AD-775 139

LOCALLY ORGANIZED PARSER FOR SPOKEN INPUT

Perry L. Miller

Massachusetts Institute of Technology

Prepared for:

Advanced Research Projects Agency
Electronic Systems Division

2 May 1973

DISCLAIMER NOTICE

THIS DOCUMENT IS THE BEST QUALITY AVAILABLE.

COPY FURNISHED CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

DOCUMENT CONTROL DATA - R&D

AD-775 139

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Lincoln Laboratory, M.I.T.	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

A Locally Organized Parser for Spoken Input

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Technical Report

5. AUTHOR(S) *(Last name, first name, initial)*

Miller, Perry L.

6. REPORT DATE 2 May 1973	7a. TOTAL NO. OF PAGES 121	7b. NO. OF REFS 38

8a. CONTRACT OR GRANT NO. F19628-73-C-0002	9a. ORIGINATOR'S REPORT NUMBER(S) Technical Report 503
b. PROJECT NO. ARPA Order 2006 c. d.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* ESD-TR-73-98

10. AVAILABILITY/LIMITATION NOTICES

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency, Department of Defense

13. ABSTRACT

This report describes LPARS, a locally organized parsing system designed to recognize continuous speech, accepting as its input a string of phonemes which contains ambiguity and error. LPARS is locally organized in the sense that it can construct local parse structures from word candidates found in all parts of the input utterance. These local structures are used as "islands of reliability" and, together with syntactic and semantic information, guide the search for words to complete the utterance.

14. KEY WORDS

parsing system
man-machine interaction
machine speech recognition
speech understanding system

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

A LOCALLY ORGANIZED PARSER FOR SPOKEN INPUT

P. L. MILLER

Group 26

TECHNICAL REPORT 503

2 MAY 1973

Approved for public release; distribution unlimited.

LEXINGTON MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Advanced Research Projects Agency of the Department of Defense under Air Force Contract F19628-73-C-0002 (ARPA Order 2006).

A LOCALLY ORGANIZED PARSER FOR SPOKEN INPUT*

ABSTRACT

This report describes LPARS, a locally organized parsing system designed to recognize continuous speech, accepting as its input a string of phonemes which contains ambiguity and error. LPARS is locally organized in the sense that it can construct local parse structures from word candidates found in all parts of the input utterance. These local structures are used as "islands of reliability" and, together with syntactic and semantic information, guide the search for words to complete the utterance.

Accepted for the Air Force
Eugene C. Raabe, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

*This report is based on a thesis of the same title submitted to the Department of Electrical Engineering at the Massachusetts Institute of Technology on 28 March 1973 in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

CONTENTS

A LOCALLY ORGANIZED PARSER FOR SPOKEN INPUT

I. INTRODUCTION

A. BRIEF OVERVIEW

Recognizing a spoken utterance by machine is a task much different from that of recognizing a text sentence. The difference arises both from the difficulty of front-end phoneme recognition and from the articulatory variability of speech. As a result, phonetic segments often cannot be identified uniquely with any real accuracy and, therefore, an utterance processed by a phoneme recognizer can contain a great deal of local ambiguity and error. In practical terms, this means that the words are not recognizable immediately, as they are in text.

Humans use a great deal of syntactic and semantic information to help them handle this ambiguity. They possess much knowleuge of the things they talk about, and of how words can be put together to describe these things. Most previous work in machine speech recognition, however, has dealt primarily with acoustical processing, such as phoneme recognition, pitch and intensity analysis, etc., with little attempt to integrate syntactic and semantic knowledge into this analysis. Any work on the recognition of continuous speech which relies solely on such a low-level approach is at the mercy of the inherent local ambiguity.

The present research demonstrates how a greater degree of language competence can be integrated into this recognition process.

This report develops an approach which uses semantic and syntactic information to assist machine recognition of speech. The approach is implemented in LPARS, a Locally organized PARSer designed to process continuous speech with the help of syntactic and semantic information. Its approach differs from traditional parsing methods in that it has no inherent left-to-right, or right-to-left, bias to its operation. Rather, it allows syntactic structures to be recognized locally in any part of the sentence. In fact, several parse structures may be built up simultaneously in different parts of the sentence and later connected by searching for words that might reasonably exist between them.

The approach is highly data-directed, letting previously recognized structures help guide the search for other words in the utterance. Both syntactic and semantic constraints are used to help guide the analysis. Thus, the system is a "vertically organized" one in the sense that it allows syntax and semantics to give feedback to, and help guide, the lower-level word-recognition process.

The emphasis in the present work has been to explore the various issues involved in the design of a local parsing system, and then to demonstrate that such a system can be created. Thus emphasis has not been on developing a finely tuned system with high performance characteristics.

The remainder of this introductory section gives an overview of the approach taken. Sections B and C describe in more detail the sources and nature of local ambiguity and error in speech. In Sec. D, two basic concepts are presented: that of phonetic distance, and that of a vertically organized system. These two basic concepts are central to understanding the approach described. Then, in Sec. E, a simplified example is worked through to give a concrete picture of how the approach is implemented. This Introduction concludes by discussing related work (Sec. F) and by outlining how the presentation of the approach is developed in the rest of this report.

B. LOCAL AMBIGUITY AND ERROR IN SPEECH

In the early 1960's, researchers working on speech recognition approached the problem from the local standpoint of recognizing sound segments in the utterance in much the same spirit as one might try to recognize letters on a printed page. At that time, most work concentrated on the recognition of isolated words and phrases; thus, no extensive use of syntactic and semantic constraints between different parts of the utterance was possible. In fact, it was initially hoped that phoneme recognition could be developed to sufficient accuracy that the parsing of continuous speech could be accomplished in much the same fashion as the parsing of text.[1, 2]

This approach was not very profitable. It was difficult to develop algorithms which could even approach the desired level of accuracy. Furthermore, it became apparent that the difficulty was not entirely the fault of the algorithms used, but rather the result of local ambiguity inherent in speech itself.

There are two fairly distinct but related sources of local ambiguity and error in speech: (1) articulatory variation, and (2) machine recognition limitations. It is worthwhile to discuss both briefly.

1. Articulatory Variation

A sequence of spoken phonemes differs from a sequence of letters written on a page, because the articulation of phonemes is a dynamic series of events which takes place as the vocal apparatus changes from one configuration to another. Each phoneme can be thought of as an idealized representation of a target configuration which the vocal apparatus attempts to attain.

The exact manner and the degree to which the target configurations are attained may vary in different attempts to vocalize a single utterance. Thus, the underlying phenomenon of speech itself is not constant or consistent; it is a dynamic phenomenon which is not reliably reproducible. Furthermore, the acoustic cues which might, if present, uniquely identify the phonetic segments are often absent or barely noticeable.

2. Machine Recognition Limitations

Granting the inherently ambiguous character of speech itself, it is not surprising that the problem of algorithmically segmenting and classifying spoken phonetic segments is a difficult one. A series of dynamic phenomena must be processed using recognition criteria which statistically are as accurate as possible. Since the cues which uniquely distinguish certain phonemes can be quite subtle, it is difficult to write an algorithm to pick up these cues only when they are "really" present. Despite a number of different attempts to develop phoneme recognizers, the amount of ambiguity and error which characterizes existing phoneme recognizers is quite large.

C. PRACTICAL IMPLICATIONS OF LOCAL AMBIGUITY

In practice, the local ambiguity described above implies that words are not easily recognizable. In this respect, it is useful to make a somewhat arbitrary distinction and identify two separate problems:

(1) Small unstressed "function" words.

(2) Longer "content" words which are considerably garbled (by excessive error and ambiguity).

1. Small Unstressed Function Words

Small unstressed words such as "a", "the", "and", "of", "is", etc. tend not to be articulated clearly and are sometimes barely articulated at all. For example, the phrase "University of Michigan" can be spoken quite intelligibly without, or with just a hint of, the word "of". This poses no problem to a human listener because semantic and syntactic knowledge primes him to expect the word "of" to be present.

The recognition of such words clearly poses difficulties for any speech recognizer which does not use context to indicate where such words might be present. Indeed, even if the word is well articulated, its small size, in the presence of front-end error, might still make it difficult to find reliably without contextual cues.

2. Longer Words Which Are Garbled

Longer "content" words offer more phonetic information to match against the input. Also, such words contain stressed syllables, which tend to be articulated more clearly than unstressed syllables. If the system is to work at all, such long words should be fairly reliably recognizable. Frequently, however, some longer words may be quite garbled by the combination of articulatory variability and front-end error. Any attempt to distinguish all such words on the first scan through the sentence would be liable to deluge the system with an overwhelming number of word candidates, most of them wrong.

It is therefore much more reasonable to look for such words very selectively, only when one has built up a great deal of contextual information indicating the exact section of input where such a word might exist, and indicating a small list of word possibilities that might meaningfully be present.

It is clear that humans deal with the ambiguity described above because they possess a great deal of knowledge of the things (objects, events, actions, etc.) being talked about, and of how utterances are formulated in English to describe these things. For instance, if a speaker pauses in the middle of a sentence, a listener can often predict that one of a limited number of words is likely to be spoken next. Certainly the listener can reasonably discard as possibilities most of the words in his vocabulary.

The present work explores how such knowledge can be incorporated into a machine so that it can be used in a systematic fashion.

3. Tradeoffs in a Speech Recognition System

An interesting issue related to this problem concerns the tradeoffs possible between increased refinement in the local processing of an utterance (i.e., refinement of phoneme recognition algorithms) on the one hand, and the use of more global information on the other. In other words, how much real information does increased local refinement yield when seen in terms of the system as a whole? One would suspect that intelligent heuristics in the higher levels of a system might make a great deal of lower-level refinement unnecessary. Other parameters in this tradeoff are vocabulary size, syntactic and semantic complexity, the degree to which small words are allowed to be used, and the degree to which the vocabulary is partitionable so that at different stages of discourse only part of the total vocabulary is likely to be used. This report takes an initial step toward exploring some of these tradeoffs.

D. TWO BASIC CONCEPTS

Some aspects of the system presented here may at first seem somewhat counter-intuitive. This section outlines two concepts central to the approach: that of a vertical system (one with vertically organized feedback), and that of phonetic distance. It is our hope that an understanding of these two concepts will help to put the approach into sharper perspective.

1. Vertically Organized System

LPARS is a vertically organized system. Vertical system organization is useful where a great deal of local ambiguity exists. A vertical system is one which allows the various levels of analysis to interact with one another. In the case of speech recognition, the various levels are semantic, syntactic, and lexical analysis (and perhaps phonemic analysis). Figure 1-1 illustrates the structure of such a system.

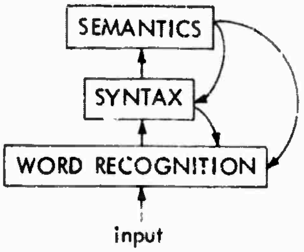


Fig. 1-1. A vertical system.

A vertical system is so named to distinguish it from a "strictly hierarchical" system which does not have the feedback from higher- to lower-level analysis. An example of a strictly hierarchical system is a programming language translator, typically having a lexical analyzer which recognizes identifiers, numbers, operators, etc. and passes them on to the parser which deduces their syntactic structure and turns this over to the semantic analyzer which generates machine instructions. Such a system is not a vertical one because, for instance, the parser cannot tell the lexical analyzer what tokens it might look for next.

Section I-F-4 discusses two other examples of vertically organized systems: the M.I.T. AI vision system,[3] and T.A. Winograd's natural language parser.[4] Vertical organization in the vision system allows higher-level analysis to be invoked even though some lines of a scene may not have been recognized. Similarly, in speech analysis, all words need not be recognized immediately by the lexical analyzer. Contextual information can be used to help determine boundaries in unclear sections of the utterance, and to propose words that might occur there.

Thus, in LPARS, the lexical analyzer first makes local tests for words which can be recognized fairly reliably. This processing results in a list of "word candidates," some of which may be wrong. The word candidates are turned over to the higher-level routines which direct tests to discover additional words. Fundamental to this analysis is the ability to parse outward from a number of recognized portions of a sentence in an organized manner.

2. Phonetic Distance

The concept of phonetic distance goes hand in glove with that of a vertically organized system. Phonetic distance is used when matching the phonetic spelling of vocabulary words in the dictionary against strings of phonemes recognized by the front end of the system. As an example, one simple measure of phonetic distance might be to define the distance of a word from a section of input as equal to the number of phonemes that differ. Thus, FROM would be a distance of 1 from both FROL and TROM.

In practice, of course, some differences are greater than others. The voicing difference between B and P, for instance, is presumably not as great as the difference between B and S. Consequently, it is useful to allow such phonetic distances wider range for more accurate discrimination. The concept of phonetic distance must also be extended to account for deleted and inserted phonemes.

It is not immediately obvious how such distances are best measured. To make such measurements certainly requires a statistical evaluation of the particular front end being used. These issues are discussed more fully in Sec. III. Hopefully, it will become clear that precisely how such phonetic distance is measured is not central to the ideas presented in this report (although the effect of these measures is definitely central).

The concept of phonetic distance fits very naturally into a vertical system. In a vertical speech system, initial scans through the input can be made at low phonetic distance to find words that are not too garbled. Higher phonetic distance tests for more garbled words can be made selectively, based on semantic and syntactic information.

E. SIMPLIFIED EXAMPLE OF LPARS IN OPERATION

LPARS is designed to process spoken input by building up parse structures in any part of a sentence, and then by using these structures to help guide the search for further words to complete the sentence. It differs from most parsers in the local nature of its syntactic processing and in its highly vertical structure.

This section gives a brief introductory description of LPARS itself and of how it performs lexical, syntactic, and semantic processing. Then, a simplified example of LPARS in operation is worked through to help put into focus the various parts of the system.

1. Some Particulars

LPARS' vocabulary contains approximately 70 words. It recognizes a very restricted, but linguistically interesting, subset of English. Its semantics are defined in terms of a particular scene (a small two-room house containing people, furniture, fixtures, etc.) about which one may make statements, ask questions, tell a very simple-minded story, or command the system to manipulate the scene.

2. Operation of LPARS

As input, LPARS expects a string of phoneme candidates from a front-end phoneme recognizer. For the present work, the input was prepared by a phonetic scrambler program which simulated front-end behavior, rather than by a real phoneme recognizer. The input phoneme candidates may be ambiguous (i.e., several possibilities may be given for one segment). The input is also expected to contain a fair amount of error.

LPARS operates by first making an initial scan through the sentence, seeking longer words which are not too garbled. The scan returns a list of word candidates which match within a given error tolerance to specified sections of the input. It is likely that some of these word candidates are incorrect, and, indeed, some may overlap.

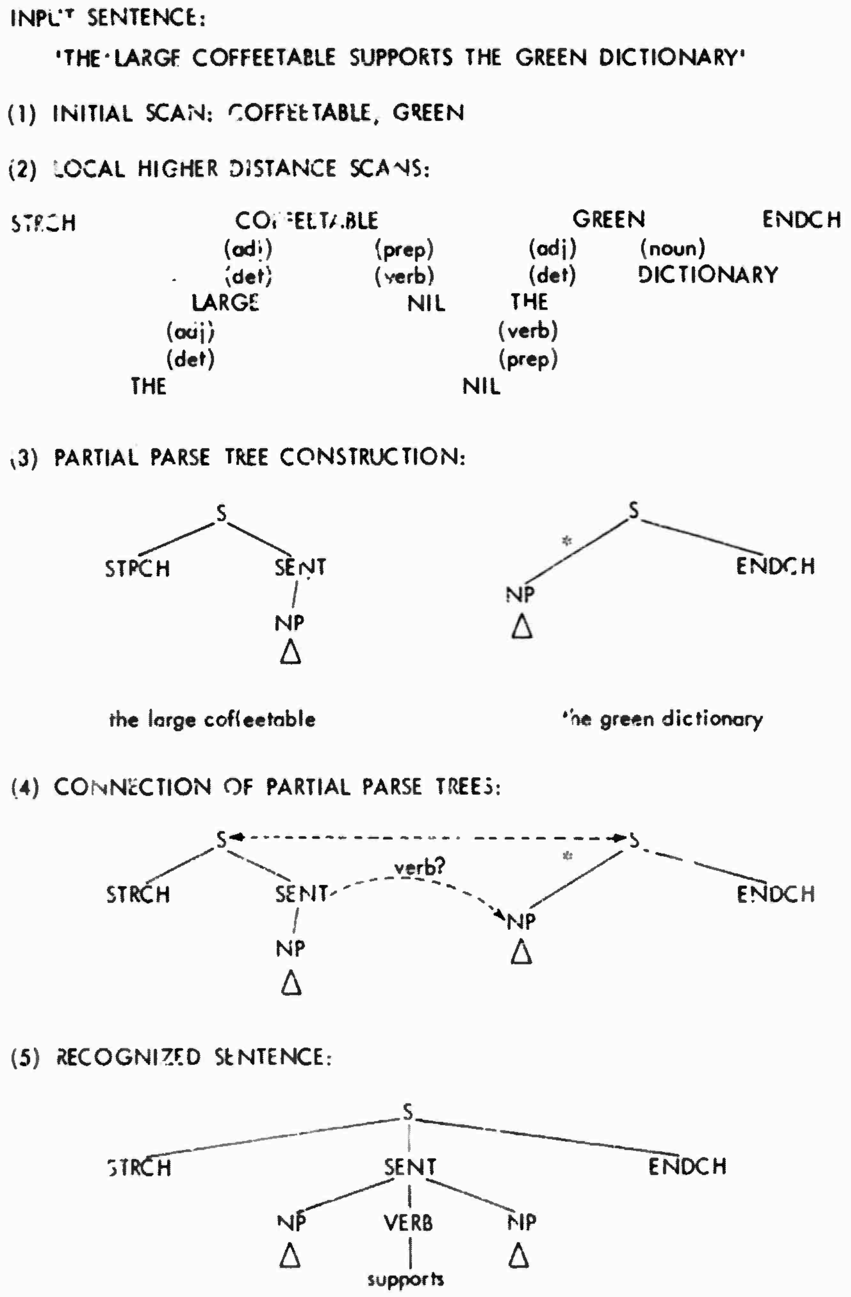


Fig. I-2. Simple example of PARS in operation.

These word candidates are turned over to the higher-level part of the system which initiates scans for small words and for more highly garbled words in the areas adjacent to and between the words found in the initial scan, and groups the words together into parse structures. This processing is done systematically in an attempt to uncover the entire utterance.

3. Simple Example

Figure I-2 gives a simplified example of LPARS in operation. The input utterance being processed, "The large coffeetable supports the green dictionary" is spoken, analyzed by a front-end phoneme recognizer, and a string of phoneme candidates is produced for input to LPARS.

LPARS first makes an initial scan through the sentence at a low phonetic distance looking for longer words which are not too garbled. Let us assume that in this instance the two word candidates "coffeetable" and "green" are found. These word candidates are turned over to the high-level part of the system, together with a start-of-sentence character (STRCH) and an end-of-sentence character (ENDCH) which the system adds.

The higher-level analysis consists of three fairly distinct stages, as Fig. I-2 indicates. First, scans at higher phonetic distance are made based on fairly local cues, in the areas surrounding the word candidates. In this example, these initial higher-distance scans are very simple. A scan in front of the noun uncovers the adjective "large", and a further scan in front of that word uncovers the determiner "the". A scan to the right of the noun for prepositions and verbs fails to find any word candidates. This means that the verb "supports" is too garbled to be picked up either by the initial scan or by the higher-phonetic-distance selective scan. Similar scans around the adjective "green" uncover the words "dictionary" and "the", but again fail to find the verb.

After these higher-distance scans have taken place, the system builds up local parse structures in the sentence. If all the words in the sentence have been found, then the entire sentence is constructed. Otherwise, the result is a number of partial parse trees in different parts of the sentence.

The system attempts to construct as many such partial parse trees as it can with the words it has found. In this example, it constructs only two trees. The first tree (STRCH the large coffeetable) is straightforward; the second tree (the green dictionary ENDCH) is somewhat unusual since it contains an "ancestor link" (the link labeled *). Such ancestor links are necessary in LPARS because of a fundamental problem of local parsing: namely, that it is not always clear from a small amount of local context what the global relationship is between two adjacent syntactic units. In this case, the exact relationship of the noun phrase to the end-of-sentence character is not clear. On the other hand, for the purposes of building local structure, LPARS need not know the actual relationship. The ancestor link allows LPARS to recognize that many syntactic relationships are possible, but to defer commitment until later when an attempt is made to join this structure to other structures by proposing words between them.

The third and final step in the parsing process consists of connecting partial parse trees to one another by using the grammar to propose words that might exist between them. Any words proposed are tested against the input at even higher phonetic distances than the previous scans.

In the example given here, the algorithm discovers that the two parse trees can be connected by a verb. It therefore initiates a higher-distance scan which succeeds in finding the verb "supports". Thus, the entire sentence is recognized.

One of the advantages of the local approach is that this algorithm can attempt to connect the most promising partial parse trees first (for instance, the longer ones). In this way, the local organization allows the system to take advantage of the maximum amount of well-matching information from all parts of the sentence to direct attention to areas where highly garbled words might exist, and to indicate what words might reasonably make sense there. A left-to-right parser would be unable to take advantage of such information.

Notice that this example is a simplified one – no erroneous words were found. In a more realistic example, erroneous words would be found, some local structures containing these words would be built up and, additionally, some erroneous local parsings of the correct words could be constructed. Hopefully, attempts to build out upon the erroneous structures would eventually prove unsuccessful, while attempts to build out upon the correct structures would usually succeed.

Notice also that, in this example, semantics have not been mentioned. Semantics enter into LPARS' analysis primarily by reducing the number of words that need be sought by the higher-distance scans. For instance, tests for adjectives preceding a noun need only look for adjectives that may legitimately modify that noun.

4. Summary Overview

Figure I-3 gives a condensed overview of the procedure we have just worked through. LPARS receives its input in the form of phoneme candidates. It makes an initial scan at a low phonetic distance yielding a list of word candidates, some of which may be incorrect. It uses these to initiate higher-distance scans, to construct partial parse trees, and, if lucky, to uncover the entire sentence. Finally, LPARS investigates ways to connect the partial parse trees together to build up the whole sentence. Section VI describes this process in more detail.

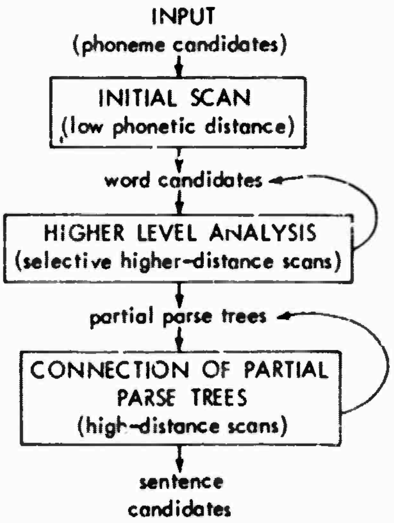


Fig. I-3. Overview of LPARS.

F. RELATED WORK

Two useful surveys of speech literature, with extensive bibliographies, are given in Refs. 1 and 2. Previous work related to the present research can be separated into four fairly distinct topics.

1. Isolated-Word Recognition

Numerous different approaches have been taken to the problem of recognizing single words spoken in isolation, given a fairly small vocabulary. Gold,[5] and Bobrow and Klatt[6] achieved 86- to 95-percent recognition, each using a vocabulary of 54 words. In both cases, recognition was done without trying to segment and identify the phonemes that made up the utterance.

Medress[7] and Vicens[8] developed systems which performed the intermediate step of attempting phoneme identification before trying to identify the word spoken. These systems achieved recognition accuracy similar to that of Gold and Bobrow. The Vicens system used a vocabulary of 561 words.

In the present research, we assume that input to LPARS is in the form which is output by the Medress and Vicens systems: a string of phoneme candidates which may contain large amounts of ambiguity and error.

2. Continuous-Speech Recognition

Vicens[8] also coupled his phoneme recognizer to a very primitive continuous-speech system which worked by making an initial scan through the utterance searching for a certain highly recognizable word, and then used a very constrained syntax together with a very small vocabulary to guide its search for the rest of the words in the utterance. LPARS can be thought of as an extensive generalization of this simple system.

Reddy[9] recently demonstrated a system which recognizes spoken chess moves, expressed in a simple nonrecursive syntax.

The system of Tappert and Dixon[10,11] achieved good results by attempting to identify "transemes" (phoneme-phoneme transitions) before matching the utterance as a whole. Rabinowitz,[12] working with Tappert and Dixon, experimented with a left-to-right continuous-speech recognizer for a simple nonrecursive syntax, with success rates ranging from 65 to 90 percent.

3. Natural-Language Processing

Most work done in recognition of natural-language input has concentrated on left-to-right algorithms. The work of Woods[13,14] and Winograd[4] on natural-language parsing is closely related to the present research.

Woods uses the formalism of an augmented transition net grammar to capture the regularities of syntactic structure. Our work also uses this augmented transition net formalism, but it is incorporated into a local rather than a left-to-right framework.

Winograd uses PROGRAMMAR procedures and a set of systemic syntactic features to embody his parsing logic. The syntactic features used in our work are very similar to Winograd's.

4. Vertical Systems

The M.I.T. AI Laboratory vision system uses the basic idea of a vertical system (see Ref. 3) which allows global structure to be used to help guide the more local processing, thus saving a great deal of unnecessary work. In recognizing a scene of blocks, for instance, not all lines must be recognized by the line recognizer before higher-level analysis is invoked. In such situations. higher-level analysis can propose possible missing lines and direct finer statistical tests to look for them.

An interesting part of Winograd's system[4] is its ability to call on semantic analysis with partially recognized structures, in a vertical fashion, allowing semantics to abort nonsensical local parses early in the analysis. Thus, semantics plays a confirmational role, answering "yes-no" questions, but does not exercise overt control.

In LPARS, semantics performs a more active function by suggesting possible action in a "creative" role, such as supplying a small list of word possibilities to test locally against a specific section of the input.

G. OVERVIEW OF SECTIONS II THROUGH VIII

Section II describes the transition network formalism, the set of syntactic features used to augment it, and the fairly simple semantic component of LPARS. The transition network formalism forms the backbone of LPARS' parsing activity.

The various sub-parts of the LPARS system are described in Secs. III, IV, and V. Section III describes how "scrambled" input is prepared for LPARS by a front-end simulator, and how word recognition is performed. Section IV describes the initial stages of LPARS' processing: the initial scan, and the fairly local higher-phonetic-distance matching. It also discusses some interesting ways in which this part of LPARS could be extended. The two main algorithms used by LPARS in its local syntactic processing are outlined in Sec. V.

Section VI describes the overall coordination of the LPARS system, and discusses some basic problems and considerations in the design of a locally organized system such as LPARS.

The experimental evaluation of LPARS is described and some sample runs are exhibited in Sec. VII.

Finally, Sec. VIII summarizes the various issues which have been dealt with in developing the local approach toward parsing, and discusses some interesting implications of the approach.

II. A TRANSITION NETWORK GRAMMAR WITH SYNTACTIC FEATURES

LPARS uses an augmented transition network grammar as the basis for its syntactic activity. The transition network formalism used by LPARS is somewhat different from previous transition net parsers such as Woods'.[13,14] A few of these differences are primarily in form: others are significant changes made to accommodate the local nature of LPARS' operation.

We now describe how LPARS has adapted the transition net grammar to accommodate its local parsing purposes. Section A introduces the transition network formalism; Sec. B describes how LPARS augments its transition net grammar by a set of syntactic features, and discusses some of the advantages gained thereby; in Sec. C, the syntax used by LPARS is presented, and its restrictions are discussed; Sec. D describes the semantic component of LPARS; and, finally, the action routines which are associated with the arcs of the grammar are discussed in Sec. E. The action routines keep track of the syntactic features and examine the semantic features of structures being parsed, to enforce consistency between different parts of the sentence.

Two basic problems are confronted here:

(1) How to organize a grammar so that recognition can be started easily anywhere within a sentence

(2) How to organize a grammar so that, when local structure building occurs, any potentially exponential inefficiencies are kept within reasonable limits.

A. TRANSITION NET FORMALISM

This section describes the basic structure of a transition net grammar. The description is kept quite informal; however, as the chapter progresses and more detailed examples are presented, the nature of the transition network as used in LPARS will become more concrete.

A transition net grammar consists of a number of finite-state networks (nets) each defining a nonterminal syntactic class (i.e., noun phrase, sentence, etc.). The idea of representing nonterminal syntactic classes by such transition diagrams was previously developed by Woods,[14] whom we quote:

> "The transition network grammar model is an extension of the notion of state transition diagram well-known to automata theory. A transition network grammar consists of a network of nodes with arcs connecting them. The nodes represent states of a hypothetical parsing machine and the arcs connecting them represent possible transitions and are labelled with the types of events in the environment of the machine which permit transitions. In the case of a transition network grammar, the types of events are the occurrences of words and phrases in the input string upon which the grammar is operating."

Figure II-1 gives a simple example of such a grammar.

Each net is made up of a number of states connected by arcs. Each arc is labeled either by the name of a terminal syntactic class (noun, verb, determiner, etc.), or by the name of a nonterminal. Terminal arcs correspond to transitions permitted by a single word being processed in the input stream. Nonterminal arcs require the recursive invocation of a nonterminal net to recognize a phrase or clause. In LPARS, an arc can also be labeled by LAMBDA; this arc allows an immediate transition without processing any input. Each net contains at least one initial state (marked ○) and at least one final state (marked ⊙).

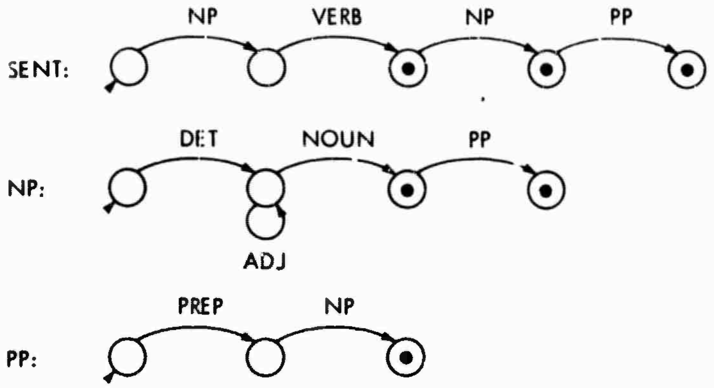


Fig. II-1.

Associated with each arc, and with each initial and each final state, is an action routine which makes tests to determine when a transition is permitted and when a state can be an initial or a final state. These tests keep track of a set of syntactic features and other information about the constructs being parsed. Thus, the transition net is "augmented" by the activity of these action routines. The operation of these routines is discussed in Sec. II-E.

Thus, for each nonterminal, a transition network defines the structures which can form that nonterminal. For instance, in the simple grammar above, a noun phrase (NP) is defined as a determiner (DET), followed by zero or more adjectives (ADJs), followed by a noun, and followed optionally by a prepositional phrase (PP). For a string of words to be accepted by a transition net, that string must permit a sequence of transitions through the net beginning on an initial state and ending on a final state. To allow this, the action routines associated with the arcs traversed, and with the initial and final states, must permit the sequence of transitions.

In a left-to-right parser, these transition nets can be thought of as a flowchart of the parser's action when analyzing a sentence. In a system such as LPARS, which performs local parsing, the transition nets are best thought of rather as defining acceptable left-to-right adjacencies for syntactic structure.

B. SYNTACTIC FEATURES

Central to the LPARS' syntactic formalism are a set of syntactic features with which LPARS augments the transition net grammar. Syntactic features make it easier to organize the grammar so that parsing can be started in the middle of a transition net, yet still proceed in an organized fashion. Syntactic features also allow a quite compact grammar to be created. This section describes how these syntactic features are used and why they are important. To this end, we first work through a simplified example.

1. Example: Relative Clauses

A relative clause can be thought of as a permutation of the structure of an English sentence. In transformational theory, a relative clause is thought of as being formed by moving a noun phrase to the front of a sentence, replacing it with a relative pronoun. For example, if one assumes the following simple underlying structure for a sentence:

NP-V-NP-PP (noun-phrase, verb, noun-phrase, prepositional-phrase) as in "the man places the book on the table", then the relative clauses shown in Fig. 11-2 can be created. (In this figure, the terminal WH represents a relative pronoun: who, whom or which.)

ORIGINAL SENTENCE:							
0.			NP	V	NP	PP	(declarative)
RELATIVE CLAUSES:							
1.			WH	V	NP	PP	(subject r.c.)
2.		WH	NP	V	–	PP	(object r.c.)
3.	PREP	WH	NP	V	NP	–	(prepositional r.c.)

Fig. 11-2.

In structure 1 (WH V NP PP), the first NP (already at the start of the clause) is relativized: as in "... who places the book on the table...".

In structure 2 (WH NP V PP), the second NP is brought to the head of the clause and relativized: as in "... which the man places on the table...".

In structure 3 (PREP WH NP V NP), the third NP together with the preceding preposition is brought to the head of the clause and relativized: as in "... on which the man places the book...". Notice that a fourth structure can also be formed with the preposition remaining at the end of the clause: as in "... which the man places the book on...".

These three relative clause structures can be labeled "subject relative clause" (SRC), "object relative clause" (ORC), and "prepositional relative clause" (PRC); these names indicate the noun phrase in the original declarative (DCL) form which was relativized. Later in this example, these names are used as "syntactic features" to help guide the parsing process.

Thus, Fig. 11-2 outlines in tabular form a set of "linguistic facts" describing the structure of a declarative sentence and of three types of relative clauses. Figure II-3 shows a possible way that these structures might be represented in a transition net grammar.

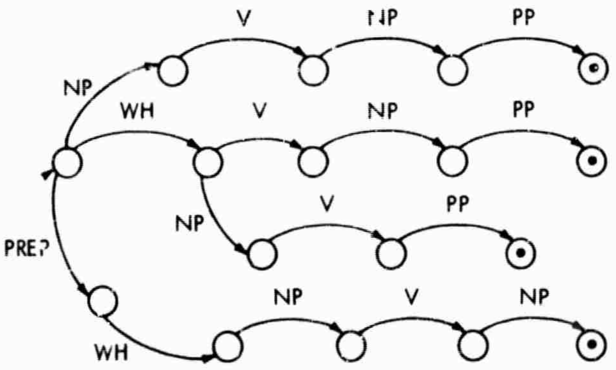


Fig. II-3.

In this transition network, each structure is spelled out by a separate branch of the network. Notice, however, that there is a great deal of similarity between the different branches. This approach to representing syntax, alt.1ough inelegant, would probably not be too costly in a left-to-right parser, since the parser would n.. progress too far down a dead-end branch before backing up.

For LPARS, however, such an approach would be very inefficient because, due to its local parsing activity, the syntactic part of LPARS must construct as many valid parse structures as possible from the words passed to it. Since LPARS has no way of knowing the correct left-to-right context in which a set of words might exist, it must assume that any context is possible. Thus, for example, if it finds a noun phrase to the left of a verb, LPARS must construct one parse tree for every instance in the grammar where an NP arc adjoins a verb arc. If there are thirty such instances, then thirty parse structures must be created.

It would be inefficient for LPARS to represent a good-sized subset of the different structural permutations of an English sentence by "spelling out" each permutation as in Fig. II-3. Such a system would not only be inelegant, but would not allow efficient extension. The inherent inefficiencies would rise exponentially as more-and-more syntactic structure was incorporated.

LPARS avoids this problem by using syntactic features which allow us to collapse a number of different permutations of a sentence, for instance, onto a single transition net skeleton. Let us see how this is done, continuing the relative clause example.

The linguistic facts portrayed in Fig. II-2 can be described using a transition net with four syntactic features: DCL, SRC, ORC, PRC, standing for each of the four possible sentence permutations. Each syntactic feature represents a particular phenomenon which might or might not be present. Therefore, each syntactic feature has associated with it a Boolean value of either "+" or "–" indicating whether that phenomenon might be present or not.

Let us see how such features can be incorporated into a transition net. Figure II-4 shows a transition net for the declarative sentence. The feature (DCL+) is associated with each arc. (In the LPARS system, these features are actually built into the action routines which are associated with each arc.)

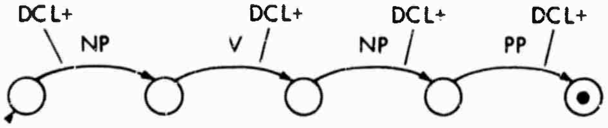


Fig. II-4.

Now let us incorporate the subject relative clause structure into this net. To do this, we add an arc labeled WH with the features (DCL–, SRC+) and label the other arcs with the appropriate SRC feature as shown in Fig. II-5.

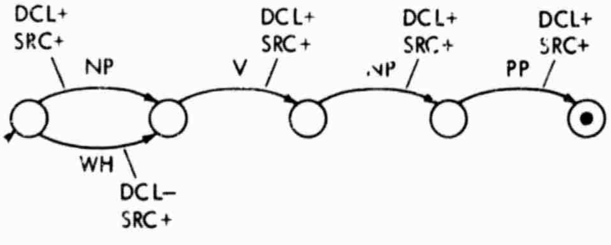


Fig. II-5.

The syntactic features associated with each arc represent tests which must be performed in the action routine associated with the arc. Exactly how these tests operate is described in Sec. II-E.

Now let us add the object relative clause structure. This can be done by adding a second possible initial state, a WH-labeled arc from that state to the first initial state, and a LAMBDA arc to bypass the second NP, as shown in Fig. II-6.

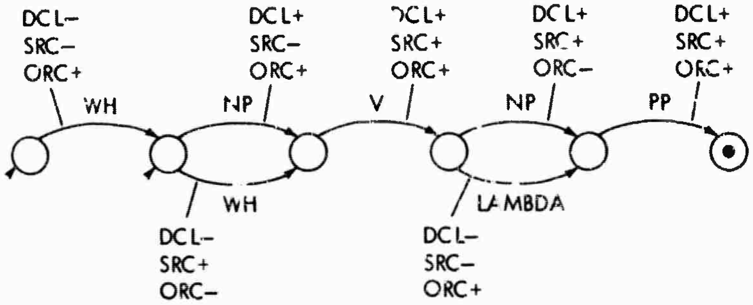


Fig. II-6.

A LAMBDA arc is one which can be traversed without parsing any input. Notice that it is only necessary to add a small amount of structure in the form of states or arcs when each new structural permutation is incorporated into the existing net. Most of the structure is carried in the syntactic features associated with the arcs.

Finally, the prepositional relative clause structure is added as shown in Fig. II-7. Here, an additional initial state and an arc labeled PREP are added, and a previous state is made a possible final state.

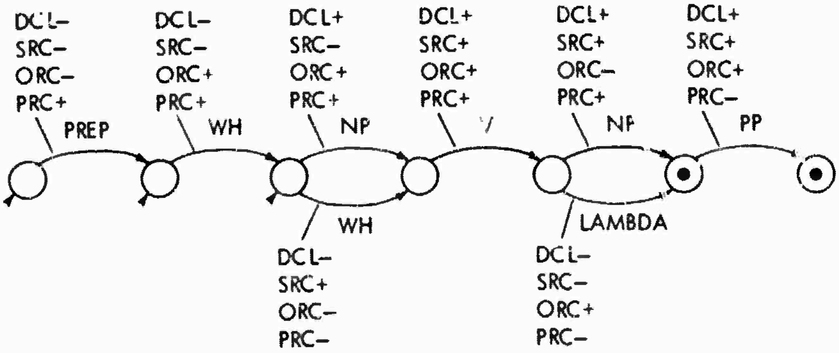


Fig. II-7.

To make this net complete, a set of initial and final state tests must be created to indicate which permutation can begin and end on the various initial and final states. Thus, the following feature sets must be associated with the initial and final states:

Initial States

1. (DCL−, SRC−, ORC−, PRC+)
2. (DCL−, SRC−, ORC+, PRC−)
3. (DCL+, SRC+, ORC−, PRC−)

Final States

6. (DCL−, SRC−, ORC−, PRC+)
7. (DCL+, SRC+, ORC+, PRC−)

This new net captures the same set of linguistic facts as Figs. 11-2 or 11-3.

The list of features associated with each arc can be considered an elimination list. In this simple example, if parsing is proceeding from left to right along the arcs, one can keep a feature list and cross off any feature marked "−" on the arc that one is about to take. If all features are crossed off, then that arc cannot be taken; if not, then any features that remain are possible global interpretations for the structure parsed so far. In LPARS, the tests which keep track of these syntactic features are performed in the action routines (as described in Sec. 11-E).

Notice that the parsing process can be started at any place in the net (not just at an initial state), and still proceed in a syntactically meaningful fashion. Notice also that this new transition net does not have the undesirable features of the net in Fig. 11-2. If a noun phrase is found to the left of a verb, only one parse structure need be created, as shown in Fig. 11-8.

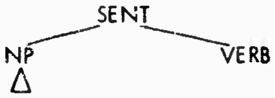


Fig. 11-8.

Section V describes the parse structures created by LPARS in more detail. This parse structure would have associated with it the syntactic features (DCL+, SRC−, ORC+, PRC+) indicating that it could be part of a declarative sentence or of an object or prepositional relative clause. As parsing continues, these features restrict the arcs which can be taken.

In this way, syntactic features provide a way of letting structure recognized in one part of a sentence influence parsing done in another part of the sentence, while keeping a very streamlined transition net skeleton.

The crossing off of syntactic features in this simple example was very straightforward. In a more complex example, the logic in the action routines might involve more complex tests before crossing off a given feature Thus, more complex functions can be associated with the arcs to test the syntactic features.

The basic concept of keeping a list of syntactic features, which indicate different possible global interpretations for the structure parsed up to a given point, is a very general one and is extendable to a wide range of syntactic phenomena.

There are two major advantages gained by using syntactic features: (1) They allow the grammar to be expressed compactly, thus cutting down on potentially exponential inefficiency; and (2) the syntactic feature logic tends to have little left-to-right bias, and therefore makes it easier to create a system in which parsing can start at any node within a network, yet still proceed in a well-defined fashion.

2. Tradeoffs in Grammatical Complexity

In using syntactic features to achieve a streamlined syntactic skeleton, we are really trading off between complexity of the transition net structure (arcs and states) and complexity of the logic in the action routines that process syntactic features. This tradeoff allows a reduction in the number of partial parse structures which must be constructed, but increases the amount of information (in the form of syntactic features) which must be associated with the parse structures. For the purposes of LPARS, this tradeoff is a useful one.

3. Generality of This Approach

As described above, LPARS uses a transition net grammar augmented by syntactic features as the basis for its parsing. The grammar which LPARS uses is, however, a very restricted one. It is therefore worth discussing how general and extendible the approach is.

This section broaches this issue by first relating syntactic features to transformational grammar[15,16] and then to the systemic grammar used by T. Winograd[4] in his natural language parser. It turns out that syntactic features, if created by looking at a set of transformations, can be related quite directly to these transformations. Syntactic features are also very similar to the systemic grammar features used by Winograd. These comparisons are made to help put the syntactic features discussed in this report into perspective with respect to other syntactic representations.

a. Syntactic Features vs Transformations

A transformational grammar includes a set of rules which take the "deep structure" of a sentence (its underlying form) and successively transform it to generate a "surface structure" (the form which would be spoken or written). Most linguistic theory is couched in this generative transformational framework.

This section argues that a transition network with a set of syntactic features can be related quite directly to a transformational grammar, if the transition network is built up by systematically applying different transformations to a simple sentence structure. Section II-B-1 gave a simple example of how this can be done for relative clause formation.

It is not a trivial task to change a transformational grammar into an equivalent transition net grammar augmented by syntactic features. There are, however, a number of principles which can serve as guidelines. The simplest principle is first to take a transition net representation of a declarative sentence (a simple example is shown in Fig. II-9).

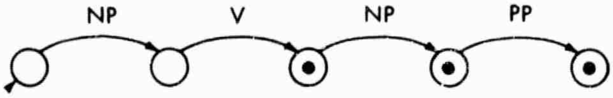


Fig. II-9.

The transformations are then taken one-by-one in the order in which they can be applied and, in a heuristic fashion, appropriate structure is added to the transition net for each transformation in turn. A very simple example of how this can be done was given in Sec. II-B-1. Incorporating the effects of a transformation into a transition net structure involves usually adding some arcs, sometimes adding additional states, and always adding one or more syntactic features.

This incremental incorporation of transformations was used to create the transition net grammar of LPARS. It is not clear how easily one can create an efficient transition net for a complex grammar by this process. The LPARS grammar does not, for instance, incorporate any of the complex cyclic effects that can arise in some transformations. In general, the creation of an efficient transition net/syntactic feature grammar from a set of transformations is probably more an art than a science, although one might be able to automate the process.

The basic point, however, is that a transition net grammar with syntactic features can be written so that its creation, and therefore its structure, bears quite a direct correlation to a set of transformations.

An incidental point in this regard is that LAMBDA transitions can model quite naturally the effect of a transformation which moves a structure from one part of a sentence to another. For example, let us assume that a transformation can move a NP from one place to another. We can create a feature labeled A to represent this change, and then put the structure shown in Fig. 11-10 into the transition net.

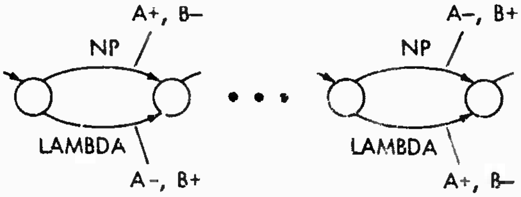


Fig. 11-10.

If the transformation can move the NP from position 2 to position 1, the feature (A+) implies that the transformation has been applied, and (B+) implies that it has not. When parsing, if the NP is recognized in position 1, then the features (A+, B–) imply that the LAMBDA transition must be taken later. Similarly, if the LAMBDA transition is taken in position 1, the features (A–, B+) imply that the NP arc must be taken in position 2. (An illustration of this very phenomenon was seen in the relative clause example. The ORC feature implies that the direct object NP has been moved to the front of the sentence and relativized. Thus, a LAMBDA transition, with the ORC feature labeled "+" and other features labeled "–", is included in the net to bypass the direct object NP.)

b. Syntactic Features vs Winograd's Grammar

In T. Winograd's natural-language parser,[4] a set of "systemic features" was used which was based on the notion of a systemic grammar. These features allowed syntactic constructs which have been recognized to interact with parsing done later in the sentence. The syntactic features which LPARS uses to augment its transition net are very similar, if not completely identical in spirit, to these systemic features.

The features used by Winograd incorporated many more different syntactic structures than does the limited set of syntactic features used by LPARS, but were implemented only in a left-to-right parser.

It would probably be quite straightforward to extend the LPARS grammar directly to incorporate the systemic set of features used by Winograd.

c. Extendability and "Syntactic Economy"

The basic issue in discussing the extendability of the LPARS parsing approach is not really whether syntactic features can accommodate a wider range of syntactic structure. We feel that Winograd's system demonstrates that this is possible. The real issue is whether, at the same time, the system can retain the desired "syntactic economy" in the sense of a streamlined transition net skeleton.

The use of syntactic features allows LPARS to represent many different permutations of the structure of a sentence on a single transition net skeleton. Thus, for instance, if a NP is found to the left of a verb, as discussed above, only one syntactic structure need be created. Therefore, a potential exponential explosion is avoided.

It is certainly conceivable that with a more comprehensive grammar, such a dramatic collapse onto a single syntactic skeleton might be difficult to achieve. On the other hand, a great deal of collapsing of syntactic structure onto common syntactic framework certainly would be possible.

Thus, the important point, from the standpoint of this report, is that the use of syntactic features to augment a transition net is a very powerful tool which allows many different syntactic structures to be combined in a single syntactic skeleton, and which dramatically reduces the potential exponential inefficiencies inherent in a local approach to parsing, and therefore makes these potential inefficiencies much more readily managed and easier to tolerate.

The problems of ultimate extendability of this approach to a comprehensive grammar have not been systematically explored.

C. LPARS' GRAMMAR

The grammar which LPARS uses contains the four syntactic features discussed above, along with seven others. These syntactic features allow LPARS to accommodate an interesting, although admittedly restricted, subset of English syntax.

LPARS' grammar allows the system to recognize the following sentence structures: declarative, imperative, the three relative clauses discussed, yes-no questions, and three types of WH-type questions. The system accommodates both active and passive forms of all these forms (except the imperative).

1. LPARS' Syntactic Features

This section discusses each of the eleven syntactic features used by LPARS and demonstrates their effect in sample sentences. These features can be separated into two groups, or "systems," of features. The implication of the separation for LPARS' operation is discussed later.

The first system of syntactic features is:

DCL (declarative)	As in "The table supports the book" or "The man puts the book on the table".
IMP (imperative)	As in "Place the book on the table".
QQ (simple question)	As in "Does the table support the book?"
SRC (subject relative clause)	As in "which supports the book".
ORC (object relative clause)	As in "which the table supports".
PRC (prepositional relative clause)	As in "on which the man places the book".

SQ (subject question)	As in "What supports the book?"
OQ (object question)	As in "What does the table support?"
PQ (prepositional question)	As in "On what does the man place the book?"

The other system of syntactic features includes only two features: ACT (active voice) and PASS (passive voice).

All the examples given above illustrate the active voice. Sample passive sentences are: "The book is supported by the table." and "What is placed on the table by the man?"

2. LPARS' Transition Network

Figure II-11 shows the set of transition networks used by LPARS; they define four nonterminals:

SENT 1	a sentence, bracketed by start and end characters
SENT	a sentence in any of the different permutations discussed above
NP	a noun phrase
PP	a prepositional phrase

(Notice that this grammar differs slightly in form from some grammars since there is no "verb phrase" nonterminal. Rather, the SENT nonterminal has incorporated directly the verb phrase structure.)

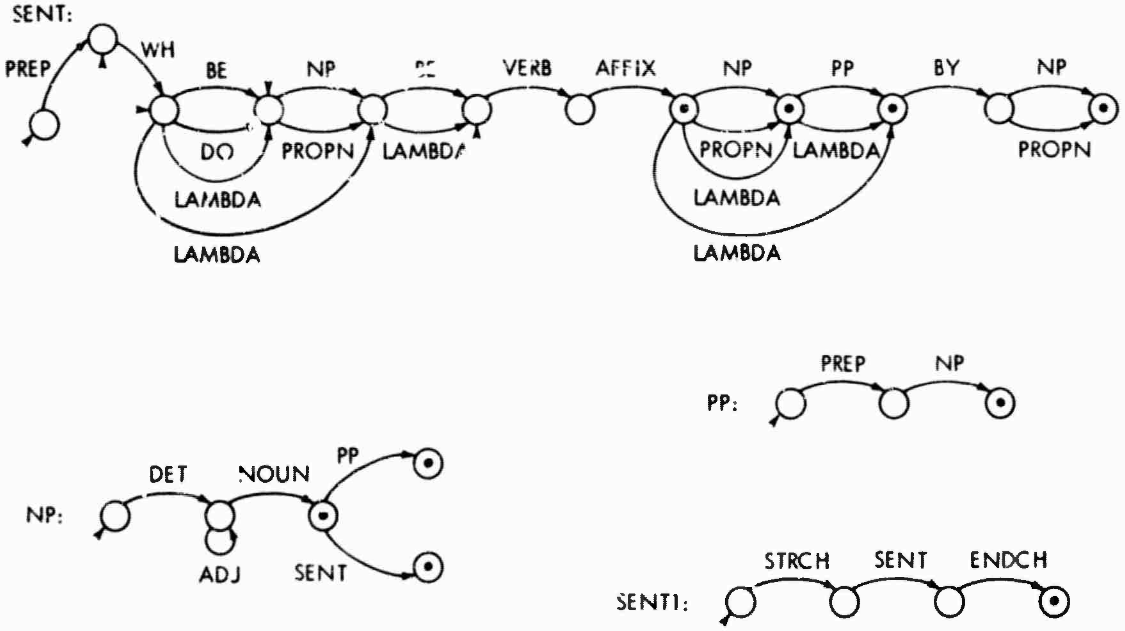


Fig. II-11. LPARS' transition network grammar.

The grammar also allows fourteen terminal syntactic classes which can be grouped into four rough categories:

(a) Six of the terminals are straightforward: noun, verb, adjective (ADJ), determiner (DET), preposition (PREP), proper noun (PROPN).

(b) Four are special function words:

WH – WH-type words (such as "who", "whom", "which", and "what").
DO – the function word "does".
BE – the function word "is".
BY – the function word used to make a passive form, "by".

(c) Two are artificial terminals included to let the grammar know whether it has recognized all the input:

STRCH – the beginning of the input
ENDCH – the end of the input.

(d) The final terminal, AFFIX, is really a subword morph (word affix). It represents the ending of a verb, which in LPARS can be "s" (active present tense), "ed" (passive), or NIL (imperative or root). (LPARS expects all verbs to be regular.) The AFFIX arc is treated very much like a LAMBDA arc by most of LPARS' algorithms.

3. Restrictions of LPARS' Grammar

As is now probably very clear, the syntax which LPARS recognizes is quite restricted. This section describes some of these restrictions explicitly and discusses why they were made.

(a) Limited sentence structures: The emphasis in LPARS is to develop a localized approach to parsing. Therefore, we have kept the syntax quite limited to allow us to focus more fully on the basic problems involved in the local approach.

(b) Small Morph Problems: Due to the crudity of current phoneme recognizers, we decided to limit the amount to which small words and small morphs were allowed. This decision results in a number of restrictions: only singular noun phrases are allowed (therefore, no subject-verb number agreement checks are necessary); only present tense is allowed, although both active and passive and root forms of the verb can be used (thus, no complex affix hopping considerations need be considered); also, the use of pronouns is not handled.

(c) To make the grammar simpler and more manageable, a few other restrictions are made: only one clause can modify a noun (although an arbitrary number of recursively nested modifying phrases is allowed); also, conjunctions are not allowed.

(d) A further restriction which deserves separate mention, is that left recursion is not allowed (i.e., allowing a nonterminal to start with itself). The local parsing algorithms discussed in Sec. V would have to be somewhat modified to accommodate left-recursive structures.

These are some of the most glaring restrictions which LPARS makes. Clearly, there are numerous complex and not-so-complex linguistic constructs which LPARS also does not account for. None of these restrictions is made necessary by any inherent incapability of the LPARS approach. Rather, we felt that, due to the crudity of current front-end analyzers, it would be overly ambitious to try to incorporate too much structure at this time, particularly structure that depended too much on small function words and small morphs such as noun and verb endings. Also, the primary focus of the present work is on developing the basic structure of the local approach, in a limited system using techniques that are general and extendable. Once such an approach is understood, extension is a much easier task.

D. SEMANTIC FEATURES

This section describes the semantic component of LPARS. Consisting primarily of semantic feature checks, LPARS' semantic processing is quite limited. Semantic features are a widely used means of incorporating a semantic component into a syntactic formalism.[17,4]

Each word in LPARS' vocabulary has a list of semantic features describing attributes of that word. Each semantic feature consists of a name and either a value or a list of possible values.

1. Semantic Feature Checks

The feature list of the noun "table" includes the features (ANIM MINUS SUPP-SUB PLUS CONT-SUB MINUS):

ANIM MINUS means that a table is inanimate.
SUPP-SUB PLUS means that a table can support objects
CONT-SUB MINUS means that a table cannot contain objects.

The feature list of the verb "support" includes the features (CONT MINUS SUPP PLUS):

CONT MINUS means that the action is not that of containing.
SUPP PLUS means that the action is that of supporting.

The feature list of the preposition "on" similarly includes the features (CONT MINUS SUPP PLUS).

Two examples of typical semantic feature checks involving these features follow.

Example 1.

If a verb has the feature (SUPP PLUS), check that the subject of that verb has the feature (SUPP-SUB PLUS). Thus, "The kitchen supports the book." would not be recognized as an allowable sentence, while "The table supports the book." would be accepted.

Example 2.

If a preposition has the feature (SUPP PLUS), check that the object of that preposition has the feature (SUPP-SUB PLUS). Thus, the phrase "on the kitchen" would not be recognized as a prepositional phrase, whereas "on the table" would be recognized.

2. Semantic Feature Checks Used by LPARS

There are five different general cases of semantic feature agreement enforced by LPARS. These are all enforced by testing appropriate features, namely:

Subject-Verb Agreement:— The subject must be capable of performing the action associated with the verb. If the sentence is passive, of course, it is the "deep-structure" subject that must be checked for this agreement

The table supports the book.
*The table contains the book.
The book is supported by the table.
*The book is contained by the table.
The man places the book on the table.
*The sofa places the book on the table.

Verb-Object Agreement:– The object must be capable of having the verb action performed to it.

*The table supports the kitchen.
*The man places the fireplace on the table.

Preposition-Object Agreement:– Since a preposition denotes a relation, the object of a preposition must be able to maintain that relationship with other objects.

on the table
*on the kitchen
*in the table

Modified Noun-Preposition Agreement:– Similarly, the noun modified by a prepositional phrase must be able to maintain the relationship specified by the preposition with other objects.

*the fireplace on the table
*the livingroom in the bookcase

Adjective-Noun Agreement:– Only certain adjectives can modify a noun.

the friendly person
*the green person
*the rectangular man
the small dish

As with most attempts to introduce order into linguistic phenomena, it is certainly possible to imagine violations of these simple semantic feature constraints. For instance, it is conceivable that in free speech one might refer to "the rectangular man." As a result, a set of feature checks of the sort used by LPARS is useful only when the scene being talked about is restricted and well understood.

Section II-E describes how the enforcement of these semantic feature checks fits into the transition net grammar.

3. Semantic Features vs Syntax

It has been argued[18] that "semantic" feature constraints of the sort used by LPARS (often called "selectional restrictions") are really syntactic in character. This argument can be understood by observing that one can transform a semantic feature grammar into an equivalent grammar without any semantic features by using many more nonterminal and terminal syntactic classes. Instead of allowing nonterminal nodes to be flagged with the semantic features of component words, the new grammar would have a special nonterminal for each combination of semantic features possible in the old grammar. In this new grammar, all the semantic feature information is implicit in the name of any given syntactic class.

This argument is a valid one. It points up the limitations of the semantic processing presently incorporated into LPARS.

4. Extensions to the Semantic Processing

One interesting extension of LPARS' semantic component would be to let the parsing take place in the context of a particular "scene" being talked about. Thus, if the words "the dictionary on the table" are parsed, these would only be formed into a noun phrase with a modifying prepositional phrase if an instance of such an object existed in the scene being discussed.

Similarly, relative clause constructs such as "which the man placed on the table" would only be recognized if some event had occurred in the course of previous interaction which could be related to the event being described in the clause.

This extension of the system could be a major project. Yet it would fit quite naturally into the overall LPARS framewor Such an approach would involve using semantic processing of a clearly non-syntactic variety.

E. ACTION ROUTINES

We now describe the operation of the action routines associated with the arcs and terminal states of the grammar. The action routines examine syntactic and semantic features to determine when use of the arc or terminal state is to be allowed. Central to these routines are the information lists which they operate upon.

1. Information Lists

As parsing proceeds, the relevant syntactic and semantic information is kept in information lists which are attached to each parse structure built up. For terminals, the information list consists of the list of semantic features associated with that word. For nonterminals, the information list is constructed by the action routines as described below.

An information list has the form of a LISP property list: an alternating sequence of tags and values (TAG1 VALUE1 TAG2 VALUE2 ...). A value may itself be a property list in turn. This property list corresponds quite closely to the set of registers which Woods[13] uses to augment his transition net grammar. Therefore, in this section, the tag-value pairs are referred to as registers.

As parsing takes place, the action routines examine the information in these registers to see if the use of an arc or terminal state is to be allowed. If so, they place updated information into the registers (actually into a new copy of the registers) which can be examined in turn by later action routines.

a. Registers Used by the SENT Net

This section describes the registers used by the action routines in the SENT net. These registers are:

FEATLIST (syntactic feature list) — The value of this register is the list of syntactic features, together with their values (PLUS or MINUS).

SSUB (surface subject) — This register contains the semantic feature information describing the surface subject of the sentence. The register is set by the action routine of the NP arc which corresponds to the surface subject.

SOBJ (surface object)
This register contains the semantic feature information describing the surface object of the sentence. The register is set by the action routine of the NP arc which corresponds to the surface object.

BY-NP ("by" noun phrase, or "passive subject")
This register contains the semantic information describing the deep structure subject of a passive sentence (i.e., the NP "the table" in "The book is supported by the table."). The register is set by the action routine corresponding to this NP in the net.

WH (WH word)
In WH-type questions and relative clauses, this register is set to contain the semantic information describing the WH word (i.e., what, who, whom, which).

PREP (preposition)
In a prepositional relative clause (PRC) or a prepositional question (PQ), this register contains the information describing the preposition.

PP (prepositional phrase)
In a sentence with a verb requiring a prepositional phrase (such as "place", or "put"), this register contains the semantic information describing that phrase.

VERB (verb)
This register contains the semantic information describing the verb.

If, during the parsing of a given sentence (or sentence fragment), any of these structures has not been recognized, then the associated register contains NIL.

2. Two Examples

We now give two examples of how these registers are used. Let us assume that we are using the SENT net to parse the phrase "which the table supports". In parsing this structure, a number of arcs are traversed in the SENT net. To determine that an arc can be traversed, the action routine associated with that arc must be called. That action routine accepts as input two information lists:

(a) The information list of the SENT net parsed so far.

(b) The information list of the terminal or nonterminal currently being parsed (in this example, the WH, NP, and VERB successively).

The action routine returns NIL if the arc cannot be traversed. Otherwise, it returns a new information list representing the entire SENT structure parsed, including the current arc.

<u>Example 1</u>

"which"
The action routine associated with the WH arc in the grammar sets the WH register to contain the information of the word "which". It also sets the FEATLIST register to indicate that the structure is either PRC, ORC, or SRC, either active or passive.

"the table"
The action routine associated with the surface subject NP sets the SSUB register to contain the semantic information associated with the NP "the table". It sets the FEATLIST register to indicate that the structure is either PRC or ORC, and that it is active.

"supports" The action routine associated with the verb arc is called, and since the sentence is active, it checks to see if the NP described by the SSUB register can be the subject of the verb. Had the sentence been passive, it would have checked to see if the SSUB NP could be the object of the verb. If the SSUB NP had not been recognized (as would have been the case when parsing a sentence fragment starting with the verb), then no check would have been made. The action routine then sets the VERB register to contain the semantic information describing the verb The information in the FEATLIST register stays the same.

Example 2

If the words being parsed are '* which the kitchen supports', the parsing would not succeed, since the semantic checks are not satisfied. The parsing of the WH and the NP would be the same as in the previous example, but the action routine associated with the verb would return NIL after it finds that the NP cannot be the subject of the verb.

3. Brief Summary of Action Routines

Each transition arc action routine takes as input two lists of information: one made up by the previous parsing in the net, the other from the terminal or nonterminal currently being parsed.

The routine processes these two lists of information to make sure that the arc can be traversed. If so, a new updated list of information is returned.

The initial and final state action routines are similar but accept only a single input, the list of information made up by parsing of previous arcs in the net, and determines whether the structure which has been parsed can begin or end (as appropriate) at that particular state.

4. Parsing Locally

Since LPARS uses the transition net to parse locally, the action routines must be written so that the system can start left-to-right parsing from any place in the net. It is not difficult to structure the action routines to allow this flexibility, but it does require some thought.

One potential problem is that semantic feature checks cannot assume that a given construct will always have been parsed just because it is to the left of the current arc being traversed. For instance, when parsing a verb in a declarative sentence, one cannot assume that the surface subject will always have been recognized. As a result, any semantic test must first ascertain that the required registers contain semantic information.

Similarly, syntactic feature checking must be set up so that it can start anywhere in the net. This is not difficult to achieve. In fact, one of the major advantages of using syntactic features is that the syntactic feature logic consists mainly of crossing off all features which cannot use an arc or terminal state, and thus has little built-in left-to-right bias.

III. LEXICAL RECOGNITION AND PHONETIC SCRAMBLING

This section describes the approach to word recognition taken by LPARS. Since LPARS is not connected to a working phoneme recognizer, there are two parts to this description: first, how LPARS generates "scrambled input" which simulates input obtained from a phoneme recognizer; and second, how the word recognition algorithm processes this input to reconstruct words which might be present.

In designing the phonetic scrambler and the word recognizer, a balance has been attempted between simplicity on the one hand, and acceptably realistic simulation of real phonetic input on the other. The main emphasis of the present work is on the use of higher-level linguistic constraints to guide lexical analysis, rather than on developing sophisticated methods of performing lexical analysis with a given front end.

There are some quite complex phenomena which can affect lexical recognition with a real front end. These phenomena include coarticulation; supra-segmental effects involving pitch, intensity, and stress; speaker-dependent effects; etc. The present research ignores these phenomena for two reasons. First, any attempt to deal meaningfully with them in the absence of a working front end would be difficult. Second, incorporating such phenomena into the lexical processing of the LPARS system might cloud the fundamental higher-level issues involved in LPARS design.

Thus, the lexical scrambling and recognition scheme used by the LPARS is not presented as the last word in lexical recognition of continuous speech. On the contrary, it is presented as a simple scheme which is easy to understand, and which preserves the basic characteristics of spoken input.

Our work makes only one fundamental assumption – that the matching of a word against a section of input can be done at a variety of levels of confidence. In the present work, phonetic distance is used to express these levels of confidence.

In LPARS, phonetic distances are determined from the confusion statistics used when the scrambler introduces error into the input it prepares for LPARS. With a real front end, of course, phonetic distances would be based on the statistical likelihood of the various errors possible.

A. PHONETIC SCRAMBLING

We now describe SCRAMBLE, the phonetic scrambler which produces input for LPARS. The scrambler accepts as input a string of phonemes correctly spelling an utterance. The scrambler transforms this correct string of phonemes into a scrambled string containing substantial error. The error is created by substitution and deletion of the input phonemes. SCRAMBLE does not, at present, insert phonemes into the input.

In making these substitutions and deletions, SCRAMBLE uses the phoneme similarity table found in Appendix 10 of the ARPA Speech Report.[19] This phoneme similarity table (the "PST"), reproduced as Appendix A of this report, ranks each of 36 phonemes against one another on a scale from 0 (meaning completely dissimilar) to 100 (meaning identical). These similarity measures are a bit artificial since they are based on a somewhat abstract notion of phoneme similarity, rather than on a rigorous evaluation of a specific phoneme recognizer. The table is, however, loosely correlated to the Vicens-Reddy front end.[8] Although this table is used by LPARS for its lexical processing, the approach taken is quite general and is not dependent on the particular statistics in this table.

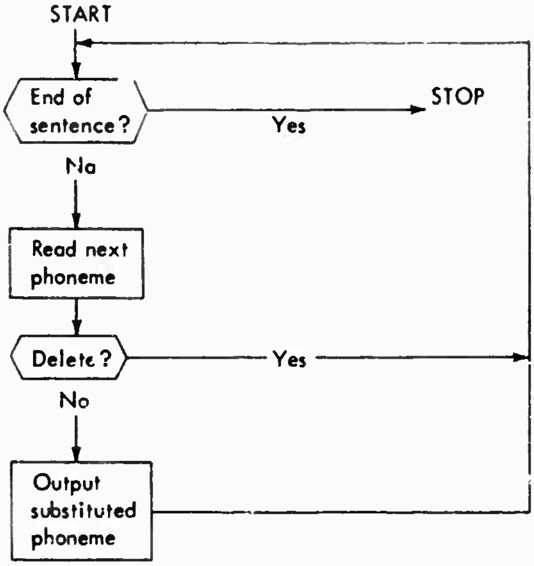


Fig. III-1.

Figure III-1 shows in flowchart form how a correctly spelled sentence is scrambled. SCRAMBLE examines each phoneme in turn, first deciding whether or not to delete that phoneme. If not, SCRAMBLE then decides what phoneme to substitute. These decisions are made randomly, weighted by the numbers in the PST as described in Secs. 1 and 2 below. The scrambling might have been more complete had insertion of phonemes been allowed. This would have required only a simple modification of the system.

1. Substitution

For each phoneme that is not deleted, SCRAMBLE must decide what phoneme is to be substituted for it. The following example describes this procedure.

Let us assume that the correct input phoneme is "T". The PST ranks the similarity of all other phonemes to "T" on a scale of 0 to 100. SCRAMBLE must determine which of these to substitute for the "T". For each input phoneme which is not deleted, the percentage likelihood that SCRAMBLE chooses a substitute phoneme in a given similarity range is:

1 percent	between 0 and 40
1 percent	between 40 and 50
2 percent	between 50 and 60
2 percent	between 60 and 70
4 percent	between 70 and 80
40 percent	between 80 and 90
50 percent	between 90 and 100

(if no phonemes exist in a particular similarity range, a phoneme from the next higher similarity range is chosen).

This particular mapping was chosen by examining tables in Appendix 10 of the ARPA Report,[19] and trying to correlate roughly the PST to the performance characteristics of the Vicens phoneme recognizer.

The actual percentages used can be varied to test the system at various levels of input error. The values given here are those used in the system evaluation described in Sec. VII. Figure III-2

90 to 100:	P,T,D,CH,K,F,TH
80 to 90:	B,J,G,V,DH
70 to 80:	S,H,SW
60 to 70:	Z,SH,ZH,M,N,NG,Y,I
50 to 60:	W,R,L,U
40 to 50:	A,E
0 to 40:	OO,O,AW,AA,AR,AE

Fig. III-2.

indicates which phonemes fall into each of the different similarity ranges with respect to "T" (A list of the phonemes used, together with pronunciation, is given in Appendix A. The phonemic representation used here is that of the ARPA Report. Most phonemes are fairly obvious, except possibly "SW" which is the neutral vowel "ə".)

The result of this scrambling is that, 10 percent of the time, "T" is scrambled quite badly (into a vocalic phoneme or a fricative). The rest of the time the substitution is made almost at random from a number of stop-like phonemes. Thus, the likelihood of preserving the "T" is quite low.

Other phonemes (especially vowels) tend to have fewer phonemes ranked as being very similar in the PST. For these phonemes, the likelihood of preserving the original phoneme is much greater. Thus, SCRAMBLE simulates a front end which recognizes vowels much better than specific consonants. This is consistent with the Vicens-Reddy front end, on which the PST is loosely based.

As discussed in Sec. III-B, the phonetic distances used by LPARS are set up to reflect these scrambling statistics. As a result, any two phonemes with similarity greater than 90 are at a phonetic distance of zero from one another, since at best substitutions are chosen at random from the group of phonemes with similarity 90 or above.

Therefore, although SCRAMBLE chooses a single phoneme to replace each input phoneme, this phoneme really represents a set of phonemes within zero phonetic distance of that phoneme (above 90 similarity in the PST). Thus, the scrambler simulates a front end which often cannot uniquely identify a phoneme, but can only roughly classify it. The Vicens-Reddy front end, for instance, recognizes a stop-like segment as a stop without trying to identify which stop it might be.

As a result, the string of individual phonemes produced by SCRAMBLE is perhaps best envisioned as a string of possible-phoneme lists, each containing several phonemes. Thus, LPARS input contains implicit ambiguity as well as error.

2. Deletions

SCRAMBLE decides whether to delete a phoneme randomly, weighted by its similarity to both of its neighbors. A phoneme bordered by similar phonemes is more likely to be deleted than one bordered by dissimilar phonemes. This approach, suggested in Appendix 10 of the ARPA Report,[19] is a reasonable approximation of front-end behavior since, for instance, one would expect a vowel to be more likely deleted when surrounded by vocalic segments, and a consonant to be more likely deleted when part of a consonant cluster.

SCRAMBLE deletes about 15 percent of the phonemes it processes. It deletes, at most, two phonemes in a row. The probability of deleting a phoneme immediately after a previous deletion is half that of the normal probability.

3. Example

We now present an example of phonetic scrambling to help give a more concrete feeling for LPARS' scrambled input. First, a string of 27 T's is scrambled:

Input

(T T T T T T T T T T T T T T T T T T T T T T T T T T T)

Scrambled String

(TH D T T TH B K F SW CH B B P F F T)

This gives an idea of the type of substitutions made. The number of deletions made here, however, is extremely high, because the likelihood of deletion is affected by the similarity of adjacent phonemes.

Next, similar scrambling is performed with a string of S's and A's:

Input

(S S S S S S S S S S S S S S S S S S S S S S S S S S S)

Scrambled String

(EE S S ZH H ZH S ZH H S H S Z Z H Z SH H)

Input

(A A A A A A A A A A A A A A A A A A A A A A A A A A A)

Scrambled String

(AE A AR AE O A A A A A AE O A TH A AE A O)

Notice that the string of A's (vowels) is not as badly scrambled as the T's or S's.

In the next example, the string "S A T S A T . . . " is scrambled.

Input

(S A T S A T S A T S A T S A T S A T S A T S A T S A T)

Scrambled String

(H A SW H O TH H A DH H A S AE P H A K H AR D S E A)

Here there is much less deletion, since adjacent phonemes are quite dissimilar.

Finally, the sentence "The green coffeetable supports the dictionary" is scrambled. Its correct spelling is:

(TH SW G R EE N K AW F EE T AA B L S SW P O R T S
TH SW D I K SH SW N AA R EE)

Two examples of this sentence, after scrambling, are:

(I D Y I NG DH AA K E SW AA K Y SH SW B O W F H
K SW G K AA SW AA R I)

(SW G L W DH AA EE H AA P U B O K L E Z DH DH I
P H SW M L EE)

Offhand, it might seem unlikely that the original words are retrievable from this scrambled hodge-podge. It is difficult for a person to even match up parts of the scrambled sentence to the original. The computer, however, can do a much more thorough, patient, systematic job of lexical matching than a person can. In fact, the scrambled sentence is somewhat misleading to a human because it is overspecified, since each phoneme really represents a set of phonemes which are equivalent under the scrambling.

B. MEASURING PHONETIC DISTANCE

In LPARS, phonetic distances of individual phoneme substitutions and deletions are added together to determine the overall phonctic distance of a word.

Since the phonetic distances are added together, they must be logarithmically related to the probabilities of substitution and deletion. Thus, if one word matches an input segment but for two errors of 1/3 likelihood each, and another word of equal length matches but for a single error of 1/9 likelihood (a more severe error), the likelihood of each word being present is equal (since the likelihood of two errors of 1/3 probability occurring is 1/3 * 1/3 = 1/9). Therefore, their total phonetic distances must be equal. This will only be true, in general, if phonetic distances are logarithmically related to the error probabilities.

The actual scale used to measure the distances is not important. LPARS measures phonctic distance on a scale from 0 to 120. Phonemes with similarity scores of 90 to 100 are of distance 0 from one another. Phonemes with similarity scores of 0 to 40 are of distance 120 from one another, representing the probability of 1 percent.

Figure III-3 snows graphically how error probability and phonetic distance are related in LPARS. With a real front end, phonetic distances would be based on the statistical probability of the various possible errors. The phonetic distance associated with both substitutions and deletions corresponds to the probability of that error occurring in the scrambler.

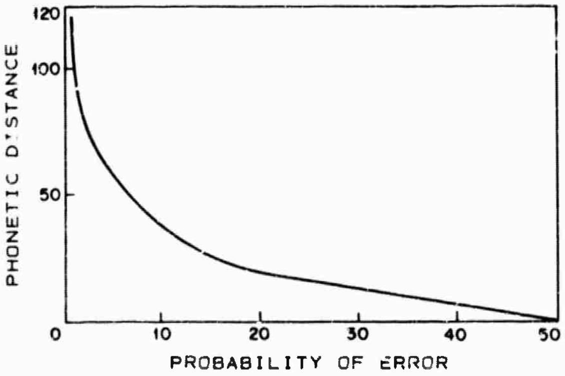


Fig. III-3.

C. WORD RECOGNITION ALGORITHM

This section describes the use of phonetic distance to measure how accurately the phonetic spelling of a word matches a particular section of input.

Figure III-4 shows a simple example of how the matching is done. The dictionary spelling of the noun "dictionary" is compared with three sample sections of input. (At present, the system does not consider multiple pronunciations of a word.) In LPARS, the input quality is much worse than these samples might seem to imply. This example is kept simple for clarity; also, the phonetic distances have been chosen arbitrarily.

Spelling									Total Distance	Distance per Phoneme
D	I	K	SH	SW	N	AE	R	EE		
Input Samples										
F	I	K	SH	SW	N	AE	R	EE	40	4
40										
D	I	K		SW	N	AE	R	EE	50	5
			50							
F	I	K		SW	N	AE	R	EE	90	10
40			50							

Fig. III-4.

In matching a word against a section of input, the phonetic distances of individual substitutions or deletions are added together yielding a total distance for the entire word. The algorithm searches for that combination of substitutions and deletions that yields the lowest total distance.

LPARS always performs its lexical matching using a threshold maximum distance-per-phoneme. During the various levels of LPARS analysis, the threshold gets progressively higher and higher as more-and-more context is used to indicate where a particular word might be present. Only words which match within the threshold are recognized. Thus, if this threshold distance-per-phoneme is 10, no matches of the word "dictionary" are accepted with total distance greater than 90 (10 * the word-length).

A very simple algorithm is used to compute the lowest-distance match of a word to an input segment. The algorithm performs a left-to-right comparison of the dictionary spelling of a word against a section of input, trying all possibilities of substitution and deletion. This process is essentially a tree search, of a substitution-deletion tree, to find that branch of the tree with the lowest score. If the score of any partial branch being explored exceeds the threshold distance, the investigation of that branch is aborted. If all branches are aborted, then the word does not match the input segment within the threshold distance. Appendix C gives a flowchart of this algorithm.

Although the algorithm is very simple, it does generate a great deal of computation. In the initial scan of the utterance, for instance, virtually every word in the vocabulary is matched against the entire input, i.e., starting at every possible phoneme position. This initial scan, of course, is done at a fairly low phonetic distance, which limits somewhat the amount of computation required.

D. SOME COMMENTS

We now make two comments on the approach taken here toward lexical recognition.

1. Local Exponential Explosion vs Vertical Organization

The lexical recognition algorithm clearly has the potential for exponential explosion of computation as the threshold is raised higher and higher, and a deeper-and-deeper search is needed of the substitution-deletion tree when matching words. On the other hand, the initial scan is made at a fairly low-distance threshold (looking for words that are not too garbled), and the amount of work required to match words at a low phonetic distance is less than the amount required for a high threshold.

One of the advantages of the vertical organization of LPARS is that most of the local lexical matching is done at a fairly low threshold, and higher-threshold scans need be done only very selectively, in specific places, looking for specific subsets of the total vocabulary.

2. Extendability of This Lexical Approach

The lexical recognition algorithm is not presented as the last word in lexical matching; it is a deliberately simple scheme. With a real front end, there are many refinements that could make the matching more accurate. The nature of these refinements, however, is not well understood and constitutes a research problem in its own right. In this report, the main emphasis is on the use of higher-level constraints to help guide local processing. Even with more refined lexical matching techniques, the basic concept of initially looking for words that match well, and later selectively searching for words that match less well, would be equally valid

IV. INITIAL STAGES: LOCAL WORD RECOGNITION

The first two stages in LPARS' processing of an utterance are now described. First, the initial low phonetic distance scan through the entire utterance looking for long words (content words) that are not too garbled; and second, the higher-distance matching in the areas adjacent to and between the word candidates found by this initial scan, looking for strings of small function words, and for long words which are more garbled.

During these two stages, no attempt is made to build any syntactic structure. Only very local lexical and boundary criteria are used in the higher-distance matching described here. The final output of this processing is a set of word candidates recognized in all parts of the sentence. Many of [illegible] words may be wrong, many may overlap. Also, there may be gaps in the utterance which are not covered by any word candidate. The goal of this initial matching is to uncover most of the words in the sentence. These words are then turned over to higher-level parts of LPARS which continue the analysis in a more comprehensive, systematic fashion.

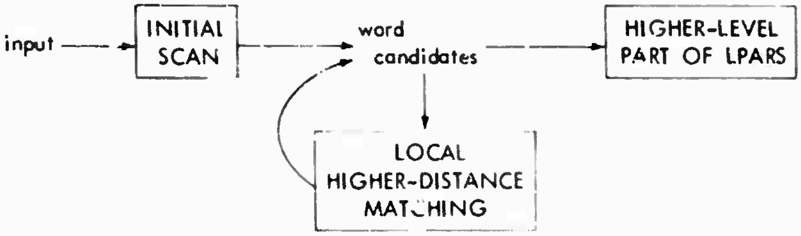


Fig. IV-1.

Figure IV-1 shows how these two stages fit into the overall LPARS framework.

A. INITIAL SCAN

When processing an utterance, LPARS first makes a scan through the entire utterance at a fairly low phonetic distance. This scan searches only for content words (which in LPARS' vocabulary are all at least 4 phonemes in length).

The scan is made by matching the spelling of each word against the input, starting at every possible phoneme position in the utterance. The scan returns a set of word candidates, each of which consists of a proposed word, its starting and ending phonemes, and its total phonetic distance.

The following example shows the results of scanning a scrambled sentence in this manner, at a number of different phonetic distances.

1. Example

The sentence being scanned in this example is "The green coffeetable supports the ashtray". This sentence is scrambled, and then scanned at a number of phonetic distances. (The LPARS vocabulary is given in Appendix B. All content words are at least 4 phonemes in length. The only unusual feature of the vocabulary is that some word strings such as "on top of" and "on the left of" are treated as single-word prepositions.)

SCAN 1: Phonetic Distance = 15, Minimum Word Length = 4

```
((CLUTTER 1 4 60)(COFFEETABLE 7 14 90))
```

This returns a list of word candidates of the following form:

```
(WORD  START-PHONEME  END-PHONEME  TOTAL-DISTANCE)
```

The word "coffeetable", for instance, has been found between phonemes 7 and 14 at a total distance of 90. Since the threshold distance-per-phoneme of the scan is 15, no word is found at a total phonetic distance greater than 15 * its wordlength.

SCAN 2: Phonetic Distance = 20, Minimum Word Length = 4

```
((CLUTTER 1 4 60)(COFFEETABLE 7 14 90)(CLUTTER 13 17 85)
(ROBERT 14 18 105)(CLUTTER 16 19 95)(CONTAIN 20 25 115)
(DICTIONARY 21 28 145))
```

SCAN 3: Phonetic Distance = 30, Minimum Word Length = 4

```
((ROBERT 1 [illegible] 160)(CLUTTER 1 4 60)(PERSON 1 5 170)
(FRIENDLY 1 5 155)(GREEN 3 5 100)(COFFEETABLE 7 14 90)
(CONTAIN 9 13 170)(CABINET 9 15 200)(INFRONTOF 10 18 250)
(TABLELAMP 11 16 210)(ONTHELEFTOF 12 18 290)
(CLUTTER 13 17 85)(ROBERT 14 18 105)(INFRONTOF 14 22 270)
(ENTRANCE 14 19 175)(CHRISTOPHER 16 22 205)
(CLUTTER 16 19 95)(PLACE 16 19 115)(FOOTSTOOL 16 22 195)
(TELEVISION 16 22 195)(INBACKOF 17 22 210)(NEXTTO 17 21 175)
(CABINET 18 23 200)(PERSON 18 22 170)(CONTAIN 20 25 115)
(GREEN 20 22 95)(DICTIONARY 21 28 145))
```

Notice that, as the phonetic distance threshold increases, the number of words found increases dramatically.

SCAN 4: Phonetic Distance = 25, Minimum Word Length = 2

```
((CLUTTER 1 4 60)(FRIENDLY 1 5 155)(THE 1 2 0)
(THE 2 2 15)(GREEN 3 5 100)(THE 3 4 5)(BY 3 5 75)
(THE 4 4 20)(COFFEETABLE 7 14 90)(IN 10 11 45)
(ON 12 13 40)(CLUTTER 13 17 85)(ROBERT 14 18 105)
(WHICH 14 16 75)(ENTRANCE 14 19 175)(CLUTTER 16 19 95)
(TELEVISION 16 22 195)(THE 16 17 45)(CONTAIN 20 25 115)
(GREEN 20 22 95)(THE 20 21 5)(DICTIONARY 21 28 145)
(THE 21 21 20)(THE 23 24 40)(ON 26 27 40)(WHICH 27 29 10)
(WHAT 27 29 75)(WHICH 28 29 25))
```

This scan is made looking for words of length 2 or greater, and illustrates why it makes sense to pick up small function words using context, rather than scanning for them in the initial scan. A large number of such words is found, most of them wrong.

2. Two Possible Extensions

a. Using Inter-utterance Context

One interesting extension of the initial scan described above is to use semantic constraints of a global nature to restrict the words initially searched for. This approach is very much in keeping with the vertical design of LPARS.

In an interactive speech system, if the previous context of the dialog can substantially restrict the words that are likely to be uttered at particular times, then only those words need be included in the initial scan. Only if the results of this scan proved unsuccessful, need one perform the initial scan using a larger part of the vocabulary.

This extension would use inter-utterance contextual information in a vertical fashion to reduce the average amount of computation required per utterance. LPARS presently does not use context of this sort. Designing a system which uses inter-utterance context in an interesting, meaningful fashion is an extensive project in its own right.

b. Using Supra-segmental Information

One of the inefficiencies inherent in LPARS' design is the necessity of matching every word at every phoneme position in the utterance. There is, however, a great deal of supra-segmental information available (see Refs. 20 and 21) in the waveform which might help indicate where stressed syllables are, where phrase boundaries are, etc. Such information could be very useful in helping a locally organized system like LPARS restrict the amount of lexical processing done in the initial scan.

B. LOCAL HIGHER-DISTANCE MATCHING

The words recognized in the initial scan are turned over to LOCALMATCH, a routine that directs higher-distance matching in areas adjacent to and between these words. The set of local scans is described in Appendix E.

Although the LOCALMATCH routine could have been written to operate automatically from an arbitrary transition net, it was written procedurally instead from an inspection of the particular gr[illegible] used by LPARS. Section IV-C discusses some possible advantages of allowing a speech-system designer the flexibility of procedurally expressing the logic which directs such local word searching.

The type of logic performed by LOCALMATCH is quite simple, due in part to the relative simplicity of LPARS' syntax and semantics. Typical actions (all performed at a higher phonetic distance than the initial scan) include:

(1) Matching for semantically acceptable adjectives to the left of nouns (and vice versa: matching for semantically acceptable nouns to the right of adjectives).

(2) Matching for verbs to the right of nouns (and vice versa).

(3) Matching for prepositions to the right of verbs and nouns, and for determiners followed by adjectives or nouns to the right of verbs.

(4) Matching for small function-word strings such as "in the", "on the", "which the", "in which the", "who the", etc., as appropriate, between two nouns or a noun and an adjective which are separated by a small number of phonemes.

(5) Matching for function-word strings such as "the", "is the", "what is the", etc. between the beginning of the sentence and a noun or adjective which starts close to the beginning of the sentence.

The LOCALMATCH routine takes advantage of the word boundary information obtained from the initial scan, together with local syntactic constraints of the grammar. This information is used to direct searches for small function-word strings that might be difficult to find without context, and to find more garbled longer words next to words already found.

1. Example

The following is an example of an initial scan and subsequent high-distance matching. The sentence being processed is "the wastebasket which the green coffeetable supports contains the dictionary".

Initial Scan Output

```
((STEPLADDER 8 15 150)(COFFEETABLE 18 25 15) (SUPPORT 26 31 5)
(CONTAIN 33 38 10)(DICTIONARY 42 49 95))
```

LOCALMATCH Output

```
((CONTAIN 1 5 135)(THE 1 2 0)(PLACE 3 5 90)
(WASTEBASKET 3 10 185)(THE 6 7 0)(STEPLADDER 8 15 150)
(WHICH 11 12 12)(CLUTTER 11 14 100)(THE 13 14 12)
(THE 15 17 0)(GREEN 15 17 60)(COFFEETABLE 18 25 15)
(SUPPORT 26 31 5)(CLUTTER 30 32 110)(ROBERT 30 33 120)
(THE 32 33 0)(THE 33 33 0)(CONTAIN 33 38 10)
(BOOKCASE 34 38 110)(ENJOY 34 38 120)(THE 39 41 0)
(PERSON 42 46 105)(DICTIONARY 42 49 95))
```

2. Semantics in the Higher-Distance Matching

The LOCALMATCH routine does not use semantics very extensively to help it restrict the various words that it matches. However, there are a few instances in which semantics are used, such as when it is looking for adjectives in front of a noun, it need only look for adjectives that are semantically compatible with that noun.

In general, however, these scans are based on such local information that it is impossible to incorporate too many semantic restrictions. For instance, in scanning for a verb next to a noun, one cannot require that the verb be able to take the noun as a subject, for instance, since the noun might be nested in prepositional phrases or clauses, modifying the subject.

As a result, the information being exploited by the LOCALMATCH routine is primarily the word boundary information together with some very low-level syntactic constraints, rather than higher-level syntactic or semantic information.

If, however, the semantic component of LPARS was extended to include a scene with objects being talked about (as in Winograd's block world[4]), and to include an interactive dialog with previous events which could be referred to (in modifying clauses), the semantic part of this lexical matching could be made much more useful and interesting as discussed below.

C. SOME SPECULATION

We now discuss, in a somewhat speculative vein, some reasons why the LOCALMATCH routine in a system like LPARS might be more conveniently expressed procedurally, rather than written to work automatically, given any transition net grammar. These arguments would be particularly valid in a system with a much more complex semantic component which made use of a scene being talked about, as mentioned above.

The crucial point is that the goal of the LOCALMATCH routine is not to systematically explore the utterance in a globally consistent fashion. Rather, the goal is to use locally available words, together with a knowledge of objects and events being talked about, to try to identify fairly local structures in the utterance describing these objects and events.

The semantic logic that might be useful in such a routine is considerably different from the syntactic constraints built into a grammar. If so, it may not be necessary (or convenient) to incorporate the semantic logic, which postulates and searches for local structures in an utterance, wholely within the syntactic parsing framework. We do not say that it would be impossible to do so, just that it might introduce a level of formalism (transition net) which is largely unnecessary for some types of semantic logic and would therefore be an inconvenient mode for expressing that logic.

1. Example 1: Recognizing Phrases and Clauses

Suppose a routine that would help search for noun phrases is wanted. Such a routine might be called whenever the lexical analyzer discovered an adjective or noun. The routine could use knowledge about which adjectives and nouns could modify one another in the task domain to drive further lexical matching to try to fill out an entire noun phrase.

This routine might also query the particular scene being discussed to see what objects are possibly being described. The routine might search the scene, obtain a list of possible objects, and if it were reasonable to assume that a determined noun group was being processed, it could direct scans for additional words that might also describe the object or that might help uniquely specify which object was being talked about.

Logic of this sort could presumably use the transition network grammar to direct these scans. On the other hand, these scans have such a local, specific character that the transition net would be basically superfluous.

The reason for this is twofold:

(a) A noun phrase is a very local structure and appears unchanged in many global contexts.

(b) The logic being used to direct the scans is highly specific; it applies only to noun phrases and does not generalize to cover many different parts of speech. On the other hand, an advantage of a syntactic formalism like the transition net grammar is that it treats all words in a systematic, homogeneous fashion. This is not true of the logic just described. One invokes the noun-phrase recognizer only when one has found a particular class of word. One knows exactly what to do next. There is no need, for instance, having found a noun, to use the transition net to discover that an adjective can modify it.

A similar argument would hold for prepositional phrases. Even when one considers more complex syntactic structures such as modifying clauses, the arguments are probably still quite valid. For instance, if the system has stored a number of events (i.e., X did Y, X did Y to Z) it might be desirable, having found descriptions of X and Y, to direct scans to see if somehow these structures could be incorporated into a relative clause describing the event stored. Here again, one is looking for specific things, and it might well be easier to express this type of logic procedurally, divorced from the formalism of the transition network grammar.

2. Example 2: Searching for Small-Word Strings

Especially with tne crudity of front-end output, small words pose a very real problem to a speech recognizer, especially when a number of them can occur together.

As an example, consider the string of words "is in the". If two nouns (N1 and N2) have been discovered, how is one to test the possibility of the string of small words being present between them? Using the transition net grammar, one would ask: "Can N1 be followed by a small verb, which can be followed by a small preposition, which can be followed by a small determiner, which can be followed by N2?" This involves a great deal of threading one way through the grammar, with very little lexical information to base any parsing choices upon.

It would be more convenient to write a program which scanned for the word-string "is in the" as a unit, whenever it observed that two nouns, N1 and N2, are x ± y phonemes apart and that N1 can be "in" N2. The answer to this question is, of course, not obvious and is probably dependent on the particular vocabulary and syntax used, on the application, and on the accuracy of the front end.

3. What Use Then Is the Syntactic Formalism?

Sections 1 and 2 above offered some arguments as to why fairly local matching might best be done outside the transition network formalism. If these somewhat speculative arguments do make sense, however, then it is reasonable to ask what such syntactic formalism is good for, if anything. The answer is that syntactic formalism is needed to assure global consistency of local structures found in the manner described above.

Since the logic described is based on fairly local criteria and is basically a set of probabilistic heuristics which looks for reasonable local constructs, there is no assurance that recognition going on in one part of the sentence is globally consistent with recognition elsewhere. The transition network can be used to assure that the entire structure is globally consistent, and to direct attention to possible problem areas (bringing information from all parts of the utterance) if a well-defined sentence is not initially discovered. The primary focus of the present work is, in fact, on the problem of how to organize a system which uses syntax (expressed in the form of a transition network) in this fashion.

V. A LOCAL APPROACH TO PARSING

A. INTRODUCTION

This section describes the two local parsing algorithms used by LPARS: TREE, which builds the local partial parse trees; and CONNECT, which connects pairs of such trees together by proposing words that might exist between them. The description of these two algorithms has been kept at the level of a verbal overview in this chapter. More-detailed flowchart descriptions are given in Appendix D.

Both algorithms described here involve only left-to-right processing of the grammar, even though they are used as part of a locally organized parsing system.

This is the most technical section of the report. A reader interested in acquiring a background of this material without (or before) going into more detail is advised to read Secs. B-1, C-1, C-2, D-1, and D-1-a which give an overview of the two algorithms described.

We first describe the structure of the partial parse trees which form a central part of LPARS processing.

B. REPRESENTATION OF PARTIAL PARSE TREES (PPTs)

This section describes how LPARS represents syntactic structures. It is assumed that the reader has some general exposure to syntactic representation.

A PPT is composed of terminal and nonterminal nodes. Each terminal node corresponds to a terminal arc in the transition net, and each nonterminal node corresponds to a nonterminal arc. Each nonterminal node must have one or more sons. The sequence of these sons corresponds to some permitted path through the network which defines that nonterminal. A number of simple PPTs are shown below (Fig. V-1).

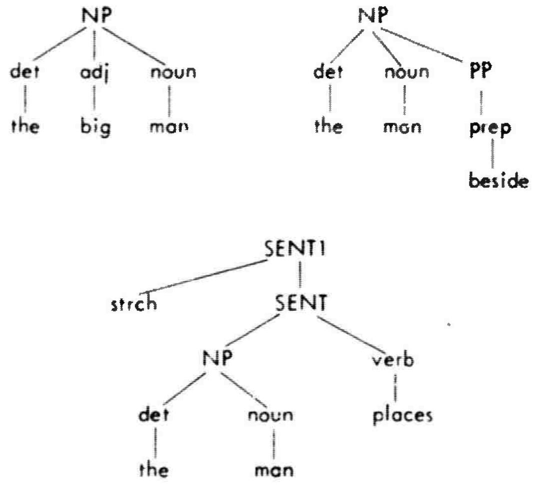


Fig. V-1.

A PPT is constructed out of words that are adjacent in the input. It can span no gaps. It is "partial" in the sense that the outer end of a path of arcs taken through an outer nonterminal node need not terminate on a permitted state.

Two points are central to the use of PPTs in LPARS:

(1) Each PPT is very closely tied to the grammar.

(2) "Ancestor links" can be used to represent the local ambiguity inherent in right recursion.

These two points are expanded upon in the following two sections.

1. Each PPT is Closely Linked to the Grammar

To understand how the local parsing algorithms operate (especially the CONNECT algorithm), one must appreciate how closely the PPTs are tied to the transition network grammar. The sequence of sons of each node represents some permissible path of arcs through the transition network which defines that nonterminal. This path of arcs is contained explicitly in each node. In fact, each node includes the following information:

(a) a list of sons

(b) a list of arcs taken (i.e., the path through the network)

(c) the start state (of the path)

(d) the end state (of the path).

Thus, given a PPT, one can get directly to the beginning and end states the path takes through the transition network in forming each nonterminal node. Therefore, one can readily investigate arcs emanating from an end state, for instance, to see how the path might be extended. This is an important capability, since the CONNECT algorithm, as discussed in Sec. V-D, consists basically of exploring ways in which the paths of two PPTs can be extended and joined.

Since the PPT is so closely tied to the grammar, we often talk of the beginning or end state of a node, meaning the beginning or end state in the grammar of the path of arcs taken in forming the node.

2. LPARS' PPTs vs Woods' Structure Building

The parse structures described here differ from those constructed by Woods' transition net parser.[13] In Woods' system, syntactic structure is built by user-written BUILD functions. These user-built structures represent a kind of deep structure rather than the path taken through the grammar. In LPARS, no attempt is made to construct deep structure as Woods' system does. The only parse structures built are created directly from the transition net as illustrated in Fig. V-1.

3. Ancestor Links

The only unusual feature of the PPTs produced by TREE is the use of "ancestor links." Ancestor links are used to express the local ambiguity caused by the existence of right recursion in a grammar.

For instance, assume that noun phrase (NP) and prepositional phrase (PP) are defined as shown in Fig. V-2.

This is a simple example of right recursion since a NP can end in a PP which can end in a NP, etc. Thus, if one sees only a noun phrase to the left of a verb, for instance, one does not know if that noun phrase is the subject of the verb, or if it is arbitrarily nested in PPs modifying

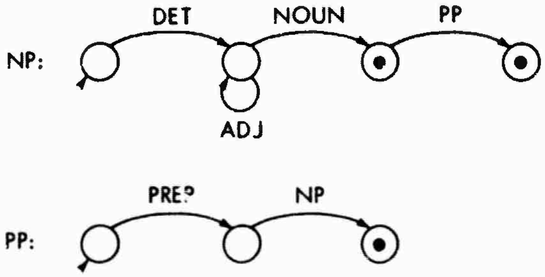


Fig. V-2.

the subject. Thus, the string of words "the box supports" could be part of ". . which the box supports. .", or "the block in the box supports. .", or "the cube on the block in the box supports. .", etc.

If ancestor links were not used, there would be an infinite number of ways to connect a NP to a verb, including those shown in Fig. V-3.

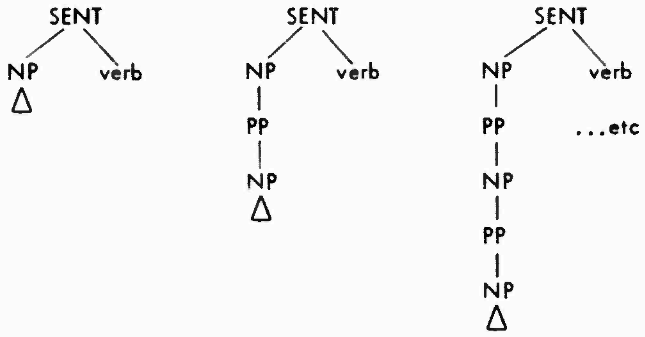


Fig. V-3.

Using ancestor links, however, whenever a NP is found to the left of a verb, the two structures shown in Fig. V-4 can be produced.

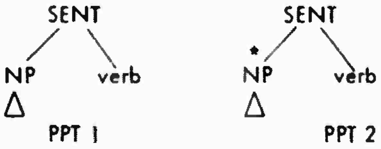


Fig. V-4.

The starred link in PPT 2 is an ancestor link, and embodies an unlimited number of possible global relationships. The ancestor link implies that the NP is the rightmost descendant of the NP arc in the SENT net, without specifying the exact relationship. Notice that if the noun phrase was not a semantically valid subject of the verb, only PPT 2 need be produced.

If only a noun had been found to the left of a verb, only one structure need be produced (see Fig. V-5).

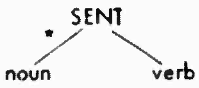


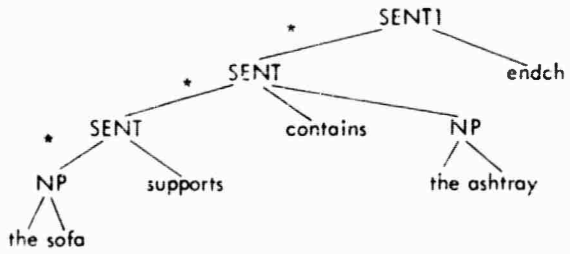
Fig. V-5.

More formally, an ancestor link represents an unspecified chain of nonterminal nodes connecting a PPT to a higher-level node. (A chain of NTs is one or more NT directly descendant from one another, each having only one son.) The presence of an ancestor link means that a chain of nonterminal nodes has been deleted and replaced by the ancestor link. As illustrated in Fig. V-5, this allows a single ancestor link to represent an infinite number of possible structures.

An ancestor link can be thought of as saying "I don't know what is to the left of me. Many structures are possible." Thus, such links can occur only on the leftmost side of a partial parse tree. (If left recursion were allowed in LPARS' grammar, then ancestor links could also appear on the rightmost side of a PPT.) TREE automatically inserts an ancestor link whenever a chain of nonterminal nodes occurs in the leftmost son of a PPT being constructed.

The concept of an ancestor link is a very simple one. It is also a very natural one in a system such as LPARS. Such links allow the system to recognize a valid, yet ambiguous, syntactic relationship without being forced to specify what the relationship is. Later, when an attempt is made to connect the PPT to other PPTs, the relationship can be resolved.

More-complex PPTs containing several ancestor links can also be made. For example, given the string of words "the sofa supports contains the ashtray endch", one of the possible PPTs produced is shown in Fig. V-6.



(as in: "The box which the table beside the sofa supports contains the ashtray")

Fig. V-6.

Since LPARS only allows right recursion, these ancestor links can occur only on the leftmost side of a PPT. Were left recursion allowed, such links would also be possible on the rightmost side of a PPT.

4. Partial Parse Tree Terms

We now define a few terms useful in talking concisely about partial parse trees.

A "Final Node": A nonterminal node of a PPT is a final node (a) if the path of arcs taken ends on a permissible final state, and (b) if its rightmost son is either terminal or is itself a final node.

An "Initial Node": A nonterminal node of a PPT is an initial node (a) if the path of arcs taken begins on a permissible initial state, and (b) if its leftmost son is either terminal or is itself an initial node.

To "Invoke": An arc in the grammar "invokes" a word if it is labeled by the syntactic class of the word. It invokes a PPT if it is labeled by the syntactic class of the top node of the PPT.

A "Context Arc": An arc of the grammar is a possible context arc for a PPT if it invokes the PPT.

C. TREE: THE PARTIAL PARSE TREE CONSTRUCTOR

This section describes TREE, the algorithm which constructs partial parse trees in various parts of the input sentence. In spirit, the algorithm is quite similar to more traditional left-to-right parsing algorithms. A number of additional considerations are made necessary, however, by the local nature of the LPARS approach.

The TREE algorithm itself does not attempt to recognize or hypothesize any words in the input. Rather, it accepts word candidates which have been recognized by previous activity of LPARS, and constructs as many different local parse structures as possible using these words. If all the words of the input sentence have been found, then a parsing of the entire sentence is among the parse structures produced. If not, TREE will output a collection of partial parse trees (PPTs) which are then turned over to a second algorithm, CONNECT. CONNECT attempts to connect pairs of parse trees together by using the grammar to propose words that might exist between them. It then tests these words out at high phonetic distance against the input (see Fig. V-7).

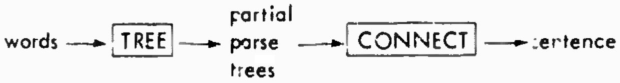


Fig. V-7.

We first present an overview of the operation of the TREE algorithm. Then we discuss two problems which do not arise in left-to-right parsing systems, but which must be dealt with to implement the local approach of LPARS.

1. Goal of TREE

The goal of the TREE algorithm is to construct as many different local syntactic structures as possible using the input words. Let us see what this means in practice. Suppose the word string "the big man" was recognized. Clearly, these words could all be incorporated into the PPT shown in Fig. V-8.

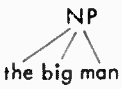


Fig. V-8.

This, in fact, is the only PPT which LPARS creates. Notice, however, that there are five other "subsumed" structures which can be formed (Fig. V-9).

A PPT (PPT 1) is "subsumed" by another PPT (PPT 2) if PPT 1 incorporates only a subset of the words incorporated by PPT 2, and if the syntactic structure connecting those words is identical in the two PPTs.

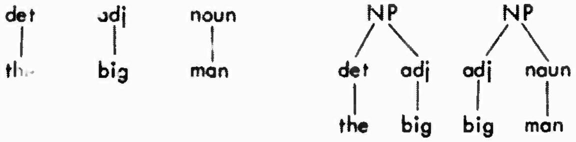


Fig. V-9.

It might seem reasonable for TREE to create all possible subsumed structures, on the theory, in the example above for instance, that "the" and "big" might be correct words but "man" might have been in error. The approach of TREE, however, is not to produce any PPT which is completely subsumed by some other PPT which it constructs. If any word is later thought to be in error, LPARS can break the structure down into smaller parts.

Thus, the goal of TREE is to construct as many local PPTs as possible, with the provision that no PPT produced is subsumed by another. For instance, given "the book beside the", TREE would produce the four structures shown in Fig. V-10.

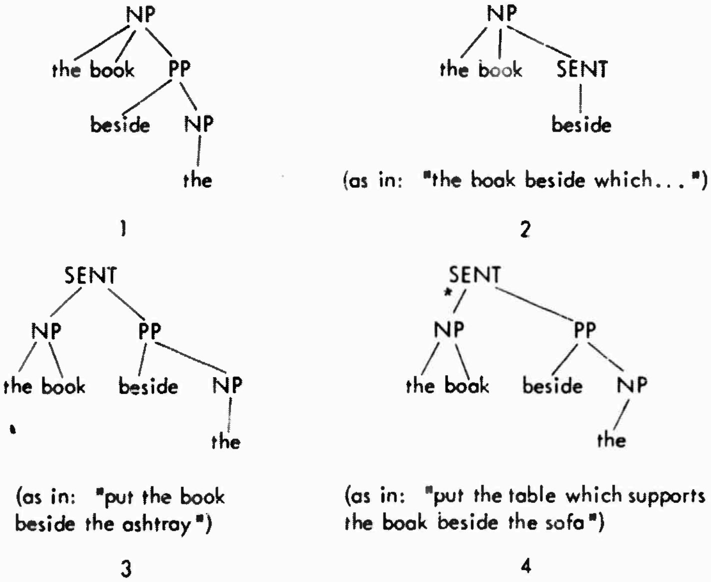

NP
the book
PP
beside
NP
the
1
NP
the book
SENT
beside
(as in: "the book beside which...")
2
SENT
NP
PP
the book
beside
NP
the
(as in: "put the book beside the ashtray")
3
SENT
*
NP
PP
the book
beside
NP
the
(as in: "put the table which supports the book beside the sofa")
4

Fig. V-10.

These four valid global interpretations of the words are all structurally different. Notice that PPT 2 does not include the word "the", since only a relative pronoun (such as "which") can follow the preposition in the SENT transition net. Notice also, that PPT 4 really represents a potentially unlimited number of global relationships because of the ancestor link.

As discussed above, the goal of TREE is to construct all possible PPTs. Another way of phrasing this goal is to say that TREE hypothesizes all possible global relationships that might exist in the grammar between all sets of adjacent words.

It is reasonable to ask why a system should go through this intermediate step of constructing all possible PPTs. The reason is that once these PPTs are built, LPARS can examine them in toto, and decide what areas of the input are best candidates for further investigation at high phonetic distance.

2. Implementation Overview

The following is a description of the TREE algorithm in general terms, to give an overview of the issues involved. We hope that this overview will suffice for most readers.

TREE accepts as input a list of words, possibly overlapping, ordered by their position in the sentence. It processes these one at a time, thus proceeding from left-to-right through the sentence. Since at each point TREE builds all possible parses, no backup is necessary.

TREE stores the PPTs it constructs in a "phoneme position table" (Fig. V-11) which has one cell for each phoneme position in the input sentence. Each cell has a list of all PPTs constructed which end at that phoneme position in the sentence.

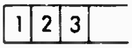


Fig. V-11. Phoneme position table.

Let us take the following simple example to see how the table is used. Suppose the following word candidates were found: (the 11 12) (big 13 15) (man 16 18). The numbers represent the beginning and ending phoneme positions of each word in the input. TREE takes each word in turn and attempts to connect it to every PPT ending immediately before that word.

Thus (the 11 12) is taken first. Since no PPTs end at position 10, no connections can be made, so the PPT "the" is recorded at position 12. Next (big 13 15) is taken; it is connected to "the", and a PPT "the big" is recorded at position 15. At this point, the PPT "the" is flagged to indicate that it is a subsumed PPT. Similarly, "man" is connected to "the big", yielding "the big man" ending at position 18, and the PPT "the big" is flagged as subsumed. The result of this activity is shown in Fig. V-12.

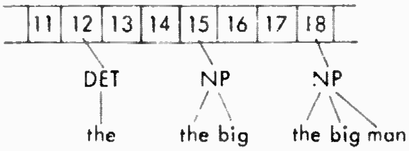


Fig. V-12.

Notice that once a PPT which has been incorporated into a larger structure is marked as subsumed, even though it will not be part of the final output of TREE, it is kept in the table. Therefore, if an overlapping word candidate has been found, such as (manufacturer 16 24), it can also be connected to "the big" yielding the PPT "the big manufacturer" ending at position 24.

This example is a very simple one. In practice, the input being processed may contain several words in any section of input. Words may overlap one another at random. Alternatively, gaps may exist (if they do, of course, no syntactic connections are made across these gaps).

Notice also that connecting a word to a single previously made PPT may yield several PPTs. As illustrated in Fig. V-8, for instance, adding "beside" to "the book" yields four different PPTs. Whenever a word is processed by TREE, it is connected in as many different ways as possible to all PPTs which end directly before that word. In addition, as discussed below in Sec. 3-f, TREE looks to see if the word can be connected to a rightmost subpart of each PPT in some other syntactic context.

Before adding a PPT to the appropriate phoneme position list, of course, TREE first checks to make sure it is not subsumed by any previously created PPT already on the list. Similarly, if the new PPT itself subsumes any PPT already on the list, the previous PPT is flagged. These two checks are necessary for reasons described below in Sec. 3-f.

3. Making Syntactic Connections

There are three different ways in which syntactic connections can be made between a previously created PPT which has been taken from the phoneme position table, and a new word being processed. These different connection modes are: (a) appending, (b) pushing, and (c) popping. Modes (a) and (b) are quite straightforward; (c) is more complex due to ancestor link considerations.

a. Appending

Appending is the simplest mode of connection and is illustrated by an example (Fig. V-13) of the word "man" being added to the PPT "the big". Here, the end state of the PPT "the big" in the grammar has an exit arc labeled "noun"; thus, the PPT "the big man" can be formed directly.

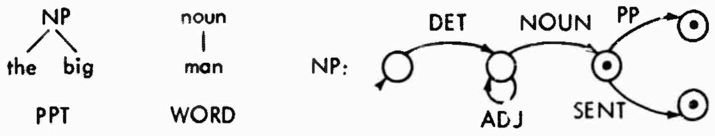


Fig. V-13.

More formally, an appending connection is made to a rightmost nonterminal node of a PPT. It requires that the rightmost son of that node be a terminal or a final nonterminal. It also requires that the rightmost state of that node have an exit arc which invokes the word being processed.

b. Pushing

Pushing is illustrated by a similar example: the word "beside" being added to the PPT "the book". Here, the end state of the PPT has an exit labeled PP, and the PP network can start with a preposition. Thus, the PPT "the book beside" can be formed (see Fig. V-14). (A similar successful push connection can be made using the SENT net.)

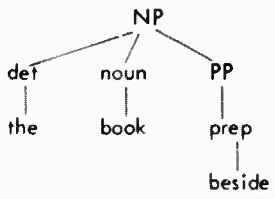


Fig. V-14.

More formally, a pushing connection can be made to a rightmost nonterminal node of a PPT. It requires that the rightmost son of that node be a terminal or a final nonterminal. It also requires that the rightmost state of that node have an exit arc invoking a nonterminal which can start (possibly recursively) with an arc invoking the word being processed.

Example: In connecting the noun "man" to the PPT "the big", an appending connection can be made. In connecting the preposition "beside" to the resulting PPT, a pushing connection must be made to the PP transition net. Figure V-15 illustrates these two connections.

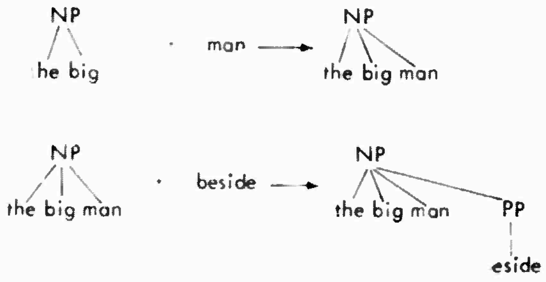


Fig. V-15.

c. Popping and Ancestor Link Formation

Popping is attempted if the PPT is a terminal or if it is a final nonterminal. Popping involves making a syntactic connection through a higher-level net.

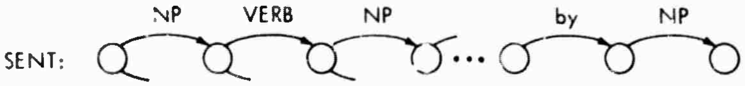


Fig. V-16.

The first step in making a "popping" connection is finding a higher-level context for the PPT which is to be connected (i.e., an arc which "invokes" that PPT). For instance, in connecting a NP to a VERB, there are three NP context arcs in the SENT net (see Fig. V-16). Each of these context arcs is taken one at a time, and an appropriate PPT of the form shown in Fig. V-17 is constructed.

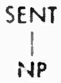


Fig. V-17.

Next, an attempt is made to connect this new PPT to the verb. In only one (the leftmost) of the three context arcs depicted in Fig. V-16 will this attempt succeed since this is the only NP arc immediately to the left of a verb arc.

In attempting to connect the new PPT to the verb, the algorithm will try all three modes of connection: appending, pushing, and popping (if the new PPT ends on a final state). Thus, the popping process is recursive, as the algorithm tries all possible combinations of multiple popping that might yield a connection (as described in Sec. 3-d below).

There is a problem involved in this recursive popping, namely, the algorithm (as stated so far) could get into an infinite loop. For instance, since a noun phrase can end a prepositional phrase which can, in turn, end a noun phrase, the algorithm could merrily pop up endlessly from one to another. It can be seen that this is the ancestor link problem in a new guise. The easiest solution to this problem is just to keep a list of the arcs already popped through at each point and abort the recursive popping if an arc already popped through is approached again.

In practice, by visually inspecting the grammar, one can insert tests that abort a number of pops so that not all possibilities are attempted. These tests are put in to improve efficiency, but do not alter the basic nature of the algorithm.

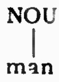
More formally, a popping connection can be made in the following circumstances: (1) If the PPT is terminal, or if the PPT is a final nonterminal node; (2) if there exists a context arc (which invokes the PPT) which has not been popped through in forming the PPT; and (3) if the new PPT formed by popping to this arc can be connected (via appending, pushing, or popping) to the word being processed.

d. Examples of Popping

The algorithm is able to abort the recursive popping successfully and still get all possible local structures because it is set up to construct ancestor link structures while making these popping connections. As described in Sec. 3 above, an ancestor link represents an unspecified chain of nonterminal nodes. Such an ancestor link is created when the algorithm pops through successive nonterminal nodes.

A number of examples using different PPTs can best serve to explain how popping and ancestor link formation take place.

<u>Example 1</u>



When popping with a terminal, one first takes each arc invoking that terminal and attempts connections. Thus, the PPT shown in Fig. V-18 would be constructed. In this case, a second pop can be made recursively since the new PPT ends on a final state. Before attempting the recursive pop, however, the algorithm first examines the PPT to see whether it should be transformed into an ancestor link structure. The rule is: if a nonterminal which has only a single son is to be popped, then the nonterminal is replaced by an ancestor link structure. It is possible to use this simple rule for two reasons:

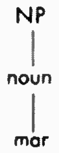


Fig. V-18.

(1) The CONNECT algorithm, which processes these PPTs, is set up to expect PPTs constructed in this fashion.

(2) In LPARS' grammar, there are no nonterminal nodes which can be made to begin and end on permitted states yet still have only a single son. The rule would have to be somewhat modified to accommodate such nonterminals.

In the above example, since the new PPT to be popped is a nonterminal with a single son, the algorithm automatically inserts an ancestor link structure, transforming the new PPT as seen in Fig. V-19.

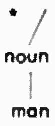


Fig. V-19.

Even though the NP node has been deleted, the algorithm must remember that it is dealing with a NP and therefore use NP context arcs in making connections. This PPT, for instance, could be connected to a verb as shown in Fig. V-20.

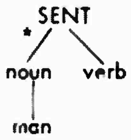


Fig. V-20.

Since a NP can end a sentence, a third recursive pop can be generated of the PPT (see Fig. V-21).

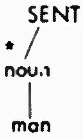


Fig. V-21.

Again, the algorithm inspects the PPT before allowing the recursive popping to take place, and, in this case, propagates the ancestor link deleting the SENT node and obtaining the PPT shown in Fig. V-22.

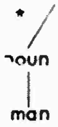


Fig. V-22.

Thus, the same ancestor link structure is passed up through successive levels of popping. Again, although now both the NP and SENT nodes have been deleted, the algorithm must remember that it is dealing with a SENT. This "propagation" of the ancestor link is done because any intervening nodes are superfluous until the structure is finally connected to the word being processed. For instance, the PPT could be connected to an "endch" as shown in Fig. V-23.

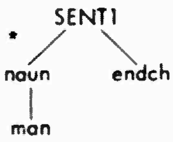
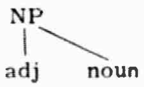


Fig. V-23.

Example 2 – a Partial NP



T PPT is handled in a fashion very similar to Example 1. Here, however, the ancestor link structure which is passed up is shown in Fig. V-24.

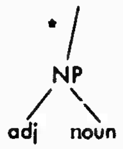
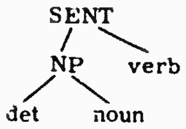


Fig. V-24.

Example 3



(as in "which the table supports")

This partial SENT PPT is handled in exactly the same fashion as Example 2. The ancestor link structure used in the popping is shown in Fig. V-25.

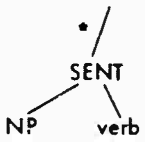


Fig. V-25.

e. Tests to Abort Popping in Certain Circumstances

The grammar used by LPARS is restricted; therefore, the efficiency problems involved in multiple popping are not prohibitive. With more complex grammars, this problem could become more significant.

One potentially useful technique for improving efficiency is to inspect the grammar and insert logic to abort various possible pops before they are attempted. This could be useful in two general situations.

(1) When the word being connected does not follow a particular PPT.

(2) When one is popping with an ancestor link structure, and there are several ways that the PPT can pop to a higher-level context. For example, there is more than one arc which allows a NP to end a SENT. Thus, the algorithm as stated can connect a NP to an ENDCH once for each of these contexts, yet the resulting structures would be identical since the intervening nodes popped through would be deleted. One obviously inefficient solution to this problem is to let several identical parse trees be formed, but let LPARS ignore the duplicates. Another approach is to inspect the grammar and insert logic to abort popping so that only one structure is ever built in such situations.

In any case, this issue (although important for practical reasons of efficiency) is not central to the basic algorithm. It is perfectly possible to let TREE explore all possible popping paths every time, using only the general mechanism that aborts if the popping starts to repeat itself. In effect, this amounts to letting TREE use the grammar "interpretively." By inspecting the grammar, one can manually "compile" a set of tests that let TREE operate more efficiently. In fact, it is probably possible to construct a set of rules that allows LPARS to inspect the grammar and construct the tests automatically. This possibility is not pursued in the present work.

f. Two Problems

There are two unique problems to be faced by a local parsing strategy, problems which do not arise in a totally left-to-right parsing approach. These problems arise in attempting to pursue the goal of finding all possible "maximal" local structures (maximal in the sense that none are subsumed by others).

Problem 1

The first problem might be called that of "finding all contexts for possible syntactic connections to rightmost subparts" and can be illustrated by considering the following hypothetical input: "the man places the book supports". Among the PPTS which can be created from this string are the two shown in Fig. V-26.

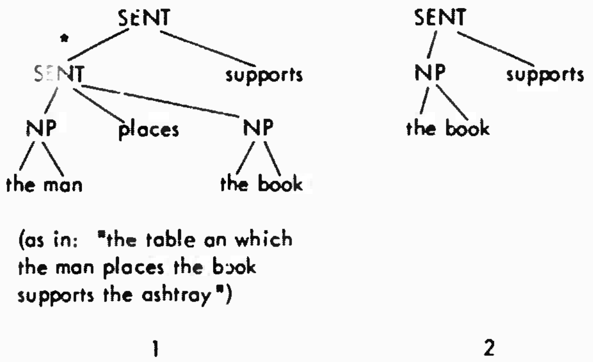

SENT
*
SENT
supports
NP
places
NP
the man
the book
(as in: "the table on which the man places the book supports the ashtray")
SENT
NP
supports
the book
1
2

Fig. V-26.

The problem is illustrated as follows. Since TREE creates PPTs by examining the words left-to-right, at some point it has created the PPT shown in Fig. V-27, and is attempting to add to this structure the verb "supports" if possible. As shown in Fig. V-26, not only can "supports" be connected to the entire structure but it also can be connected to the noun phrase "the book" in a different syntactic context.

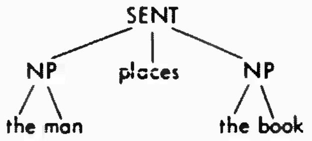


Fig. V-27.

Thus when examining a new word, TREE must do more than just continue any parsing that it has already developed. It must also see if any rightmost subparts of structures that it has built up can be connected to the new word. There are two general cases of this problem:

(1) In the example above (Fig. V-26) where a rightmost son of the PPT can be connected to the word in a different syntactic context.

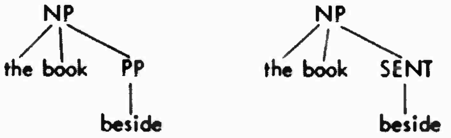
(2) The other example is where one can connect the word to the PPT but only if some of the leftmost sons of the PPT are deleted. For instance, consider the word string "which the table supports the ashtray". The string "which the table supports" can be formed into a sentence fragment. Yet "the ashtray" cannot be joined to this

entire fragment. The word "which" must be removed first, after which the structure "the table supports the ashtray" can be formed.

This problem is not a difficult one to solve once it has been understood, but it does require a somewhat different parsing strategy than does a purely top-down left-to-right system.

Problem 2

The second problem arises directly from the solution of the first. Namely, in solving Problem 1, TREE may at times construct a parse structure using a right nost subpart of one parse structure, but this parse structure may be subsumed by some other parse structure which TREE has created. This second problem, the elimination of redundant PPTs, can be illustrated by the string of words "the book beside the". The string "the book beside" generates, among others, the two PPTs shown in Fig. V-28.



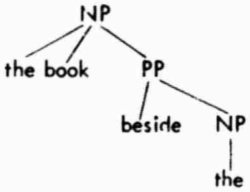
(as in: "the book beside which...")

PPT 1

PPT 2

Fig. V-28.

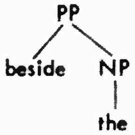
When "the" is added to PPT 1, it yields PPT 3 (see Fig. V-29). However, any attempt to



PPT 3

Fig. V-29.

add "the" to PPT 2 will clearly fail. On the other hand, "the" can be connected to a rightmost subpart of PPT 2 ("beside") in another syntactic context, yielding PPT 4 (see Fig. V-30). But,



PPT 4

Fig. V-30.

PPT 4 is subsumed by PPT 3 and therefore is not wanted. In practice, LPARS creates each PPT tentatively and throws it away if it is subsumed by a previously constructed PPT, or by a PPT constructed later.

The two problems discussed above arise from the local nature of LPARS' parsing. Problem 1 arises from the need to form all possible syntactic connections. In the process of finding these, however, the algorithm may also find some redundant structures. This gives rise to Problem 2, the need to throw such redundant structures away. Possibly, a more elegant solution allows the construction of all valid parse trees without the redundancy inherent in the present algorithm.

4. Some Remarks on TREE

This section makes two remarks about the nature of the TREE algorithm and its relationship to other parsing methods.

a. Efficiency Considerations for the Local Approach

The TREE algorithm may seem to be inefficient. It certainly can construct a fair number of different parse structures out of the same words. It must be remembered, however, that all parsers do this to some degree using backup, the only difference being that, when backup is used, only one partial structure exists at a time.

Some left-to-right parsers, such as Earley's,[22] do construct multiple parse trees. In theory, such an algorithm is just as efficient as one using backup. Of course, TREE would probably construct more partial parse structures than Earley's parser, given the same grammar and the same input, because Earley's algorithm expects to be given all the tokens in the sentence and thus need only construct PPTs which make sense in the possible left-to-right contexts present.

However, the basic point is that, although the number of PPTs constructed by TREE may at first appear large, this is partly because TREE does not use backup to effectively erase part of its work, but rather sets it all up for everyone to see. In fact, by doing all the work once and for all before calling the CONNECT routine, it helps make LPARS a more efficient system than if it had to redo this parsing many times as CONNECT tried out different possible ways of hypothesizing unfound words between structures in different parts of the sentence.

b. Unique Features of the TREE Algorithm

The local nature of the TREE algorithm results in a number of unique problems which do not occur in other parsers. It is useful to outline these specifically:

(1) The ancestor link problem is unique to an approach which attempts to create local structures.

(2) The necessity of finding all possible syntactic connections between structures being built up is a unique part of the local approach. In the particular left-to-right approach to building local structures used by TREE, this necessity gave rise to the two interesting problems discussed in Sec. C-3-f above.

(3) The need to have a syntactic formalism which lets parsing originate anywhere in the grammar and proceed in an organized fashion is unique to the local approach.

D. THE CONNECT ALGORITHM

The task of the CONNECT algorithm is to take two PPTs constructed by the TREE algorithm and use the grammar to propose possible word-strings which might exist between the two PPTs (see Fig. V-31).

Fig. V-31.

We often refer to the two PPTs as the left-hand PPT (LPPT) and the right-hand PPT (RPPT). These PPTs bound an input segment which is to be investigated. CONNECT proposes word-strings, consisting of at most one content word plus a small number of function words, which might fill the input segment. Then, CONNECT tests these hypotheses against the input segment at a high phonetic distance. If any match succeeds, the combined PPT is added to the collection of PPTs being investigated. When processing two input PPTs in this fashion, CONNECT tests out all possible hypotheses which are syntactically and semantically permissible.

1. Why Only One Content Word?

It is reasonable to ask why CONNECT searches only for word-strings containing, at most, one content word plus a small number of function words. Clearly, one might have allowed CONNECT to propose a longer string of words. Some limit must be set, however, and the design philosophy of LPARS is to keep this limit fairly low. At the opposite extreme to this philosophy, one approach to sentence recognition might be to generate all possible sentences and match each against the entire input. This approach is clearly inefficient. The whole structure of LPARS is designed to avoid such an approach. Thus, the idea of proposing a large number of long word-strings without looking at the actual phonemic input is avoided as much as possible by LPARS. If there is a large section of input between two PPTs, LPARS can make a higher distance scan through that section and use the words found as input to the TREE algorithm, or use other fall-back mechanisms as is discussed in Sec. VI.

It would be simple to expand the "single content word" constraint described above to make it more flexible. One might also allow word-strings consisting of two content words and, at most, one function word. Alternately, if the input section between is fairly small, one might want to look for, at most, one or two function words. The choice made in LPARS is somewhat arbitrary, and is certainly not the only possibility.

2. An Introductory Example

As discussed above, the task of the CONNECT algorithm is to find all word-strings that might exist between two PPTs, with the constraint that no word-string contain more than one content word. This section discusses a few examples to illustrate what is involved in this process.

Consider the two PPTs shown in Fig. V-32.

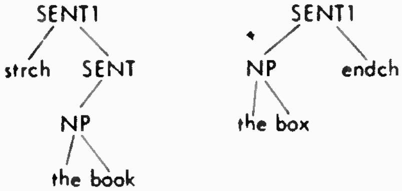


Fig. V-32.

Sample word-strings which might connect these PPTs are "supports the", or "is contained by the".

Consider the next two PPTs (Fig. V-33).

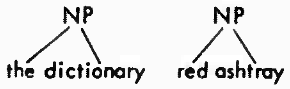


Fig. V-33.

Sample word-strings which might connect these PPTs include: "beside the", "in the big", "which the", "in which the", "supports in the" (as in "put the pipe which the dictionary supports in the red ashtray"). In fact, there are a great many possible word-strings which could connect these two PPTs. (This fact raises the efficiency question of whether one might want to use the CONNECT algorithm selectively. The answer to this question is "yes," and is discussed more fully in Sec. VI.)

Now let us try to get a feeling for the logic involved in generating these possible word-strings.

3. Extending Paths Through the Net

This section makes some preliminary comments to help make the CONNECT algorithm's activity easier to visualize.

Fundamental to this activity is the fact that every PPT is very closely tied to the transition network grammar (as discussed above in Sec. B). The sequence of sons of each node represents a permitted path through the associated network, and each PPT node has pointers to the states in the net on which the path begins and ends. Therefore, it is very easy to explore ways in which the path might be extended (by examining arcs emanating from the end state).

Let us first take a simple example. Suppose we are given the partial noun phrase "the big" and asked to use the grammar to see if we can make this PPT end on a final state by hypothesizing at most one content word (see Fig. V-34).

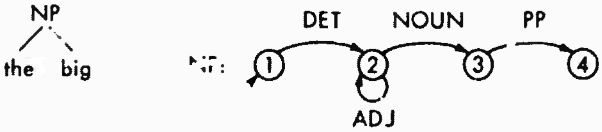


Fig. V-34.

To answer this question, we must look at the end state of the PPT in the net (state NP2) and follow the arcs that emanate from it. We then see that the PPT can be made to end on a final state by hypothesizing a noun. The resulting "hypothesized PPT," including the proposed word is shown in Fig. V-35. In this case, no other structures are possible.

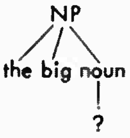


Fig. V-35.

Now let us look at a more complex example of which the previous example is a sub-pa1*. Suppose that the sentence (SENT) net is as shown in Fig. V-36.

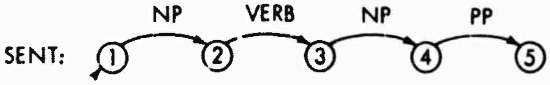


Fig. V-36.

Further suppose that the two PPTs of Fig. V-37 have been found

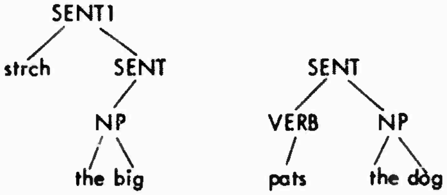


Fig. V-37.

The question now is: Can these two PPTs be joined by proposing at most one content word? For an answer, we first look at the SENT node of each PPT and observe that: (1) the SENT node of PPT 1 ends on state 2, and (2) the SENT node of PPT 2 begins on state 2. Thus, these two paths can be directly concatonated. But, for the resulting structure to be a valid PPT, all innermost nonterminal sons must begin and end on permitted states.

Hence, the two SENT nodes can be connected only if the noun phrase "the big" can be made to end on a final state by proposing, at most, one content word. As we saw above, this can be done by proposing a noun. Therefore, the two PPTs can be joined into the "hypothesized PPT" shown in Fig. V-38.

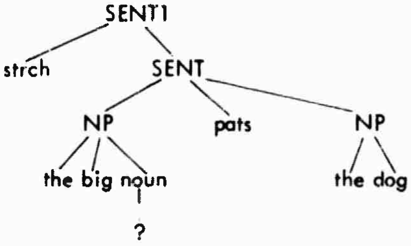


Fig. V-38.

This simple example illustrates quite clearly the heart of the CONNECT algorithm, namely:

(1) On some level, paths from both PPTs must be joined.

(2) All inner nodes below this joining must be made to begin or end (as appropriate) on permitted states.

These two basic principles are the key to the operation of the CONNECT algorithm. There are, unfortunately, a number of complications which make the algorithm slightly more complex than the above example might seem to indicate. Nevertheless, the basic principles remain those stated above.

The two main complications are:

(1) Having to pop to a higher-level context to make a connection: For instance, given two noun phrases, one possible word-string connection is a verb. To find this possibility, the algorithm must pop to the SENT net (see Fig. V-39).

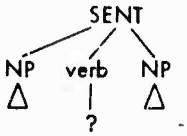


Fig. V-39.

(2) Ancestor link considerations: The necessity to accommodate ancestor links poses further considerations. For instance, when connecting the two PPTs shown in Fig. V-40.

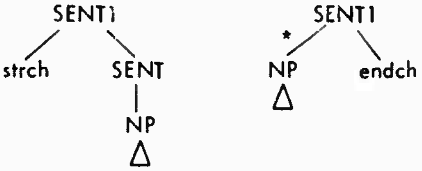


Fig. V-40.

First, the SENT1 nodes are joined, and then the SENT node in PPT 1 is extended to join the ancestor link son of PPT 2, generating the combined PPT (see Fig. V-41).

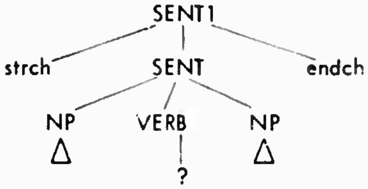


Fig. V-41.

Notice that, in this example (due to the ancestor link), paths in the two PPTs were joined at two levels of structure. In a more complex example, paths can be joined at several levels. (In fact, one can think of the CONNECT algorithm as being able to "zip" two partial structures together.)

4. Two Parts of CONNECT

The preceding section discussed the basic principles involved in the operation of the CONNECT algorithm. We now look at this algorithm in more detail. There are two distinct parts of the CONNECT algorithm:

(a) First is the JOIN routine which joins the higher-level structure of the PPTs, "zipping" them together from top to bottom, thereby constructing a hypothetical combined tree. This combined tree indicates a possible word-string that might exist between the input PPTs. The JOIN routine by itself determines only tne parts of speech of the possible word-strings.

(b) Second is the CONSTRAIN routine which inspects a tree, created by JOIN, with its hypothesized words, and uses the contextual syntactic and semantic information to compile additional constraints beyond mere parts of speech, further narrowing down what the hypothesized string of words might be. (For instance, if the unclear section might contain a verb, inspection of the context might allow one to rule out a number of verbs in the vocabulary.)

The following sections give a simple example of these two routines in operation, and then describe each in turn. The description of the algorithm ignores lambda transitions. This simplification allows the description to be somewhat cleaner, and does not lose any of the overall flavor of the algorithms themselves.

a. Simple Example

We now give a simple example to help clarify the individual functions of the two parts of CONNECT, and show how they fit together before describing them in detail. Let us take the example given in the introduction: two noun phrases at the beginning and end of the sentence, with a gap between them (Fig. V-42).

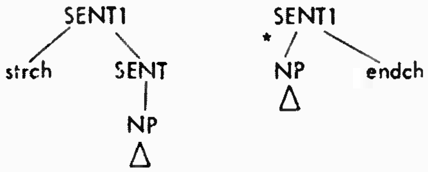


Fig. V-42.

The task of CONNECT is to find all the ways in which the two PPTs can be joined together by words which might exist between them. The algorithm starts at the top of the two PPTs and recognizes that each is headed by a SENT1 node, and that these two nodes can be joined. The algorithm therefore constructs the structure presented in Fig. V-43.

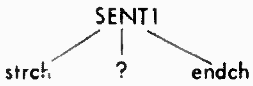


Fig. V-43.

The algorithm places a pointer to this combined structure in a global location. The son labeled "?" represents a cell in a list of sons which is left temporarily empty. It will later be filled in as processing proceeds. The routine now calls itself recursively with the two PPTs of Fig. V-44 as arguments. The address of the empty son of the higher-level structure is also

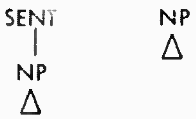


Fig. V-44.

passed down. These two PPTs are the inner sons of the original two PPTs. Along with these two structures, the algorithm is passed additional information indicating:

(1) That it is no longer processing the highest-level nodes of the trees and thus, for instance, no popping connections can be attempted.

(2) That both PPTs are bounded by outside context (the start and end characters) and, therefore, that when processing is finished all inner nonterminals must begin or end (as appropriate) on permitted states.

The algorithm proceeds to test for possible intervening words by extending the path of the SENT node of the LPPT starting at the end state. In so doing, it tries to complete the SENT node, incorporating the RPPT (the NP) into the structure. It finds that it can successfully do this by following a verb arc and an affix arc to the direct object NP arc. It therefore constructs the PPT shown in Fig. V-45.

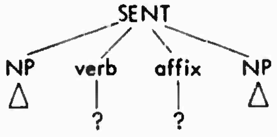


Fig. V-45.

This PPT is then attached to the previously built structure by placing a pointer to this PPT in the cell marked "?" in the higher-level structure above. Thus, the PPT of Fig. V-46 is built and is accessible through a global pointer.

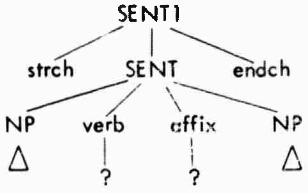


Fig. V-46.

Now the first part of the CONNECT process, the joining of higher-level structure, is completed. In more complex examples, this process can take longer and involve several levels of recursion as several levels of structure are "zipped" together.

The PPT is now passed on to the second part of the CONNECT algorithm, CONSTRAIN, which examines the entire structure to find additional constraints which could be placed on the verb, such as "transitive," "must be able to take a certain class of noun as subject and as object," etc. It then tests all verbs satisfying these constraints against the input.

After the processing of this first possibility is complete, CONNECT resumes trying to find different ways of joining the two structures. Thus, it would also find the passive sentence form, proposing a word-string of the form "BE VERB AFFIX BY" as in "is supported by".

Also, had the SENT node of the LPPT ended on a permitted state, the algorithm could have dropped down recursively and attempted to connect the two structures given in Fig. V-47.

Fig. V-47.

These could be connected by a number of word-strings including: PREP, PREP WH, or WH. In fact, if the right-hand NP had not originally been connected to the ENDCH, the algorithm would indeed have found these possibilities. The necessity that all innermost nodes begin and end on permitted states, however, made it impossible for the algorithm to drop down below the SENT node in trying to connect the NP.

The following sections describe the two parts of this algorithm in more detail.

b. JOIN Routine

This section describes the logic of the JOIN routine which takes two PPTs and creates all possible hypothetical combined PPTs which can be made connecting them.

The JOIN routine does all its processing of the transition network grammar in a left-to-right fashion, starting from the end states of the left-hand PPT (LPPT) and exploring ways in which paths can be extended from these states to join the RPPT.

To illustrate how this processing is performed, we first break the problem down into several components and describe each first.

(1) How an existing path can be extended in a single net.

(2) How a hypothesized path can be initiated in a net.

(3) How a path can be continued in a lower net.

(4) How a path extended from the LPPT can be joined to the RPPT.

Finally, we discuss how the entire algorithm is organized to explore all possible connections.

(1) Extending a Path

A PPT node has a sequence of sons which represents a path through the associated transition network. This path can be extended by looking at each exit arc emanating from its end state.

(a) If the arc is a terminal arc, then an extension to the path can be made readily. To do this, JOIN constructs a new extended PPT and turns this over recursively for further extension (see Fig. V-48).

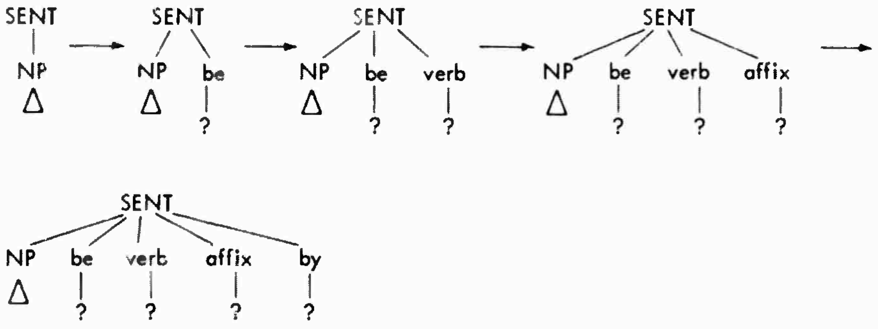


Fig. V-48.

(b) If the arc is a nonterminal, however, the algorithm cannot continue to extend the path in the current net, unless the nonterminal itself can be completed with, at most, one content word. There are two such nonterminals in LPARS' grammar: NP, which can be completed by (DET NOUN); and PP, which can be completed by (PREP DET NOUN) where the preposition is "in" or "on". JOIN therefore treats these nonterminals like terminals when extending a path and inserts the appropriate structures into the hypothesized extended PPT (see Fig. V-49).

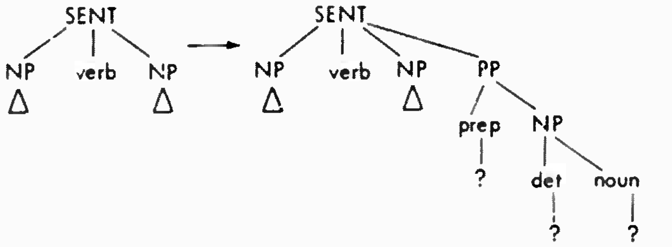


Fig. V-49.

The algorithm can continue recursively to extend a path until a state with no permitted exits is reached, or until it reaches a state from which it could only continue by hypothesizing two content words.

(2) Starting a Proposed Path from Scratch

The JOIN algorithm can also start up a hypothesized path from scratch in a given transition network in a fashion very similar to that described above. It merely takes each initial state of the net and looks at the arcs emanating from it and proceeds as above.

(3) Continuing an Extended Path in a Lower Net

Another possible way to extend a path involves pushing to a lower net, and continuing the extension of the path in that lower net. This involves a combination (Fig. V-50) of the techniques described in the previous two sections.

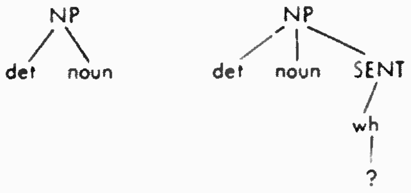


Fig. V-50.

(4) Joining an LPPT Path to the RPPT

There are three fundamental ways (called Modes 1, 2, and 3) in which an LPPT with an extended path can be joined to an RPPT.

Mode 1:- If the end state of the LPPT equals the start state of the RPPT.

In this case, the two paths can be directly concatonated, assuming that all exit action routines are compatible (which can be checked since each node includes a list of arcs taken by its path) (see Fig. V-51).

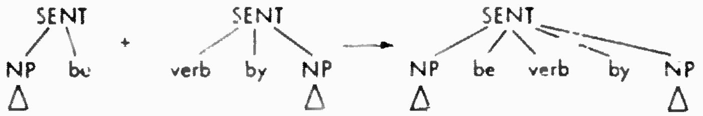


Fig. V-51.

The algorithm must then ascertain that the inner nodes begin and end on permitted states. This is done in LPARS by left-to-right extension of paths. Thus, a node is made to end on a final state by following arcs from the end state of its path. Conversely, a node is made to begin on an initial state by initiating a path at each possible initial state and attempting to extend this path to join the node.

Mode 2:- If an exit arc from the endstate of the LPPT invokes the RPPT (Fig. V-52). Again, the algorithm must still make the inner nodes begin and end on permitted states.

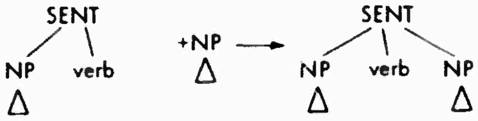


Fig. V-52.

Mode 3:- If the RPPT has an ancestor link son, and the end state of the LPPT equals the start state of the RPPT (Fig. V-53). This example is somewhat harder to visualize since, although

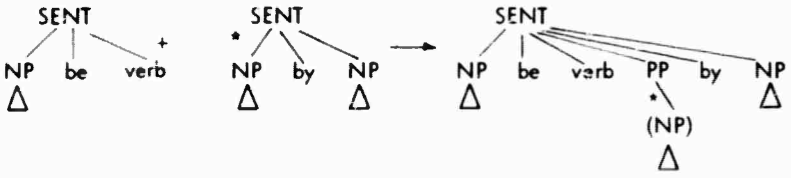
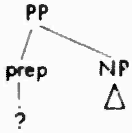


Fig. V-53.

the two nodes have been joined, additional processing must be done to complete the connection. In the example above, one must initiate a path in the PP net to be joined to the ancestor link son, thus generating the following possible structure (Fig. V-54) and creating the following entire structure (Fig. V-55).



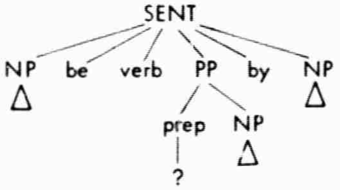


Fig. V-54.

Fig. V-55.

(5) Overall Control of CONNECT

The preceding sections attempted to give a sound verbal overview of the logic involved in the CONNECT routine. The following program outline describes the overall control program which coordinates the CONNECT algorithm so that, given two PPTs, it will find all possible intervening word-strings.

The only part of CONNECT which it does not include is the popping to higher-level contexts with the LPPT, and the application of this algorithm to the PPT resulting from the pop.

```
BEGIN CONNECT(LPPT, RPPT)
    (1) If LPPT is NIL, or RPPT is NIL return.
    (2) If LPPT can be directly joined (with no extension of the path) to RPPT by
        Mode 1, and if ANCESTOR(RPPT) equals true then call CONNECT(RSON(LPPT),
        LSON(RPPT)).
        (Comment: This is the step that handles the zipping together of higher-level
        structure. As discussed in Sec. D-3-a, before CONNECT is called recursively,
        a merged structure is constructed from the current nodes, containing a tem-
        porarily empty son, and the address of this son is available to the lower invo-
        cation of CONNECT.)
    (3) Call JOIN(LPPT, RPPT).
    (4) If this is the top level of CONNECT or if FINAL(LPPT) equals true, then call
        CONNECT(RSON(LPPT), RPPT).
END
```

ANCESTOR(N) returns true if the leftmost son of node N is an ancestor link son.

RSON(N) returns a pointer to the rightmost son of node N, or NIL if N is terminal.

LSON(N) returns a pointer to the leftmost son of node N, or NIL if N is terminal.

FINAL(N) returns true if node N ends on a permitted final state.

JOIN(L, R): The JOIN routine performs the path extension and joining described in the previous sections. It also takes care of making sure innermost nodes begin and end on permitted states. If there is a higher level of merged structure, it places a pointer to any current "joined" PPT into the "empty" son cell of that higher structure. Finally, it calls the CONSTRAIN routine whenever any combined PPT is constructed.

(6) Attempting All Possible Connections

As described above, the CONNECT algorithm automatically attempts to connect the RPPT to lower nodes of the LPPT as well as just to the top node (by dropping down recursively). Clearly, the converse situation is also desired: namely, to attempt to join the top node of the LPPT to lower nodes of the RPPT (for instance, see Fig. V-56).

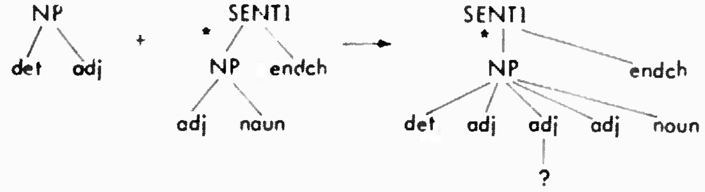


Fig. V-56.

To allow this to happen, CONNECT is first called with the top nodes of the two PPTs as arguments, and then drops down to successive inner sons of the RPPT and calls JOIN with each of those nodes, and the LPPT as arguments. When attempting these connections, the algorithm does not allow the top level of the RPPT to drop down recursively.

The only other possible connections which must be attempted are those made possible by popping with the LPPT (see Fig. V-57). The only difference between this popping and the popping connections discussed in the TREE algorithm is that, in this case, the PPT being popped need

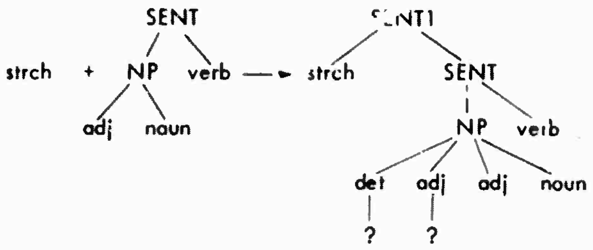


Fig. V-57.

not end on a final state. It is only necessary that it can be made to end or a final state by proposing a string of words containing, at ,most, one content word.

Thus, the CONNECT algorithm tries all possible pops it can make with the LPPT, and attempts to join each resulting structure to the RPPT.

5. CONSTRAIN: Compiling Additional Constraints

The part of the CONNECT algorithm described so far connects two PPTs by proposing words that might exist in betv :en, but only determines the parts of speech of the words proposed. From examining the context of these hypothesized wo^rds (i.e., the entire combined PPT), one can deduce additional const^raints as to what the words might be. The information needed for deducing these additional constraints is contained in the conditions which augment the arcs in the grammar.

An example is probably useful to highlight exactly what the problem here is. Suppose the two PPTs shown in Fig. V-58 were joined by hypothesizing a verb between them.

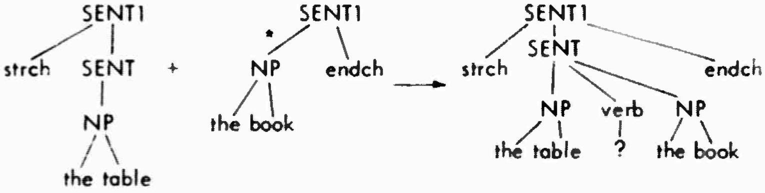


Fig. V-58.

A person looking a' this structure would know that not all verbs could exist in this particular context. The verb must be transitive; it must not require an animate subject, for instance; and it must be something a table can do to a book. In the normal operation of the transition net grammar, the tests that enforce these constraints are contained in the conditions associated with the arcs in the gramm.ar.

Thus, one possible way to determine what verb could fit into this context is to take each verb in tu^rn, and reparse the sentence with that verb and see if the parsing succeeds. This is a very time-consuming and inelegant approach. Somehow we want a way of "turning the constraints around" and have them tell us what they want, rather than just telling whether a particular string of words is acceptable or not.

In other words, the constraints must actively suggest action rather than merely answer yes-no questions. This, in fact is exactly what LPARS allows. The constraint tests in the arcs are written so that they can be run in two modes:

(1) In one mode, the tests are given two lists of information representing features associated with two structures which are to be checked for consistency. The test will return NIL if the arc cannot be traversed, or else return TRUE.

(2) In the other mode, the tests are passed one feature list and the atom "QUESTION" in place of the other feature list, indicating that LPARS wants the test to tell it what constraints must be satisfied, and then to return TRUE just as if those conditions had indeed been met. Each such constraint which must be met is appended to a global list.

(To allow constraints to be placed on a string of words, the system actually uses a set of distinct QUESTION atoms (Q1, Q2, Q3, Q4, . . .) depending on the position of the word in the proposed word string.)

When a combined PPT has been found and passed to the CONSTRAIN algorithm, a QUESTION atom is associated with each word which has been hypothesized. After processing the combined PPT, CONNECT then has a list of additional constraints which it can use to search its dictionary quickly to come up with a list of words which would fit.

This "turning around of the constraints" allows the algorithm to process the PPT only once, rather than once for each word of the proposed part of speech.

A simple example is probably useful at this point. Suppose we have the situation described above: a verb has been hypothesized between the two noun phrases "the table" and "the book".

Mode 1: Normally, when parsing a sentence, the algorithm would do the following.

(1) When parsing "the table", it associates the semantic information describing this noun phrase with a marker called SSUB in the list of information associated with the SENT mode.

(2) When parsing the verb, it assures subject-verb agreement between the information tagged SSUB and the information associated with the verb. For instance, if the verb required an inanimate subject, the test assures that the subject had been animate.

(3) When parsing the second NP, it similarly assures whatever agreement was appropriate.

(4) When making the final state test, it makes sure that the verb was transitive.

Mode 2: Since we do not know what the verb is, we construct the PPT given in Fig. V-59.

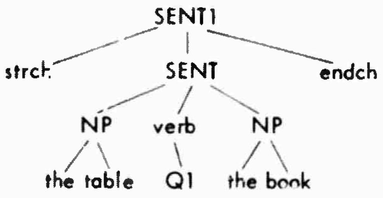


Fig. V-59.

This PPT is then subjected to the following parsing process:

(1) When parsing the first NP, the parser behaves just as in Mode 1 since nothing is changed.

(2) When parsing the verb, Mode 2 is activated. The algorithm must determine what constraints are to be satisfied. Since the subject is not animate, the constraint (ANIM MINUS) is added to the global list of constraints associated with the QUESTION atom Q1, indicating that the verb must not be one which requires an animate subject. In this case, the constraint (CONT MINUS) can also be added, indicating that the verb could not be a "containing" verb such as "contain" or "hold" since "table" does not make sense as the subject of such a verb. A new list of information to be associated with the SENT node is constructed, with a QUESTION atom (Q1) being associated with the tag VERB.

(3) When parsing the second NP, additional constraints might be found in much the same fashion.

(4) When performing the final state test, the test which assures that the verb is transitive sees the atom QUESTION associated with VERB, and would therefore add the constraint (VERB TRANS) to the list.

The net effect of this activity is to create the following list of constraints ((ANIM MINUS) (CONT MINUS) (VERB TRANS)). A list of verbs that match these specifications is made, and tested against the input. Any which succeed are incorporated into new PPTs and added to the list of PPTs which LPARS is using.

VI. OVERALL DESIGN OF THE LPARS SYSTEM

In this section, we review the whole of the LPARS system and discuss how the different parts fit together. We then describe the fallback (or backup) mechanism built into LPARS. Finally, some interesting issues involved in coordinating the various subparts of the system are discussed.

A. OVERVIEW OF THE LPARS SYSTEM

Figure VI-1 outlines the basic flow of control in LPARS; this involves 4 basic steps.

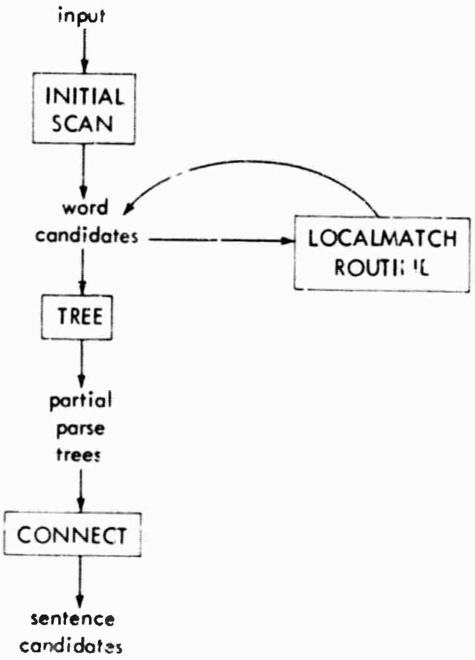


Fig. VI-1.

1. Initial Scan

The first step is the initial scan made at low phonetic distance through the entire utterance looking for fairly long words that are not too garbled.

2. LOCALMATCH Routine

The second step is the local higher-distance matching, searching for strings of small words and for more highly garbled longer words, in the areas beside and between the words found in the initial scan. This matching is done based on very local criteria, with no attempt to enforce any global coordination between processing going on in different parts of the utterance.

3. TREE Algorithm

All words found by the initial scan and the LOCALMATCH routine are input to the TREE algorithm which constructs as many different syntactic structures as it can, in all parts of the utterance, using these words. If all the words of the sentence have been found, then TREE will construct the entire sentence.

4. CONNECT Algorithm

These PPTs are then turned over to the CONNECT algorithm which attempts to join together pairs of the parse structures by proposing word-strings which might exist between them, and testing these word-strings out at high phonetic distance against the input. If any word-string matches successfully, a new combined parse structure is created.

B. TWO HEURISTICS USED BY LPARS

Two heuristics are used to augment the basic system described above.

One heuristic aids the initial local word recognition process. If the set of word candidates output by the initial scan and the LOCALMATCH routine leaves substantial gaps in the utterance, then the system reinitiates local word recognition in those gaps. It rescans the gaps at a higher phonetic distance, and processes any words found using the LOCALMATCH routine. This is a simple heuristic, which uses very local information to direct higher-distance scans in areas where further attention is obviously needed. In practice, it is not invoked very often.

The second heuristic helps the CONNECT algorithm to compensate for any partially erroneous structures which may have been built by the TREE algorithm. CONNECT will only accept as input PPTs which do not overlap, and which are within some maximum number of phonemes of one another. However, the TREE algorithm may have attached an erroneous word onto an otherwise correct structure. To compensate for this possibility, the system has two functions (RIP-L and RIP-R) which rip leftmost and rightmost words off a PPT, thus obtaining smaller subsumed structures.

By using these functions, the system can take two partially overlapping PPTs and rip words off either or both, to obtain two smaller nonoverlapping structures which can be turned over to CONNECT. Also, even if the PPTs do not overlap, the system can rip words off either or both so long as the number of phonemes between them does not exceed the maximum permissible, and turn the smaller PPTs over to CONNECT.

The two heuristics described above are useful tricks which help the system work more successfully. They do not, however, make any pretense of providing a systematic means of finding the correct sentence, if processing by the whole of LPARS fails.

C. SYSTEMATIC FALLBACK

We now describe a systematic method for fallback available to LPARS if the initial attempt at sentence recognition fails. This fallback method is exhaustive, and will eventually find any sentence (although it might well find erroneous sentences first). In concept, the fallback consists of reactivating the entire LPARS system at a higher set of phonetic distance thresholds.

Three significant phonetic distance thresholds used by LPARS are those used by (1) the initial scan, (2) the LOCALMATCH routine, and (3) the CONNECT algorithm.

These thresholds are initially set at levels which yield reasonable success rates without exhaustive amounts of computation. However, if the system fails to find a sentence using the standard levels for these thresholds, the system can raise the thresholds and try again. Furthermore, it can do this without having to repeat most of the work already done (as discussed below in Sec. 1). In fact, LPARS can continue this fallback process for several iterations until a sentence is found.

In theory, of course, continuing the iterative fallback process to absurdly high-distance thresholds would result in the system generating every possible acceptable sentence that it could construct with its vocabulary, since eventually, at a high enough threshold, every word would match everywhere in the utterance. In practice, one must decide upon some maximum threshold level, after which one acknowledges failure.

The remainder of this section discusses how one might allow the fallback to higher-distance thresholds to take place without requiring that all the previous work be repeated.

It is easy to achieve this goal in the TREE algorithm. The additional words found by the higher-threshold initial processing are given to TREE together with the old words and the phoneme position table containing the previously constructed PPTs. TREE can proceed with the provision that now each syntactic connection must be made either between any PPT and a new word, or between a PPT containing a new word and any word. In other words, one does not attempt to connect any of the old PPTs to any of the old words. This restriction allows TREE to find all possible syntactic structures, yet the work done in the previous TREE processing remains intact.

To allow fallback to higher-distance thresholds without repeating the lexical work done previously is simple in concept, but costly in storage. It requires that each partial lexical match which was aborted, because it exceeded the previous threshold, be recorded so that it can be restarted. This involves a great deal of storage because the initial scan, for instance, matches most words in the vocabulary against every phoneme position in the sentence. Each one of these matching attempts usually involves several substitution-deletion combinations.

Thus, if there were 50 words, 50 phonemes in the utterance, an average of 10 partial matches per recognition attempt, and if two words were required to record each aborted attempt, then 50,000 words of storage would be required (all to process one sentence). On the other hand, perhaps this is reasonable if one considers that it represents the state of 25,000 parallel, independent processes that have been temporarily suspended. LPARS at present does not implement this approach, but rather recomputes all the lexical matching at the higher-distance thresholds if it is forced to use this fallback.

D. DESIGN CONSIDERATIONS FOR THE LPARS SYSTEM

Sections II through V discussed the various sub-parts of the LPARS system: the initial scan, the LOCALMATCH routine, and the TREE and CONNECT algorithms. These are the tools which the LPARS system uses to recognize an utterance. But just building these tools is not enough. One must understard what the various tools are useful for, what they are not useful for, and then design the system as a whole around this understanding.

Therefore, in this section we examine the tools which have been developed to see how they are best used.

1. Two Modes of Joining PPTs

Consider the task of examining two PPTs developed from an input utterance, together with an intervening input segment, to construct a combined PPT. There are two fundamentally different ways to proceed. The alternatives could be called the "bottom-up" (or "data-directed") approach vs the "top-down" (or "grammar-directed") approach.

a. Bottom-Up (BU) Connection

In BU (or data-directed) connection, the system scans the segment between the two PPTs at a high phonetic distance, and then tries to construct a combined PPT from the two input PPTs together with any words found. In this mode of connection, the previous PPTs are used to indicate a section of input to be examined at high phonetic distance, but not to indicate explicitly what word-strings might be expected to occur there.

b. Top-Down Connection

Top-down (or grammar-directed) connection is that performed by the CONNECT routine described in Sec. V. Here, the grammar is used to indicate what word-strings might possibly exist in the gap, and only these specific word-strings are tested against the input.

A number of issues might be discussed to evaluate the relative merits of these two strategies. Here, we look at only one of these issues, namely, how the two strategies perform as a function of the amount of "context" contained in the two PPTs. To illustrate this issue, consider the following two examples.

Example 1

Suppose LPARS is attempting to connect the two PPTs shown in Fig. VI-2.

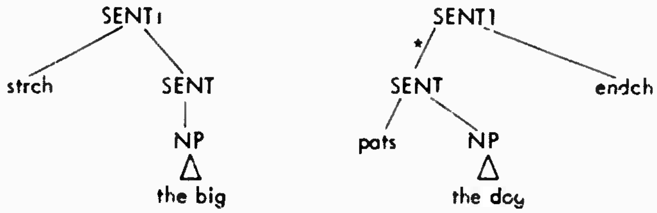


Fig. VI-2.

Using the top-down, grammar-directed approach, LPARS discovers that only one possible word-string can fill this gap, namely a noun. (Remember that only word-strings containing, at most, one content word are considered.) Therefore, it compiles a list of appropriate nouns and tests them against the input to see if any match.

Example 2

Suppose, on the other hand, LPARS is attempting to connect the two PPTs shown in Fig. VI-3

NP
the dictionary
NP
red ashtray

Fig. VI-3.

Using the top-down, grammar-directed approach, LPARS discovers (even with its very simple grammar) that there is a very large number of possible word-string connections; these include:

(1) is contained by the
(2) supports the
(3) in the
(4) in the big
(5) by the
(6) by the big
(7) on the book in the

(8) which the
(9) on which the
(10) on the book which the
(11) which the book on the
(12) supports by the
.
.
. (and many more).

As a result, using the top-down algorithm with these two PPTs is not very efficient. Doing so will exercise almost the entire grammar, and generate almost every word-string conceivable, each of which must be matched against the input at high phonetic distance. Clearly, this approach is not very selective in bringing context to our aid.

On the other hand, is this surprising? After all, a noun phrase is a very basic building block of an English sentence. It can occur in many contexts, and is structurally the same in each context. Therefore, it is reasonable that two such primitive structures can be connected many ways.

What then does this imply?

(1) Need for "Sufficient Context"

One implication is that we should not use the top-down CONNECT algorithm unless the PPTs involved provide "sufficient context," in some sense, to make the word-string possibilities quite selective. This restriction is not surprising when we consider that the CONNECT algorithm is a very powerful tool; given any two parse structures conceivable, it processes the grammar in a very rigorous and systematic fashion to investigate every possible way that they might be connected. The price paid for this power and generality is that we must be selective about the problems we ask the routine to solve.

This, in fact, is why we have the LOCALMATCH routine. As currently implemented in LPARS, the LOCALMATCH routine works on very local criteria (words and word pairs) to investigate the areas around them. (In an extension to the LPARS system, this routine might investigate somewhat more-complex structures, particularly in searching for particular word-strings which are semantically quite likely to be present.)

Moreover, the LOCALMATCH routine investigates around and between fairly simple local constructs in a way that avoids a very systematic, costly processing of the grammar. Also, it can be primed to look primarily for highly likely connections. Thus, the LOCALMATCH routine cuts corners (by using simple tests and heuristics). It can do so because its goal is not necessarily to uncover the entire utterance, merely substantial portions of it. Once these portions are uncovered, the full power and generality of the CONNECT algorithm is standing by to complete the job.

(2) Implication for Scan Threshold Levels

A second, related implication is that, to use the CONNECT routine efficiently, we must tune the system so that the PPTs constructed after the initial·local processing usually have sufficient context to be input to that routine. Only when this does not happen, does the BU connect routine need be called, or the more-general fallback be invoked.

In other words, to use the CONNECT routine effectively, the phonetic distance thresholds of the initial scan and LOCALMATCH routine must be set so that a high percentage of the correct words in the utterance is usually found.

2. Vocabulary Density

A second important concept that enters into this discussion is the somewhat vague term "vocabulary density." Let us take two examples to see what vocabulary density might mean.

(a) In a sparse (non-dense) vocabulary, one might set the local scan thresholds to uncover on the average 85 percent of the correct words in the sentence, and generally only have a few incorrect words found.

(b) In a denser vocabulary, when one sets the local scan thresholds so that 85 percent of the correct words are found, a large number of incorrect words are usually found as well.

This is certainly a highly specific concept of vocabulary density, but a useful one for our purposes.

a. Factors Influencing Density

There are clearly a number of factors influencing vocabulary density; these include

(1) vocabulary size

(2) front-end accuracy

(3) average size of word

(4) amount of semantic selectivity which can be brought to bear in the initial local processing

(5) utilization of supra-segmental information.

b. Embedding of Correct Sentence Fragments

Vocabulary density affects the two connection strategies discussed above because it influences the degree to which correct sentence fragments are likely to be embedded in larger structures which include some of the erroneous words.

The more erroneous words discovered, the more likely it is that some of these words will become attached by the TREE algorithm to a string of correct words. Of course, if all the correct words are found, or if the erroneous word overlaps with a correct word which has been found, no harm will be done. In general, however, the presence of incorrect words attached to correct sentence fragments makes the job of isolating the correct sentence fragments difficult, and therefore complicates the job of finding the appropriate input segments to be investigated next.

c. How CONNECT Deals with Embedding

The top-down CONNECT algorithm can deal with this embedding problem by using the RIP-R and RIP-L routines described in Sec. B above. In so doing, the system postulates that certain fragments are correct embedded fragments, and tries to connect them. The denser the vocabulary, the more such trials the algorithm would have to make with a given set of PPTs. The degree to which this approach is practical depends on an evaluation of the work involved in postulating embedded fragments and in processing them with the CONNECT algorithm.

The BU algorithm has potentially even more difficulty working in the presence of a great deal of embedding. This difficulty arises because, in investigating a set of PPTs for the connection of embedded fragments, one might investigate a number of different input segments (perhaps, in total, consuming a substantial part of the input). With TD connection, during such investigation a great deal of selectivity is brought to bear, as far as the word-strings looked for in different places. With the BU algorithm, however, such selectivity is impossible. If a large portion of the input is involved, one probably might just as well rescan the whole input at a high phonetic distance.

Thus, the denser the vocabulary, the more difficult it is for both algorithms to work efficiently.

The vocabulary used in LPARS during its testing was fairly sparse, so these problems did not occur to too large a degree. When experimenting with the system with higher scrambling levels, and using smaller words, the problems were observed and, since they were interesting problems, this section has tried to isolate and summarize them. The present system has only attempted to attack these problems in a fairly limited fashion.

3. Current LPARS Approach

During evaluation of the LPARS system, described in Sec. VII, only the grammar-directed CONNECT algorithm was used, and only PPTs containing "sufficient context" were passed to this algorithm. A PPT was deemed to have "sufficient context" (a) if it included a STRCH or an ENDCH, (b) if it was not a single word, and (c) if it was not a NP or partial NP. When using a more-complex grammar, of course, these tests would probably need to be more sophisticated.

The phonetic distance thresholds were set so that most of the correct words were found after the initial processing. Thus, in general, PPTs containing sufficient context were built up and could be passed to the CONNECT algorithm. If not, the system would fall back and iterate at higher phonetic distance thresholds.

VII. EXPERIMENTAL EVALUATION OF LPARS, AND SAMPLE RUNS

This section describes the experimental evaluation of the LPARS system, and then discusses a few sample runs which illustrate LPARS' activity when investigating a sentence.

A. EXPERIMENTAL EVALUATION

During the experimental evaluation of LPARS, fifty sentences were input to the system to be recognized (the sentences used are listed in Appendix B). The following steps were taken during the recognition of each sentence.

1. Scrambling:— First, the correct phonetic spelling of the sentence is given to the scrambling program (the front-end simulator) which transforms the sentence into a string of "scrambled" phonemes containing a great deal of error in the form of deletion and substitution. The statistics used in performing this scrambling are exactly those described in Sec. III. Since the scrambling is based on random error probabilities, some sentences produced contain relatively small amounts of error, while others contain a great deal.

2. Initial Local Processing:— The string of phonemes is then input to the LPARS system which performs its initial scan and higher-distance local matching, attempting to find most of the words in the sentence. The phonetic distance thresholds of these scans were set so that approximately 85 percent of the correct words are usually found by this local processing.

3. PPT Construction:— Next, the words found by the initial local processing are input to the TREE algorithm which builds as many different parse structures as it can in all parts of the utterance (as described in Sec. V-C). If one or more entire sentence is found, then LPARS proceeds no further.

4. PPT Connection:— Next, pairs of PPTs are input to the CONNECT routine which attempts to construct combined PPTs as described in Sec. V-D. Any such combined PPT which is found is included in the set of PPTs being investigated. If this process discovers one or more possible sentences, then LPARS proceeds no further.

5. Iteration at Higher Thresholds:— If the PPT connection is unsuccessful, then LPARS raises the thresholds at which the various scans are made and repeats steps 2, 3, and 4 again. During the evaluation, the phonetic distance thresholds used for the initial scan, the local higher-distance matching, and the PPT connection were 17, 24, and 34, respectively during the first iteration; and 22, 26, and 39 during the second iteration. If no sentence has been found at the end of the second iteration, then processing stops and failure is acknowledged.

An attempt to recognize a given sentence, as outlined above, can result in any of the following results.

(1) An entire sentence is constructed by the TREE algorithm from the words found by the local processing.

(2) A sentence is found by the algorithm that connects PPTs.

(3) No sentence is found at all.

Processing ceases at the end of any stage in which one or more possible sentences is found. When sentences are found, it may be during either the first or second iteration of the system. Of course, it may be that none of the sentences found is correct.

In evaluating LPARS, if several possible sentences were found, the recognition of the sentence was considered successful if the correct sentence was included among them. When several sentences were found, the correct sentence was almost always the best match.

1. Evaluation

During the actual evaluation of LPARS, fifty sentences were input with the following results:

> 23 sentences were correctly recognized from words found only by the local processing in the first iteration.
>
> 19 sentences were correctly recognized by the PPT CONNECTION algorithm in the first iteration.
>
> 3 sentences were correctly recognized in the second iteration of the system.
>
> 2 sentences resulted in incorrect recognition. In both cases, the sentence found differed from the input sentence in a single content word.
>
> 3 sentences resulted in failure to find any possible sentence at all. In two cases, two adjacent content words were badly garbled, and therefore the CONNECT algorithm was unable to propose the correct word-string. In the third case, LPARS crashed irretrievably due to a program bug.

Thus, the overall success rate of LPARS with the fifty input sentences was 90 percent.

2. Comments on LPARS' Evaluation

The LPARS evaluation described above primarily establishes that, given a simulation of a fairly crude front end, the ideas embodied in LPARS can be made to work acceptably.

Certainly there are a number of interesting further questions which one might like to ask, such as how the overall system performance is affected by varying such parameters as:

(a) scrambling probabilities
(b) scrambling characteristics
(c) vocabulary size
(d) word size
(e) the phonetic distance levels of the various scans.

It would be very difficult to pursue such issues systematically with the current implementation of LPARS since it is very slow. Often, 10 to 15 minutes or more of dedicated CPU time is required per sentence. (This inefficiency is not inherent in the LPARS design, but rather of the current implementation.)

In any case, the primary goal of the present research is on developing an approach to parsing which allows syntactic analysis to proceed systematically from anywhere in a sentence, and to understand the problems and issues involved in such an approach.

In the present evaluation of LPARS, we have described a form of scrambled input, argued that it has the basic characteristics of spoken input for our present purposes, and then demonstrated that LPARS can operate successfully given such input. This success demonstrates that the ideas embedded in LPARS are fundamentally sound and coherently formulated, and that, for instance, there are no unforeseen factors which somehow invalidate the approach taken.

B. LPARS SAMPLE RUNS

We now exhibit a number of sample runs of the LPARS system, illustrating various aspects of the system's operation. In these examples, comments output by the LPARS system appear in capitals. Additional comments in small letters have been added by the author to make the examples more intelligible.

Example 1

This example shows a sentence which is recognized entirely from words found by the initial local processing.

```
(UNSCRAMBLED  SENTENCE)
(THE  WASTEBASKET  WHICH  THE  GREEN  COFFEETABLE  SUPPORT  S
     CONTAIN  S  THE  DICTIONARY)
(TH  SW  W  AA  S  T  B  E  S  K  SW  T  W  I  CH  TH  SW  G  R  EE  N  K
AW  F  EE  T  AA  B  L  S  SW  P  O  R  T  Z  K  SW  N  T  AA  N  Z  TH  SW
D  I  K  SH  SW  N  AA  R  EE)
```

```
(SCRAMBLED  SENTENCE)
(T  Y  W  AW  D  P  E  ZH  P  ZH  W  I  CH  Y  L  I  NG  G  AA  B  EE  P  AW
P  R  H  SW  TH  O  L  P  J  B  SW  U  T  AW  M  H  D  SW  G  TH  Z  Y  M  AW
I  EE)
```

```
(INITIAL  SCAN)
( (STRCH  0  0)  (STEPLADDER  8  15  150)  (COFFEETABLE  18  25  15)
(SUPPORT  26  31  5)  (CONTAIN  33  38  10)  (DICTIONARY  42  49  95)
(ENDCH  50  50) )
```

The initial scan finds three correct and one erroneous words. The words are ordered as they appear in the sentence. (Recall that the three numbers associated with each word candidate are start phoneme, end phoneme, and total phonetic distance of the match.)

```
(LOCAL  HIGHER  DISTANCE  MATCHING)
( (STRCH  0  0)  (CONTAIN  1  5  5)  (THE  1  2  0)  (PLACE  3  5  90)
(WASTEBASKET  3  10  185)  (THE  6  7  0)  (STEPLADDER  8  15  150)
(WHICH  11  12  12)  (CLUTTER  11  14  [illegible])  (THE  13  14  12)
(THE 15 17 0) (GREEN 15 17 60) (COFFEETABLE 18 25 15)
(SUPPORT 26 31 5) (CLUTTER 30 32 110) (ROBERT 30 33 120)
(THE 32 33 0) (THE 33 33 0) (CONTAIN 33 38 10)
(BOOKCASE 34 38 110) (ENJOY 34 38 120) (THE 39 41 0)
(PERSON 42 46 105) (DICTIONARY 42 49 95) (ENDCH 50 50) )
```

A number of additional words are found by the higher-distance matching, including all the correct words in the sentence. Next, the TREE algorithm constructs all possible local partial parse structures from the words found. These trees are printed out below.

```
(PARTIAL PARSE TREES SORTED BY LENGTH)
(STRCH ( (THE WASTEBASKET (WHICH (THE GREEN COFFEETABLE)
    SUPPORT AFX) ) CONTAIN AFX (THE DICTIONARY) ) ENDCH)
(* (* (* (THE GREEN COFFEETABLE) SUPPORT AFX)
    CONTAIN AFX (THE DICTIONARY) ) ENDCH)
(* (* (* (THE GREEN COFFEETABLE) SUPPORT AFX)
    ENJOY AFX (THE DICTIONARY) ) ENDCH)
(STRCH ( (THE WASTEBASKET) CLUTTER AFX (THE COFFEETABLE) ) )
(* (* (THE GREEN COFFEETABLE) SUPPORT AFX) ENJOY
    AFX (THE PERSON) )
(* (* (CLUTTER A      CONTAIN AFX (THE DICTIONARY) )
    ENDCH)
(* (ROBERT ENJOY AFX (THE DICTIONARY) ) ENDCH)
(* (* ROBERT ENJOY AFX (THE DICTIONARY) ) ENDCH)
(* (* (CLUTTER AFX) ENJOY AFX (THE DICTIONARY) ) ENDCH)
(* (THE DICTIONARY) ENDCH)
(STRCH (CONTAIN AFX (THE) ) )
(ROBERT ENJOY AFX (THE PERSON) )
(* ROBERT ENJOY AFX (THE PERSON) )
(* (CLUTTER AFX) ENJOY AFX (THE PERSON) )
(PLACE AFX (THE STEPLADDER) )
(CLUTTER AFX (THE BOOKCASE) )
```

The symbol * means that the following son is an ancestor link son. The symbol AFX represents an AFFIX arc which has been traversed. (Any LAMBDA arcs traversed are not indicated in the printout.) Thus, for instance, the PPT (* (THE DICTIONARY) ENDCH) represents the structure shown in Fig. VII-1.

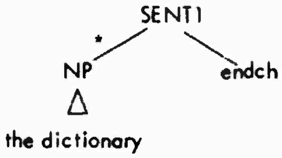


Fig. VII-1.

Since an entire sentence (see Fig. VII-2) is among these PPTs, LPARS proceeds no further.

```
(SENTENCES  FOUND)
(STRCH  ( (THE  WASTEBASKET  (WHICH  (THE  GREEN  COFFEETABLE)
SUPPORT  AFX) )  CONTAIN  AFX  (THE  DICTIONARY) )  ENDCH)
```

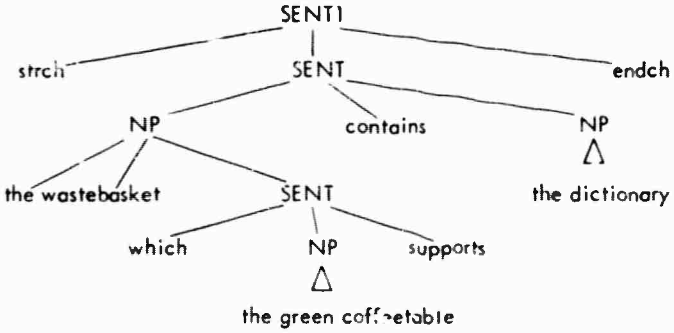


Fig. VII-2.

<u>Example 2</u>

```
(UNSCRAMBLED  SENTENCE)
(THE  COFFEETABLE  NEXTTO  THE  BOOKCASE  IS  ILLUMINATE  D
      BY  THE  TABLELAMP)
(TH  SW  K  AW  F  EE  T  AA  B  I  N  SW  K  S  T  OO  TH  SW  B  U  K  AA
S  I  Z  I  L  OO  M  I  N  AA  I  T  B  A  I  TH  SW  T  AA  B  L  AE  M  P)


(SCRAMBLED  SENTENCE)
(SW  G  AA  V  EE  G  AW  V  R  I  SW  T  Z  P  OO  SW  DH  U  G  AW  S  J
S  I  R  OO  N  I  M  AA  J  P  A  EE  F  SW  P  AA  K  I  I  SW)
```

```
(INITIAL SCAN)
( (STRCH 0 0) (COFFEETABLE 2 9 15) (TABLELAMP 6 12 115)
(GREEN 8 11 50) (NEXTTO 10 15 40) (BOOKCASE 17 21 5)
(ILLUMINATE 24 31 5) (ENDCH 43 43) )
```

Here, four correct and two erroneous words are found. (Note that the word candidate "tablelamp" is erroneous since it is in the wrong place.)

```
(LOCAL HIGHER DISTANCE MATCHING)
( (STRCH 0 0) (THE 1 1 0) (COFFEETABLE 2 9 15)
(TABLELAMP 6 12 115) (GREEN 8 11 50) (IS 10 12 40)
(NEXTTO 10 15 40) (SUPPORT 13 17 140) (THE 16 16 0)
(BOOKCASE 17 21 5) (IS 22 23 0) (ILLUMINATE 24 31 5)
(THE 32 34 0) (BOOKCASE 35 39 115) (ENDCH 43 43) )
```

In this case, not all the words are found by the initial local processing. Therefore, the system constructs only the partial structures shown below.

```
(PARTIAL PARSE TREES SORTED BY LENGTH)
(STRCH ( (THE COFFEETABLE (NEXTTO (THE BOOKCASE) ) ) IS
     ILLUMINATE AFX) )
(STRCH ( (THE COFFEETABLE (NEXTTO) ) ) )
(This PPT is not subsumed by the previous one, even though it might appear to be.
It represents the PRC structure "the coffeetable nextto which. . .".)
(STRCH ( (THE COFFEETABLE) IS) )
(* (NEXTTO (THE BOOKCASE) ) IS ILLUMINATE AFX)
(* (THE BOOKCASE) IS ILLUMINATE AFX)
( (THE BOOKCASE) IS ILLUMINATE AFX)
(* TABLELAMP SUPPORT AFX)
ENDCH
(IS SUPPORT AFX)
(THE BOOKCASE)
GREEN
```

The system must now use the CONNECT algorithm to attempt to discover an entire sentence.

```
(PPT CONNECTION)
```

```
(PPT PAIR)
     (STRCH ((THE COFFEETABLE (NEXTTO (THE BOOKCASE)))
          IS ILLUMINATE AFX))
     ENDCH


PPTS TO BE CONNECTED
     (STRCH ((THE COFFEETABLE (NEXTTO (THE BOOKCASE)))
          IS ILLUMINATE AFX))
     ENDCH
```

The system now calls the CONNECT algorithm with the two PPTs shown in Fig. VII-3.

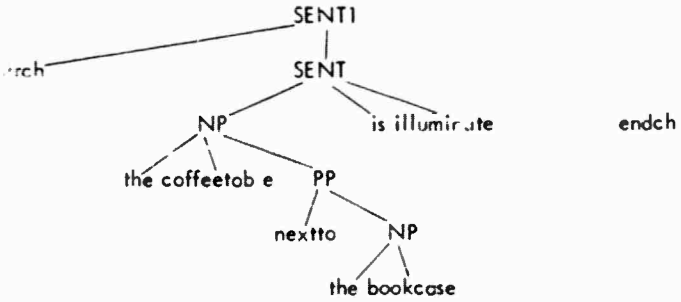


Fig. VII-3.

```
PROPOSED PPT
(STRCH ((THE COFFEETABLE (NEXTTO (THE BOOKCASE)))
     IS ILLUMINATE AFX BY NP) ENDCH)
```

A proposed combined PPT is discovered (see Fig. VII-4)

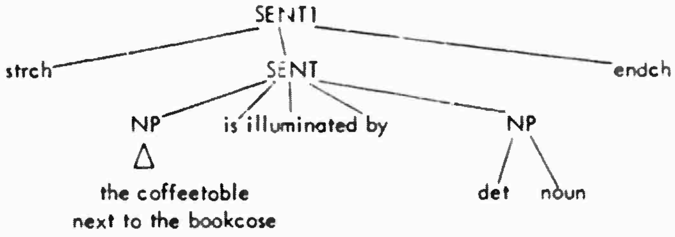


Fig. VII-4.

All word-strings which match the hypothesized portion of this proposed PPT are tested against the input. To help us, LPARS types out one such word-string.

```
POSSIBLE WORD STRING (D BY THE CHANDELIER)
```

This word-string does, in fact, match.

```
WORD STRING MATCHED ((D BY THE CHANDELIER) 32 42 395)
COMBINED PPT
(STRCH ((THE COFFEETABLE (NEXTTO (THE BOOKCASE)))
    IS ILLUMINATE D BY (THE CHANDELIER)) ENDCH)
```

So do others. (Notice that only semantically reasonable nouns are matched.)

```
WORD STRING MATCHED ((D BY THE FLASHLIGHT) 32 42 415)
COMBINED PPT
(STRCH ((THE COFFEETABLE (NEXTTO (THE BOOKCASE)))
    IS ILLUMINATE D BY (THE FLASHLIGHT)) ENDCH)
WORD STRING MATCHED ((D BY THE TABLELAMP) 32 42 225)
COMBINED PPT
(STRCH ((THE COFFEETABLE (NEXTTO (THE BOOKCASE)))
    IS ILLUMINATE D BY (THE TABLELAMP)) ENDCH)
```

Three possible word-strings match within the high phonetic distance of this match. Notice, however, that the word-string containing "tablelamp" matches at the lowest phonetic distance.

This example illustrates an interesting aspect of LPARS. Even though words in the vocabulary are not very close phonetically, when LPARS is performing matches at very high phonetic distances, often several word-strings may match. This, of course, is not too surprising. The more garbled a section is, the more difficult it would be to determine which words match there.

To deal with this problem, a system would have to be able to do one of three things: call front-end routines to help choose between the different possibilities, use semantics to rule some out, or ask the speaker which he actually meant.

The advantage of a vertically organized system like LPARS is that it only investigates such segments very selectively: when a great deal of contextual information indicates the necessity, and when a great deal of context can be used to help constrain the words that might be present.

Example 3

```
(UNSCRAMBLED SENTENCE)
(PLACE THE GREEN DICTIONARY ON THE BOOKCASE)
(P L AA S TH SW G R EE N D I K SH SW N AA R EE AW N TH
SW B U K AA S)
```

```
(SCRAMBLED SENTENCE)
(L AA D M SW B W EE F EE K H SW N AW R EE AW NG B Y K
U P AW ZH)


(INITIAL SCAN)
((STRCH 0 0) (DICTIONARY 9 17 15) (CLUTTER 20 23 65)
(BOOKCASE 22 26 5) (ENDCH 27 27))


(LOCAL HIGHER DISTANCE MATCHING)
((STRCH 0 0) (THE 1 2 0) (FRIENDLY 3 8 135) (GREEN 6 8 90)
(DICTIONARY 9 17 15) (ONTHELEFTOF 18 24 185)
(ONTOPOF 18 24 120) (ON 18 19 5) (IS 18 19 0) (THE 20 21 5)
(CLUTTER 20 23 65) (BOOKCASE 22 26 5) (ENDCH 27 27))


(PARTIAL PARSE TREES SORTED BY LENGTH)
(* (* (GREEN DICTIONARY) (ON (THE BOOKCASE))) ENDCH)
(* (GREEN DICTIONARY (ON (THE BOOKCASE))) ENDCH)
(* (ON (THE BOOKCASE)) ENDCH)
(STRCH ((THE FRIENDLY)))
(* (GREEN DICTIONARY) (ONTHELEFTOF))
(GREEN DICTIONARY (ONTHELEFTOF))
(GREEN DICTIONARY (ONTHELEFTOF))
(* (GREEN DICTIONARY) (ONTOPOF))
(GREEN DICTIONARY (ONTOPOF))
(GREEN DICTIONARY (ONTOPOF))
(* (GREEN DICTIONARY) IS CLUTTER AFX)
(* (THE BOOKCASE) ENDCH)
(GREEN DICTIONARY (ON))
(IS (THE BOOKCASE))
```

In this example, when it comes time to connect PPTs, two PPTs partially overlap, so the system rips words off to try to connect subtrees. First, it rips "the friendly" of the left-hand PPT, and attempts connection with the two resulting structures (see Figs. VII-5 and VII-6).

```
(PPT CONNECTION)
(PPT PAIR)
     (STRCH ( (THE FRIENDLY) ) )
     (* (* (GREEN DICTIONARY) (ON (THE BOOKCASE) ) ) ENDCH)
```

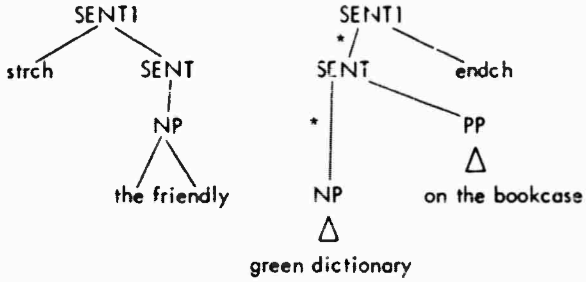


Fig. VII-5.

```
PPTS TO BE CONNECTED
     STRCH
     (* (* (GREEN DICTIONARY) (ON (THE BOOKCASE) ) ) ENDCH)
```

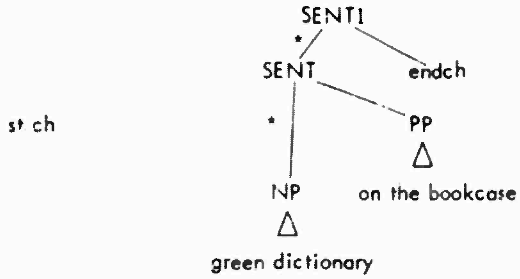


Fig. VII-6.

```
PROPOSED PPT
(STRCH (WH VERB AFX (DET GREEN DICTIONARY)
     (ON (THE BOOKCASE) ) ) ENDCH)
POSSIBLE WORD STRING (WHO PLACE S THE)
```

The first possible combined PPT hypothesized is a question (Fig. VII-7).

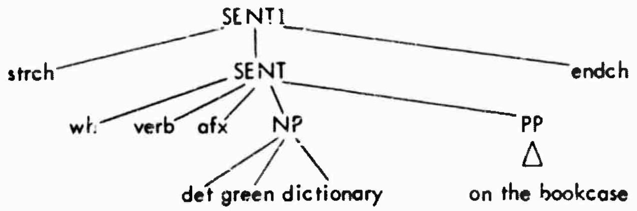


Fig. VII-7.

```
PROPOSED PPT
(STRCH (VERB AFX (DET GREEN DICTIONARY)
    (ON (THE BOOKCASE) ) ) ENDCH)
POSSIBLE WORD STRING (PLACE O THE)


WORD STRING MATCHED ( (PLACE O THE) 1 5 155)
COMBINED PPT
(STRCH (PLACE O (THE GREEN DICTIONARY) (ON (THE BOOKCASE) ) )
    ENDCH)
```

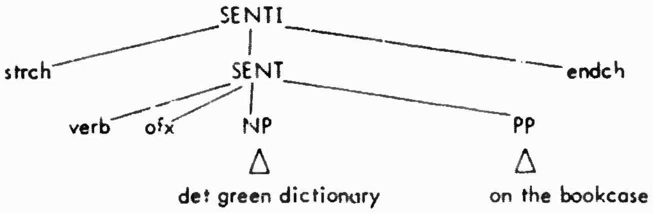


Fig. VII-8.

The next possibility is an imperative command. One of these word-strings matches and a combined PPT is constructed (Fig. VII-8). (The word "O" is the root affix, the null string.)

(These examples stop when the correct sentence is found. The algorithm itself is set up to continue to explore different possible connections. It is quite striking, however, watching LPARS work, how quickly the system can often zero-in on promising connections, once given a set of PPTs.)

Example 4

In this example, the CONNECT algorithm is called to match word-strings in the middle of an utterance.

```
(UNSCRAMBLED SENTENCE)
(THE BROWN SIDETABLE BESIDE THE FIREPLACE CONCEAL S THE
    ARMCHAIR)
(TH SW B R A OO N S A I D T AA B L B EE S A I D TH SW
F A I R P L AA S P SW N S EE L Z TH SW A R M CH AA R)
```

```
(SCRAMBLED SENTENCE)
(DH I R E OO NG Z AR EE N AW V D EE AW AE E TH Y P A I
W P AW H D SW W H EE R H Y A SW M AW R)
```

```
(INITIAL SCAN)
( (STRCH 0 0) (FIREPLACE 20 26 95) (CONCEAL 27 32 15)
(ENDCH 40 40) )
```

```
(LOCAL HIGHER DISTANCE MATCHING)
( (STRCH 0 0) (FIREPLACE 20 26 95) (IS 27 28 0)
(CONCEAL 27 32 15  (CLUTTER 29 31 120) (THE 32 34 0)
(THE 33 34 0) (ARMCHAIR 35 39 125) (ENDCH 40 40) )
```

No words have been found between phonemes 0 to 20, so an immediate higher-distance scan is made.

```
(SPECIAL SCAN)
( (THE 1 2 0) (BROWN 3 6 55) (SIDETABLE 7 13 140) )
```

```
(PARTIAL PARSE TREES SORTED BY LENGTH)
(* (* FIREPLACE CONCEAL AFX (THE ARMCHAIR) ) ENDCH)
(STRCH ( (THE BROWN SIDETABLE) ) )
(* (THE ARMCHAIR) ENDCH)
(* FIREPLACE IS CLUTTER AFX)
```

```
(PPT  CONNECTION)


(PPT  PAIR)
     (STRCH  ((THE  BROWN  SIDETABLE)))
     (*  (*  FIREPLACE  CONCEAL  AFX  (THE  ARMCHAIR))
          ENDCH)


PPTS  TO  BE  CONNECTED
     (STRCH  ((THE  BROWN  SIDETABLE)))
     (*  (*  FIREPLACE  CONCEAL  AFX  (THE  ARMCHAIR))
          ENDCH)
```

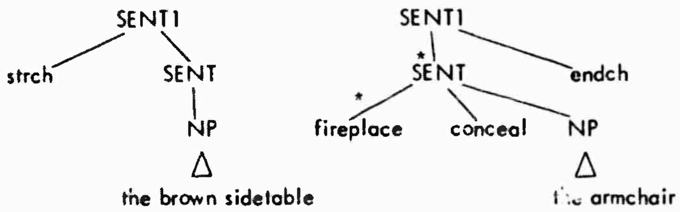


Fig. VII-9.

```
PROPOSED  PPT
(STRCH  ((THE  BROWN  SIDETABLE  (PREP  (DET  FIREPLACE)))
     CONCEAL  AFX  (THE  ARMCHAIR))  ENDCH)
POSSIBLE  WORD  STRING  (INFRONTOF  THE)
```

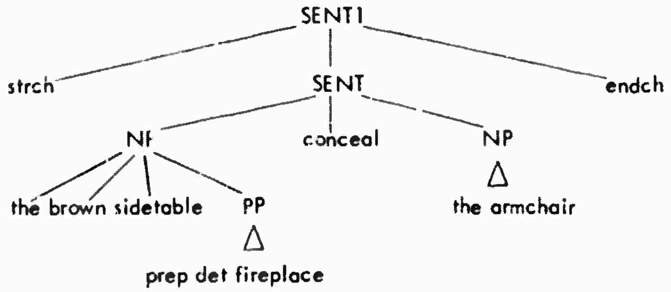


Fig. VII-10.

```
WORD STRING MATCHED ( (BESIDE THE) 14 19 200)
COMBINED PPT
(STRCH ( (THE BROWN SIDETABLE (BESIDE (THE FIREPLACE) ) )
     LAM CONCEAL AFX (THE ARMCHAIR) ) ENDCH)
```

VIII. CONCLUSIONS, BASIC ISSUES, AND AREAS FOR FURTHER WORK

This section places the present work into broader focus by discussing some interesting issues brought to light. Most of these issues relate to the local nature of LPARS' design. In this discussion, we describe some of the fundamental problems involved in designing a system like LPARS, and how the approach described solves these problems. Finally, we conclude by discussing a number of interesting areas for further work.

A. BASIC ISSUES BROUGHT TO LIGHT

1. Advantages of a Local Approach

The most obvious issue concerns the relative merits of LPARS' local approach which has the disadvantage of requiring a great deal of initial lexical processing. The entire vocabulary is matched against the entire input. If words had been recognized with sufficient accuracy in the early part of the sentence, some of this lexical work might not really be necessary.

The most obvious advantage of the local approach over a more left-to-right approach is that the left-to-right approach would have difficulty if a relatively large amount of error occurred near the beginning of the sentence. It would be unable to get at the necessary context to help it deal with this ambiguity, to isolate it, and then make specific suggestions of words to look for. A local approach, on the other hand, allows information on both sides of an erroneous area to be brought to the assistance of the parser.

There is another way of stating this argument. Namely, the local approach allows the highest phonetic distance scans to be made taking advantage of the maximum possible amount of lower phonetic distance (well matching) information available. Thus, the higher-distance scans are "data directed" as completely as possible. (This is not true of a left-to-right approach.)

In particular, the algorithm that connects PPTs has available to it all possible information from previous lower-level parsing and processing. As a result, all partial parse structures constructed from lower phonetic distance processing can be examined, and the most promising connections can be tried first. This is a very powerful capability, and one which no left-to-right algorithm can achieve.

In a left-to-right system, it would be much more difficult for the system to know when a high-distance match might be desirable, especially if one wants to do such high-distance matching quite selectively. It is certainly impossible to get information as to the right-hand boundary for such a match.

2. Syntactic Economy

A second issue might be called "syntactic economy." Syntactic conciseness or economy is a vague term, especially in reference to natural language parsing where there are many different formalisms for defining syntactic relationships: transformations with side conditions, transition nets augmented by tests on arcs, PROGRAMMAR procedures, etc. Syntactic conciseness is usually a qualitative judgment made subjectively by looking at a grammatical representation of a set of linguistic facts.

An interesting aspect of local parsing as described here is that it provides motivation for talking explicitly about a particular type of syntactic economy: namely, the degree to which the parsing logic for similar structures is collapsed down to a minimum syntactic skeleton.

In the LPARS local parser, there is very real incentive to try to collapse the grammatical skeleton as much as possible, since every time a local syntactic structure is recognized, it is asserted for every instance that occurs in the grammar. Therefore, if the grammar contains thirty separate instances where a noun-phrase can be followed by a verb, when such a pattern is recognized, all thirty possibilities must be constructed, even though many of them might never even be considered in a left-to-right system analyzing a text version of the sentence.

Such "ineconomy" would lead to excessive inefficiency in the construction of local PPTs and would also complicate the job of connecting different segments since there would be so many to choose from.

LPARS achieves conciseness by using a transition net grammar augmented by syntactic features. The transition net itself is a very powerful tool for achieving syntactic conciseness since it can capture very naturally the regularities of many different permutations of a basic structure. BNF, for example, does not have this capability because it provides no mechanism similar to the transition net's ability to let parsing paths diverge and later reconverge.

To achieve syntactic compactness in a systematic fashion, the transition net is married in LPARS to a system of syntactic features which allow a number of different permutations of a sentence (such as relative clauses, declarative sentences, questions, etc.) to be consolidated into a single transition net structure. Thus, an efficient syntactic skeleton can be created for local parsing.

3. Local Representation of Global Relationships

An interesting problem inherent in the local approach is that of locally representing different possible global relationships between adjacent syntactic elements. This problem arises because the global relationship between adjacent words often cannot be determined from limited context.

LPARS handles this problem in three ways:

(a) One mechanism used is the ancestor link described in Sec. V-B. Ancestor links are used to represent potential ambiguity caused by right recursion in the grammar.

(b) LPARS also takes advantage of the syntactic economy achieved by collapsing a number of different syntactic patterns down to a single syntactic skeleton. Thus, a number of potential global relationships can be combined in a single local structure.

(c) The third mechanism is the fallback of constructing several differen parse trees if different parses of some set of words cannot be conveniently merged by the two techniques just described. This fallback, although less desirable, is not too inefficient if it is only necessary on a limited basis.

4. Formalism vs Procedures in Parsers

A very interesting issue brought to light concerns the relative merits of expressing logic in the form of a nonprocedural formalism (such as a transition net grammar), or in the form of user-written procedures (such as LPARS' LOCALMATCH routine).

A number of speculations made in the present work indicate that the type of probabilistic, semantically oriented logic, which might be used in the LOCALMATCH routine to search for local structures that describe objects and events in a scene, might be written most conveniently in the form of procedures, rather than written to operate from an arbitrary transition net. Much of this logic could be quite divorced from the syntactic representation of the phrase involved. (Remember that the goal of the LOCALMATCH routine is not to perform a systematic search for a globally consistent sentence, but rather the use of heuristics to find plausible structures in different parts of the sentence.)

Once these local structures are found, however, the system needs to have a means of assuring overall consistency and global compatibility of the entire utterance and to direct attention to areas which might be investigated to attain this global consistency. The transition net formalism is used to achieve this. It would be much more difficult to write the logic that directs this processing (the CONNECT algorithm) in purely procedural form, without some structural skeleton such as a transition net to work from. This would be difficult because the logic that pieces the entire sentence together must be able to start processing at any place in the sentence, and must be able to join together any two arbitrary sentence fragments by proposing words between them. A transition net grammar provides a very natural framework to structure this logic around. It is hard to imagine how one would write such logic without some such abstract, manipulable, non-procedural representation of the grammar.

Thus, procedures seem to be the best medium (in the current state of the art) for expressing the logic that uses semantic and pragmatic information to help look for local structures, whereas the transition net formalism is best suited for systematically trying to enforce global structural consistency.

5. Implications for Hardware Design

It is interesting to consider what the design of the LPARS system might imply about the design of efficient hardware for continuous speech processing. One of the most striking aspects of the LPARS system is the large degree of initial lexical processing done in the initial scan and the LOCALMATCH routine.

The lexical matching involves a great deal of computation, even though it is done at a low phonetic distance. On the other hand, virtually all this processing can be done in parallel. Thus, there is great potential for performing this computation very quickly, if one has special-purpose hardware which allows parallel computation on a massive scale. Such a system might contain many special-purpose mini-processors especially designed to perform lexical recognition. Such hardware, although perhaps not entirely feasible at present, is liable to become perfectly viaole in the near future, given the present rate at which costs of computational hardware are dropping.

6. Parallel Process Independence vs Coordination

It is worth discussing exactly to what degree LPARS allows parallel independent processing to take place and to what extent processing in various parts of the utterance is coordinated.

The entire processing of LPARS is globally coordinated in the sense that the phonetic distance thresholds are set for processing in all parts of the utterance, in a globally coordinated fashion.

Within this overall global coordination of phonetic distance thresholds, however, the processing in the initial scan consists of many independent lexical matching processes which are conceptually independent. Similarly, the LOCALMATCH routine consists conceptually of a number of independent processes which use local information to direct their actions.

Once this initial local processing ends, more global processing begins: namely, the construction of PPTs, the inspection of these trees from all parts of the utterance, and the attempt to connect them together by proposing intervening words, using the most promising first. The nature of th. later processing is not independent or potentially parallel since it requires examination of all parse structures together (to see which is most promising and which input segments are worth investigating first).

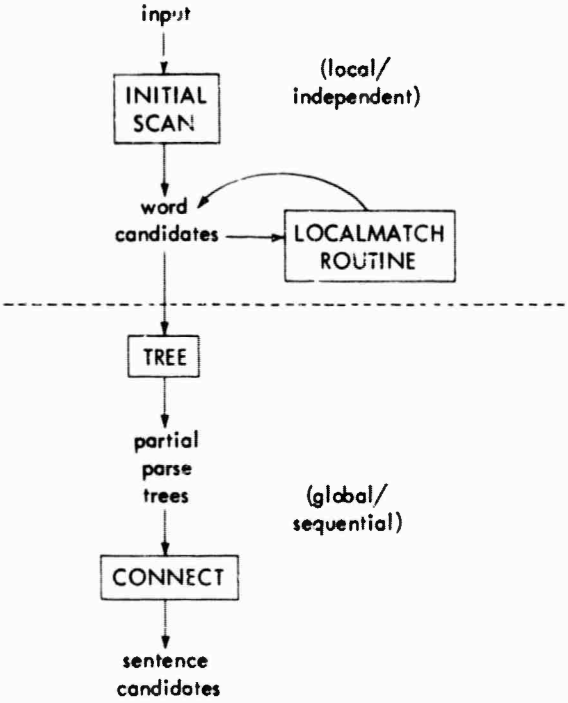


Fig. VIII-1.

Thus, in LPARS, the distinction between local/independent vs global/sequential processing is clearly marked. This distinction separates the initial scan/LOCALMATCH processing from the analysis done by the TREE and CONNECT algorithms (see Fig. VIII-1).

B. AREAS FOR FURTHER WORK

The primary focus of the present work has been on developing a local approach to parsing which allows structures to be built up in all parts of the sentence, so that the system can explore (using the transition network) how these local structures might be pieced together in a globally consistent fashion.

1. Elimination of Current Restrictions

There are, of course, a number of restrictions in the scope of the present work. A number of potential areas for further work are extensions of the system which remove these restrictions:

(a) Only a limited syntax is used. It would be interesting to explore the degree to which the problems of local parsing are compounded when a much larger syntax is used, and to explore techniques useful in dealing with these problems.

(b) Only a relatively small vocabulary is used by LPAPS. It would be interesting to explore the problems that arise as the vocabulary is substantially increased.

(e) The LPARS system uses scrambled input rather than input produced by a real front end. An obvious extension is to join LPARS to a working front end, at which point one could begin to deal with such phenomena as coarticulation, supersegmental effects, etc. into the LPARS lexical matching scheme.

The most interesting extensions of the present work are those to the semantic component of LPARS. Semantic extensions could be incorporated at almost all levels of the LPARS system. In general, these extensions have been mentioned as appropriate in several sections of this report. Such proposed extensions generally involve using the context of a scene being discussed and of an interactive dialog to assist in the processing. Context of this sort might assist in the following ways:

(a) Allow the system to do its processing using only a portion of its entire vocabulary.

(b) Augment the LOCALMATCH routine to look for local structures which described objects or events in the scene.

(c) Augment the action routines so that noun-phrases and clauses are recognized only if they make sense in terms of the scene being described.

2. Tradeoffs in a Higher-Level Speech System

As discussed in Sec. I, high-level systems such as LPARS can make an important contribution by providing a medium in which one can begin to evaluate various tradeoffs involved in building an integrated speech system - tradeoffs involving such things as:

(a) The quality of front-end information,

(b) The size of the vocabulary,

(c) The complexity of the syntax,

(d) The sophistication of the semantic routines available to assist the processing,

(e) The frequency of occurrence of small function words that must be handled, and

(f) The degree to which the vocabulary can be partitioned.

Once a concrete feeling for such tradeoffs is obtained, one can begin to get a feeling for where research effort can be most productively allocated. It would be interesting to systematically explore some of these tradeoffs.

3. Integration of Supra-segmental Information

Another very interesting area for further work is the integration of the use of supra-segmental information into a local system such as LPARS. In a speech waveform, there is a great deal of such information which could help indicate where syllables are, where stressed morphs are, where phrase boundaries are, and possibly even information about the overall structure of the sentence.

Such information could be very helpful in allowing a system to establish "islands of reliability" from which to work in various parts of a sentence. One would suspect, therefore, that such information could be utilized particularly effectively in locally organized systems like LPARS. Without local organization, even if one did use local supra-segmental information one would eventually have to fall back on a left-to-right approach.

Hopefully, the development of locally organized parsing systems will give added impetus to research into promising supra-segmental effects, since such parser organization would provide a natural vehicle for exploiting such information effectively.

APPENDIX A
THE PHONEME SIMILARITY TABLE

Appendix A describes the phonemes used in the LPARS system, and the phoneme similarity table.

I. LPARS' PHONEMES

Table A-1 lists the phoneme set used in our work. With each symbol is the IPA symbol and, if not obvious, a sample pronunciation.

TABLE A-1

LPARS' PHONEMES

P	p		N	n	
B	b		NG	ŋ	sing
T	t		W	w	
D	d		R	r	
CH	č		L	l	
J	j		Y	y	yet
K	k		OO	u	boot
G	g		U	ʊ	put
F	f		O	o	open
V	v		AW		
TH	θ	with	AA	e	table
DH	ð	this	A	a	father
S	s		AR		
Z	z		AE	æ	bad
SH	š		E	ε	bet
ZH	ž		I	I	bit
H	h		EE	i	beet
M	m		SW	ə	about

II. PHONEME SIMILARITY TABLE

Table A-2 is a reproduction of the table in Appendix 10 of the ARPA Speech Report.[19] This phoneme similarity table is used by the lexical scrambler and word-recognition algorithm as described in Sec. III.

TABLE A-2

PHONEME SIMILARITY TABLE

	--	P	B	T	D	CH	J	K	G	F	V	TH	DH	S	Z	SH	ZH	H	M	N	NG	W	R	L	Y	OO	U	O	AW	AA	A	AR	AE	E	I	EF	SW
--	100																																				
P	99	100																																			
B	89	90	100																																		
T	99	96	86	100																																	
D	89	86	96	90	100																																
CH	90	86	76	90	80	100																															
J	82	78	84	82	88	92	100																														
K	99	92	82	96	86	88	80	100																													
G	89	82	92	86	96	78	86	90	100																												
F	95	95	85	91	81	85	77	87	77	100																											
V	85	83	95	81	91	75	83	77	87	90	100																										
TH	95	92	82	92	82	86	78	88	78	99	89	100																									
DH	86	85	91	85	91	77	85	79	87	90	98	91	100																								
S	71	66	56	70	60	78	70	66	56	75	63	74	65	100																							
Z	66	61	61	65	65	75	75	61	61	68	68	69	70	95	100																						
SH	72	65	55	69	59	79	71	69	59	72	62	75	64	97	92	100																					
ZH	67	60	60	64	64	74	76	64	64	67	67	58	69	92	97	95	100																				
H	79	68	58	72	62	76	68	76	66	75	65	76	67	86	81	89	84	100																			
M	66	65	75	61	71	51	59	57	67	66	76	65	74	55	60	54	59	57	100																		
N	66	61	71	65	75	55	65	61	71	64	74	65	74	59	64	58	63	61	96	100																	
NG	66	57	67	61	71	53	61	65	75	60	70	61	70	55	60	58	65	65	92	96	100																
W	59	58	68	54	64	44	52	50	60	59	69	58	67	48	55	47	52	50	87	83	79	100															
R	59	54	54	56	56	46	48	52	52	57	57	58	59	50	55	49	54	52	69	71	67	82	100														
L	58	55	55	57	57	47	49	55	53	56	56	57	58	51	56	50	55	53	68	72	68	81	99	100													
Y	66	59	59	65	65	55	57	63	63	62	62	65	64	57	62	60	65	65	74	78	78	87	89	90	100												
OO	45	35	45	59	49	51	39	43	55	38	48	59	48	49	54	52	57	55	64	68	72	57	61	62	56	100											
U	65	54	64	58	68	50	58	62	72	57	67	58	67	52	57	55	60	64	79	85	87	72	80	81	75	75	100										
O	58	28	28	52	52	24	26	36	56	51	51	32	55	38	43	41	46	46	45	47	51	56	54	55	49	71	62	100									
AW	20	9	9	13	13	5	7	17	17	12	12	15	14	23	28	26	31	29	24	28	52	17	55	56	50	44	51	69	100								
AA	20	9	9	13	15	5	7	17	17	12	12	15	14	23	28	26	51	29	24	28	32	17	55	36	50	42	29	67	98	100							
A	44	58	36	42	42	54	56	40	40	41	41	42	45	36	41	57	42	40	55	57	55	46	64	65	57	55	62	82	51	49	100						
AR	40	35	35	59	59	29	31	35	55	58	58	59	40	41	46	40	45	45	50	54	50	43	61	62	52	58	51	79	48	46	89	100					
AE	40	56	56	38	58	28	30	54	54	59	39	40	41	40	45	39	44	42	51	55	49	44	62	61	51	45	46	74	55	55	84	87	100				
E	44	41	41	41	41	31	55	57	37	44	44	45	46	51	56	50	55	45	56	56	52	49	65	64	54	60	53	69	42	40	75	78	79	100			
I	60	58	58	56	56	46	58	52	52	61	61	60	61	50	55	49	54	52	75	71	67	66	80	79	69	61	62	50	19	17	68	71	64	77	100		
EF	52	51	51	47	47	37	39	45	43	52	52	51	52	57	62	56	61	51	66	62	58	59	71	70	60	62	47	47	20	18	53	64	57	78	85	100	
SW	80	74	74	78	78	70	72	76	76	77	77	78	79	72	77	75	73	76	75	77	75	74	76	77	85	55	72	46	27	27	56	51	50	55	68	59	100

APPENDIX B

LPARS VOCABULARY AND TEST SENTENCES

Appendix B lists the words in LPARS' vocabulary with their parts of speech and phonetic spellings, and the sentences used in LPARS' evaluation.

I. LPARS VOCABULARY

```
(DISLIKE VERB D I S L A I K)
(ENJOY VERB SW N J O I)
(HANDLE VERB H AE N D L)
(DISCARD VERB D I S K AR D)
(CLUTTER VERB K L SW T R)
(COMPLEMENT VERB K A M P L SW M SW N T)
(ILLUMINATE VERB I L OO M I N AA T)
(BRIGHTEN VERB B R A I T N)
(CONCEAL VERB C SW N S EE L)
(CONTAIN VERB K SW N T AA N)
(SUPPORT VERB S SW P O R T)
(OBSCURE VERB A B S K I OO R)
(PLACE VERB P L AA S)
(APPROACH VERB A P R O CH)

(CHRISTOPHER PROPN K R I S T SW F R)
(MARIANNE PROPN M E R I AE N)
(ROBERT PROPN R SW B SW R T)
(PERIWINKLE PROPN P SW R EE W I N K L)
(ANASTASIA PROPN E N E S T AA S EE SW)

(CHANDELIER NOUN SH AE N D SW L EE R)
(FLASHLIGHT NOUN F L AE SH L A I T)
(TABLELAMP NOUN T AA B L AE M P)
(FIREPLACE NOUN F A I R P L AA S)
(SIDETABLE NOUN S A I D T AA B L)
(COFFEETABLE NOUN K AW F EE T AA B L)
(ARMCHAIR NOUN A R M CH AA R)
(BOOKCASE NOUN B U K AA S)
(RADIATOR NOUN R AA D EE AA T O R)
(NEWSPAPER NOUN N OO S P AA P R)
(MAGAZINE NOUN M AE G SW Z EE N)
(ASHTRAY NOUN A SH T R AA)
(DICTIONARY NOUN D I K SH SW N AA R EE)
(REFRIGERATOR NOUN R EE F R I D J SW R AA T O R)
(CABINET NOUN K E B I N SW T)
(PERSON NOUN P SW R S SW N)
(TELEVISION NOUN T SW L SW V I J SW N)
(BREAKFASTTABLE NOUN B R SW K F E S T AA B L)
(WASTEBASKET NOUN W AA S T B E S K SW T)
(STEPLADDER NOUN S T SW P L E DD SW R)
(FOOTSTOOL NOUN F U T S T OO L)
(VACUUMCLEANER NOUN V E K I OO M K L EE N SW R)
(ENTRANCE NOUN SW N T R E N S)

(INFRONTOF PREP I N F R SW N T SW V)
(ONTOPOF PREP AW N T A P SW V)
(BESIDE PREP B EE S A I D)
(INBACKOF PREP I N B E K SW V)
(NEXTTO PREP N SW K S T OO)
(ONTHERIGHTOF PREP AW N TH SW R A I T SW V)
(ONTHELEFTOF PREP AW N TH SW L SW F T SW V)
(BEHIND PREP B EE H A I N D)

(RECTANGULAR ADJ R SW K T E N G I OO L A R)
(CIRCULAR ADJ S I R K I OO L A R)
(BROWN ADJ B R A OO N)
```

```
(ORANGE ADJ O R A N J)
(GREEN ADJ G R EE N)
(FRIENDLY ADJ F R SW N D L EE)

(DOES DO D U Z)
(WHO WH H OO)
(WHICH WH W 1 CH)
(WHOM WH H OO M)
(WHAT WH W A T)
(IS BE 1 Z)
(THE DET TH SW)
(BY BBY B A 1)
(S AFXS Z)
(D AFXS T)
(O AFXS )
(IN PREPA 1 N)
(ON PREPA AW N)
```

II. SENTENCES USED IN LPARS' EVALUATION

```
(the dictionary infrontof the sidetable is place d in the
wastebasket by periwinkle)
(the wastebasket which the green coffeetable support s
contain s the dictionary)
(the coffeetable which support s the green ashtray obscure s
the entrance)
(the green coffeetable support s the dictionary)
(the sidetable which support s the ashtray obscure s the
fireplace)

(the sidetable which support s the ashtray in the bookcase
obscure s the fireplace)
(does the sidetable support the ashtray)
(what does the sidetable support)
(the dictionary on the sidetable is brighten d by the
chandelier)
(does the chandelier infrontof the fireplace brighten the
rectangular coffeetable)

(the chandelier infrontof the fireplace brighten s the
coffeetable)
(what does the chandelier infrontof the fireplace
brighten)
(the coffeetable nextto the bookcase is illuminate d by
the tablelamp)
(is the dictionary support d by the sidetable)
(does periwinkle place the dictionary on the coffeetable)

(what does periwinkle place beside the refrigerator)
(what support s the dictionary ontherightof the fireplace)
(place the circular ashtray in the wastebasket)
(the dictionary which is support d by the coffeetable
support s the tablelamp which illuminate s the fireplace)
(who dislike s periwinkle)

(the friendly person place s the ashtray inbackof the
cabinet)
(periwinkle place s the dictionary on the sidetable beside
the bookcase)
(place the green dictionary on the bookcase)
(the coffeetable beside  the armchair infrontof the
fireplace support s the ashtray)
(the tablelamp beside which robert place s the dictionary
illuminate s the armchair)
```

(the refrigerator in which the ashtray is place d by robert conceal s the entrance)
(whom does marianne dislike)
(on what does christopher place the dictionary)
(beside what does christopher place the ashtray)
(beside what is the green rectangular ashtray place d by robert)

(is robert dislike d by the friendly person beside the coffeetable)
(what is support d by the circular sidetable)
(what is place d beside the newspaper on the coffeetable by christopher)
(on what does periwinkle place the tablelamp)
(beside what does the friendly person place the magazine)

(infrontof what is the magazine place d by marianne)
(in what is the newspaper place d by robert)
(the bookcase contain s the flashlight)
(the flashlight illuminate s the circular wastebasket beside the television)
(the magazine is support d by the dictionary ontopof the coffeetable)

(the newspaper on the sidetable conceal s the ashtray)
(the brown sidetable beside the fireplace conceal s the armchair)
(place the green dictionary beside the refrigerator)
(place the dictionary which support s the ashtray beside the cabinet)
(does the newspaper conceal the circular ashtray)

(does christopher enjoy marianne)
(what does the tablelamp infrontof the fireplace illuminate)
(place the ashtray infrontof the sidetable)
(what brighten s the breakfasttable)
(the bookcase behind the coffeetable support s the magazine)

APPENDIX C
WORD-RECOGNITION ALGORITHM

Appendix C describes the routine used for lexical recognition. This routine receives the following arguments:

WORD — the correct phonetic spelling of the word to be matched (an array of characters).

SENT — the phonetic spelling of the scrambled input utterance (an array of characters).

I — the phoneme position in the sentence at which the match is to be started.

WLNGTH — the length of the word.

SLNGTH — the length of the sentence.

WDIST — the maximum distance permissible for the match.

The algorithm calls the following routines:

SUBDIST(PHON1, PHON2) returns the phonetic distance of substitution between two phonemes.

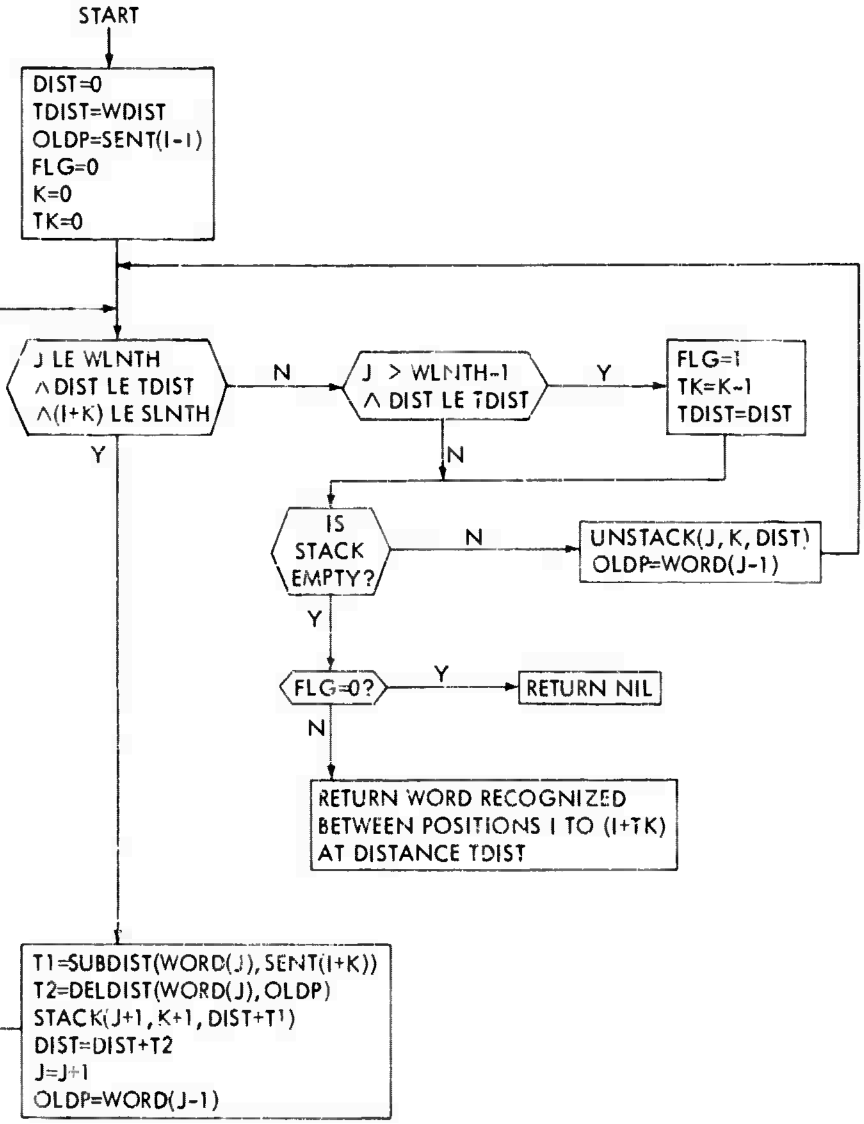
DELDIST(PHON, LASTPHON) returns the phonetic distance of deletion of PHON when preceded by LASTPHON.

STACK(J, K, DIST) puts three values onto a stack.

UNSTACK(J, K, DIST) pops the three values from the stack.

The algorithm explores all possibilities of substitution and deletion in matching the word to the section of input which starts at position I.

LEX(WORD,SENT,I,WLNTH,SLNTH,WDIST)



APPENDIX D
TREE AND CONNECT ALGORITHMS

Appendix D contains flowcharts of the logic of the TREE and CONNECT algorithms. In creating these flowcharts, we have tried to leave in enough detail to convey the basic operations being performed, but not so much as to obscure them. The algorithms described in this Appendix are written to operate on LISP data structures (lists); therefore, they use the following LISP functions:

CAR, which obtains the first member of a list,

CDR, which obtains the remainder of the list,

LIST, which returns a list of the elements passed to it, and

APPEND, which appends two lists.

The algorithms also invoke the following functions (listed alphabetically):

ADDSON(PPT,SON,ARC) returns a new PPT created by adding a rightmost son to the input PPT (providing that the action routine associated with the ARC permits, else NIL).

ANCESTOR(PPT) returns true if the leftmost son of the top node of the PPT is an ancestor link son.

CATPATH(LPT,RPT) returns a new PPT consisting of a concatonation of the two input PPTs (providing that the action routines are compatible, else NIL). If the leftmost son of the RPT is an ancestor link son, it is not included in the resulting structure.

CONTENTW(ARC) returns true if the ARC is labeled by a content word terminal.

CONTEXTLIST(PPT) returns a list of all arcs which invoke the PPT.

ENDSTATE(PPT) returns the end state of the top node of the PPT.

ENDTEST(ENDFLAG, PPT) returns true if ENDFLAG is NIL or if the PPT is a final PPT.

EXITARCS(PPT) returns a list of all arcs emanating from the end state of the top node of the PPT.

FINAL(PPT) returns true if PPT ends on a permitted final state or is a terminal node.

FIRSTARC(PPT) returns the first arc in the top node of the PPT.

INITIAL(PPT) returns true if the PPT begins on a permitted initial state.

INITSTATES(ARC) accepts as input an arc invoking a nonterminal, and returns a list of initial states of that nonterminal's net.

LAMBDA(ARC) returns true if the ARC is a LAMBDA arc.

LASTARC(PPT) returns the rightmost arc in the top node of the PPT.

LSON(PPT) returns the leftmost son of the PPT, or NIL if the PPT is a terminal node.

NT(ARC) returns true if the ARC invokes a nonterminal.

POPPPT(PPT, ARC) returns a new PPT, constructed by popping with the PPT to an ARC invoking the PPT in a higher-level net.

PPT(ARC) accepts a terminal arc and constructs a "hypothetical" terminal PPT of the same part of speech as the arc.

PTSP(N), if N is an arc or a PPT, this function returns the part of speech associated with that arc of PPT.

RIPFIRSTSON(PPT) returns NIL if the top node of PPT has only one son, or returns a new PPT identical to the input PPT but minus the leftmost son of the top node.

RIPLASTSON(PPT) is similar to RIPFIRSTSON but removes the rightmost son.

RSON(PPT) returns the rightmost son of the PPT, or NIL if the PPT is a terminal node.

STARTPATH(P, ARC) returns a new PPT created by starting a path in a net (where ARC emanates from an initial state of that net). P becomes the only son in this new PPT.

STARTSTATE(PPT) returns the starting state of the top node of the PPT.

STORE(PPT) stores a PPT in the phoneme position table.

STORELIST(PPTL) stores a list of PPTs in the phoneme position table.

TERMT(PPT) returns true if the PPT is a terminal.

i. TREE ALGORITHM

The basic top-level routine of the TREE algorithm is the TREE-WORD routine which takes as input a PPT and a word and constructs as many possible resulting PPTs as possible by appending, pushing, and popping.

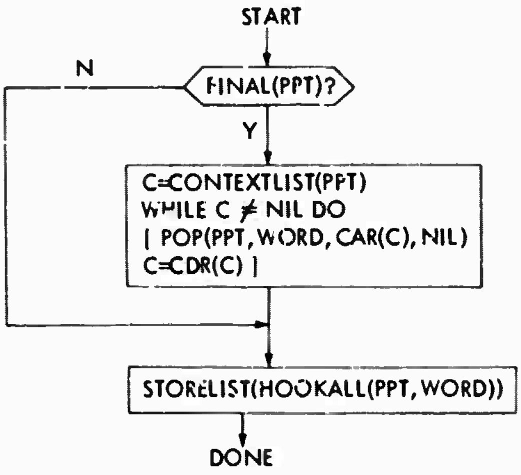
The HOOK routine coordinates all apperding and pushing connections between the top node of a PPT and the word. If a given connection is not possible, the routine must try to effect the connection to a rightmost sub-part of that node.

The PUSH routine handles pushing connections.

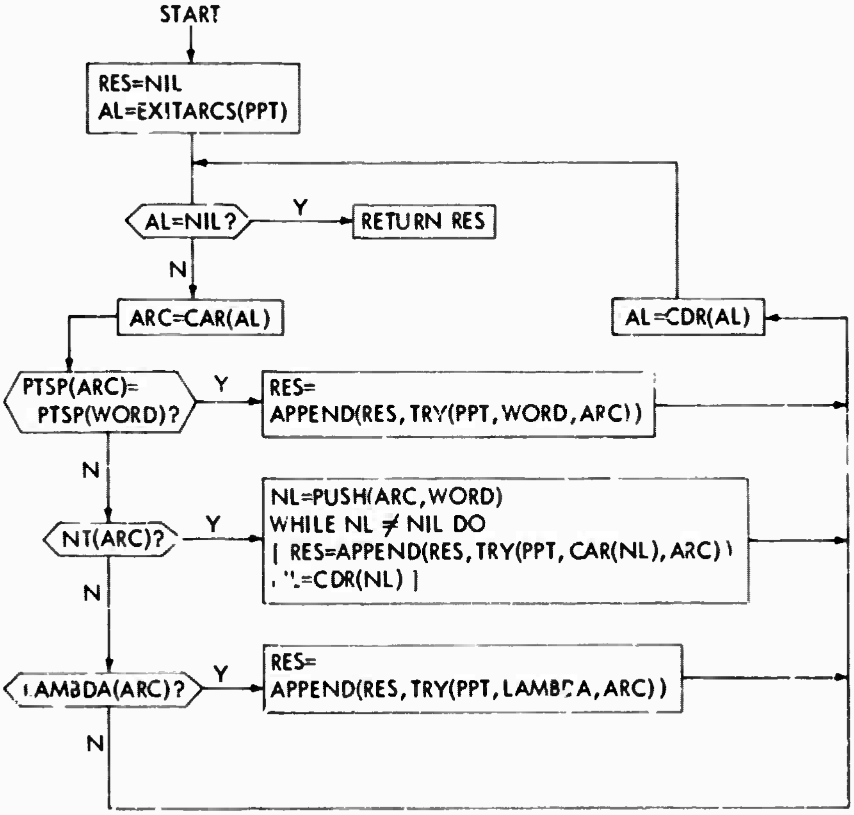
The HOOKALL routine attempts to "hook" the word, not only to the top node of the PPT, but also to lower rightmost nodes. Also, it checks to see if lower rightmost descendents of the PPT can be joined to the word in other syntactic contexts.

The POP routine handles popping connections. In the process, it calls the ABORT routine which prevents infinite looping, and the POPPPT routine which must also handle creating ancestor link structures.

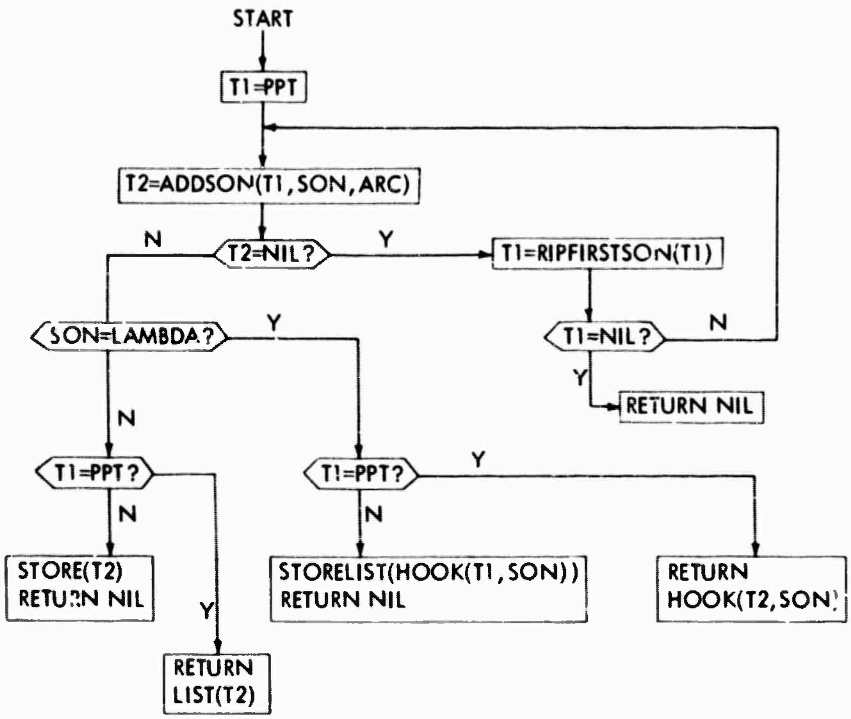
WORD(PPT, WORD)



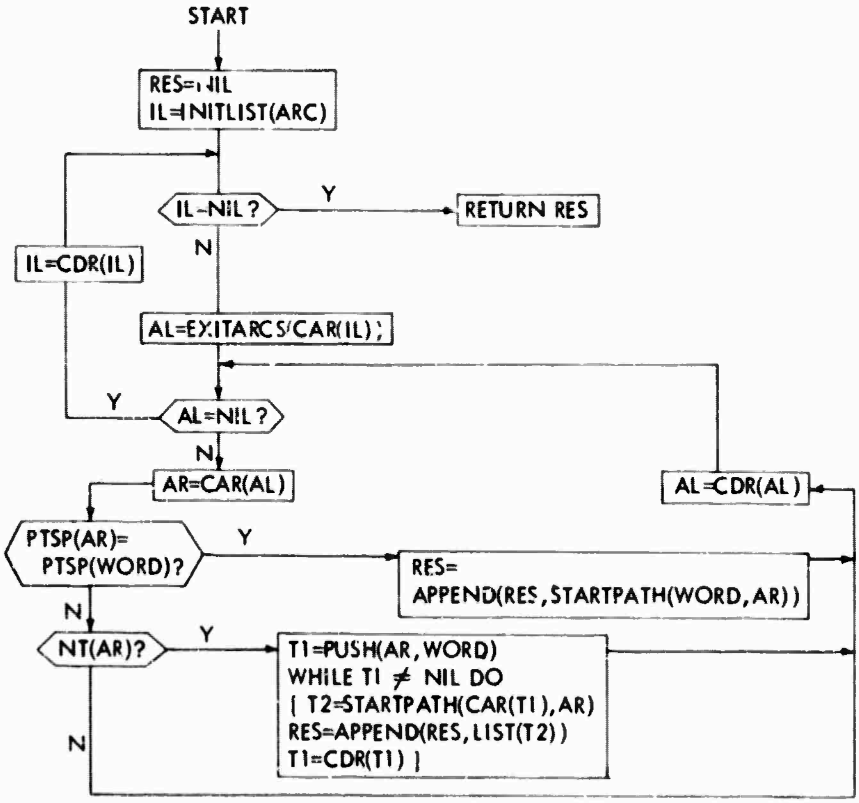
HOOK(PPT, WORD)



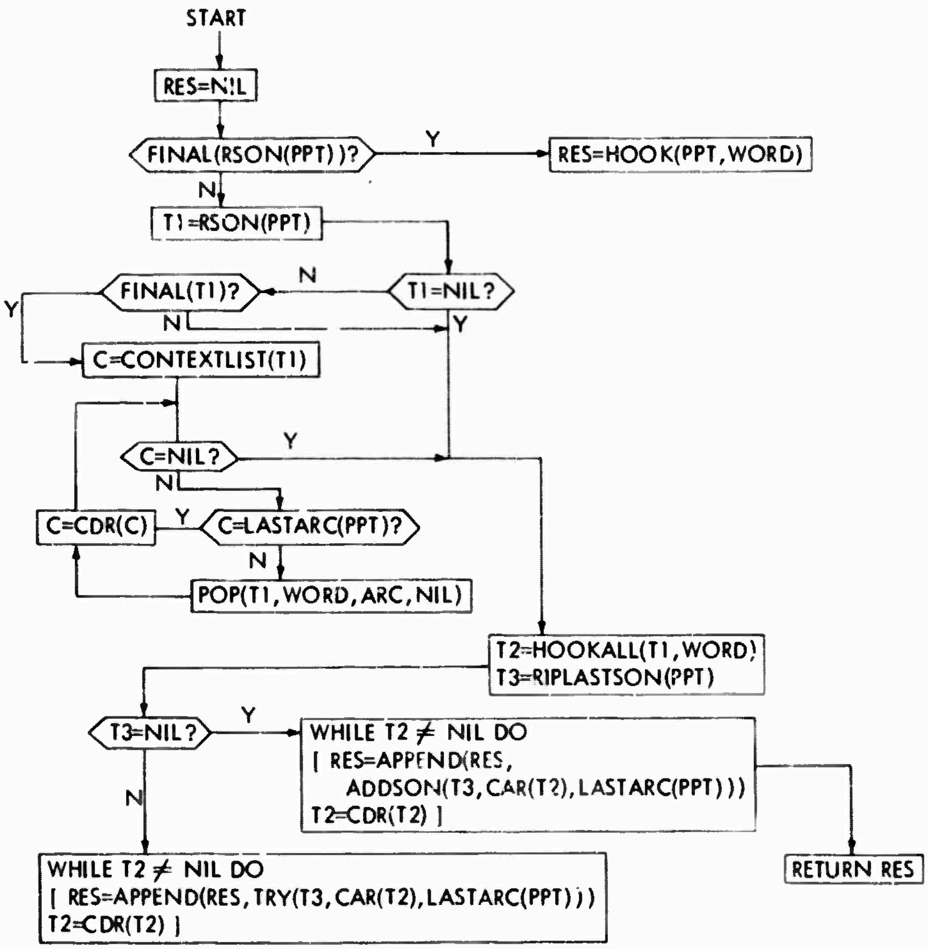
TRY(PPT,SON,ARC)



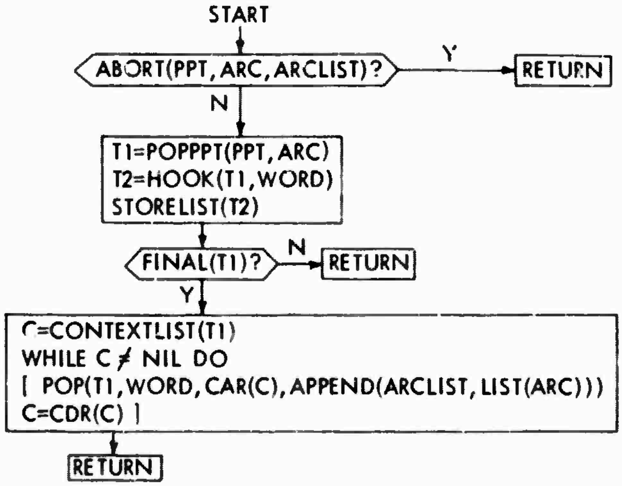
PUSH(ARC,WORD)



HOOKALL(PPT,WORD)



POP(PPT,WORD,ARC,ARCLIST)



II. CONNECT ALGORITHM

The following flowcharts outline the two main routines of the JOIN sub-part of the CONNECT algorithm.

The EXTEND routine coordinates the extending of a path from an LPPT and the joining of that path to an RPPT.

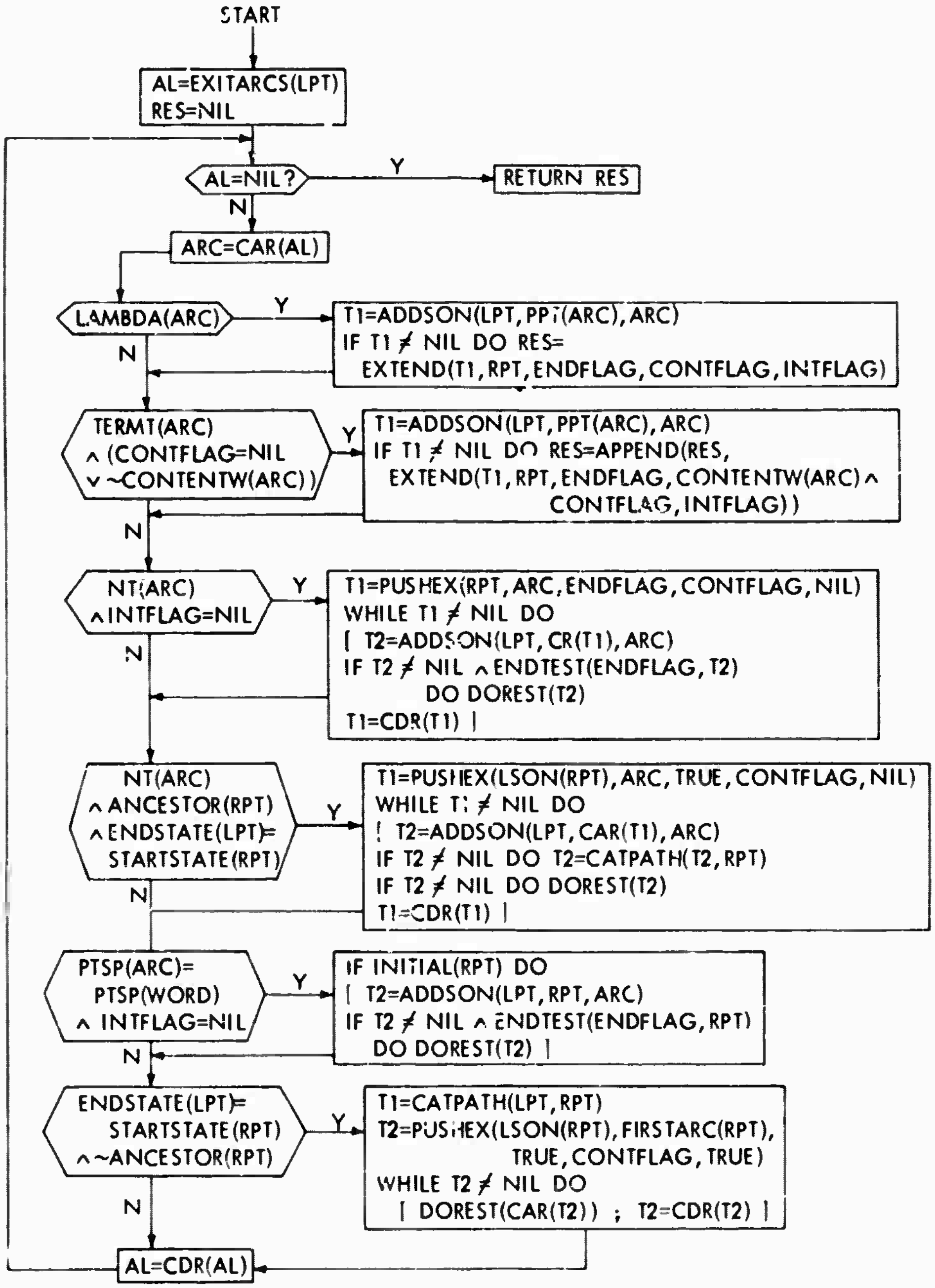
The PUSHEX routine is called with INTFLAG=NIL when the extended path is to continue on a lower level, and is called with INTFLAG=TRUE to make an RPPT start on a permitted initial state.

The DOREST routine is called by the EXTEND routine after the paths have been joined. It assures that the rightmost sons of the LPPT end on permitted final states, and call the CONSTRAIN routine.

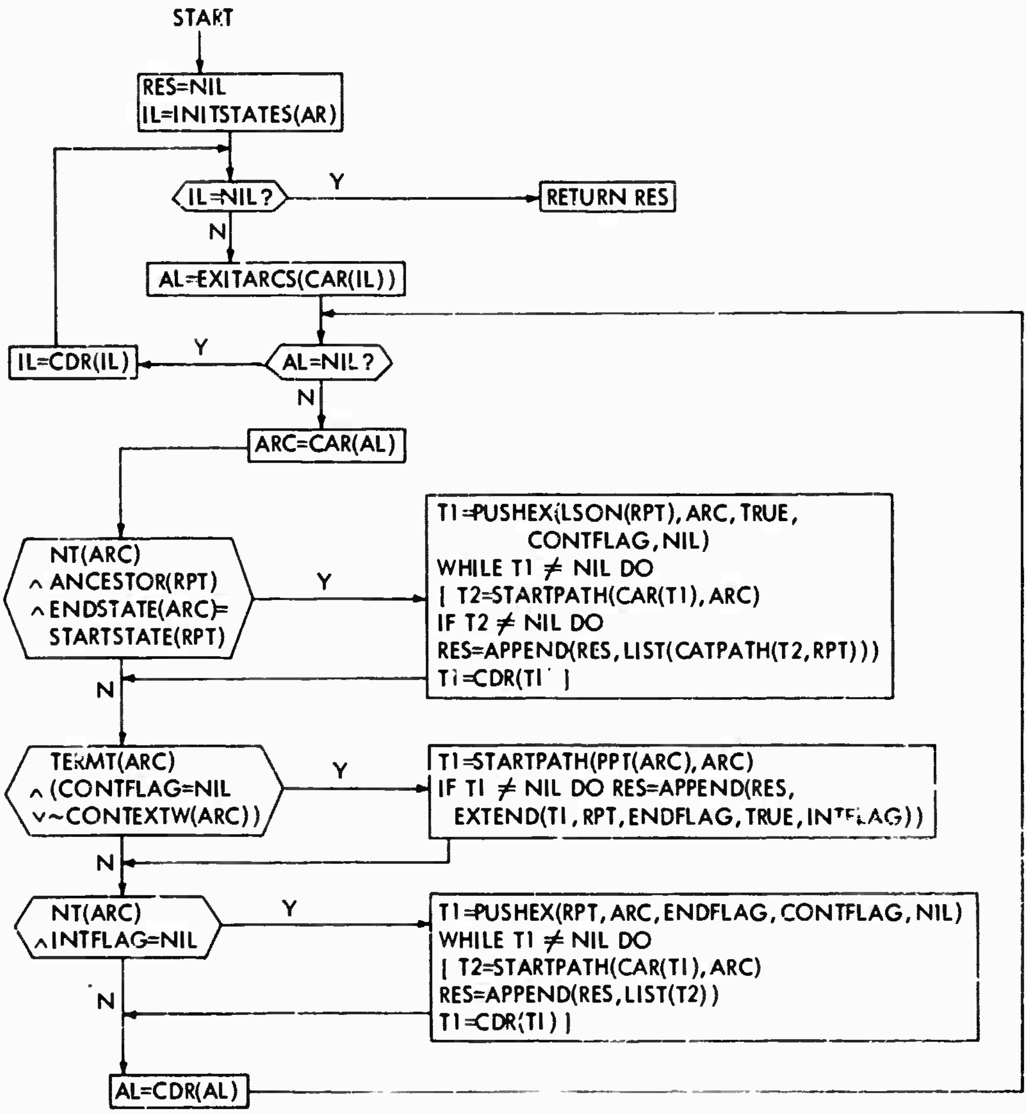
Three arguments are used for communication between these routines:

CONTFLAG is true when an extended path has already traversed a content word,
ENDFLAG is true if the extended path must end on a final state, and
INTFLAG is used in the PUSHEX routine as described above.

EXTEND(LPT,RPT,ENDFLAG,CONTFLAG,INTFLAG)



PUSHEX(RPT, AR, ENDFLAG, CONTFLAG, INTFLAG)



APPENDIX E
LOCALMATCH LOGIC

Appendix E describes the higher phonetic distance lexical matching done by the LOCALMATCH routine. This matching is done in areas adjacent to and between words found either by the initial scan or by previous local matching.

Three types of matching are done:

(1) Matching for words immediately adjacent to a word already found.

(2) Matching for nearby words separated by only a few phonemes from a word already found.

(3) Matching for function word-strings in the area between two words already found.

I. ADJACENT MATCHING

Below, we indicate for each word class which words are searched for immediately adjacent on each side:

NOUN	left	- adj
	right	- verb, prep
VERB	left	- noun, propn
ADJ	left	- adj
	right	- adj, noun
PROPN	right	- verb
PREP	left	- noun
ENDCH	left	- noun, propn, verb

II. NEARBY MATCHING

Below, we indicate for each word class the words which are searched for in areas separated by only a few phonemes on each side:

NOUN	left	- verb, prep
VERB	left	- verb
	right	- noun, adj
ADJ	left	- verb, prep
PREP	right	- noun

III. FUNCTION WORD-STRINGS BETWEEN WORDS

Below, we show the set of function word-strings, and the word-pair contexts between which each is matched for. (These tests are made only if the two words are separated by a roughly appropriate number of phonemes.)

the

- strch adj/noun
  (i.e., between a strch and an adj or a noun)
- prep adj/noun
- verb adj/noun

is

- strch propn

which

- prep propn
- noun propn
- noun verb

by

- noun propn
- verb propn

does

- strch propn
- strch-prep propn
  (i.e., between a prep, at the start of the sentence, and a propn)

which is

- noun verb

what does

- strch propn
- strch-prep propn

whom does

- strch propn

on what does

- strch propn

in what does

- strch propn

by the

- verb adj/noun
- noun adj/noun

on the

- verb adj/noun
- noun adj/noun

in the

- verb adj/noun
- noun adj/noun

which the

- noun adj/noun
- prep adj/noun

whom the

- noun adj/noun

is the

- strch adj/noun

does the

- strch adj/noun

in which the

- noun adj/noun

on which the

- noun adj/noun

what does the

- strch adj/noun
- strch-prep adj/noun

what is the

- strch-prep adj/noun

whom does the

- strch adj/noun

on what does the

- strch adj/noun

in what does the

- strch adj/noun

on what is the

- strch adj/noun

in what is the

- strch adj/noun

in which

- noun propn

on which

- noun propn

who

- strch verb
- noun verb

who is

- strch verb
- noun verb

what

- strch verb

what is

- strch verb

APPENDIX F
BIBLIOGRAPHY

Alter, R., "Utilization of Contextual Constraints in Automatic Speech Recognition," IEEE Trans. Audio Electroacoust. *AU-16*, No. 1, 6-11 (March 1968).

Barnett, J., "A Vocal Data Management System," IEEE Conference on Speech Communication and Processing, Boston, Massachusetts, 1972, pp. 340-343.

Bobrow, D. G., and J. B. Fraser, "An Augmented State Transition Network Analysis Procedure," Proc. International Joint Conference on Artificial Intelligence, Washington, D. C., 1969, pp. 557-567.

Chomsky, N., "Formal Properties of Grammars." In *Handbook of Mathematical Psychology, Vol. 2*, R. D. Luce, R. R. Bush and E. Galanter, Editors (Wiley, New York, 1963).

Feigenbaum, E. A., and J. Feldman, *Computers and Thought* (McGraw-Hill, New York, 1963).

Jacobs, R. A., and D. S. Rosenbaum, *English Transformational Grammar* (Blaisdell Publishing Co., Waltham, Massachusetts, 1968).

Klatt, D. H., and K. N. Stevens, "Strategies for Recognition of Spoken Sentences from Visual Examination of Spectrograms," Massachusetts Institute of Technology (1971), unpublished document.

Kuno, S., "The Predictive Analyzer and a Path Elimination Technique," Commun. ACM *8*, No. 7, 453-462 (July 1965).

McCarthy, J., *et al.*, *LISP 1.5 Programmer's Manual* (M.I.T. Press, Cambridge, Massachusetts, 1965).

Minsky, M., *Semantic Information Processing* (M.I.T. Press, Cambridge, Massachusetts, 1969).

Petrick, S. R., "A Recognition Procedure for Transformational Grammars," Ph.D. Thesis, Massachusetts Instiiute of Technology (1965).

Pierce, J. R., "Whither Speech Recognition," J. Acoust. Soc. Am. *46*, No. 4 (Part 2), 1049-1051 (October 1969).

Quillian, M. R., "Semantic Memory," Ph.D. Thesis, Carnegie Institute of Technology (1966).

Reddy, D. R., "Computer Recognition of Connected Speech," J. Acoust. Soc. Am. *42*, No. 2, 329-347 (August 1967).

__________, "On the Use of Environmental Syntactic, and Probabilistic Constraints in Vision and Speech," AI-78, Stanford University (1969).

Sussman, G., T. A. Winograd and E. Charniak, "Micro-Planner Reference Manual," AI Memo 203, Project MAC, M.I.T. (July 1970).

ACKNOWLEDGMENTS

I wish to express my great appreciation to Professor Jonathan Allen for many hours of discussion, advice, and encouragement during the course of this work. I am also indebted to Professors Terry A. Winograd and J. C. R. Licklider for having initially suggested this area of research and for their continuing advice and support, and to Mr. James Forgie for numerous patient hours spent listening while I tried to formulate various ideas intelligibly.

I also wish to thank various people who helped me during the course of this work, including Professor Dennis H. Klatt, Mr. Louis N. Gross, Dr. Amedio Armenti, Mr. Donald E. Hall, Mr. James Sloane, Mr. Jack Klovstad, and Mr. Robert Goldberg.

REFERENCES

1. N. Lindgren, "Machine Recognition of Human Language – Parts 1 and 2," IEEE Spectrum 2, No. 3, 114, and No. 4, 44 (1965).

2. S. R. Hyde, "Automatic Speech Recognition: Literature, Survey and Discussion," Research Department Report No. 35, P. O. Research Department, Dollis Hill, London NW2, 1968.

3. A. Guzman-Arenas, "Computer Recognition of Three-Dimensional Objects in a Visual Scene," Report MAC-TR-59, Project MAC, M.I.T. (December 1968).

4. T. A. Winograd, "Procedures as a Representation of Knowledge in a Computer Program for Understanding Natural Language," Report MAC-TR-84, Project MAC, M.I.T. (February 1971).

5. B. Gold, "Word Recognition Computer Program," Technical Report 452, Research Laboratory of Electronics, M.I.T. (June 1966).

6. D. G. Bobrow and D. H. Klatt, "A Limited Speech Recognition System," Proc. AFIPS Fall Joint Computer Conference, Washington, D.C., 1968, p. 33.

7. M. Medress, "Computer Recognition of Single-Syllable English Words," Ph. D. Thesis, M.I.T. (1969).

8. P. Vicens, "Aspects of Speech Recognition by Computer," Report CS-127, Computer Science Department, Stanford University (1969).

9. D. R. Reddy, et al., "Working Papers in Speech Recognition I," Carnegie-Mellon University (April 1971).

10. C. C. Tappert, et al., "The Use of Dynamic Segments in the Automatic Recognition of Continuous Speech," IBM Corporation, RADC-TR-70-72 (1970).

11. ____________________, "Automatic Recognition of Continuous Speech Utilizing Dynamic Segmentation, Dual Classification, Sequential Decoding, and Error Recovery," IBM Corporation, RADC-TR-71-146 (1971).

12. A. S. Rabinowitz, "On the Value of the Fano Algorithm in Establishing the Graphemic Form of Machine Derived Phonetic Strings," Ph. D. Thesis, North Carolina State University (1972).

13. W. A. Woods, "Transition Network Grammars for Natural Language Analysis," Commun. ACM 13, 591-606 (1970).

14. W. A. Woods and R. M. Kaplan, "The Lunar Sciences Natural Language Information System," Report No. 2265, Bolt Beranek, and Newman Inc. (September 1971).

15. N. Chomsky, "A Transformational Approach to Syntax," in The Structure of Language, J. A. Fodor and J. J. Katz, Editors (Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1964).

16. ___________, Aspects of the Theory of Syntax (M.I.T. Press, Cambridge, Massachusetts, 1965).

17. J. J. Katz and J. A. Fodor, "The Structure of a Semantic Theory," in The Structure of Language, J. A. Fodor and J. J. Katz, Editors (Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1964), pp. 479-518.

18. J. C. Nyiri, "No Place for Semantics," Foundations of Language 7 (1971).

19. A. Newell (Chairman), et al., "Speech-Understanding Systems, Final Report of a Study Group," Computer Science Department, Carnegie-Mellon University (May 1971).

20. W. D. Lea, "Intonational Cues to the Constituent Structure and Phonemics of Spoken English," Ph. D. Thesis, Purdue University (June 1972).

21. Y. D. Willems, "The Use of Prosodics in the Automatic Recognition of Spoken English Numbers," Ph. D. Thesis, M.I.T. (June 1972).

22. J. Earley, "An Efficient Context-Free Parsing Algorithm," Commun. ACM 13, 94-102 (1970).