SOME FUNDAMENTALS OF ESL COMPOSITION ANALYSIS

© Gary Libben

A Thesis

in

The Centre for Teaching English

as a Second Language

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Arts at
Concordia University
Montreal, Quebec, Canada

January 1982

ABSTRACT

SOME FUNDAMENTALS OF ESL
COMPOSITION ANALYSIS

Gary Libben

This study describes a system for the preparation and computerized linguistic analysis of written material produced by learners of English as a second language. The system includes rules and conventions for characterizing student errors that make compositions amenable to computerized analysis, as well as a grammatical parser designed to meet the needs of applied linguists.

The parsing system, now ready for programming, uses a dictionary of five hundred and twenty-five entries and a set of ordered context sensitive rules. Multiple left-to-right analyses, executed by the computer, assign part of speech labels to words, mark and label phrase boundaries, and determine clausal structure. No semantic marking is done.

In addition to a description of the correction scheme and parser, the thesis discusses the fundamental role a system of this sort may play in any linguistic analysis of second language data.

ACKNOWLEDGEMENTS

Much of this work is the result of the efforts of many people involved in the ESL teaching and Learning Project directed by Patsy Lightbown and Bruce Barkman. I am very grateful for having been given the opportunity to work as a research assistant on the project and would like to express my indebtedness in particular to Catherine Faure who accomplished most expertly the enormous and painstaking task of creating the many text files we work with.

Bruce Barkman, both as project director and thesis supervisor, has been integral to the development of the system. As much gratitude as I could express here would not be nearly sufficient recompense for the many months of work and valuable insights that he has unselfishly contributed.

None of the computer work represented in this thesis would have been possible without the aid of Anne Barkman. Through many hours of consulting and emergency calls she has taught me the art of incisive problem-solving and 'thinking on the bottom line'. It should be noted, that she is in no way responsible for my continued ignorance.

I would also like to thank Tammi Rossman for her friendship and collaboration. Her interest in this research resulted in valuable conversation which has helped me to clarify and re-organize many of the key concepts in this thesis. I would also like to thank Mark Takefman for his characteristically skillful work in drawing the flow

charts and syntax diagrams that appear in this paper, and Walter Okshevsky for reading and commenting on the final drafts of the manuscript.

Finally, my wife Oda Lindner deserves a very special statement of gratitude. She has lived as I have through the many successes and failures of the system as it was being put together, endured printouts of the lexicon Scotch Taped to our kitchen wall, and spent much time editing most of the manuscript. It is to her that the thesis is dedicated.

TABLE OF CONTENTS

CHAPTER 1 Preliminaries

1.1 Background..... 1

1.2 Overview..... 2

CHAPTER 2 INTRODUCTION..... 4

CHAPTER 3 ERROR CORRECTION

3.0 Introduction..... 13

3.1 Word Error Correction..... 15

3.1.1 Substitution..... 16

3.1.2 Insertion & Deletion..... 16

3.1.3 Transposition..... 18

3.1.4 Summary of Error Correction Rules..... 19

3.2 Spelling Error Correction..... 25

3.3 Punctuation Errors..... 32

3.4 Capitalization Errors..... 34

3.5 Intrusions & Mazes: The Special Cases..... 36

3.6 Implications..... 40

CHAPTER 4 THE DICTIONARY & RULES

4.0 Motivation..... 46

4.1 The System for Grammatical Labelling: An Overview... 48

4.2 The Dictionary..... 50

4.2.1 Part of Speech Labels..... 50

4.2.1.1 Closed System Items That Occur in Verb Phrases..... 51

4.2.1.2 Closed Systems that Appear in Noun Phrases..... 58

4.2.2.0 The Dictionary of Open System Items..... 65

4.2.2.1 Open System Verbs..... 66

4.2.2.2 Open System Nouns..... 70

4.3 The Dictionary and Phrase Structure..... 73

4.4 The Dictionary - Independent Rules..... 80

4.4.1 A Note on Clauses..... 80

4.4.2 The Rules..... 81

CHAPTER 5 APPLICATIONS OF THE SYSTEM

5.1 Pre-Set Functions..... 91

5.2 User Definition..... 94

REFERENCES 96

LIST OF FIGURES

Figure		Page
1	Error Correction Bracket Syntax	30
2	Punctuation Error Bracket Syntax	33
3	The Counting Program	41
4	Program to Produce a Corrected Composition	44
5	Closed System Code for 'Have' and 'Do'	53
6	Closed System Code for 'Be'	54
7	Dictionary Codes for Modals	55
8	Dictionary Codes for Pronouns	59
9	Dictionary Codes for Non-Personal Pronouns	61
10	Dictionary Codes for Interrogative Pronouns	62
11	Code for Noun Limiters and Articles	63
12	Code for Open System Verbs	67
13	Open System Nouns	71
14	The Nominalization Rule	85
15	Infinitives	86
16	Verb Phrase Insert	87
17	Discontinuous Verb Phrases	88
18	The 'By' Rule	89
19	Changes in Part of Speech Labels	90

CHAPTER 1

Preliminaries

1.1 BACKGROUND

This paper describes a system for the treatment of written material produced by learners of English as a second language. In particular, it is an attempt to deal with the problem of analyzing a large number of ESL compositions which were collected by the ESL Teaching and Learning Project at Concordia University (Lightbown and Barkman 1978).

Because it was envisioned from the outset that the compositions obtained during the data collection phase of the project would serve as the corpus for a great many and varied analyses, the compositions were entered into the university's computer system. This in itself provided for great ease and flexibility in handling the data. It was decided by us, however, that if the resources of the computer were to be used to the project's greatest advantage, a principled system for dealing with second language data that would meet the specific needs of applied linguists was necessary. This is an outline of such a system.

The system presented here is the latest generation of a series. I made the first rudimentary proposals in October 1979 for the elaboration of programs developed by Anne Barkman and used in Barkman and Winer (1979).

Since that time I have worked closely with Bruce Barkman toward the development of a methodology that we believe would be of value to any computerized analysis of second language data.

At the time of writing, the system is being programmed in Pascal.

Although it has been successful under simulation tests using samples of both native and non-native speaker data, it is our opinion that further refinements will most probably be necessary once the system is programmed and in operation. This paper, then, must be considered as an interim report.

1.2 OVERVIEW

This paper is divided into four parts. The first deals with the motivation for a system of this sort. The reasons for which compositions have been used by researchers in applied linguistics will be outlined, as well as the specific tools now in use for these analyses. In addition, I will attempt to outline what in my estimation are the fundamentals of ESL composition analysis- that is the structure that is necesary before the analysis of compositions can be done.

Finally, the general question of what aid computerized analysis can be in the analysis of ESL compositions will be dealt with. I will briefly outline some other approaches to computer-aided linguistic

analysis and their application to the specific area of the analysis of ESL data.

In the second and third sections, the stucture of the proposed system will be presented. This includes the error correction scheme, the dictionary, the phrase marking system, and the power rules.

In the fourth section, I will describe the various uses of the system, including such pre-set functions as the statistical package hook-up and the generation of a learner probability grammar as well as the system's ability to accept user-defined variables.

CHAPTER 2

INTRODUCTION

At the higher levels of ESL instruction, compositions are frequently written by students. These compositions constitute a permanent record of a student's written English performance under more or less unconstrained circumstances.

Because of their availability and ease of collection, one would expect to see applied linguists, interested in the relevant aspects of second language acquisition, flocking to the field of composition analysis.

This is not generally the case. Although performance on a composition-writing task would seem to be an eminently valid measure of written English proficiency and allow for an analysis of the 'components' of second language acquisition (e.g. morphology and syntax), the field of composition analysis still suffers from some serious problems. Of these problems, the two most important are efficiency and reliability.

Even if one is analyzing compositions to arrive at some qualitative measure of global proficiency, as teachers often do, the processing of compositions requires a rather large amount of time.

For researchers in second language acquisition, the problems of efficiency are much greater. Typically, the questions that these

researchers address require close scrutiny of individual compositions and a large sample size. A complete error analysis of the type proposed by Corder (1967) or Richards (1974) is a mammoth undertaking for a significantly large number of compositions. As a result, most analyses restrict themselves to a small, well-defined area of investigation. Other more ambitious projects require so much time as to almost preclude the possibility of the study ever being replicated. If one accepts replicability to be one of the criteria of a successful experiment, then the use of an inefficient method diminishes the value of the study in the scientific community. I do not intend to claim, of course, that this would be the intention of any researcher, but only that the effect is an unfortunate consequence of the research tools employed.

The other great problem with composition analysis is that of reliability. By reliability, I mean the consistency of an analyst or grader over time and across compositions. It is this problem more than any other that has prevented the full use of compositions in second language research.

Probably all researchers who have attempted to analyze student compositions have had to deal with this problem. While we may assume that many have limited the reliability problem by setting up a set of criteria for, let us say, the characterization of an error as a grammatical error as opposed to a lexical error, these criteria are rarely noted in the literature. The problem has, however, been noted in Barkman and Winer (1979) who do report on the 'intermediate stages'

of their study. By far the most widely accepted method of dealing with reliability problems in this, and similiar types of research, is to have the same compositions analyzed independently by two researchers. This technique allows the results to be calculated after a test of reliability has been perfomed between analysts.

While this practice does much to make a particular analysis less 'subjective', it also has the effect of making an already inefficient method twice as inefficient.

Many researchers, recognizing the problems of reliability and efficiency but wishing to study 'free written expression' have employed the method of T- Unit analysis. The technique, used by Hunt (1965) to measure the sytactic development of first language learners, uses the terminable unit (T-Unit) as the basic unit of analysis. A T-Unit is considered to be a main clause and any subordinate clauses that are attached to it. Typically, the number of T-Units or the number of error-free T-Units in a composition are counted as well as the mean T-Unit length. Although researchers do find problems with this type of analysis (Malcolm, 1981), it has become popular primarily because the measures are relatively easy to calculate and allow for the processing of large sample sizes.

The most radical method of avoiding the pitfalls stated above is to avoid the analysis of compositions entirely. Tests are given to language learners precisely because they are more reliable than composition analysis and can be corrected easily and quickly. A test,

however, measures language proficiency under 'unnatural' circumstances if at all. To be sure, there are better and poorer tests, but all share that feature.

It is my opinion that when a researcher in the field of second language acquisition decides to administer a test rather than analyze compositions to investigate, let us say, the order of morpheme acquisition, he is trading off validity for reliability and efficiency. Given constraints on time and money, this trade-off is most often justified.

It is of little use to lament the underuse of composition analysis in applied linguistics unless one can propose a plan for making it a more attractive option to choose. This requires some method of dealing with the two problems discussed above.

In this paper, such a plan is proposed. The goals are quite modest, and remain well below the level of the analysis of discourse. The proposal deals with what I will call the fundamentals of ESL composition analysis.

In the last decade, the social and physical sciences have been transformed by a single technical device. That device is, of course, the computer. The use of the computer seems to be the ideal way of dealing with the problem of inefficiency. If one could simply feed in compositions and get out analyses, the computer would also take care of the problem of reliability. This unfortunately is not possible.

Because student compositions contain errors by definition, entirely computerized analyses would require that the computer be able to correct compositions. The requirement of relalive well-formedness is a feature that all attemts at processing natural language have in common. Although many word-processors claim to be able to correct spelling errors, inspection of the algorithms used reveal that the approach taken could not be adequate to the task of correcting student compositions. In brief, these word-processors compare the words in a text to a computer based dictionary. If the word is not found, then it is tagged as a probable spelling error. Clearly, aside from the fact that such a program requires an extremely large dictionary, the approach cannot be used to correct syntax because sentences are, for all practical purposes, unique language events.

Given then, that all student compositions contain errors, that the compositions must be corrected before the computer can analyze them, and that researchers differ in the way they correct compositions, we cannot rely on the computer to solve the reliability problem entirely.

Notice, however, that the computer requires only that the sentences in the compositions be grammatical. If we could limit the hand analysis to the task of making sentences well-formed, then we would be reducing the problem greatly.

Normally the variability in the hand analysis of compositions has at least three sources. The first is that researchers may vary in

their judgement of a sentence being grammatical or ungrammatical. The second is that researchers all may agree that a sentence is ungrammatical but vary in their judgement of what must be done to make it correct. The third is that researchers may vary in their classification of errors according to type and attribution.

All three decisions must be made by the researcher, who at least in the first case must rely on native speaker intuitions. The second decision, however, can be made with reference to a set of criteria that would ideally be both public and standard. After the first two decisions have been made, the computer can process the compositions. Although it cannot make the third decision, it can allow the decision to be uniformly applied. For example given the sentence: "He want milk." The researcher must decide that an error exists and that the sentence should be: "He wants milk."

The computer can now do a grammatical analysis of the sentence and retain the information that an error was corrected in the sentence. If the researcher later decides that non-inflected verbs are to be characterized as grammatical errors, then this instruction is given to the computer which scans all sentences, finds that the change made to this particular sentence by the researcher corrects for an non-inflected verb, and tags that error as a grammatical error.

The use of the computer in such analyses would greatly increase the efficiency of a study, by performing tabulations quickly and with uniformity and accuracy. A complete performance analysis based on a

sample of compositions, however, would still remain a time-consuming task.

The value of computerization in this area is not that it significantly reduces the time required to perform an initial analysis, but rather that it allows all subsequent analyses of the same compositions to be done in a small fraction of the time needed for the first.

The time consuming part of such a computerized analysis is the time required to enter the compositions into the computer and to correct them. Once the compositions are so 'prepared' a multitude of separate investigations may be carried out. Entered and corrected compositions provide the second language researcher with a re-usable 'data reservoir'.

So far, little has been said about the automated system that produces a grammatical characterization of the sentences in this reservoir. It is clear that some sort of natural language processor is required.

The field of natural language processing, previously thought to be beyond the scope of computers, has received much attention since the publication of "Understanding Natural Language" (Winograd, 1972). Although the system created by Winograd could understand and produce natural language, it depended on the computer's knowlege of the world and as such could only be effective in a very small universe of things

and relations (In fact, the size of a table top!).

Since then, many programs to process natural language have been created and are reviewed by Sager (1981). In that same work, Sager describes the rather impressive achievements of the Linguistic String Project (LSP) in this area. The parsers developed by these projects were created with a view toward a man-machine natural language interface. In other words, these parsers are intended to allow a human to communicate with a computer in English rather than Fortran or Pascal.

These programs are typically complicated and expensive to run. The reason for this is that the machine must not only be able to grammatically characterize natural language sentences, but must also be able to understand them and respond appropriately.

It is our opinion that for the purposes of composition analysis we require only a program that can economically produce a detailed grammatical characterization of sentences.

While programs that exclusively produce grammatical analyses of sentences do exist, (Cherry, 1980), they are mainly intended for use by writers and editors. Because these users would not require a very detailed grammatical characterization of words, these programs don't provide them.

For ESL composition analysis, detailed word characterization is

necessary. For this purpose we employ a dictionary design that tags words with a rather elaborate part of speech code. The dictionary, as discussed in chapter 4, incorporates features of different dictionary types.

Salton (1968) provides a detailed description of the different types of computer based dictionaries, their construction, manipulation, and evaluation. Formally, a computer based dictionary can consist of single words, phrases, or word segments (stems and affixes). A full-form dictionary is relatively easy to compile and reduces the possibility of mis-matching words. The segment dictionary has the general advantage of requiring less storage space. This is particularly true for languages that commonly inflect verbs (Pacak, 1974).

In our system, stems are related to the appropriate affixes by rule. Full forms are only listed for irregular nouns and verbs. The dictionary thus combines the features of segment and full-form dictionaries.

CHAPTER 3

ERROR CORRECTION

3.0 Introduction

Our ability to make profitable use of the computer rests, to a large extent, on the regularity of the input we can provide. By regularity, I mean two things: 1) the individual symbols must have a unique and unambiguous meaning such that combinations of them also have unique and unambiguous meanings, and 2) the input must be well-formed such that implicational rules can be specified that will limit the information we must explicitly provide. In other words if we are to use the computer in further analyses of the structure of student compositions, these compositions must be first corrected so that they have a structure the computer can analyze.

This issue becomes particularly important in the following section where the structure of the dictionary is discussed. Effecting a dictionary match-up of words in a text requires, for example, that all these words are correctly spelled. In addition, we will want to disambiguate words with respect to part of speech. Consider the following example:

(1a) The pictures of Martian canals were shot in the dark.

(1b) The pictures of Martian canals were a shot in the dark.

The different interpretations of these sentences are based on our assignment of different part of speech values to the word 'shot' in each sentence. While our assignment of one or the other value to

'shot' might have been based on the meaning of the preceding or following sentences in the text as well as the sytactic structure of the rest of the sentence, the computer would have only the latter cue to work with. Hence, if we are relying on the computer to assign a part of speech value to 'shot', the erroneous omission of the indefinite article or any error in another part of the sentence might preclude the computer's chances of making a correct assignment.

It follows then that the correction of errors has a two-fold function. It allows for subsequent analysis in general and for the subsequent analysis of errors in particular.

In a system of this sort, errors must be corrected even if the investigator has no intention whatsoever of analysing those errors. I emphasize that what follows is not an analysis of errors, but rather is simply a system for their correction.

3.1 Word Error Correction

Basic to the correction of errors is the retention of what was produced and a statement of what should have been produced. If a student wrote:

(2a) He go to the store.

the least we want to be able to state is that this example contains an error and that this error involves the word 'go' which should have been 'goes' or 'went' etc. depending on the context.

In calling these errors "word errors", I mean only that they involve words rather than other things such as punctuation marks. In this sense, both lexical and grammatical errors are word errors.

The conventions that we have adopted for the correction of compositions reflect our decision to allow the context of a word or sentence to be the primary determinant of correctness or incorrectness. In cases, however, where the the role of the semantic context is not clear, we have established a hierarchical organization of rules that apply, all other things being equal.

I will proceed first with a description of the conventions for the correction of word errors assuming no over-riding contexual constraints, and then discuss how they interact with the 'meaning' in their linguistic environment.

3.1.1 Substitution

Assuming then that the proper correction of example (2a) involves the substitution of the word 'goes' for 'go' this would then be indicated by editing the utterance such that it looks like this:

(2b) He (goes<go) to the store.

where '<' means: "has been corrected from"

Example (3) is simiiar but involves a noun which must be corrected from (3a) to (3b).

(3a) kill the mans

(3b) kill the (men<mans)

This type of correction, which we will call substitution makes no special claims about the relatedness of the words involved in the correction. The following correction is perfectly acceptable.

(4a) He has a drive

(4b) He has a (car<drive)

In substitutions we are simply stating that word (x) occupying position (n) must be changed to word (y) occupying that same position (n).

3.1.2 Insertion and Deletion

Other simple correction types are insetion and deletion. When insertion is required such as in example (5a), this is signified with

the notation used in (5b).

(5a) Ron dead.

(5b) Ron (is+) dead.

where '+' means: "This word has been inserted"

The case of a sentence containing multiple errors of different types is given in (6). Here the errors require correction by both substitution and insertion.

(6a) He want find Ron.

(6b) He (wants<want) (to+) find Ron.

The preceding example illustrates one of the basic priciples in this system of error bracketing: Namely that the basic unit of correction is the word. None of the correction symbols apply to affixes. The third correction type, deletion, therefore always refers to deletion of an entire lexical unit as in example (7).

(7a) I don't know me

(7b) I don't know(-me)

where '-' means: "The following word has been deleted".

Insertion and deletion differ not only in the symbols that represent them but also in the position of those symbols within the correction brackets. As will become apparent in the discussion of other error types, the value of this format is that it allows the machine to distinguish between correct and incorrect word simply by noting whether they follow or are preceded by a correction symbol.

3.1.3 Transposition

Transposition correction or correction of word order does not necessarily require the use of a new symbol. It could also be handled using the insertion and deletion conventions, since every transposition, at least at the mechanical level, involves a deletion of a word in one position and an insertion of it in another position. This approach to the problem, however, is not adequate since it would result in a transposition error appearing to be two errors of unrelated types. In addition, the insertion-deletion approach does not allow us, in a sentence containing multiple errors to determine which of these are related and which are not. A possible solution to this problem might be to co-index insertion-deletion pairs. This solution, however, would require further elaboration of the programs, with no gain in usefulness or information, so it seems better to treat word order errors independently.

The convention that we have adopted and whose use is illustrated in example (8) indicates both where the word was found and where it should have been found.

(8a) Because he the old man saw

(8b) Because he (saw=) the old man (=saw)

where '=' means either: "Has been moved to" or "has been moved from", depending on its position within the correction bracket.

There are two special subtypes of transposition errors. The first involves a sentence which can only be corrected through the use

of multiple transpositions. In this case some sort of co-indexing is necessary. We have handled the problem by allowing for multiple consecutive occurrences of the transposition symbol within an error correction bracket. Thus the symbol indicates a transposition error and at the same time its number ties it to the other transposition symbol in a different bracket but in the same sentence. The corrections in (9) exemplify these kinds of sentences and how they are handled in the system.

(9a) Because he the old man seen had

(9b) Because he (had=) (seen==) the old man(==seen) (=had)

where individual transpositions are denoted by '=', '==', etc.

The second type of special case is one in which a word in a sentence is not only in the wrong place but also in the wrong form. We have indicated this by allowing the transposition symbol and the substitution symbol to combine forming a single expression. The coding convention for this is given in example (10).

(10a) Yesterday, the child ran away because he the old man see.

(10b) because he (saw=<) the old man (=<see)

where '=<' indicates that both substitution and transposition have been performed.

3.1.4 Summary of Error Correction Rules

To summarize, the possible corrections are:

<	substitution
+	insertion
-	deletion
=	transposition
=<	transposition and substitution

The rules for their use are:

1. They apply to units neither smaller nor greater than the word.
2. The error is always preceded by a correction symbol.
3. Any word supplied by the coder to make an utterance correct is followed by a correction symbol.

The following corrections therefore are not possible:

EXAMPLE	VIOLATES RULE	SHOULD BE
(-have been)	1	(-have) (-been)
(was<have been)	1	(-have) (was<been)
(have been<was)	1	(have+) (been<was)
(him-)	2	(-him)
(+him)	3	(him+)

The three usage rules satisfy the conditions stated above for the profitable use of computerization. The constrained use of these symbols allows us to assign a unique interpretation to language events in the compositions. The constraints on well-formedness of error

correction statements allow us to detect coding errors and to give the machine further instructions on the basis of our knowlege of the correction bracket structure.

Although they do impose a certain rigidity of form upon the corrections, the rules do not specify a strategy for choosing between correction possibilities. For example, the sentence in (11a) could be corrected to (11b), (11c), or (11d).

(11a) She is come

(11b) She (-is) (comes<come)

(11c) She is (coming<come)

(11d) She (has<is) come

Similiarly, in the following example there are two transpositions possible. Sentence (12a) could be corrected to (12b) or (12c).

(12a) She has a dress blue

(12b) She has a (blue=) dress (=blue)

(12c) She has a (=dress) blue (dress=)

It is usually the case that the context of the sentence determines which of the alternative corrections is most appropriate. In some cases, however, there exist two or more alternative corrections that are appropriate to the context, or the context does not provide enough information to allow for a choice. In these cases we want to attempt to minimize secondary variance by making explicit the choices consistently made in the absence of unambiguous contexual cues and by stating the principles that underlie them. This seems to

be the best approach for increasing reliability across coders and across studies.

The following set of ordered rules has been formulated as guidelines for the correction of sentences in vacuo.

The rules reflect the basic principle of the system as a whole, namely to make the analysis of compositions possible while sacrificing as little as possible of their original structure. The rules and their ordering are an operationalization of that principle.

1. If a number of ways to correct an sentence are possible, choose the way that requires the least number of corrections.

2. If more than one possibility requiring the same number of corrections remain, choose transposition over substitution, substitution over insertion, and insertion over deletion.

3. If more than one possibility requiring the same number of corrections of the same type remains, choose to correct an adjective or adverb rather than a verb or noun.

4. If there still remains more than one possibility, choose to correct the word closest to the end of the sentence.

Contextual constraints have priority over any of these rules and interact with them as in the following example of an exchange between

a teacher and student in example (13a).

Teacher: Are your parents coming to parent night?

(13a) Student: My father come.

In accordance with the rules, we would be obliged to correct the utterance in (13a) to (13b). But this solution would violate a contexual constraint. The question that the teacher asked clearly requires an answer that contains an expression of future time. Our first correction is therefore vetoed. The two remaining possibilities are (13c) and (13d). Because both of these latter alternatives are fine with respect to the context, we must choose the correction in (13d) in accordance with rule 1.

(13b) My father (comes<come)

(13c) My father (is+) (coming<come)

(13d) My father (will+) come

Example (12) which was presented above, now poses no problem. In accordance with rule 3, the adjective 'blue' is moved around the noun 'dress'. This is shown in (12b).

(12b) She has a (blue=) dress (=blue)

The notation that has been presented in this section and the rules for the use of this notation form the foundation of the system as a whole. They reflect the assumptions that we have made about the nature of ESL composition analysis, the appropriate unit of correction, and the balance that should exist between evaluator intuitions and formal correction rules. The functioning of the system

will be necessarily tied to the advantages and limitations of these assumptions.

3.2 Spelling Error Correction

In correcting spelling errors our goal is again to preserve information about exactly what was produced while making possible the analyses which require that words be correctly spelled. As was mentioned above, the dictionary match-up of words is possible only upon this assumption. The reason for this lies in the nature of computer operation. For example, in a normal computerized search through compositions for the word 'cigarette', the machine divides the universe of words into those which are exactly 'cigarette' and those which are not. This has the effect of making the word 'sigarette' no more similiar to 'cigarette than any other word is. The machine's inability to abstract away from detail in the way that is so natural for humans, requires that in any computerized analysis of compositions, spelling errors must be corrected.

Now that we have stated the case for the correction of spelling errors even if the researcher does not intend to analyze those errors, we come to the question of what a spelling error looks like.

It can be argued that all errors are basically spelling errors in that all errors can be reduced to errors of letter deletion, addition and substitution.

This is a somewhat trivial point which ignores the qualitative change that occurs when meaningless letters combine to form units of meaning. No one would seriously want to argue that the word 'bird' is

the word 'horse' with a different spelling. The point becomes less trivial, however, when one considers example (14).

(14) It is (his<him) book.

In such a case it is usually our assumptions about the writer that form the basis of our decision. If you, the reader, were to find an error like (14) in the text of this paper, you would probably judge it to be a typographical error. If it were found in the text of an ESL composition however, your judgement would probably be not nearly as automatic. But a convention which states that this error is a grammatical error for ESL beginners and is a spelling error for advanced students would be of little use. The terms beginner and advanced are difficult to operationalize and we would want to have a correction scheme that allows for evaluator-blind experiments.

A different approach to the problem is to not allow any error that results in the creation of another word to be counted as a spelling error. This rule errs, however, because it is too conservative and far too sweeping. Its application would result in the coding of the errors in examples (15), (16), and (17) as word errors.

(15) Sweet (wine<mine) gives me a headache.

(16) I need a piece of (rope<grope).

(17) He (bet<best) me ten dollars.

The errors in these examples must be spelling errors. We cannot form any hypotheses that might account for their being anything but

spelling errors. Furthermore, in example (16) we suspect that the new word that was erroneously created might not even be recognized as being an English word by the student. The non-word rule is therefore not adequate for the coding of spelling errors.

We want to be very careful in our characterization of spelling errors precisely because they are usually ignored in second language research. If for example, a researcher performs an error analysis of compositions deciding not to count spelling errors, the implementation of a rule that assigns the label 'spelling error' to all ambiguous cases would result in an overestimation of proficiency and reduce the population of analyzable errors. On the other hand, in a situation where students are typing compositions on a terminal, the implementation of the non-word rule might lead to conclusions of low English proficiency when in fact only typing proficiency is low.

We would like to be able to state that all errors resulting in non-words are spelling errors and that some errors resulting in well-formed English words can also be characterized as spelling errors. I am excluding here errors that require the deletion or insertion of whole words. e.g (-me), (I+).

In deciding upon a rule for the correction of spelling errors that does not result in a systematic overestimation or underestimation of student proficiency we have asked the following question: how do the errors in examples (15) (16) and (17) differ from the following errors which we would not be inclined to code as spelling errors?

(18) He (learns<learnt) quickly.

(19) He (bummed<bumped) a cigarette from me.

(20) (has<had) she ever been there?

(21) She (eats<eat) fish on Fridays.

Examples (18) to (21) differ from the others in two ways: they result in the creation or the deletion of morphemes, and the erroneous words are of the same type as the correct ones. The rules that we propose for the treatment of spelling errors attempt to capture these differences.

1. All non-words are considered to be spelling errors.
2. All errors that are words and can be corrected without deleting or inserting morphemes are spelling errors unless the error and its correction have the same part of speech label.

These rules would result in the coding of examples (15) to (17) as spelling errors and examples (18) to (21) as word errors. The rules are context sensitive to the extent that part of speech labels are context sensitive. In examples (22a) and (22b) the first error would be characterized as a word error and the second would be characterized as a spelling error.

(22a) He (set<sat) the book on the table

(22b) He won the first (set<sat)

The different characterizations result from the shift of 'set' from verb to noun in the two contexts.

So far I have been using the substitution symbol in coding

spelling errors as well as word errors. The symbol that will be used henceforth to code spelling errors is the '*' which functions as the '<' does. Because the basic unit of correction within correction brackets is the word, the '*' is the only symbol that is necessary for spelling errors. The use of any of the other symbols thus far introduced would require a statement of the use of a unit smaller than the word if they were applied to spelling errors. We can now recode example (22).

(22a) He (set<sat) the book on the table

(22b) He won the first (set*sat)

Within error correction brackets, spelling errors may occur alone or in combination with other corrections. In such cases, the spelling error is always coded last. Consider the following example:

(23a) I want hte train for Christmas

(23b) I want (a<the*hte) train for Christmas

The symbol '*' then always occurs inside the error correction bracket and may co-occur with all symbols except '+' (for obvious reasons) and the transposition symbol when that symbol '=' follows a word. The reason for this is that for spelling errors in transposed words, we need only indicate this in one of the pairs because all transpositions are inherently co-indexed.

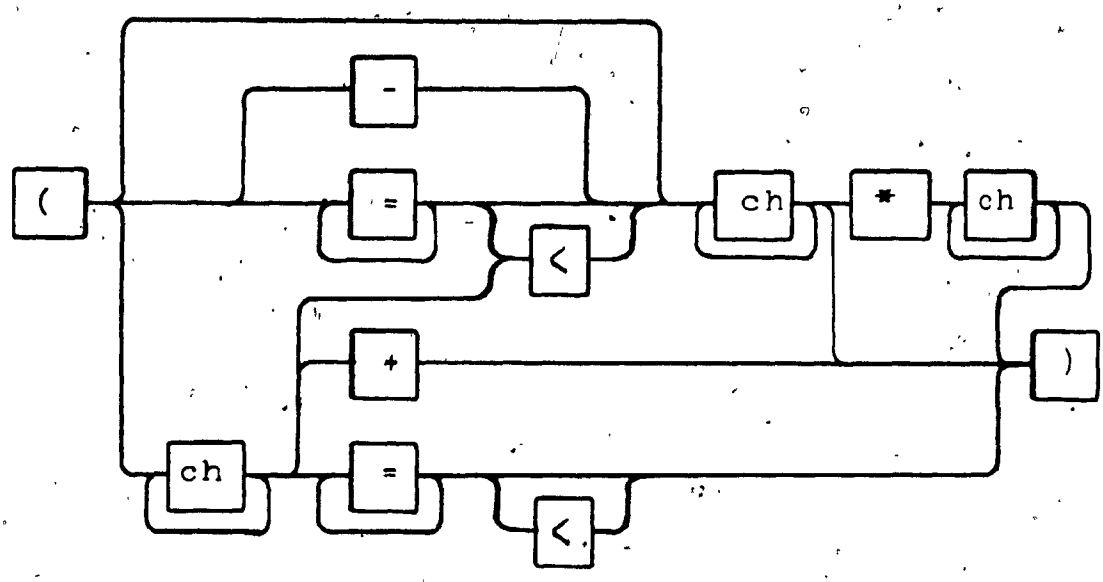
To summarize, the word correction brackets may identify the specific ways in which a word must be changed to be correct and may also signify that a spelling change is required.

The rigidity of form imposed by the rules and conventions that have been presented above allows the computer to predict what it will find if certain conditions hold true.

In figure 1 the syntax of the error correction bracket is diagrammed. This represents the possible sequences of characters and symbols that the computer can expect to find.

Figure 1

Error Correction Bracket Syntax



Before any analyses of data are performed, a coder-error detection program is run by the system which scans all the brackets.

The program aborts further analyses and sends a message plus address when it finds an impossible sequence inside the error correction bracket. (e.g. unbalanced brackets, blanks, the absence of a correction symbol etc.)

This program partially controls for the errors that a coder is bound to make when coding a large number of compositions. There are, however, coder-errors which this program will not detect such as the misspelling of words within the brackets. The elimination of these requires another check at a further stage of analysis.

3.3 Punctuation Errors

The correction of punctuation errors is considerably simpler than the correction of either word or spelling errors because of the limited number of elements involed. We will include as punctuation marks only periods(.), commas(,), colons(:), semicolons(;), exclamation marks(!), quotation marks(" or') and question marks(?).

We also want to restrict the ways in which punctuation errors can be corrected. Because of their limited number the movement of puntuation marks by the coder should not be allowed. Therefore the symbol '=' will not be used in the correction of these errors. The bracketing of punctuation errors is also different. They are enclosed in curly brackets as examples (24) and (25) illustrate.

(24a) Snakes which I hate are reptiles

(24b) Snakes{,+} which I hate{,+} are reptiles{.+}

(25a) Only some, people are nice;

(25b) Only some{-,} people are nice{.<;}

This notation allows the computer to recognize puntuation errors by their distinct bracketing while retaining the general internal form of the word error brackets. Punctuation may now be used in the analysis of sentences and clause types.

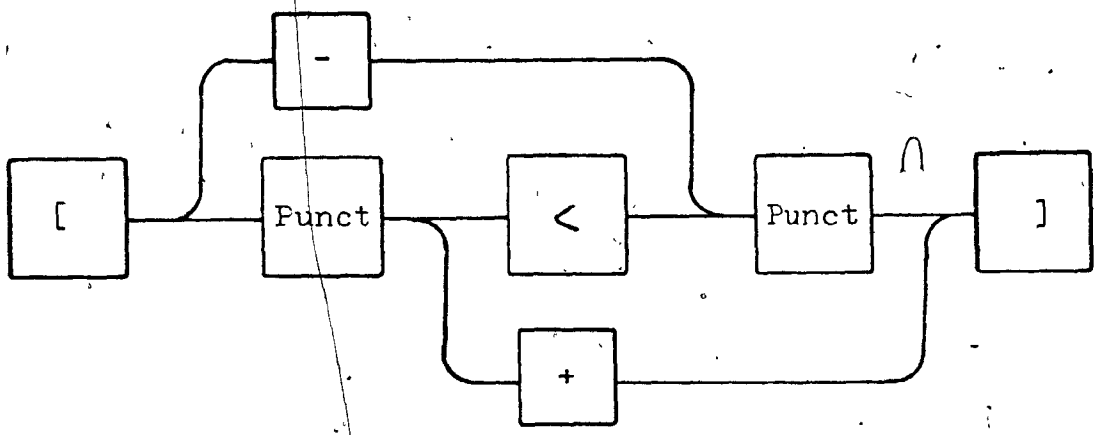
The correction of punctuation is subordinate to word correction and should be done after words and spelling have already been corrected. The purpose of this is not to allow punctuation changes to

serve as grounds for correcting words.

The coder-error detection program that is used to check the word error brackets also extends to punctuation error brackets. Figure 2 is a description of the considerably simpler syntax of punctuation error brackets.

Figure 2

Punctuation Error Bracket Syntax



3.4 Capitalization Errors

In this section I would like to make the case for not including capitalization errors in the correction scheme.

If we do not include capitalization, we eliminate the necessity for using both lower and upper case letters in the programing of the system. This paper has been written using the computer and a word processing program. I am using both upper and lower case letters for the sake of readability, but it would have been considerably less expensive to use upper case letters only. The reason for this is that all lower case letters are internally represented by the machine as a letter preceded by a '^' The word 'Horse' would actually look like this: H^O^R^S^E This means that lower case letters take up twice as much storage space as upper case letters. In addition, assuming the dictionary that will be presented in the following chapter will have only upper case entries, the '^' will have to be removed before a word is looked up and then replaced again.

Finally, another problem with the correction of capitalization errors is their interaction with punctuation errors. We would not like to have capitalization errors created every time a period (.) is inserted or deleted by the coder.

These considerations have lead us to question what information would be lost by ignoring the differences between upper and lower case letters.

Capitalization is not the unique marker of sentence onset as this can be adequately predicted by the preceding punctuation mark or its absence. We are left then with the identification of proper nouns. Capitalization does identify proper nouns, but not adequately; for we do not know whether the word that begins a sentence is a proper noun or not. If we are interested in having a formal identifier of proper nouns we would do better to create one that does the job adequately. This is exactly what we have done. In this system the coder places the symbol '@' before all proper nouns in a composition. Capitalizations errors are not corrected and the system operates with upper case letters only.

3.5 Intrusions and Mazes: The Special Cases

In our discussion of the error correction system two special cases have not yet been considered. The first is the intrusion.

An intrusion is defined here as a word or sequence of words that belongs to another language and not to English. It is not uncommon for a student to include a mother tongue word in an English composition, falsely assuming it to be a cognate. Another possibility is that the student includes this word knowing that it cannot be used in English, in order to indicate to the evaluator that he does not know the required English word. Here the student's assumption is of course that the evaluator understands the student's mother tongue. In this latter case, the student might be inclined to include whole sentences in his mother tongue. Consider example (26a):

(26a) It's like a citron. Je ne sais pas le mot en anglais.

In the case of 'citron', the system does not have a mechanism for proper label assignment. We wouldn't want it to be counted as a spelling error, but allowing it to be considered to be a substitution error would cause the system to fail. The reason for this is that all words, excluding misspellings, will be assigned part of speech labels by the dictionary rules. The word 'citron' would of course not be found in the dictionary and therefore in the absence of any special instructions to the contrary, the machine would assign it the default value of regular noun.

We have attempted to handle this problem without changes in the correction bracket structure. The dictionary includes a pre-lexicon which is consulted before the normal part of speech rules are applied. In this pre-lexicon the codeword 'frc' is read signifying an intrusion of a French word. (The code would of course be different for other languages.) Because we only require that the machine be able to retrieve the word 'citron' if necessary, we can use the correction scheme in example (26b).

(26b) It looked like a (lemon<frc*citron).

where '<frc*' means: "This word has been substituted for a French word whose exact spelling is: 'citron'.

The only symbols that are permitted by the system to co-occur with 'frc' are '<', '-', and '=<'. In the case of the intrusion of an entire sentence only deletions '-' are permitted.

In addition to the restriction on the co-occurrence there is another way in which the treatment of multiword intrusions differs from that of single word intrusions and corrections in general. Coding whole sentence intrusions word by word would be rather tedious. For this reason, the keyword 'frc' also lifts the constraint that there be no blanks within correction brackets. As has been stated above, under normal circumstances this would force a coder-error message.

Sentence (26a) can now be conveniently coded in the following manner:

(26c) (-frc*Je ne sais pas le mot en anglais)

The use of this device for correcting intrusions does not entail any loss of information. The exact French words can be retrieved using the general mechanism for the retrieval of misspellings, and the number of French words can be calculated in the following way:

N of French intrusions = N of 'frc'
N of French words = N of 'frc' + N of blanks

The problem of mazes is somewhat simpler. A maze is defined as a word that can be recognized as nothing other than a string of characters. An example of a maze is found in example (27a).

(27a) He went hrze the store

We suspect that this is a spelling error but have no idea what word could have been intended. To solve this coding problem we can employ the same mechanism that was used for the handling of intrusions. In this case, the codeword in the pre-lexicon is 'mze'. The sentence is corrected in (27b).

(27b) He went (to<mze*hrze) the store

The co-occurrence contraints for 'mze' are the same as those for single word intrusions.

The use of a codeword for mazes rather than the simple spelling error symbol allows us to capture the difference between mazes and other errors. The difference is that mazes result in a loss of sentence structure. This interferes with the application of correction rules 2 and 3 and forces the coder to choose an appropriate correction on the basis of a weaker context. In other words, the use of the 'mze' keyword is a formal indicator of the reduced confidence we must have in the appropriateness of that particular correction.

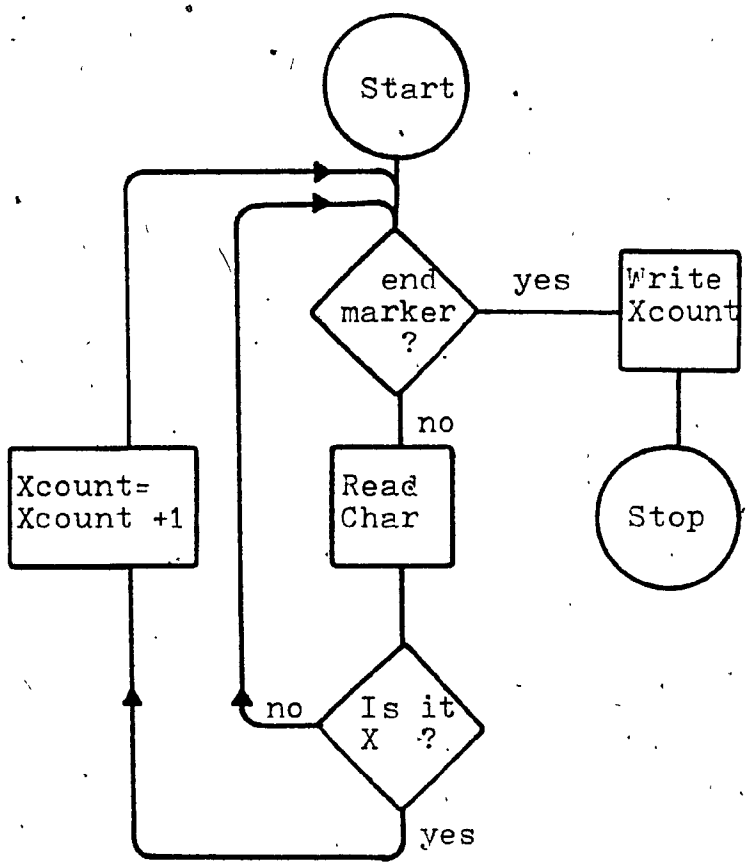
3.6 Implications

In this section I would like to consider what control over the data has been gained through the constraints on error correction. The operations that are possible with this degree of structure are still quite elementary, but represent a large step forward in the facilitation of automated analysis.

The computer now has the ability to recognize a certain number of concepts that we have defined. The symbol '{' for example, is now the operational definition of 'punctuation error'. Similiarly, a word is defined as a string of alphabetic characters that is surrounded by non-alphabetic characters. The well-defined concepts that we have can now be counted over a large number of compositions or a single composition. The general program for counting is presented in figure 3.

Figure 3

The Counting Program



This is a generalized program in the sense that the test items are variable. In this particular program the two variables are 'end marker' and 'X'. The end marker can either be 'end of record' which would presumably separate compositions or 'end of file' which marks the end of all the text. The value that we choose for this variable allows us to choose between an analysis of all compositions or a single one. The variable 'X' is the operational definition of the thing that we want to count. For punctuation errors 'X' would be defined as '[' and the final value of 'Xcount' would be '[count' or the total number of punctuation errors.

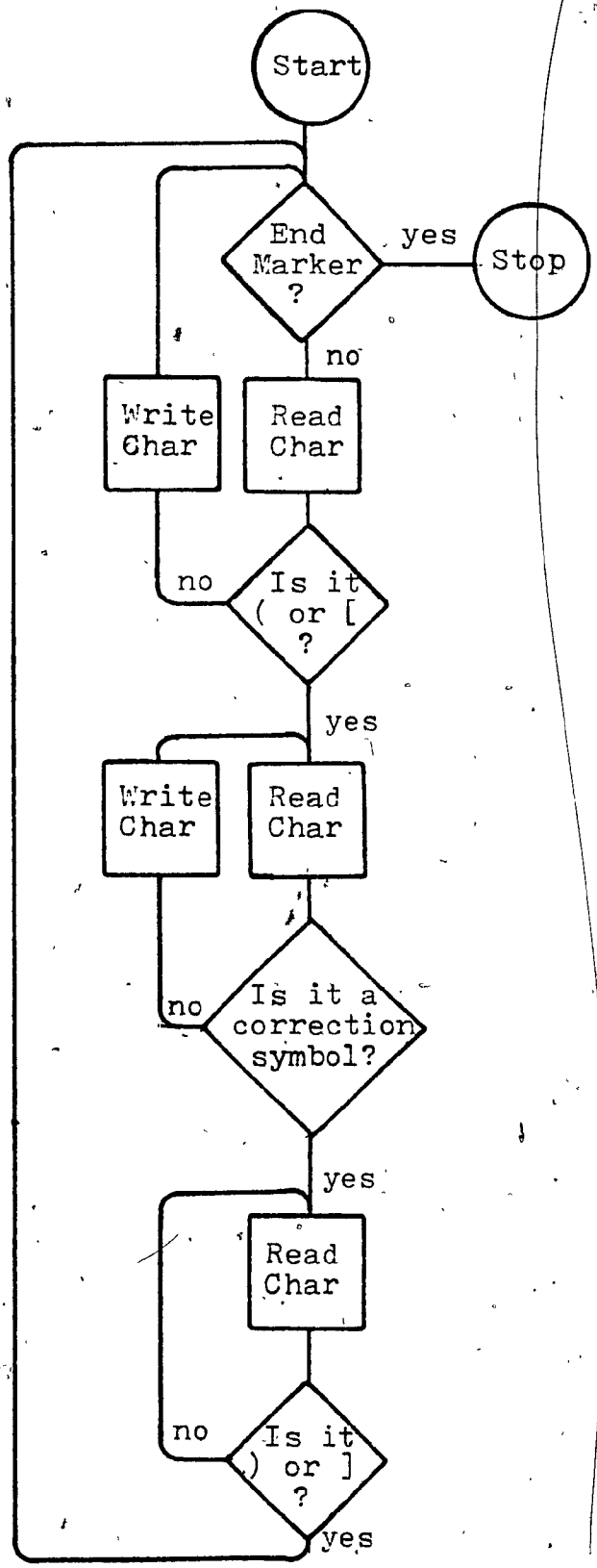
The variable 'X' may also be a logical expression. If for example, we wished to know the total number of true spelling errors, a count of '*' would not be sufficient. We would have to define 'X' as: {* and not (frc or mze)}.

Some analyses may require more than one counter running either in series or simultaneously. This is easily handled. A more interesting use of a counter is one in which a subset of a group of compositions is to be analyzed. In this case the end marker would be set to 'end of record' and another counter would be associated with that test condition. This second counter would stop the program and print the results when 'end of record', had been encountered a certain prespecified number of times. This would effectively give data for, let us say, the first fifty compositions in the file. In this way, several groups may be defined in an experiment.

The structure that we have given can also be used to produce different versions of those compositions. We might imagine that in some pedagogical application of this system, it would be desirable to produce for the student both the original version of his compositions and the corrected version. Very little manipulation is required to accomplish this because inherent in the error correction bracket structure is a statement of where, for all corrections, wrong words are to be found and where correct words are to be found. The program that would produce a perfect corrected version of a student composition is presented in figure 4.

Figure 4

Program to Produce a Corrected Composition



Using this same principle we could produce semi-original compositions, with for example, only spelling errors corrected.

Some of the things that can be done with compositions at this level of structuring appear to be rather useful like having the ability to produce for the student a cloze test of his own composition where specific error types are blanked out. At the outset, however, this system was intended to provide for the needs of researchers in applied linguistics. Presumably the type of analyses that would be carried out would go beyond the recognition of a word as a correct word or an incorrect word. We also need the ability to state what kind of words are correct or what kinds of words are incorrect.

The following chapter describes the part of speech assignment operations that allow for these more detailed analyses.

CHAPTER 4

The Dictionary and Rules

4.0 Motivation

The preceding chapter concluded with a statement of the need for grammatical labeling of words and groups of words. This is necessary before any serious analysis of compositions can be done. The present chapter illustrates how this task can be accomplished by machine.

It is rather odd that my argument for the implementation of computers in the analysis of ESL compositions should be based on the need for the grammatical characterization of compositions. Researchers often avoid computerized analysis precisely because they do not wish to be limited to counting only words, punctuation marks, and sentence length. Researchers opt for hand analysis because the computer cannot provide them with the syntactic characterizations that they need.

The result is not generally a happy one. Because the hand assignment of part of speech labels, and clause boundaries for a significantly large number of compositions might require years, the wise researcher limits himself to the analysis of specific units, and codes only those. This means that the researcher looking at students' accuracy on certain verb inflections must find and code these for, let us say, one hundred compositions. It also means that another researcher looking at the same compositions for accuracy on other verb

inflections cannot benefit from the previous work and must go through the entire set of compositions all over again.

This is not a particularily efficient procedure, and places a severe limit on the quantity and quality of research in the field.

That automated syntactic labeling would be of great value need not be restated. With automated grammatical labeling, we believe that research which formerly required weeks to do could be done literally in minutes.

Automation, however, is only possible under certain circumstances. Those conditions are that sentences obey the structural and orthographic regularities of English. The preparation of ESL compositions according to the conventions and rules given in chapter 2 would satisfy these conditions.

Assuming then, that compositions have been corrected in this manner, we can proceed to describe the program for grammatical labeling.

4.1 The System for Grammatical Labeling: An Overview

Our goals in this project were, in a sense, quite limited. We aimed at creating a system that would correctly do the following:

1) Assign part of speech labels to words.

2) Mark the boundaries of phrases and label them.

3) Mark and label the boundaries of clauses.

The system has two major components: a dictionary and a set of ordered rules.

Basically, the system operates by scanning a sentence from left to right. Each word that it encounters is looked up in the dictionary. If that word is found, then all the characteristics that are associated with the word and the word itself are put back into the sentence. In this manner, the entire composition or set of compositions is scanned and modified.

After this has been done, the system begins again to scan the now partially coded sentences. Aided by the new information provided by the dictionary assignments, it assigns more labels by applying rules of word co-occurrence and order. These also are inserted in the text and provide input for the phrase and clause boundary rules which end the process. If the machine is unable to assign any unit with a label, or if it has little confidence in a particular assignment, the sentence containing the unit is printed out, and the computer waits for a correction from the person running the program.

In the following sections of this chapter the structure and function of each of the systems components will be examined.

4.2 The Dictionary

The dictionary component includes the actual entries in the dictionary and certain rules or procedures that are associated with them. These rules are part of the dictionary component because they are applied only to words that have been recognized by some part of the dictionary.

A single dictionary entry may carry three kinds of information. All entries carry a part of speech label for the particular lexical item. Some entries also specify a phrase boundary or clause boundary that is associated with the word. We will begin the discussion of the dictionary by looking at the part of speech labels that accompany entries.

4.2.1 Part of Speech Labels

The dictionary is composed of two smaller dictionaries. The first and most important is the dictionary of closed system items, and the second is a dictionary of some open system words. The code for closed system items all begin with two letters, whereas all open system codes begin with a single letter. This distinction allows us, for example, to indicate that auxiliaries are verb-like but yet share only some of the properties of open-class verbs.

4.2.1.1 Closed System Items That Occur in Verb Phrases

Closed system verbs then, will begin with the letter 'v' indicating that they occur in verb phrases and will have another letter in column 2 of the code indicating the type of closed class verb.

There are four possibilities for the first two columns of a closed system verb code.

VB Indicates some form of the verb 'to be'.

VH Indicates some form of the verb 'to have'.

VD Indicales some form of the verb 'to do'.

VM Indicates that the verb is a modal.

The reader must already realize that if we are defining closed system verbs as those verbs which do not function as main verbs in a verb phrase, then we will immediately run into trouble. Not all forms of the verb 'to have' function only as closed system verbs. We surely want to be able to distinguish between the different uses of 'had' in the following examples.

(1) He had a cold.

(2) He had had a cold.

(3) He had to stay home.

The approach that we have taken to the solution of this kind of problem is to allow the dictionary component to assign a tentative label to a word. This label is changed by a routine that takes an

alternative label for the word and replaces the original label with the new label if certain conditions are met.

An example of this procedure is the treatment of 'd. This can, of course, be a contracted form of both 'had' and 'would', as examples (4) and (5) illustrate.

(4) He'd buy that coat.

(5) He'd bought that coat.

For both examples above, the dictionary would first assign the label 'contracted form of have'. The subroutine, which is run after the dictionary component, searches for the next verb following 'd. If that next verb is in the base form, the code for 'd is changed to 'contracted form of would'. If the next verb is not in the base form, the original dictionary assignment remains.

Subroutines of this type apply to all entries for which there is more than one possible label. In the following description of word entries, the leftmost code is the main code that will be first assigned by the dictionary. The alternative codes are listed further to the right.

The main codes for the closed sysystem forms of 'have' and 'do' have four columns. The possible values of columns 3 and 4 as well as the entries are given in figure 5.

Figure 5

Closed System Code for 'Have' and 'Do'

CODE FOR VH, VD
COLUMN 3
1= FULL FORM
2= CONTRACTED FORM

COLUMN 4
0= BASE FORM
1= THIRD PERSON SINGULAR
2= PRESENT PARTICIPLE (ING)
3= PAST (ED)
4= PAST PARTICIPLE (EN)

LEXICAL ITEM	MAIN CODE	CODE 2	CODE 3	CODE 4
HAVE	VH10	V20		
'VE	VH20			
HAS	VH11	V21		
'S	VH21	VB212		
HAD	VH13	V23	V24	
HAVING	VH12	V12		
'D	VH23	VM262		
DO	VD10	V10		
DOES	VD11	V11		
DID	VD13	V13		

the reader will notice that not all forms of the verb 'to do' are in the list. This is because the verbs 'done' and 'doing' are never closed system items.

In the labeling of the various forms of 'to be', we considered 'be' to be closed system regardless of its function in the sentence The specific funtioning of be in a sentence is determined by the rule component of the system. Unlike the code for 'd, however,all'VB' labels are permanent and have the form shown in figure 6.

The two other types of closed system items that are considered to

Figure 6

Closed System Code for 'Be'

```
CODE FOR VB
     COLUMN 3
       1= FULL FORM
       2= CONTRACTED FORM

     COLUMN 4
       0= BASE FORM
       1= PRESENT TENSE
       2= PRESENT PARTICIPLE (ING)
       3= PAST (ED)
       4= PAST PARTICIPLE

     COLUMN 5
       0= UNMARKED
       1= FIRST PERSON ONLY
       2= SINGULAR
       3= PLURAL

BE              VB100
BEING           VB120
AM              VB111
'M              VB211
IS              VB112
ARE             VB123
'RE             VB223
WAS             VB132
WERE            VB133
BEEN            VB140
```

be verb-like are the modals and the negators 'not' and 'n't' in word final position. Both the negators are assigned the code 'VN' which has two possible values in column 3. The value '1' signifies full form and the value '2' signifies contracted form.

Only what we consider to be true modals are included in the dictionary. We have defined a true modal to be one which undergoes interrogative inversion. Items such as 'used to' and 'want to' are therefore not on the list. The entries for modals (VM) are shown in figure 7.

Figure 7

Dictionary Codes for Modals

CODE FOR VM
COLUMN 3
1= FULL FORM
2= CONTRACTED FORM
COLUMN 4
1= CAN SERIES
2= MAY SERIES
3= MUST SERIES
4= OUGHT SERIES
5= SHALL SERIES
6= WILL SERIES

COLUMN 5
1= PRESENT
2= PAST

CAN	VM111	VOO	NOO
COULD	VM112		
MAY	VM121	N40	
MIGHT	VM122		
MUST	VM131	NOO	
OUGHT	VM141		
SHALL	VM151		
SHOULD	VM152		
WILL	VM161	VOO	NOO
'LL	VM261		
WOULD	VM162		

The entries in figures 6 and 7 should give the reader a basic familiarity with the format of the dictionary and allow for a brief discussion of why this particular format was chosen.

There are great cost advantages to having a short dictionary. It was felt by us, however, that no matter what the length of the dictionary, each entry should be as information laden as possible. In constructing this dictionary we have developed rather elaborate labels for entries. In a sense what we have done is attempted to pre-guess the needs of future users.

In distinguishing between open and closed system words, our aim was to take advantage of the fact the number of closed system words is both finite and countable. In this system then, all such words are given a unique label. It is, however, probably more often the case that researchers are interested in classes of words rather than individual words in the analysis of composition samples. With this in mind we have organized the word codes so that certain concepts may be extracted from the corpus. By concept we mean here things such as 'verbness' 'regularity' or 'markedness'.

With words pre-classified in this way, a great number of questions may easily be asked. If, for example, a researcher is interested in the error rate of contracted verb forms as opposed to full verb forms, the question might take the following form:

1. The computer counts all the words that have two letter codes where the first letter is 'V'.
2. The computer counts how many of those have the value '2' in the third column of their codes.
3. The results of step 2 are subtracted from the results of step 1.
4. Steps 1, 2, and 3 are repeated. But this time the field of search is limited to only those occurrences which are inside error correction brackets and are followed by a correction symbol.

This process would yield absolute frequencies and relative frequencies for correct and incorrect verbs which are either

contracted or not contracted. What the researcher would have done in looking for the answer to his question, is simply to operationally define the concepts that he uses in terms of the dictionary codes.

4.2.1.2 Closed System Items That Appear in Noun Phrases

The closed system items that appear in noun phrase begin with a two-letter code as do all closed system entries. The four possibilities are:

NP Indicates that the word is a pronoun.

NQ Indicates that the word is an interrogative pronoun.

NL Indicates that the word is a limiter.

NA Indicaltes that the word is an article.

In figure 8, the breakdown for NP is given as well as the personal pronoun entries. These codes utilise the concepts of case, person, gender, and markedness.

Figure 8

Dictionary Codes for Pronouns

CODE FOR NP
COLUMN 3
1= PERSONAL PRONOUN
2= DEMONSTRATIVE PRONOUN
3= INDEFINITE PRONOUN

CODE FOR NP1
COLUMN 4
1= FIRST PERSON
2= SECOND PERSON
3= THIRD PERSON
4= THIRD PERSON MASCULINE
5= THIRD PERSON FEMININE

COLUMN 5
0= MORE THAN ONE POSSIBILITY
1= SUBJECTIVE CASE
2= OBJECTIVE CASE
3= POSSESSIVE CASE DETERMINER FUNCTION
4= POSSESSIVE CASE NOMINAL FUNCTION
5= REFLEXIVE

COLUMN 6
0= UNMARKED
1= SINGULAR
2= PLURAL

I	NP1111
ME	NP1121
MYSELF	NP1151
MY	NP1131
MINE	NP1141
WE	NP1112
US	NP1122
OURSELVES	NP1152
OUR	NP1132
OURS	NP1142
YOU	NP1200
YOURSELF	NP1251
YOURSELVES	NP1252
YOUR	NP1230
YOURS	NP1241
HE	NP1411
HIM	NP1421
HIMSELF	NP1451
HIS	NP1401
SHE	NP1511
HER	NP1501
HERSELF	NP1551

HERS	NP1541
IT	NP1301
ITSELF	NP1351
ITS	NP1331
THEY	NP1312
THEM	NP1322
THEMSELVES	NP1352
THEIR	NP1332
THEIRS	NP1342

The criteria of classification that have been used for the closed system codes are often idiosyncratic to the class of words, This is evident in the following excerpts from the dictionary. As far as possible, we have tried to keep the meaning of particular values in columns constant. This has proved to be much easier for the open system words than for the closed system items.

The dictionary excerpts diplayed in figures 9 to 11 on the following pages give an indication of the kind of classification that has been implemented. As I have stated above, the importance of the classififcation scheme in dictionary labeling is that it determines which concepts may be operationally defined and how.

Figure 9

Dictionary Codes for Non-Personal Pronouns

CODE FOR NP2
COLUMN 4
1= NEAR
2= DISTANT

COLUMN 5
1= SINGULAR
2= PLURAL

THIS	NP211
THAT	NP221
THESE	NP212
THOSE	NP222

CODE FOR NP3
COLUMN 4
1= POSITIVE/ASSERTIVE
2= NEGATIVE/NON-ASSERTIVE

COLUMN 5
1= REFERENCE TO THE ENTIRE SET
2= REFERENCE TO A PART OF THE SET

COLUMN 6
1= ONE
2= BODY
3= THING

EVERYONE	NP3111
EVERYBODY	NP3112
EVERYTHING	NP3113
NOONE	NP3211
NOBODY	NP3212
NOTHING	NP3213
SOMEONE	NP3121
SOMEBODY	NP3122
SOMETHING	NP3123
ANYONE	NP3221
ANYBODY	NP3222
ANYTHING	NP3223

Figure 10

Dictionary Codes for Interrogative Pronouns

CODE FOR NQ

COLUMN 3
- 1= PERSON
- 2= NON-PERSON

COLUMN 4
- 1= NON-SPECIFIC
- 2= SPECIFIC
- 3= SUBJECTIVE CASE (PERSON)
- 4= OBJECTIVE CASE (PERSON)
- 5= POSSESSIVE CASE (PERSON)

COLUMN 5
- 0= SIMPLE
- 1= EVER

WHAT	NQ210
WHATEVER	NQ211
WHICH	NQ220
WHICHEVER	NQ221
WHO	NQ130
WHOEVER	NQ131
WHOM	NQ140
WHOMEVER	NQ141
WHOSE	NQ150
WHOSEVER	NQ151

Figure 11

Code for Noun Limiters and Articles

```
CODE FOR NL
     COLUMN 3
       1= POSITIVE/ASSERTIVE
       2= NEGATIVE/NON-ASSERTIVE
     COLUMN 4
       1= REFERENCE TO THE ENTIRE TWO-MEMBERED SET
       2= REFERENCE TO THE ENTIRE SET
       3= REFERENCE TO PART OF A TWO-MEMBERED SET
       4= NEUTRAL REFERENCE TO PART OF THE SET
       5= PAUCAL REFERENCE TO PART OF THE SET
       6= MULTAL REFERENCE TO PART OF THE SET
       7= REFERENCE TO A SUFFICIENT PART OF THE SET
     COLUMN 5
       1= CAN PRECEDE A SINGULAR NOUN ONLY
       2= CAN PRECEED COUNT NOUNS ONLY
       3= CAN PRECEED BOTH COUNT AND MASS NOUNS
       4= CANNOT DIRECTLY PRECEED A NOUN
     COLUMN 6
       0= BASE
       1= COMPARATIVE
       2= SUPERLATIVE
     COLUMN 7
       1= CAN EXIST AS THE ONLY WORD IN A NOUN PHRASE
       2= CANNOT EXIST AS THE ONLY WORD IN THE NOUN PHRASE
```

ALL	NL12301
EACH	NL12101
EVERY	NL12102
BOTH	NL11201
EITHER	NL13101
NO	NL22302
ANY	NL24302
NONE	NL22401
NEITHER	NL21101
SOME	NL14301
FEW	NL15201
FEWER	NL15211
FEWEST	NL15221
MANY	NL16201
MORE	NL16311
MOST	NL16321
ENOUGH	NL17301
LESS	NL15311
LEAST	NL15321
SEVERAL	NL14201
LITTLE	NL15101
OTHER	NL14302

ANOTHER NL14101

CODE FOR NA
COLUMN 3
1= DEFINITE ARTICLE
2= INDEFINITE ARTICLE BEFORE CONSONANT
3= INDEFINITE ARTICLE BEFORE VOWEL

THE	NA1
A	NA2
AN	NA3

In some cases it was not possible to find an classification scheme for a group of words. In the case of prepositions, it was felt that there was no adequate way to categorize the individual words according to syntactic or semantic criteria. Because of this, prepositions are listed in the dictionary according to their frequency in written American English. There are forty-two uniquely identified prepositions in the list. The topic of preposition codes will be

4.2.2.0 The Dictionary of Open System Items

The dictionary of open system items contains a list of irregular verbs, a list of irregular nouns, and a list of suffixes.

As a dictionary, its function is different from that of the dictionary of closed system words. Because the number of entries in, for example the dictionary of open system verbs, is very small in comparison to the number of of open system verbs in English, most of the verbs that are compared to the dictionary in the 'look-up' process will not be matched. Therefore in the majority of cases, the dictionary will specifiy what the verb is not, rather than what it is. This allows us to use a set of default rules that state: "If we know this word is a verb and we also know that it is not in the dictionary of irregular verbs, then it must be given the code for regular verb."

In chapter 2 the various forms that a dictionary can take was discussed. This dictionay of open system words uses all three types of entries: whole words, stems, and affixes. In the following section the interaction of these three types can be seen.

4.2.2.1 Open System Verbs

At present, the dictionary of open system verbs contains 192 irregular forms and three suffixes. Figure 12 includes the coding scheme as well as a sample of these entries.

Column 2 of the codes for these words distinguishes between regular verbs and four types of irregular verbs. We suspect that the process of overgeneralization is students' verb errors might apply differentially to these four irregular verb types. The classification built into the dictionary allows for the testing of this hypothesis and others which require a finely tuned verb taxonomy. The user retains the ability, however, to ignore the classification of irregular verbs and to treat only the distinction between regular and irregular verbs. This is done by creating a category of 'not zero' in column 2 of the verb code.

Figure 12

Code for Open System Verbs

CODE FOR V

COLUMN 2

0= REGULAR VERB. THERE IS NO STEM CHANGE IN EITHER THE PAST TENSE OR THE PAST PARTICIPLE. BOTH END IN ED.

1= THE BASE, THE PAST TENSE, AND THE PAST PARTICIPLE FORMS ARE ALL DIFFERENT.

2= THE PAST TENSE AND THE PAST PARTICIPLE ARE THE SAME.

3= THE BASE AND THE PAST PARTICIPLE ARE THE SAME.

4= THE BASE AND THE PAST TENSE ARE THE SAME.

5= THE BASE, THE PAST TENSE, AND THE PAST PARTICIPLE ARE ALIKE.

COLUMN 3

0= BASE FORM OF VERB

1= THIRD PERSON SINGULAR PRESENT

2= PRESENT PARTICIPLE

3= PAST TENSE

4= PAST PARTICIPLE

BEGIN	V10	
BITE	V10	
BLOW	V10	
BREAK	V10	
CHOOSE	V10	
WRITE	V10	
BEGAN	V13	
BIT	V13	
BLEW	V13	
BROKE	V13	
CHOSE	V13	
WROTE	V13	
BEGUN	V14	
BITTEN	V14	
BROKEN	V14	
CHOSEN	V14	
WRITTEN	V14	
BRING	V20	
BUILD	V20	
BURN	V20	
BUY	V20	
CATCH	V20	
CREEP	V20	
WIN	V20	
BROUGHT	V23	V24
BUILT	V23	V24
BURNT	V23	V24
BOUGHT	V23	V24
CAUGHT	V23	V24
CREPT	V23	V24
RUN	V30	V34

BEAT	V40	V43	
COST	V50	V53	V54
CUT	V50	V53	V54
HIT	V50	V53	V54
HURT	V50	V53	V54

The dictionary does not contain any entries with the values '1' or '2' in column 3. When 'ing' is added to a verb the only changes that occur are consonant doubling and silent 'e' dropping. When 's' is added to a verb there are no changes. We therefore decided not to increase the size of the dictionary by including these forms, but rather decided to have the codes assigned by a set of rules. When the program finds a word that ends in 'ing' it performs the following operations:

1. The 'ing' is removed.
2. The remaining word is compared to all dictionary entries that have the value '0' in column 3. If a match is found, the word in the text is assigned the code of the first two columns in the dictionay entry plus the value '2' in column 3.
3. If a match has not been found and the remaining word ends in a double consonant, the final consonant is dropped. If it does not end in a double consonant, an 'e' is added to the word. Step 2 is then repeated.
4. If no match has still been found, it is assumed that the stem is regular. The word is assigned the code: V02.

In the case of the third person singular marking, rules similiar to 1,2, and 4 are used. A rule similiar to 3 is not necessary because the addition of 's' produces no orthographic changes.

The examples given above are specific cases of the general principle employed in the open system dictionary. The codes for all open system regular verbs are assigned in this manner. The suffix is dropped, the word is looked up, and a default code is assigned when no match is made.

A final note is necessary concerning the labeling of open system verbs. It is impossible, without considering the context, for the computer to distinguish between the past tense and the past participle of verbs. This applies, of course, only to regular verbs and those which have the values '2' or '5' in column 2. When these words are encountered, they are first given the code for past participle. If, after subsequent analysis, they are found not to be a part of a perfect or passive construction, and do not begin reduced clauses, then the code is changed to 'past'.

4.2.2.2 Open System Nouns

The dictionary of open system nouns is similiar in both structure and function to the dictionary of open system verbs. In this section therefore, I will only discuss the general characteristics of the dictionary.

The entries that we have included are those which we felt could not be labeled by rule. they are: (1) irregular nouns (2) cardinal numbers (3) ordinal numbers and (4) noun suffixes. The symbol '@' which is used by the coder to mark off proper nouns is also in the dictionary and any word preceded by that symbol will be given the label for proper noun. Of course not all the cardinal and ordinal numbers are in the dictionary. They are treated as stems and suffixes and the proper assignment is made by rule. Thus the word 'twentieth' is given the label 'ordinal number' by the recognition of 'twenty' as a stem, the recognition of 'th' as an ordinal suffix, and a 'y' conversion rule. The program also recognizes Arabic numerals as cardinal numbers, but at present there is no mechanism for dealing with Roman numerals.

The dictionary of open system nouns also contains a sub-classification of irregular forms. This can be seen in figure 13 which displays the column codes for this section of the dictionary.

The system of default rules that we have developed for open system nouns is quite similiar to the system for verbs. When a plural

Figure 13

Open System Nouns

```
CODE FOR N
  COLUMN 2
    0= REGULAR NOUN- THOSE WHICH SHOW NO CHANGE IN THE  BASE
       BEFORE PLURAL AND TAKE A REGULAR -S PLURAL.
    1= THE NOUN STEM IS DIFFERENT FROM THE BASE FORM BEFORE
       AN -S PLURAL.
    2= THE NOUN STEM IS DIFFERENT FROM THE BASE FORM, NO
       -S PLURAL.
    3= THE PLURAL FORM IS IDENTICAL TO THE BASE FORM.
    4= PROPER NOUN
    5= CARDINAL NUMBER
    6= ORDINAL NUMBER
  COLUMN 3
    0= BASE FORM OF NOUN
    1= POSSESIVE FORM OF NOUN
    2= PLURAL FORM OF NOUN
    3= PLURAL AND POSSESIVE REGULAR
    4= PLURAL AND POSESSIVE IRREGULAR
```

form (a noun ending in 's') is encountered, it is looked up. If it is located on the list with a code ending in'2' or '4' the word is assigned that code. If it is not found on the list it is assumed that the noun stem is regular and the word is assigned the code N02. Basically the same procedure is followed for words ending in ('s) and (s').

It is important to mention here that although I have been using the term'suffix' to refer to noun indicators such as '-ment' and the plural morpheme 's', the computer sees '-ment' as a main entry in the dictionary. In other words, '-ment' is an abbreviatory device for a large number of words that are to be given the code 'regular noun'. In the same way, the symbol '@' is not seen as a prefix but rather stands for a set of words that are to be coded N4. This allows us to deal with the concatenation of suffixes. When the system encounters a word such as 'establishments' it will first compare the entire word to

the dictionary and of course find no match. The 's' will then be dropped from the word and another comparison will be made. This time, the ending '-ment' will be found, telling the system that the word is a regular noun. 'Establishments' will be given the code N02.

The above has raised a question which must already be troubling the reader: Isn't it necessary to know what category the word belongs to so that it may be looked up on the right list and have the appropriate default rules applied to it? The answer is of course, yes and no. We must know whether a word is a verb or a noun before we apply the 's' rule because the 's' rule might be checking for either 'third person singular' or 'plural'. When the stem of the word is in the dictionary, we can use the information obtained through its location to make the correct category assignment. When the stem is not on the dictionary, we must rely on the environment of the word for information about its probable part of speech value. The next section of this chapter describes the way in which the dictionary entries themselves can be used to assign phrase boundaries and labels. These phrase markers will, of course, constitute the environment of the word. What we have strived toward is a system in which the part of speech value of a word listed in the dictionary will also generate information about phrasal structure. That same phrasal structure will be used to yield information about the part of speech values of other words.

4.3 The Dictionary and Phrase Structure

This section describes another aspect of the dictionary. So far we have been dealing with the dictionary's ability to correctly assign a part of speech value to a word found in some part of a text. We have employed a straight matching technique and a small set of rules to capture some generalizations that can be made about English morphology. In this section, we are using the same dictionary but elaborating its use, It is our belief that one of the greatest strengths of this look-up system is its ability of utilize all the information that a single entry carries. If it is true that in human natural language processing, individual words serve as multi-dimensional cues, tapping this multi dimensionality would greatly increase the efficiency of any automated system dealing with natural language.

Specifically, we are interested in the ability of dictionary entries to yield information about the structure of the phrases of which they are a part. It is necessary here to stop and explain what we mean by the term 'phrase'.

All phrases refered to in this paper are simple phrases. The system has no hierarchical phrase structure and phrases are not nested. The only exception to this last statement is our treatment of prepositional phrases. A preposition is considered to contain an immediately following noun phrase. This is the only type of nesting that is permitted, but even in this case, the noun phrase contained by

the preposition is dissolved and only the label 'prepositional phrase' is given. This can be best explained by example. In (1) below, the entire string is considered to be a single prepositional phrase. In (2), however, there are two phrases: 'a hot tin roof' is marked as a noun phrase and 'on the prairies' is marked as a separate prepositional phrase.

(1) on a hot tin roof

(2) a hot tin roof on the prairies

In the case of verb phrases, the restrictions that we have set are even tighter. A verb phrase may contain only those words which are coded 'V' in column 1. Thus, in example (3), the sentence contains a single verb phrase that is discontinuous because it is interrupted by an adverb.

(3) He has already gone.

In this system then, the phrase is literally the smallest syntactic unit larger than the word. It follows also that the phrase labels that are generated from the dictionary may apply only to the concept 'phrase' as defined by us. For example, we will claim below that all prepositions carry with them a 'beginning of phrase marker' and automatically label that phrase as a prepositional phrase. This is hardly surprising, given that we have defined a prepositional phrase as one which begins with a preposition and have not allowed such a phrase to be nested within another phrase.

To return now to the properties of specific dictionary entries

with respect to phrase labeling, our decision to use the dictionary in this manner stemmed from some simple observations. Within the framework of our restricted definition of 'phrase' it seemed that some words always began phrases, some words always ended phrases, and others never ended phrases. We noticed, for example that if a verb phrase contains a modal that modal is always the first word in the verb phrase. Of course English modals cannot co-occur, so it follows that all modals begin verb phrases. In the dictionary then, all the codes for 'VM' are preceded by the code '{V' which means :"Verb phrase is on from this point until it it turned off". Some other words have the exact opposit properties. They don't allow anything to follow them in the same phrase. An example of this is the pronoun 'mine' which, with the exeption of post-modifiers like 'too' and 'also', always end noun phrases. The code in the dictionary for 'mine' therefore is: NP1141 }N. as one would suspect, the code '}N' means that the noun phrase ends at this point.

The third type of relationship between words and phrase labels is one where a particular word prevents a phrase which is 'on' to be turned 'off'. The word 'the' is an example of this. Wherever and whenever the word 'the' occurs, the following word must belong to the same phrase. We might imagine that the following word is held by it. All dictionary entries for words which have this property of being able to hold onto following words are given the additional code '~'. This signifies that no phrase marker can occur in that position.

Before I elaborate the rules any further, it might be wise to

stop here and work through an example. Using this example, we will attempt to simulate the computer's coding of a sentence using only the tools that have been thus far presented. This example was first presented in chapter 3 to illustrate the deletion convention.

(7a) I don't know me.

As has been discussed above, the coder first reads this sentence. He recognizes that an error exists and codes it in the following manner:

(7b) I don't know (-me)

This marks the end of the human coding phase. All the following operations are done by computer. For this particular sentence, the task is not a difficult one since all the words in the sentence are listed in the dictionary. When the words are matched on the list, the part of speech labels are inserted into the text. The sentence is given the following form:

(7c) NP1111/I VD10/DO VN2/N'T V10/KNOW (-NP1121/ME).

In (7c), blanks still separate words but each word has two distinct parts. The symbol '/' functions as a delimiter and allows for the selective recovery of either the code or the orthographic representation of the word. The value of this extra symbol is most evident in the case of Arabic numerals occurring in the text.

We have also stated that the dictionary entries assign phrase labels. I have separated the two processes here for the sake of exposition, but in the actual operation of the system, dictionary

based phrase labels and part of speech labels are assigned simultaneously.

All phrase labels begin with the symbol '{' or '}'. The open curly bracket signals the beginning of a phrase and its couterpart signals the end of a phrase. The letter that follows the phrase marker is the label for the phrase. Because the system requires that both '{' and '}' be followed by a label, a phrase assignment is complete only if the brackets as well as the labels are balanced. The value of this stipulation for program de-bugging should be transparent.

In example (7) therefore, the dictionary inserts the phrase labels that are associated with the individual lexical items to produce the string displayed in example (7d).

(7d) {N NP1111/I }N {V VD10/DO

~ VN2/N'T V10/KNOW }V (-NP1121/ME).

The reader might feel that he has witnessed in the last four examples, the metamorphosis of a simple three word sentence into a monster. It must be remembered that the characterization of a sentence displayed in (7d) is meant to be read only by the computer. The string in (7d) then, is a translation of the computer's internal representation of the sentence. It is, in a sense, what the machine 'knows' about the sentence.

The tools that we have thus far presented allows the computer to almost completely code the simple sentence in (7). The dictionary entry for the word 'I' contains its part of speech label and two phrase markers. This means that whenever the code for 'I' is inserted into the text, it is surrounded by noun phrase markers. The dictionary assignment has the effect of ensuring that 'I' is always described as a single-word noun phrase (The string 'He and I' would therefore contain two noun phrases).

It has been stated above that modals always begin verb phrases. Here, the word 'do' creates the 'beginning of verb phrase' marker. This is dictated by the first code in the dictionary entry. In this case, 'do' cannot be assigned its alternate main verb code because of the presence of (n't) immediately following.

It will be noticed that the word 'do' is followed by the symbol '~' which means that no phrase markers may occur in the position that the symbol occupies. This 'hold' symbol is not inserted by the word 'do', but rather is part of the dictionary entry for (n't). Although it may be obvious to us that a phrase could not begin with a contraction, this must be specified for the computer.

By far, the most powerful dictionary-based phrase label rule is one which states that all open-system verbs end in verb phrases. This means that all words in the dictionary of open-system verbs as well as all words coded 'V' in column 1 by default rules are followed by the symbol'}V'. The effect of this is seen in example (7d) where the '}V'

is inserted after 'V10/KNOW'.

Finally, it is interesting to note that in example (7), the word following the error correction symbol is assigneda part of speech value by the dictionary. Words following error correction symbols, however, are not permitted to affect the phrase structure of a sentence.

So, although the word 'me' has an 'end of noun phrase' marker associated with it in the dictionary, that marker is not inserted into the text. For that reason, the last phrase marker in the sentence occurs before the error correction bracket.

The sentence in example (7) is far too simple to be an adequate test of a system that claims to characterize English sentences. The sentence is extremely short, has only two phrases, and contains only dictionary-listed words. Without more rules, however, the system could not deal with much greater complexity. In the following section, the dictionary-independent rules are presented and their function exemplified.

4.4 The Dictionary-Independent Rules

4.4.1 A Note on Clauses

At present, the system has a simple set of rules that assign clause boundaries in sentences. The method is quite similiar to that employed in the assignment of phrase labels. All sentences begin with a 'beginning of clause' marker (<), and end with a clause marker (>). All subordinators carry with them '<' and all coordinators carry with them both '<' and '>'. Finally, the dictionary entries for commas also contains a clause boundary. A string between two commas is considered to be a clause if a verb exists in the sentence after the two commas.

All these dictionay assign clause labels are tentative. The boundaries are destroyed if a verb phrase is not found within them. In addition, non-finite verb phrases that are not immediately preceded by another verb phrase are considered to begin clauses.

The rules stated above result in the creation of the following clause structure displayed in examples (1), (2), and (3).

(1) <She is eating yogurt>.

(2) <The woman <eating yogurt> is my friend>.

(3) <She eats yogurt><because she likes it>.

This scheme for assigning clause structure is extremely simple, but adequate for the characterization of sentences produced by ESL

students.

The clauses need not be labeled because the first phrase of the clause determines its type. The primary function of the clause rules is to provide a framework in which the rules discussed in section 4.3 can operate.

4.4.2 The rules

In this section, the set of rules that apply to the text which has been partially coded by the dictionary rules is discussed.

Generally, these rules formalize the way in which frequently occurring words determine the labeling of other sentence parts.

Rule 1. The Nominalization Rule

We have discussed above that the word 'the' exerts some influence on the word that folows it. In the dictionary, 'the' is followed by the symbol '~', indicating that the next word must belong to the same phrase. In fact, an article also determines what part of speech value the next word may be assigned. It has the effect of nominalizing that word.

In this system, whatever follows an article (NA) is given the symbol 'N' in column 1. If that word is always coded as a noun, then the code is not changed. If, however, it is not coded or has some letter other than 'N' in column 1, then the noun code is either

inserted or made to precede the existing code. The phrases which are partially coded in examples (1) and (2) would be changed by this rule to the form given in examples (1a) and (2a).

(1) {N NA2/A V32/RUNNING N20/MAN }N

(2) {N NA1/THE ?/TURTLE }N

(1A) {N NA2/A NV32/RUNNING N20/MAN }N

(2A) {N NA1/THE NOO/TURTLE }N

Rule 1 aids greatly in the coding of unrecognized nouns. When the 'N' is placed by the rule in column 1, then the dictionary default rules can re-apply to the word and, in the case of turtle, give it the code NOO. The rule also creates a class of verbs that occur in noun phrases. Figure 14 displays the structure of the rule in flow chart form.

Rule 2. Infinitives

The purpose of this rule is to distinguish between the occurrence of 'to' as a preposition 'PP02' and as an infinitive marker 'VI'. The rule in fugure 15 states That if the word 'to' is found and the following word is not coded and does not have a suffix (e.g. -ing, -ed, -ly), then the word is coded as a regular verb in the base form. The code for two is then changed to 'VI'.

Rule 3. Verb Phrase Insert

Although the dictionary entries create boundaries for verb phrases

that begin with auxiliaries, we do not yet have a rule that inserts '{V' before single word verb phrases. This rule, as shown in figure 16, inserts the boundary after checking that an auxiliary does not precede the verb.

Rule 4. Discontinuous verb phrases

A verb phrase is considered to be discontinuous if any number of words separate the auxiliary and the main verb. We have used the definition of discontinuous verb phrases to treat interrogatives with verb inversion as well as discontinuous phrases in an affirmative sentence.

All auxiliaries are implicitly coded 'discontinous'. If they are immediately followed by a main verb, the disc. marker is removed. If on the other hand, a verb does not follow, a search is made of the rest of the sentence. When the main verb is found, it is also given a disc. marker before. This indicates that it belongs to the auxiliary.

In figure 17 it can be seen that the search also skips over intervening clauses.

Rule 5. The 'by' rule

This rule simply captures the facT that tan unrecognized word before 'by' (PP09) is almost always a verb. When the program finds PP09, it goes back one word, and, if that word is not coded, it is given the

label 'VOO'.

Rule 6. Changes in part of speech labels

After all the above rules have been applied, the program creates the remaining phrase boundaries by looking for the differences between the first column letter of two adjacent words. If a difference does exist and there is no '~' in the space between the two words, a phrase boundary is inserted.

Figure 14

The Nominalization Rule

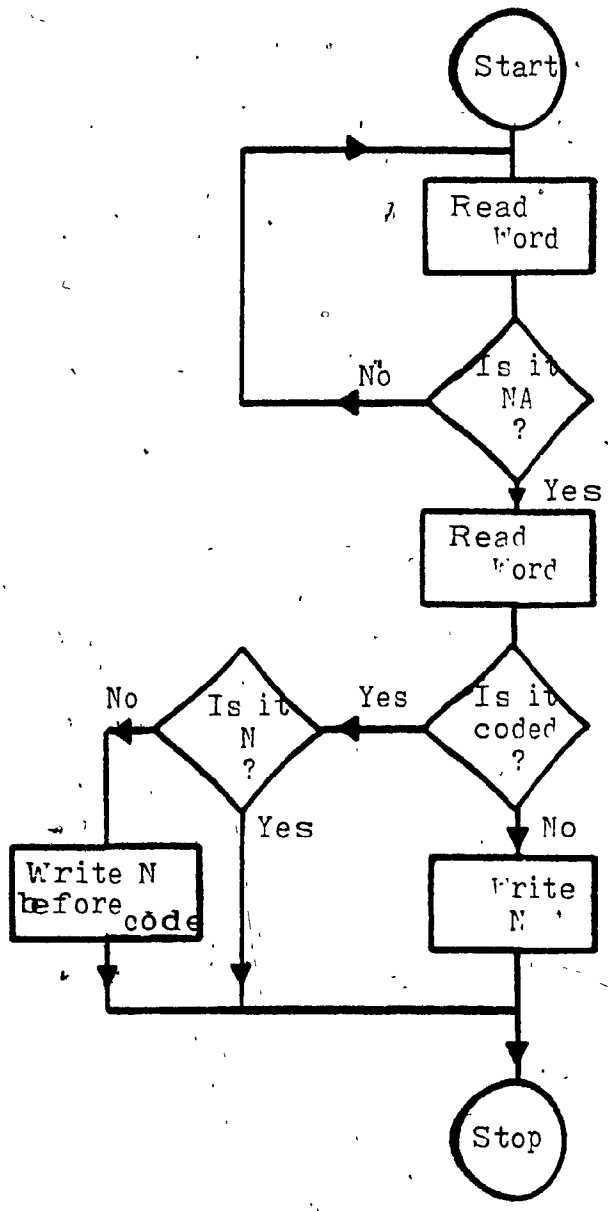


Figure 15

Infinitives

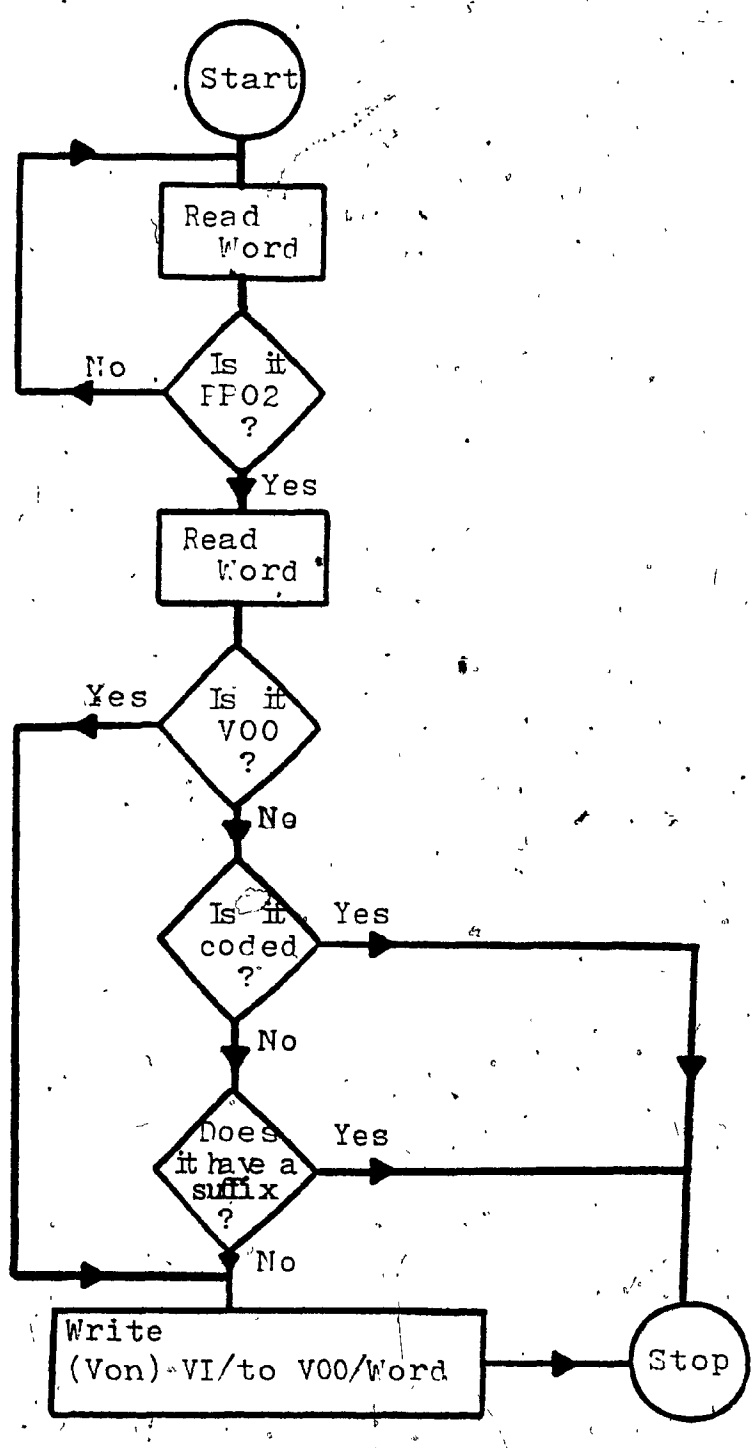


Figure 16

Verb Phrase Insert

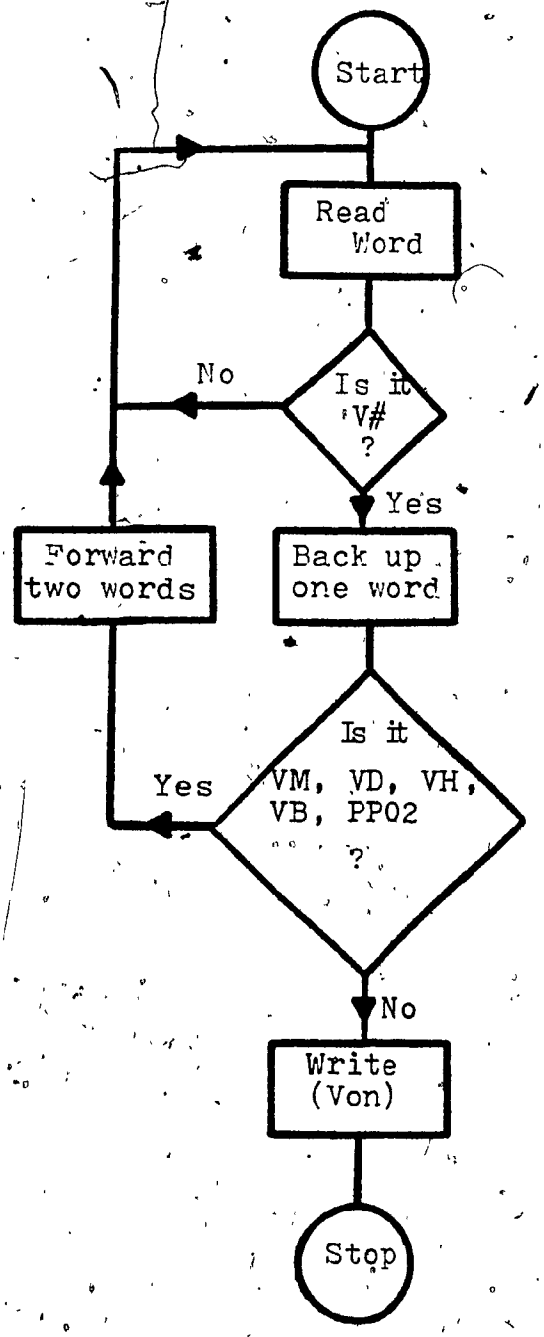


Figure 17

Discontinuous Verb Phrases

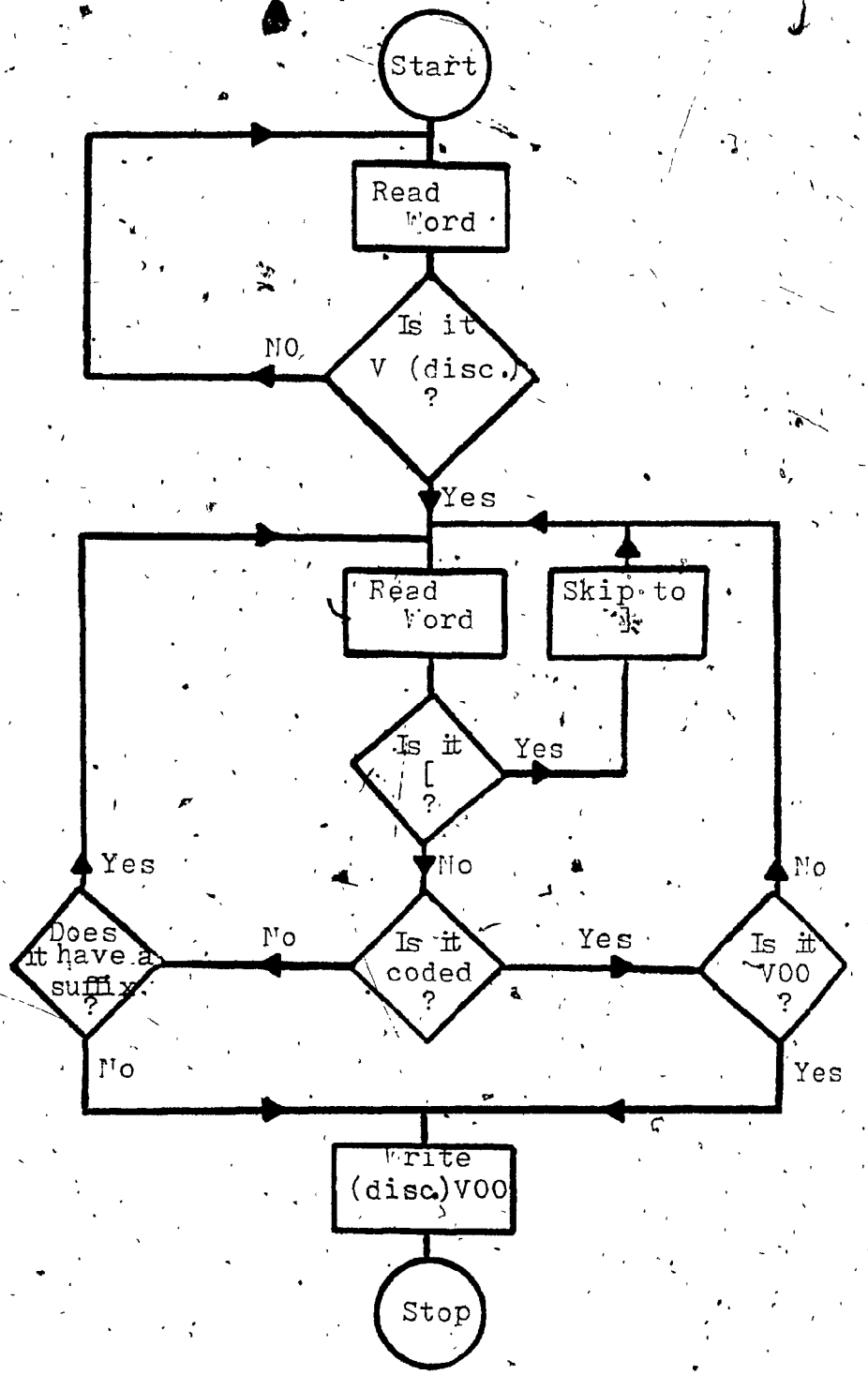


Figure 18

The 'By' Rule

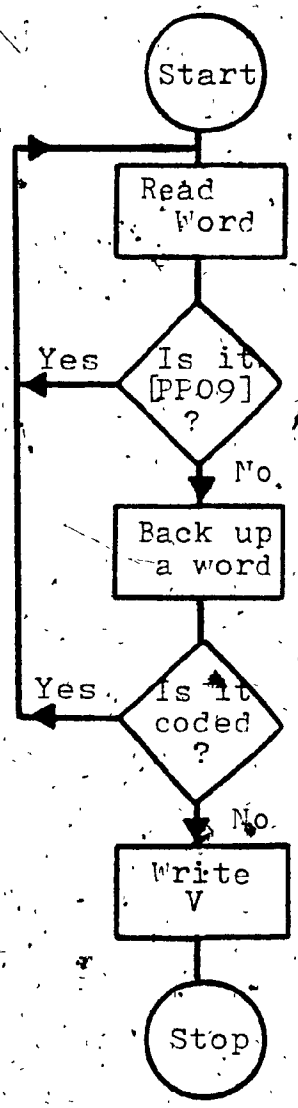
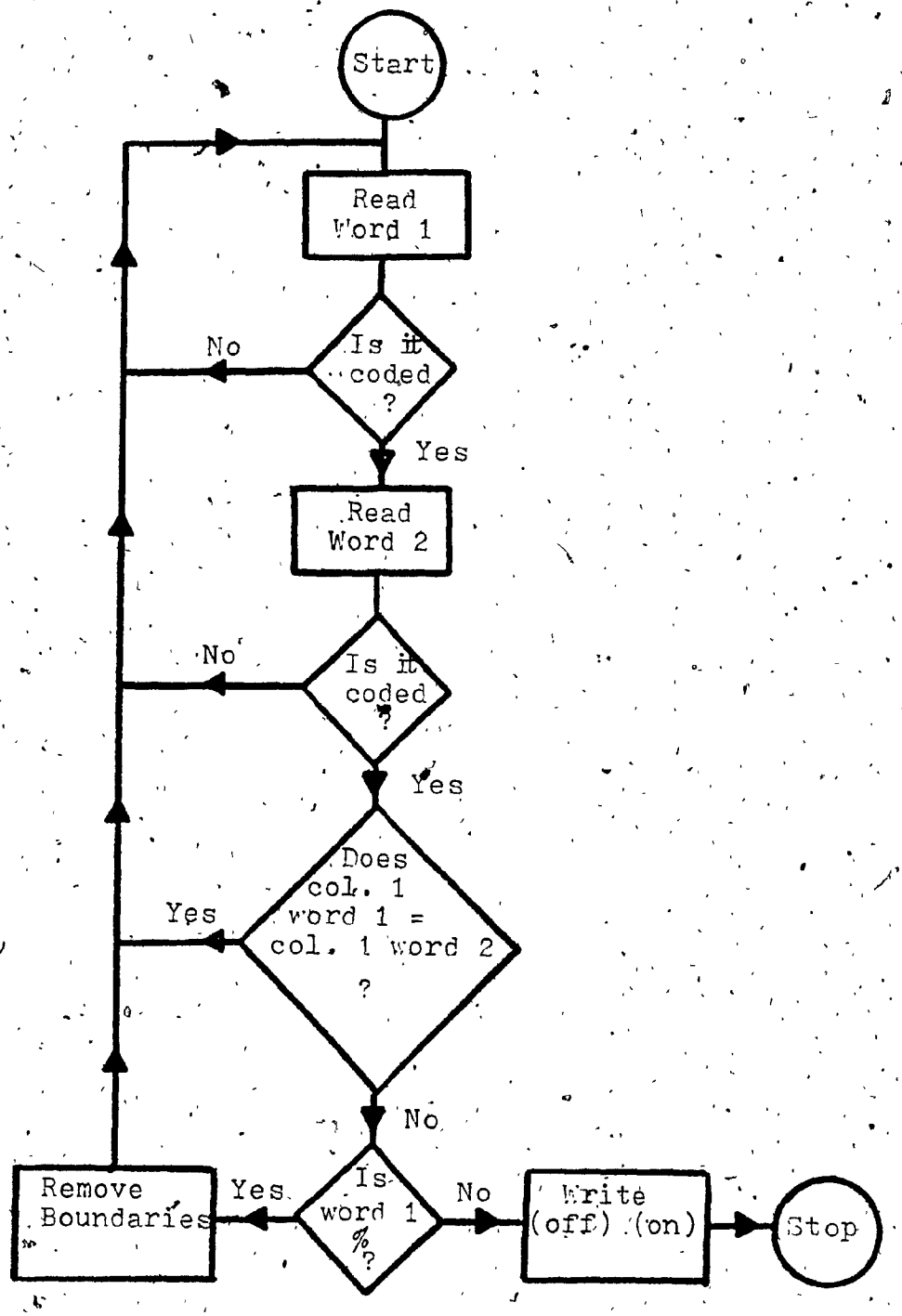


Figure 19

Changes in Part of Speech Labels



CHAPTER 5

Applications of the System

Throughout this paper, I have been claiming that the system does not represent any particular analysis of second language data but rather is that which is fundamental to any analysis.

Although in the preceding chapters I have given examples of the uses to which the system may be put in morpheme studies and studies of lexical diversity or syntactic complexity, I have not yet embarked upon any extensive discussion of the uses of the system in practical research. In this chapter, two areas will be dealt with. I will first sketch our plans for the pre-set functions. This will be followed by a discussion of the user's ability to create his own independent and dependent variables and to use them in statistical analyses.

5.1 Pre-set Functions

The system is extremely powerful in that it allows for a multitude of linguistic analyses in a very short period of time.

This is at once an advantage and a disadvantage. When analyses are easy to perform there is a great tendency for the researcher to go on 'fishing trips'. In other words, the ability to look at everything often allows one to forget the importance of constructing hypotheses prior to statistical analysis of the data.

This problem must be felt quite acutely by the authors of the Statistical Package for the Social Sciences (Nie, Hull, Jenkins, Steinbrenner and Bent, 1970) who often warn their readers of the danger of being swamped with data. We too would like to discourage the user from running all possible analyses.

On the other hand, we would like the system to be not only a mechanism for hypothesis testing but also a mechanism for the generation of new hypotheses. It seemed desirable, for this reason, to have a feature built into the system that would produce a linguistic 'summary' of the compositions that it operates on. The researcher, after examining the summary, could decide to investigate one or more particular phenomena that seem interesting.

To this end, we have chosen to have the computer create a probability grammar of the sentences being studied.

The concept of probability grammar as discussed in Klein and Dittmar (1979) is not a unitary concept, and subsumes a fairly large variety of procedures.

We propose to use one of the simpler types of probability grammars for this system. Basically, the computer would go through a word and label counting routine and produce one of the familiar kinds of phrase structure grammars. If we take noun phrases as an example, it might look like this:

np -----> n

np -----> adj n

np -----> art adj n

A student, or group of students with a noun phrase grammar like this one would only produce three types of noun phrases.

A probability grammar takes this basic structure and attaches probabilities to each of the possibilities. It is assumed that the probability of 1 is distributed over the left hand symbol. In other words, the numbers associated with this set could only add up to one. This is, of course just one step further than a 'frequency of use' count.

In brief, we propose that in the future, a probability grammar be generated by the system by request. This grammar could attempt to characterize sentence types, single compositions, or groups of students that differ on some language-related measure.

The probability grammar would also have to be modified to deal with second language data. We could not write grammatical rules for what students actually do, because much of this is not rule governed. It would seem that the best way of dealing with a summary of second language written production would be to generate a probability grammar based on what students attempt to do. This would be operationally defined as the way a sentence that contained an error was corrected. In addition we plan to have a probabilistic measure of the students' success in attempting to produce that string. Our phrase structure

grammar may now look like this:

np ----->n .75 .82

np ----->adj n .50 .60

np ----->adj art n .25 .50

If this were a single student, it would tell us that the student attempted to produce a single word noun phrase seventy-five percent of the time that he attempted to produce any noun phrase. When he did this, the noun was correct eighty-two percent of the time.

5.2 User Definition

Although the statistical package hookup would be one of the pre-set functions, it would also be under user control and is therefore best discussed here. The hookup refers to the system's ability to generate an SPSS-style matrix that would be used by the statistical package.

We have discussed the user's ability to define variables such as grammatical error by operationalizing the concept in terms of the symbols used by the system. All defined variables such as complex sentence are created in the same manner. These allow for correlation studies of two such variables.

There are two major advantages to this approach. The first and most obvious, is that data is kept internally by the computer, therefore reducing the possibility of error. The second is that all

categories remain re-definable.

The greatest flexibility that is built into the system is the user's flexibility in determining the unit of analysis, which we could simply call group. For linguistic analysis, we might imagine that sentence type would form the basis of a group. Groups could also be formed by linking some attitude data to student compositions.

It is our opinion that the system's applicability to diverse research interests extends well beyond our original anticipations. When the system is operational, this will clearly be the focus of our interest.

REFERENCES

Barkman, B., & Winer, L. The ESL performance of grade 10 francophones in Quebec: Some methodological considerations. 1979 Proceedings of the First Delaware Symposium on Language Studies, October 1979. In press.

Cherry, L.L. PARTS - A system for assigning word classes to English text. Computing Science Technical Report, No. 81. Murray Hill, New Jersey: Bell Laboratories, 1980.

Corder, P. The significance of learners' errors. IRAL, Vol. v/4, 1967.

Hunt, K. Grammatical structures written at three grade levels. Research Report No. 3. Urbana, Ill.: National Council of Teachers of English, 1965.

Klein, W., & Dittmar, N. Developing grammars. Berlin: Springer Verlag, 1979.

Lightbown, P.M., & Barkman, B. Interactions among learners, teachers, texts, and methods of English as a second language. Progress Report 1977-78. Concordia University, Montreal, Canada.

Malcolm, D. A comparison of English compositions by native and non-native speakers. Unpublished Master's thesis, Concordia University, 1981.

Nie, H.H., Hull, C.H., Jenkins, J.G., Steinbrenner, K., & Bent, D.H. Statistical package for the social sciences. New York, N.Y.: McGraw-Hill, 1970.

Pacak, M. Computational linguistics and information handling. In Howerton, P.W. (Ed.), Management of information handling systems. Rochelle Park, N.J.: Hayden Book Co., Inc., 1974.

Richards, J.C. A non-contrastive approach to error analysis. In J.C. Richards (ed.), Error analysis. London: Longman Group Limited, 1974.

Sager, N. Natural language information processing. Reading, Mass.: Addison-Wesley Publishing Company, Inc., 1981.

Salton, G. Automatic information organization and retrieval. New York, N.Y.: McGraw-Hill, 1968.

Winograd, T. Understanding natural language. New York, N.Y.: Academic Press, 1976.