentence also pointed out an ambiguity we had originally failed to observe. Notice these sentences:

"What was the college entered by John? "

"The college that was is no more. "

Since both of these are grammatical sentences, the sentence

What was the college that was entered by John?

produced the surprising ambiguity.

In connection with a recent conference on computational linguistics [9], the comment was made that "[computational linguists] may use the computer much as economists use gold, without actually touching it. " We "touch" the computer. After all, gold is good, hard currency.

IV. The Details of Implementation

Possibly the most important aspect of the feature method is its computing efficiency. This, of course, sharply depends upon its implementation. The details of our implementation of features as a method for controlling syntactic ambiguity reflects our particular computer environment and language processor. However, once this particular implementation is seen, it can be easily carried over to other environments.

The basic parsing algorithms for handling various classes of languages are continuing to be refined. Among the best are the Martin Kay parser (a modified version of which we use in REL) [4], the Earley parser [10] and the augmented network parser of Woods [11]. However, over and beyond the basic parsing algorithm, there are two ways parsing efficiency can be lowered; first, if the number of rules that are applicable to a sentence is very large, and second, if checking on special conditions requires special subroutines. *

A large number of rules makes the parsing tree grow to great size and the time expended by even the best parsing algorithms climb. Such is the case if the number of parts-of-speech proliferates or if syntactic ambiguity, especially multiple groupings of constituents, is not controlled. These difficulties can inevitably force the computation to exceed the capacity of high speed storage; and the costs of peripheral access are prohibitive, whether it be dictionary overflow, working storage overflow, or subroutine paging. These problems are exacerbated by the small high speed memory partitions, and roll in/roll out costs of time shared systems. It is in light of these common difficulties experienced in building operational systems that we believe our implementation of features is particularly interesting.

In order to convey the computational details, we refer to the particular data structure formats we use that are associated with the parsing procedures [Fig. 1].

The dictionary uses a list structure with two-word list elements. The main portion is a binary tree holding the rules of grammar and lex-

* Parsing time is often expressed as proportional to a power of the length of the sentence being parsed, i.e. as knP where n is the number of words in the sentence. Parsers are compared in terms of their respective values for p. Thus, good context-free parsers are said to be proportional to n^2. However, the factor of proportionality k, which depends sharply on the implementation, is often overlooked. When disk access time, 2 to 3 orders of magnitude greater than high speed memory time, and other similar factors come into play, and since n is small in normal applications, k can be the controlling variable.

Dictionary Formats:
(2 word, 8 byte, list element)

(1) typical dictionary element

byte 1:	part of speech
byte 2-4:	address of definition
byte 5-6:	match link
byte 7-8:	fail link

(2) definition element - mask part

byte 1:	code (indicating mask part)
byte 2-4:	link
byte 5-6:	feature on mask
byte 7-8:	feature off mask

(3) definition element - second part

byte 1:	code (indicating nature of definition, e.g. ambiguous)
byte 2-4:	address of actual definition, including associated semantic aspects
byte 5-8:	address of syntax conditions routine

Phrase Format (typical entry in parsing graph):
(3 word, 12 byte list elem.)

byte 1:	part of speech
byte 2:	(system use)
bytes 3-4:	features
bytes 5-8:	parsing graph link
bytes 9-12:	link to associated semantic information, e.g. pointers into the data base.

Figure 1

ical items. Figure 2 gives a stylized illustration of such a dictionary containing just three simple rules of grammar.

Words and phrases are held in the parsing graph in phrase format [Fig. 1] using a list structure with three-word list elements. A stylized illustration of such a graph for "Bob is John's son" is shown in Figure 3. In the case of the Kay parser, the parsing graph consists of a series of nodes interconnected by labeled, directed arcs, the labels being the consituent words and phrases.

Parsing proceeds by selecting a node in the graph, say Q_i, and attempting to match rules of grammar to words and phrases immediately to the right of Q_i. The Q_i's are successively selected starting from the right and working to the left. As this matching process proceeds, phrases successfully matched are accumulated in a pushdown stack. One use of this stack is as a back track trail, so that when one path through the graph has been completely analyzed, the process can back up a step and proceed along an alternate path. When a successful match of a rule of grammar to a path segment in the graph has been found, this pushdown stack constitutes the list of constituent phrases to which the rule has applied.

Features are held in the phrase format as a 16 bit field, each bit allocated to a given feature. The bit position corresponding to a given feature thus functions as a binary switch indicating whether the feature is on or off for the particular phrase. Thus in our implementation, we allow a maximum of 16 features to each part-of-speech. We have found this tight, but adequate.

The definition found in the dictionary has three parts: (a) a stack of feature masks, (b) a syntax conditions routine, (c) the phrase which will be inserted as a labeled arc in the parsing graph as a result of successful application of the rule. The stack of feature masks contains an on-mask and an off-mask corresponding to the constituent phrases. Each of these on/off masks is 16 bits long. If, for example, a given rule specifies that the possessive feature must be on and the determiner feature must be off for the first constituent (as in the rule <name> → <name>'s <relation>), then the bit corresponding to the possessive feature is one in the on-mask and the bit corresponding to the determiner feature is one in the off-mask.

It will be recalled that we do not have a separate lexicon and that individual characters are our terminal vocabulary. Since these characters are taken directly from the user's input sentence, they never carry features. They are recognized to be characters, so one need not carry feature masks for them in the feature mask stack, thus reducing to an acceptable minimum the memory requirements [see Fig. 1].

It now becomes a straightforward and efficient procedure to read down the stack of constituent phrases and the stack of on/off masks in tandem. For each phrase, the on-mask and off-mask are respectively "and"-ed to the features for the phrase, and the results of these two tests are used to accept or reject the rules applicability in the obvious way. Since the consituent stack must be accumulated as a back track trail in any case, this matching procedure is particularly efficient, as it must be since it is one of the tight inner loops of the algorithm.

Although most rules require no further checks on applicability than those indicated above, there are rules of a more clearly transformational type, where further checks may be desired. For this purpose, the definition may contain what we refer to as a "syntax conditions routine." The syntax conditions routine, if it exists for a given rule, is called after the indicated feature mask checks have been made and prior to acceptance and insertion of the new phrase in the parsing graph. This routine may, for example, go into the parsing graph to check on structural context; it may set aside and/or check subsidiary information; it could even rearrange the parsing graph or call the parser recursively. Our direct interest here is in its ability to make more extensive feature checks. How this may function can be seen from an example.

The conjunction rules insist on parallel construction for the conjoint phrases, e.g.:

Mary's (uncle and aunt)

* (Mary's uncle) and aunt

(Mary's uncle) and (John's aunt)

* Mary's (uncle and John's aunt)

Thus the condition that must be met for the application of the conjunction rule is that certain of the features must be either on or off in the same way for both constituents. This clearly cannot be checked by the feature mask tests as applied independently to each constituent phrase. Therefore, such feature checks are embodied in the syntax conditions routines. Since one usually cannot afford the space in high speed memory to keep all such subroutines resident, access to peripheral storage is required, and therefore, computing time is increased. It is our experience that only a small fraction of the rules require such more expensive tests. In our current grammar for REL English, only 29 rules out of a total of 239 have syntax condition routines.

	part of speech	def. add.	match link	fail link
d_0:	<relation>	0	d_1	d_6
d_1:	(blank)	0	d_2	0
d_2:	0	0	d_3	0
d_3:	F	0	d_4	0
d_4:	(blank)	0	d_5	0
d_5:	<name>	$definition_1$	0	0
d_6:	<name>	0	d_7	d_{11}
d_7:	'	0	d_8	0
d_8:	s	0	d_9	0
d_9:	(blank)	0	d_{10}	0
d_{10}:	<relation>	$definition_2$	0	0
d_{11}:	(blank)	0	d_{12}	0
d_{12}:	<rel. clause>	$definition_3$	0	0

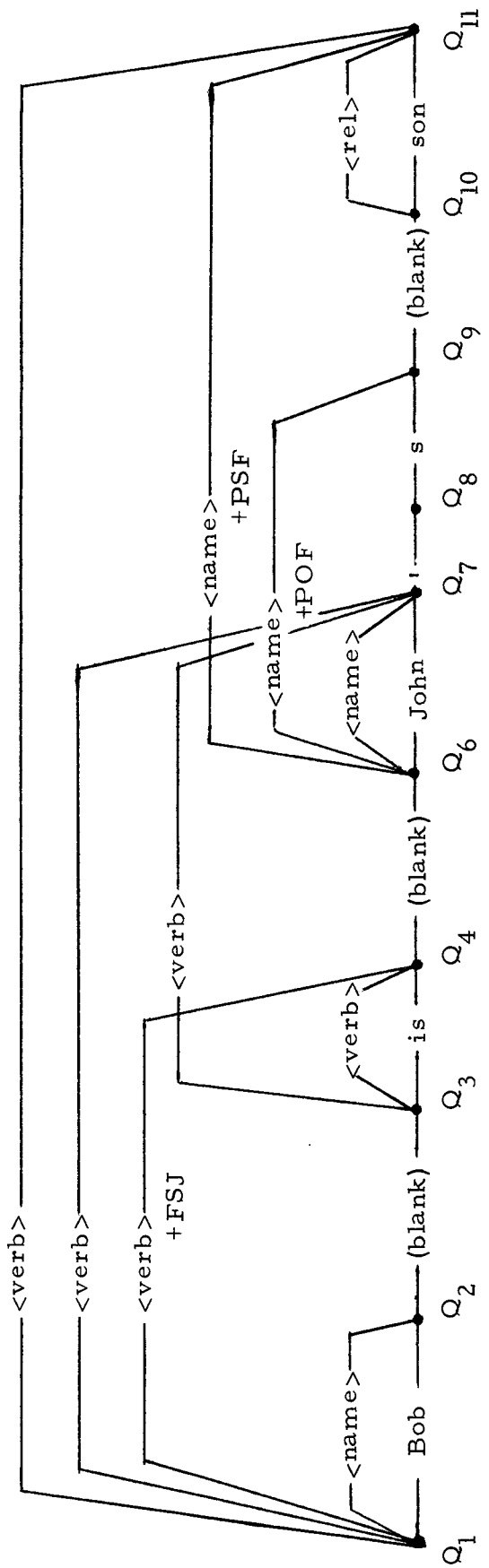
Illustration of main part of dictionary containing rules:

<name> → <relation> of <name>

<name> → <name>'s <relation>

<name> → <name> <rel. clause>

Figure 2



Typical Parsing Graph

Figure 3

REFERENCES

(1) THOMPSON, F. B.; LOCKEMANN, P. C.; DOSTERT, B. H.; et al. REL: a rapidly extensible language system. In: Association for Computing Machinery National Conference, 24th, San Francisco, Calif., August 1969. New York, 1969, p. 399-419.

(2) DOSTERT, B. H.; THOMPSON, F. B. A rapidly extensible language system: REL English. International Conference on Computational Linguistics, 3d, Stockholm, 1-4 September 1969. Preprint No. 35, 37 p.

(3) LOCKEMANN, P. C.; THOMPSON, F. B. A rapidly extensible language system: the REL language processor. International Conference on Computational Linguistics, 3d, Stockholm, 1-4 September 1969. Preprint No. 34, 31 p.

(4) KAY, MARTIN. Experiments with a powerful parser. Deuxième Conference Internationale sur le Traitement Automatique des Langues, Grenoble, 1967.

(5) FRIES, C. C. The Structure of English. Harcourt, Brace and World, New York, 1952, 304 p.

(6) HARMAN, G. H. Generative grammars without transformational rules: a defense of phrase structure. Language, 39, p. 597-616.

(7) CHOMSKY, NOAM. Aspects of the Theory of Syntax. MIT Press, Cambridge, Mass., 1965, 251 p.

(8) DONNE, JOHN. Devotions.

(9) KARLGREN, H. (Circulated letter) International Conference on Computational Linguistics, 3d, Stockholm, 1-4 September 1969.

(10) EARLEY, J. C. Generating a recognizer for a BNF grammar. Computer Science Center, Carnegie Institute of Technology, Pittsburgh, 1965.

(11) WOODS, W. A. Transition network grammars for natural language analysis. Communications of the ACM, 13 (1970) p. 591-606.