June 1977

A FRAMEWORK FOR SPEECH UNDERSTANDING

by William H. Paxton

Artificial Intelligence Center
Technical Note 142

This research has been funded under the following ARPA contracts, all administered through the Army Research Office: DAHC04-72-C-0009, DAHC04-75-C-0006, DAAG29-76-C-001, and DAAG29-76-C-0012.

ABSTRACT

This paper reports the author's results in designing, implementing, and testing a framework for a speech-understanding system. The work was done as part of a multi-disciplinary effort based on state-of-the-art advances in computational linguistics, artificial intelligence, systems programming, and speech science. The overall project goal was to develop one or more computer systems that would recognize continuous speech uttered in the context of some well-specified task by making extensive use of grammatical, semantic, and contextual constraints. We call a system emphasizing such linguistic constraints a 'speech-understanding system' to distinguish it from speech-recognition systems, which rely on acoustic information alone.

Two major aspects of a framework for speech understanding are integration -- the process of forming a unified system out of the collection of components -- and control -- the dynamic direction of the overall activity of the system during the processing of an input utterance. Our method of system integration gives a central role to the input-language definition, which is based on augmented phrase-structure rules. A rule consists of a phrase-structure declaration, which specifies the possible constituents of a phrase, and an augmentation, which is a procedure for computing 'attributes' and 'factors'. Attribute statements determine the properties of particular phrases constructed by the rule; factor statements make acceptability judgments on phrases. Together, these statements contain specifications for most of the potential interactions among system components.

Our approach to system control centers on a system 'Executive' applying the rules of the language definition, organizing hypotheses and results, and assigning priorities. Phrases with their attributes and factors are the basic entities manipulated by the Executive, which takes on the role of a parser in carrying out its integration and control functions. The Executive controls the overall activity of the system by setting priorities on the basis of acoustic and linguistic acceptability judgments. These data are combined to form scores and ratings. A phrase score reflects a quality judgment independent of the phrase's context and gives useful local information, but in setting priorities we want to use global information concerning the sentential context. To get early and efficient access to

the contextual information, we have developed a technique for calculating phrase ratings by a heuristic search of possible interpretations that would use the phrase. One of our experiments shows that this context-checking method results in significant improvements in system performance.

These experiments are important for evaluating a complex system framework such as ours. It is not enough simply to demonstrate that a system with certain features can be implemented; a working system shows that the features are not disastrous, but it does not show the good effects, if any, the features have on performance. Experimentation is a valuable technique for use in discovering and explaining the actual effects and interactions of the design features. In a series of experiments, we have studied system features by comparing the performance with a particular feature to the performance with a simpler alternative in place of that feature. The observed difference indicates the importance of that feature, and interactions are revealed by comparing different combinations of features. The results of the experiments give a better understanding of system performance and suggest new lines of development.

PREFACE

This paper is my Ph.D. dissertation for the Computer Science Department of Stanford University. I want to thank my reading committee -- Terry Winograd, Daniel Bobrow, Cordell Green, and Edward Smith -- and my speech-understanding colleagues at SRI -- Ann Robinson, Barbara Grosz, Gary Hendrix, Jane Robinson, and Don Walker. I also want to thank Joyce Friedman for reading an early draft and Jerry Bolzano for helping me design the experiments reported in Chapter IV. Finally, I want to thank my wife Deb for her support during the stressful years of research and writing that went into this dissertation.

CONTENTS

I INTRODUCTION

CONTENTS:

A. ORIENTATION

This paper reports the author's results in designing, implementing, and testing a framework for a speech-understanding system. The work was done as part of a multi-disciplinary effort based on state-of-the-art advances in computational linguistics, artificial intelligence, systems programming, and speech science (see Newell et al., 1973).* The overall project goal was to develop one or more systems that would recognize continuous speech uttered in the context of some well-specified task by making extensive use of grammatical, semantic, and contextual contraints. We

---

* This research has been funded under the following ARPA contracts, all administered through the Army Research Office: DAHC04-72-C-0009, DAHC04-75-C-0006, DAAG29-76-C-0011, and DAAG29-76-C-0012.

call a system emphasizing such linguistic constraints a 'speech-understanding system' to distinguish it from speech-recognition systems, which rely on acoustic information alone.*

Two major aspects of a framework for speech understanding are integration -- the process of forming a unified system out of the collection of components -- and control -- the dynamic direction of the overall activity of the system during the processing of an input utterance. Figure I-1 lists some distinguishing characteristics of our framework in the areas of integration and control.

Our method of system integration gives a central role to the input-language definition, which is based on augmented phrase-structure rules. A rule consists of a phrase-structure declaration, which specifies the possible constituents of a phrase, and an augmentation, which is a procedure for computing 'attributes' and 'factors'. Attribute statements determine the properties of particular phrases constructed by the rule; they may compute values for

---

* The framework discussed in this paper evolved as part of a joint effort by Stanford Research Institute (SRI) and System Development Corporation (SDC). SRI was responsible for overall system control and for developing components to handle syntax, semantics, and discourse. SDC was responsible for the acoustic components -- signal processing, acoustic-phonetics, and phonology (see Bernstein, 1975; Ritea, 1975; and Barnett, 1976).

SYSTEM INTEGRATION

Provides for interaction of information from various sources of knowledge -- syntax, semantics, discourse -- in a procedural representation.

Provides a means for adjusting ('tuning') the language definition to particular domains without loss of generality.

Avoids commitment to particular system control strategy, and allows flexible use of various strategies for combining words and phrases.

SYSTEM CONTROL

Provides special techniques to assign priorities by using contextual constraints.

Allows combinations of top-down, bottom-up, and bidirecrectional strategies.

Organizes and constructs data structures for hypotheses in a manner that greatly reduces duplication of effort.

Is based on extensive experimental studies to evaluate design alternatives.

Figure I-1. DISTINGUISHING CHARACTERISTICS.

attributes that relate to syntax, semantics, or discourse. Factor statements make acceptability judgments on phrases. Scores for factors are non-Boolean; they are represented internally by integers and are referenced symbolically in the language definition by a set of variables such as 'good,' 'ok,' 'bad,' and 'out.' A proposed phrase is not

simply accepted or rejected; it is rated as more or less acceptable, depending on factor values. The language definition can be 'tuned' to particular domains by modifying the judgments made in factor statements. Like attributes, factors may be syntactic-, semantic-, or discourse-related.

Together, the attribute and factor statements in the procedural parts of the rules contain specifications for most of the potential interactions among system components. The form of the specifications avoids commitments to particular system control strategies. For example, the rule procedures can be executed when any subset of constituents are present, so incomplete phrases can be constructed to provide intermediate results. It is not necessary to acquire constituents in a strictly left-to-right order.

Our approach to system control centers on a system 'Executive' applying the rules of the language definition, organizing hypotheses and results, and assigning priorities. Phrases with their attributes and factors are the basic entities manipulated by the Executive, which takes on the role of a parser in carrying out its integration and control functions. It builds a 'parse net' to hold intermediate data, and it uses the net to eliminate wasteful duplication of effort. Two types of tasks interact to build the net: the predict task, which leads to predictions for words in

the input, and the word task, which gets words from the acoustics and uses them to construct new phrases. The predict task operates in a top-down manner and ends by scheduling the word task; the word task operates in a bottom-up manner and ends by scheduling the predict task.* Both tasks can operate bidirectionally through the input and are guided by a lookahead mechanism to avoid unnecessary operations.

The Executive controls the overall activity of the system by setting priorities, which it does on the basis of acoustic and linguistic acceptability judgments. These data are combined to form scores and ratings. A phrase score reflects a quality judgment independent of the phrase's context. For example, the score of a nonterminal phrase depends on the scores of its constituents and the factors that indicate how well the constituents go together; it does not depend on higher level phrases that might include it as a constituent. Because phrase scores are independent of context, they do not have to be recalculated for each possible use of the phrase. If a phrase gets a subthreshold score, it can be discarded without concern that in a

---

* Readers unfamiliar with the terms 'top-down' and 'bottom-up' may wish to consult a text on parsing such as Aho and Ullman (1972). Basically, a top-down method uses incomplete structures to make predictions for missing substructures, while a bottom-up method takes complete substructures and looks for ways to organize them into larger units.

different context it might be acceptable. Moreover, the language definition allows scores to be calculated for incomplete phrases, so the Executive has access to quality judgments from scores at each step as it adds constituents.

The score gives useful local information about a phrase, but in setting priorities we want to make use of global information concerning the sentential context; ideally, we do not want to waste time working on a phrase that is not part of the best current hypothesis on the entire utterance. To get early and efficient access to the contextual information, we have developed a technique for calculating phrase ratings, which are intended to provide an estimate of the best score for an interpretation that can be constructed with the particular phrase. Phrase ratings are calculated by a heuristic search of the tree of possible interpretations that would use the phrase. This technique provides the Executive with an effective way of estimating how well the phrase fits its possible sentential contexts while avoiding an exhaustive search of the possibilities. One of our experiments shows that this context-checking method results in significant improvements in system performance.

These experiments are important for evaluating a complex system framework such as ours. It is not enough

simply to demonstrate that a system with certain features can be implemented; a working system shows that the features are not disastrous, but it does not show the good effects, if any, the features have on performance. Experimentation is a valuable technique for use in discovering and explaining the actual effects and interactions of the design features. In a series of experiments, we have studied system features by comparing the performance with a particular feature to the performance with a simpler alternative in place of that feature. The observed difference indicates the importance of that feature, and interactions are revealed by comparing different combinations of features. The results of the experiments give a better understanding of system performance and suggest new lines of development.

The following section contains brief descriptions of the various components of the speech-understanding system, including those developed by SDC. These sketches allow the reader to understand what the framework integrates and controls; more details are available on the various system components in Walker et al. (1976) and in the SDC publications referenced above.

B. AN OVERVIEW OF THE SPEECH-UNDERSTANDING SYSTEM

The domain for the speech-understanding system is information about the ships of the U.S., Soviet, and British fleets. The data base of the system contains information such as owner, builder, size, and speed for several hundred ships. The user can get this data from the system by employing a subset of English that includes many common forms of questions and commands. The system's internal organization is shown in Figure I-2. The directions of the arrows in the figure indicate the general flow of information when an utterance is interpreted by the system and a response returned to the speaker.

1. ACOUSTIC COMPONENTS

The acoustic-phonetic processor, the mapper, the phonological lexicon, and the lexical subsetter were developed by SDC. The acoustic processor digitizes and records the input from the human speaker at a rate of 20,000 samples per second. RMS-energy values are calculated for each 10-millisecond frame of speech; this is followed by fundamental frequency extraction, formant frequency analysis, syllable segmentation, phrase segmentation, and other analyses. From these parameters, rough segment labels are derived; subsequent processes use the information

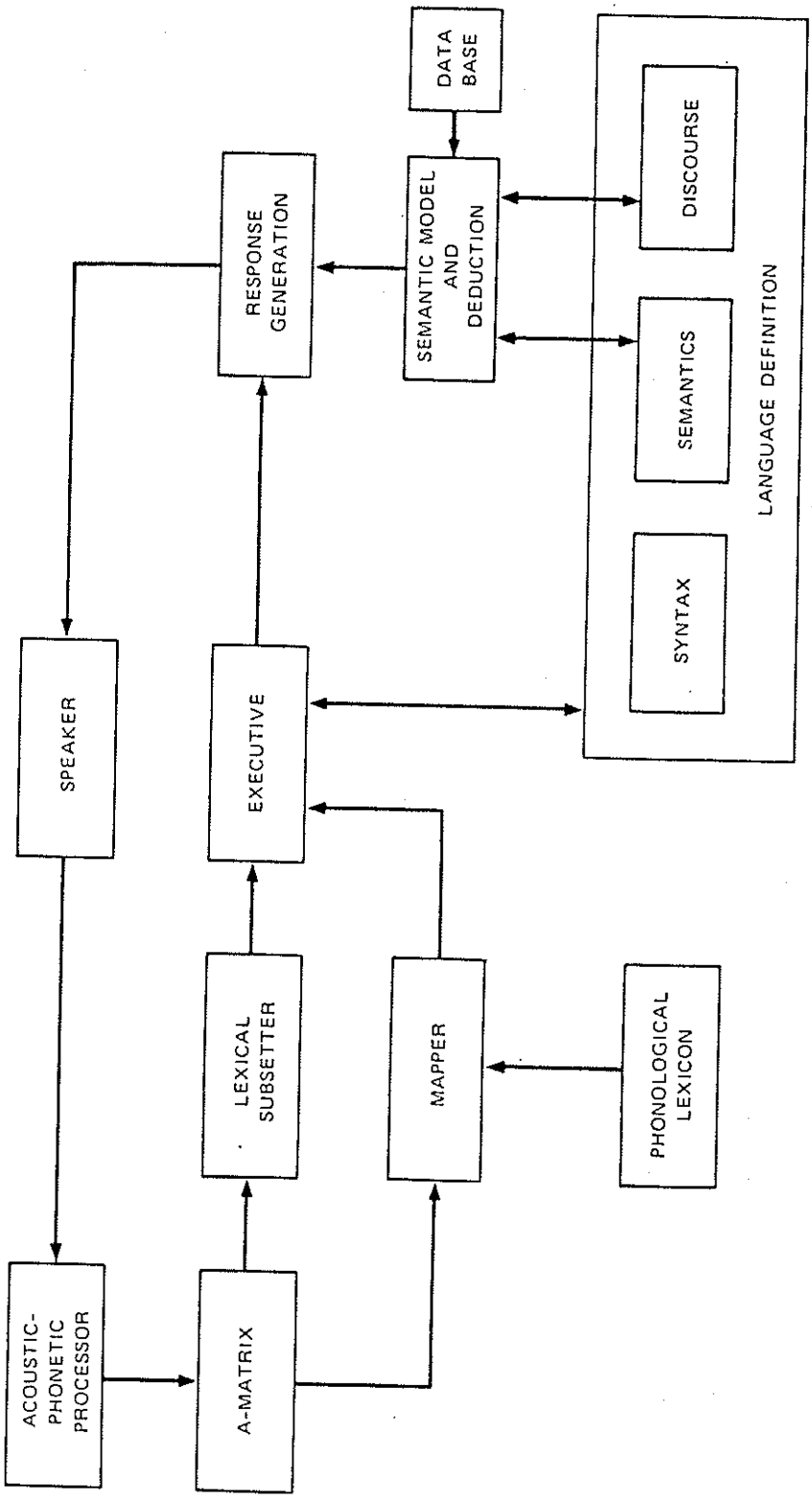

Figure I-2. SYSTEM ORGANIZATION

available to segment the speech into phoneme-like units, assign feature bits such as nasal or retroflexed, and generate phonemic labels with associated merit scores for each segment. All of the acoustic-phonetic information for the utterance is stored as an array called the 'A-matrix.'

Using the A-matrix data, the mapper carries out acoustic tests. Given a word predicted by the Executive and a location in the speech input, the mapper compares possible pronunciations of the word with the acoustic data at that point. The location can be specified with a left time, a right time, or both. The mapper assigns a score between 0 and 100 indicating how well the word matches the input. If the value exceeds a given threshold, the mapper reports the beginning and ending times of the word with the score.

The lexical subsetter performs an analysis of the A-matrix at a specified location in the utterance and returns a list of words that could begin (or end) at that time. This capability reduces the number of words that would otherwise have to be checked by the mapper.

2. SYNTAX

The syntactic knowledge in the system is represented both in the phrase-structure part of the

language-definition rules and in the attribute and factor statements in the procedure part of the rules. Syntax provides computationally inexpensive information on the words or phrases that may combine and on how well they go together. Syntactic information alone can often reject an incorrect word or phrase without requiring costly semantic and discourse tests.

Factors are used for traditional syntactic tests such as agreement for person or number; they are also used to reduce the scores of unexpected phrases. For example, certain negative questions (e.g., "What submarine doesn't the U.S. own?") are not expected to occur. A factor statement lowers the rating for this interpretation but does not eliminate it completely. If no better hypothesis can be formed to account for the input utterance, this interpretation will be accepted.

Since the language-definition system provides the capability for rating phrases by means of non-Boolean factors, the grammar can be tuned to particular discourse situations and language users simply by adjusting factors that enhance or diminish the acceptability of particular interpretations. It is not necessary to rewrite the language definition for each new domain (see Robinson 1975a,1975b).

3. SEMANTICS

The system's knowledge of the domain is embodied in a partitioned semantic network (see Hendrix 1975). A semantic network consists of a collection of nodes and arcs, each node representing an object (a physical object, situation, event, set, or the like) and each arc a binary relation. The structure of the network differs from that of conventional nets in that nodes and arcs are partitioned into spaces. These spaces organize information into bundles and have a role in networks analogous to that of parentheses in logical notation.

The network also serves as the medium for recording and communicating semantic information among the relevant system components. During the interpretation of an utterance, semantic composition routines, which are called directly from the language-definition rules, relate the constituents of a phrase to the network model. These routines build new network structures to reflect the underlying meanings of acceptable phrases and eliminate phrases not satisfying semantic criteria.

To supplement the knowledge encoded in the network, a relational data base is maintained. It can be accessed directly from the network, which contains a representation of the contents of the data base.

4. DISCOURSE

The discourse component of the speech-understanding system relates a given utterance (or a portion of it) to the overall dialog context and to entities and structures in the domain (see Deutsch 1974, 1975; Grosz 1977). The current domain of the speech-understanding system provides for interaction with information in a data base. In this domain, the discourse context is limited to a linear history of the preceding interactions. For complex task-oriented dialogs, the linear discourse history can be replaced by a structured history mirroring the organization of the task execution.

An important function of the discourse component is to expand elliptical expressions to their full meaning. In the SRI/SDC system, a single noun phrase can be accepted as a complete utterance if it can be expanded into a meaningful sentence using information from the previous dialog context. For example, the phrase "length" is unacceptable in isolation, but, following "What is the speed of the ship?" it can be expanded to mean "What is the length of the ship?"

Another capability is the identification of the referents of definite noun phrases. Partitions in the

semantic network focus the attention of deductive procedures on those items that have been previously mentioned in the dialog. A representation of the referent of a definite noun phrase is kept as the discourse attribute of the phrase. If no referent is found, the phrase is given a low score.

5. DEDUCTION

The deduction component of the system provides an inference mechanism for retrieving information from the semantic network. During the interpretation of an utterance, it supplies information needed by both the semantic composition routines and the discourse procedures. When an interpretation has been found for a question, the deduction component is used to find an answer.

6. GENERATION

The generation component of the speech-understanding system contains procedures to produce an English phrase or sentence corresponding to a semantic network substructure (see Slocum 1975). Usually, this substructure is the answer to a question asked by the user. A distributed generation grammar enables the generator to express the content of the input nodes and arcs by employing the closest applicable templates (rules) in the superset

hierarchy of those nodes. The answer to a question, for example, can be either a noun phrase or a complete sentence, depending on the exact content of the input.

C. PREVIEW OF THE FOLLOWING CHAPTERS

The following chapters discuss in detail the framework of the speech-understanding system. The Definition System, consisting of a metalanguage and a compiler, is covered in Chapter II, which also contains both a description of a version of the SRI language definition and a discussion of some alternative definitional approaches used in previous natural language systems.

Chapters III and IV cover the design and testing of the Executive, which is the major contribution of this work. Chapter III gives both an overview and a complete explanation of the Executive's operation. The chapter ends with a discussion of system integration and control in some other speech-understanding systems. Chapter IV describes a series of experiments testing the system framework and, in particular, analyzing the effects and interactions of four major choices of control strategy. Chapter V outlines some possible extensions and revisions of the framework.

II THE DEFINITION SYSTEM

CONTENTS:

A. INTRODUCTION

The Definition System consists of a metalanguage for writing definitions of the input language for the speech understanding system and a compiler to convert such definitions into a form for use by the Executive System.* In Sections B and C, the metalanguage is described, and its

---

* We make the usual distinction between the metalanguage and the object language: the object language is the language being defined (in our case, it is the system's input language); the metalanguage is the language used to state the definition. The 'language definition' is written in the metalanguage and specifies the object language.

use is illustrated by a sketch of the SRI language definition. Section D contains a discussion of the Definition Compiler, focusing on the process of rule translation and describing the internal representation of the structural and procedural information. The final section, Section E, compares our approach to that of some earlier efforts. The use of the translated language definition in understanding utterances is described in Chapter III, The Executive System.

B. THE METALANGUAGE

The metalanguage is designed for specifying the definition of the input language for the speech-understanding system. A definition written in the metalanguage consists of a lexicon containing the input language vocabulary, a set of composition rules for combining words into phrases and smaller phrases into larger ones, and some global declarations giving information needed by the Definition Compiler and the Executive. The lexicon is separated into categories, such as noun and verb, and the words in each category are assigned values for various attributes such as grammatical features and semantic representation. The composition rules are phrase structure rules augmented by a procedure which is executed whenever

the rule constructs a phrase. Information provided by the procedure includes both attributes of the phrase based on the attributes of its constituents, and factors for use in judging the acceptability of the phrase. The global declarations in a language definition give information such as lists of attributes for the different categories.

1. COMPOSITION RULES

The structure declaration in each rule gives the alternatives and options for the immediate constituents, and the rule procedure specifies attributes and factors for phrases constructed by the rule. The rule procedure is organized as nested conditional statements depending on the particular phrase structure. Unwanted phrases are blocked by tests referring to constituent attributes. These tests are embedded in conditionals in a manner ensuring that as soon as the relevant constituents are available, the tests are made. The tests are independent of the presence or absence or other constituents and are also insensitive to the order in which the constituents are acquired.

In the rules we have developed at SRI, the procedures typically contain much more information than the structure declarations. For example, in the language definition discussed below, there are only ten

rules, but there are about ten pages of rule procedures. In this definition the choice was made to use standard syntactic categories [such as sentence (S), noun phrase (NP), and verb phrase (VP)] and general structure declarations [such as S = NP VP]. The size of the rule procedures reflects the large number of constraints that are not captured by these structure declarations alone. A language definition for text input might omit such restrictions of the grounds that no one would ever violate them in actual conversation. However, in speech understanding, difficulties in acoustic recognition can cause the system to 'hear' almost anything, so the language definition must take advantage of every opportunity to block unacceptable phrases and downgrade unlikely ones. This point will be illustrated in the discussion of the language definition in Section C.3 below.

Part of a composition rule is shown in Figure II-1. The rule starts with the keyword RULE.DEF followed by the rule name (S1), the structure declaration, and the procedure. In the structure declaration, vertical bars separate alternatives, braces are used to delimit a set of alternatives, parentheses delimit optional items, and angle brackets mark an optional set of alternatives. Category names can be terminated with a number to provide

PART OF A COMPOSITION RULE

```
RULE.DEF S1 S=NP1 <(DO NP2) VP1 | BE {VP2 | NP3 | "THERE}>;

BEGIN
  .

MOOD = IF DEIX(NP1) EQ "WH THEN "WH
       ELSE IF DEIX(NP1) NQ "UNDEFINED THEN "DEC
       ELSE "UNDEFINED;
IF MOOD EQ "WH THEN F.MOOD = GOOD;
IF OMITALL(VP1,BE) AND SUBCAT(NP1) EQ "PRO AND MOOD EQ "DEC
       THEN F.REJECT(F.PROSENT);
  .

END;
```

Figure II-1.

unique names for different occurrences in the rule of the same category (e.g., NP1, NP2, and NP3, are all noun phrases or NPs). A quote mark indicates that the next word is to be taken literally rather than being interpreted as a category name. Thus, the phrase structure declaration in Figure II-1 states that a phrase of category S can be composed of a noun phrase, NP1, optionally followed by either a predicate with a verb phrase, VP1, or a predicate with a BE verb (such as "is" or "are"). The constituent VP1 can optionally be preceded by a DO verb (such as "did" or "does") and a noun phrase, NP2. The BE verb must be followed by either a verb phrase, VP2, a noun phrase, NP3, or the word "there". Examples of sentences using this rule are given in Figure II-2.

EXAMPLE SENTENCES

S = NP1 VP1	Who built it?
S = NP1 DO NP2 VP1	What does it do?
S = NP1 BE VP2	How many were built?
S = NP1 BE NP3	Who was its builder?
S = NP1 BE THERE	How many are there?
S = NP1	Which ones?

Figure II-2.

The portion of Figure II-1 starting with the word "BEGIN" contains an excerpt from the procedure for the rule. The first statement assigns a value to the MOOD attribute. The expression DEIX(NP1) refers to the attribute named DEIX of the constituent NP1. If the value of the DEIX attribute of NP1 is WH, the MOOD attribute of the sentence is set to WH (indicating a question like "What ..." or "Who ..."). The MOOD is set to DEC (indicating a declarative sentence) if DEIX of NP1 is not WH and is not UNDEFINED. (The default value of attributes is the special symbol UNDEFINED.) However, if DEIX of NP1 is UNDEFINED, MOOD is also set to UNDEFINED. The next statement sets the MOOD factor, F.MOOD, to GOOD if the MOOD attribute of the sentence is WH. This is a nonBoolean factor indicating a high expectation for WH questions. The last statement in the figure is a restriction blocking elliptical sentences formed of a single nonWH pronoun such as "we". In other words, if both kinds

of predicates are omitted (OMITALL(VP1,BE)), if NP1 is a pronoun (SUBCAT(NP1) EQ "PRO), and if the sentence is declarative (MOOD EQ "DEC), then the phrase is blocked (F.REJECT(F.PROSENT)). The full procedure for the rule contains several pages of such attribute and factor statements.

In this and other rules, there are attributes that specify acoustic properties related to the input signal, syntactic properties such as mood and number (singular or plural), semantic properties such as the semantic network representation of the meaning of the phrase, and discourse properties for anaphora and ellipsis. The values of constituent attributes are used in computing the attributes of larger phrases, and the attributes of complete interpretations are used in generating responses.

The factors also use acoustic, syntactic, semantic, and discourse information. Acoustic factors reflect how well the words match the actual input, syntactic factors deal with tests such as number agreement between various constituents, semantic factors ensure that the meaning of the phrase is reasonable, and discourse factors indicate whether an elliptical or anaphoric phrase makes sense in the given dialog context. The values of factors are included in a composite score for the phrase. The

scores of constituents are combined with the factor scores to produce the scores of larger phrases, and the scores of complete interpretations are used in setting Executive priorities.

Attributes and factors either have constant values or have values that depend on attributes of constituents and global information (such as a model of the discourse or the results of preliminary, low-level acoustic processing). By design, the attributes and factors of a phrase are not allowed to depend on the context formed by other phrases that can combine with it to produce larger structures. By giving up explicit context dependency, it becomes easier to share phrases among different contexts which allows the Executive to reduce duplication of effort. (The importance of this sharing is documented in Chapter IV, Experimental Studies.) However, contextual restrictions provide heuristic information that in practice is too valuable to ignore, so the Executive alogorithms for priority setting take them into account by special techniques (see Chapter III, Sections C.2 and D.5). Essentially, we have taken the explicit context dependencies out of the rules so that phrases can be shared by different contexts, and developed methods in the Executive so that contextual restrictions can still be used in controlling the operation of the system.

Another restriction on the rule attribute and factor definitions is that they must cover cases in which the value of a referenced attribute has the special value UNDEFINED. The primary reason for UNDEFINED attributes is the desire to allow Executive control strategies that depend on information regarding incomplete phrases -- phrases missing one or more constituents. With the attribute and factor definitions required to handle UNDEFINED attributes of missing constituents, the Executive can execute the rule procedure with a partial set of constituents, and the results will be indicative of possible completions of the phrase.

2. THE LEXICON

Like the composition rules described above, the lexicon combines declarative and procedural information. However, while the rules are predominantly procedural, the lexicon mainly contains static declarations of words and their attributes. The structure of the lexicon is illustrated by considering the information for the word "length".

Figure II-3 contains an extract from the SRI lexicon containing information related to "length". "Length" is in the subcategory RELN.MEASURES (relational

SAMPLE LEXICAL ENTRY

```
WORDS.DEF N
 .

WORDFN LAMBDA(C)
    NULLDEFAULTATTRS(C,"(DETREQ MEAS RELN UNIT INDFLG));
SUBCATEGORY RELN.MEASURES
   ATTRIBUTES   MEAS=T, NBR=SG, ... ;
   WORDS
      LENGTH
         PDGM=(S.HAVE.LENGTH PG.INVH),
         SUPSET=N.LENGTHS;
      SIZE
      .
      .

      ENDWORDS;
END;
```

Figure II-3.

measures) of the lexical category N (nouns). Some other RELN.MEASURES are "size" and "speed". The noun subcategories correspond to semantic classes that are important in the task domain of the speech system. Attributes declared for a subcategory are shared by all of its members. Thus, because it is a RELN.MEASURE, "length" is automatically given several attributes including one that marks it as a measure term (MEAS=T) and another that marks it as singular (NBR=SG). Default values for some other attributes are shared by all the N subcategories. For instance, because "length" is not marked otherwise, the WORDFN redundancy function for N sets attributes DETREQ,

UNIT, and INDFLG, to the value NIL to indicate, respectively, that "length" does not require a determiner, that it is not a unit of measurement (such as "feet"), and that it does not refer to an individual (such as "England"). Most of the attributes of "length" are set according to category and subcategory redundancies. The only attributes explicitly given for the particular word "length" are PDGM and SUPSET which relate to its meaning, the information that distinguishes "length" from other RELN.MEASURES. (Phonological information that would also distinguish "length" is stored separately for technical reasons.)

In addition to word and attribute declarations, lexical categories have an associated procedure that is invoked whenever a word from the category is found in an utterance. For example, the lexical NP procedure, for words like "it" and "who", calls semantic routines to build nodes in a semantic network and also calls discourse routines to find possible referents.

3. GLOBAL DECLARATIONS

The global declarations at the start of a language definition provide information needed by the Definition Compiler and the Executive. This information includes a list of categories to be used in the definition and lists of

attributes for the categories. The global declarations can also contain redundancy functions for rewriting category and rule definitions. The metalanguage provides for these functions because they provide ways to simplify the definition; however, they have not been used in the current system.

There are several reasons why redundancy functions have not been used. First, the language definition is still relatively small. As its scope is extended, the definition's size will grow and mechanisms to simplify the statement of rules will become more important. Second, we have not yet provided special metalanguage facilities for expressing redundancies. The current mechanism simply provides a chance for a user-defined LISP program to operate on the definition before it is compiled. As a result, it is often easier to make several changes to a set of rules rather than expressing the generalization in a redundancy function. Finally, we have not yet developed facilities to allow redundancy functions to create new composition rules rather than just making modifications to existing ones. We expect this to be a major contribution of redundancy rules in future extensions of the metalanguage (see Chapter V).

4. ANNOTATED FORMAL SYNTAX

An annotated formal syntax of the metalanguage is given below using phrase structure rules with the notation described previously. Vertical bars separate alternatives, braces delimit a set of alternatives, parentheses enclose an optional set of items, angle brackets bound an optional set of alternatives, and a single preceding quote mark indicates a literal. Any item whose name ends with the string "name" is an identifier, and items with names ending with the string "names" refer to a series of one or more identifiers separated by commas. In this formalism, the first part of a language definition is shown in Figure II-4.

The global declarations include a list of categories, the name of the root category (typically the sentence category, S), specification of various functions, and declarations of attributes. The RESPONSEFN function is called by the Executive whenever a root category phrase is constructed or when some resource limit is reached. The SCOREFN function is responsible for combining individual factor values into a composite rating for the phrase. (The particular procedures used for RESPONSEFN and SCOREFN in the speech understanding system are described in Chapter III, Sections C.4 and D.5.b.) The CATEGORYFN, RULEFN, and WORDFN, are functions that make changes in the definitions

DECLARATIONS

```
language.def = "LANGUAGE.DEF decls "END ";
               rules.and.categories

decls = decl "; (decls)

decl = "CATEGORIES categorynames |
       "ROOT "CATEGORY categoryname |
       decl.function functionspec |
       "ATTRIBUTES attr.decls "ENDATTRS

decl.function = "RESPONSEFN | "SCOREFN | "WORDFN |
                "CATEGORYFN | "RULEFN

functionspec = functionname |
               "LAMBDA "( (variablenames) ") expression

attr.decls = attr.decl "; (attr.decls)

attr.decl = "{categorynames | "ALL ("EXCEPT categorynames)}
            {"HAVE | "HAS} attributenames

rules.and.categories = {lexical.category | composition.rule}
                       (rules.and.categories)
```

Figure II-4.

for lexical categories, composition rules, and words, respectively, before the definitions are compiled. The expression appearing in the function specification is an arbitrary LISP expression written in an infix notation developed at SDC (see Barnett, 1973). The attribute declarations give lists of the various categories and their attributes for use by the Compiler.

The syntax for the lexicon is shown in Figure II-5. The optional expression in the lexical category

```
                        LEXICON

lexical.category = "WORDS.DEF categoryname (expression ";)
                    lexcatparts "END ";

lexcatparts = {"WORDFN functionspec |
               "WORDS catwords "ENDWORDS |
               "SUBCATEGORY subcatname
                     ("ATTRIBUTES catwordattrs ";)
                     catwords "ENDWORDS}
               ";  (lexcatparts)

catwords = lexentryname (catwordattrs) "; (catwords)

catwordattrs = attrname "= attrvalue (", catwordattrs)
```

Figure II-5.

specifies the category procedure. Following it can come a WORDFN function to modify the word definitions in the category before they are compiled. A typical use of a WORDFN is to supply default values for attributes. A category definition can contain either a set of words or a series of subcategories. Each word can have an arbitrary number of attribute-value pairs. Each subcategory can have a set of attribute-value pairs in addition to its set of words. These attribute-value pairs provide defaults for the words in the subcategory. For example, if attribute A is listed with value B in the subcategory attributes, all words in the subcategory that do not explicitly assign a value to A get B as a default assignment. The attribute values in the lexicon are LISP data items such as atoms, numbers, or lists.

The syntax for composition rules is shown in Figure II-6. The rule structure declarations use the same notation for phrase structure as is employed in this section. The rule subfunctions and the rule expression form the procedural part of the rule. They are written in a dialect of LISP (see Barnett, 1973) with extensions for testing constituent structure and computing attribute and factor values. The Definition Compiler recognizes references to attributes by means of the global declarations of their names. Factor names are identified by an "F." prefix. A rule application can be blocked by the statement "F.REJECT(factorname)". This statement causes immediate termination of the rule. Both attributes and factors can be used in expressions and can be assigned values. Attributes of constituents can be accessed by an expression of the form "attributename(constituentname)". Attributes are often set to the same value as an attribute of the same name in some constituent, so a special statement is provided for this operation:

^ATTRS attributenames FROM constituentname.

This statement produces for each attribute in the list an assignment statement of the form

attributename=attributename(constituentname).

RULES

```
composition.rule = "RULE.DEF rulename structure
                   (subfunctions) expression "END ";

structure = categoryname "= rhsalts ";

rhsalts = rhsseries ("| rhsalts)

rhsseries = rhsitem (rhsseries)

rhsitem = "( rhsseries ") | "{ rhsalts "} | "< rhsalts "> |
          "" literalname | categoryname

subfunctions = "RULE.SUBFN functionname
               "( (variablenames) ") expression ";
               (subfunctions)
```

Figure II-6.

The main forms for testing constituent structure are 'HAVE constituentname' and 'OMIT constituentname'. HAVE implies that the constituent position is filled with a phrase. OMIT implies that the constituent position is not going to be filled because some other alternative has been selected. The rule procedures are sometimes invoked by the Executive with only a partial set of constituents, and it is possible in such cases for both HAVE and OMIT to be false for a missing constituent. Once the phrase is complete, however, either HAVE or OMIT, and not both, will be true for each constituent. For tests with HAVE and OMIT that refer to more than one constituent, logical connectives AND, OR, and NOT are available, or one of the following special

operators can be used: HAVEALL, HAVEANY, OMITALL, and OMITANY. These operators take a list of constituent names as arguments and have the obvious meanings.

C. A VERSION OF THE SRI LANGUAGE DEFINITION

A version of the SRI language definition will serve as an illustration of the use of the Definition System. The definition described below is of moderate complexity: it is less complex than those used in some current natural language text systems, but more complex than the languages of most previous speech systems. It was derived from a larger definition and used in the series of experiments described in Chapter IV.*

The domain of discourse for the language is a data base of information about ships of the U.S., Soviet, and British fleets. There is information in the data base about several hundred ships and a large number of ship classes and categories. For each ship, the data base contains characteristics such as name, type, owner, builder, length, beam, draft, displacement, speed, complement, and power.

---

* The larger definition was developed by Jane Robinson and Ann Robinson, with assistance from Gary Hendrix, Joyce Friedman, and myself. As designer of the Definition System, I influenced the general structure of the definition but did not work out the details. After the definition was relatively complete, I extracted a subset, made some revisions, and used the result in a series of experiments.

The particular domain of discourse determines a large portion of the vocabulary, and, hence, the lexicon. A change in the domain would require corresponding changes in the vocabulary and lexicon. The composition rules, however, are quite general and the effect on them of a change in discourse domain would be relatively small. Some attributes and factors in the rules have been 'tuned' to the particular domain (see Robinson, 1975), but most of them deal with general features of English.

1. GLOBAL DECLARATIONS

Figure II-7 contains an abbreviated version of the global declarations. There are 18 categories with S as the root category. There are 41 attributes, which can be divided into four sets: 12 attributes for syntax, 13 for case semantics, 9 for semantic translation, and 7 for discourse. The category with the most attributes is NP with 24. On the average, each category has about eight attributes.

2. LEXICON

The lexicon is divided into twelve categories. There are three categories of verbs, BE, DO, and V, illustrated by "is", "does", and "own", respectively. All

GLOBAL DECLARATIONS

```
LANGUAGE.DEFINITION
     CATEGORIES S,NP,VP,CLASSIFIER,PREPP,PREP,N,BE,DO,V,
          NUMBER,CENTI,SMALLNUM,TEEN,DIGIT,DET,WHDET,ADJ;
     ROOT CATEGORY S;
     ATTRIBUTES
          S HAS REPLY;
          V,VP,BE,DO HAVE TENSE;
          NP,ADJ,VP,WHDET,DET,PREPP,NP HAVE SUPSET,SUPCASE;
          .
          .

          ENDATTRS;
     END;
```

Figure II-7.

have attributes for number (singular or plural) and tense (present or past), and verbs in category V also have attributes for voice (active or passive) and case semantics that resemble the case grammar of Fillmore (1968) as adapted to computer use by Celce Murcia (1976) and others. There are two categories of numbers: DIGIT and TEEN. The TEENs ("ten", "eleven", and "twelve") are separate because they do not combine in the same manner as DIGITs to form larger numbers (for example, 31 cannot be said as "twenty eleven"). Both DIGITs and TEENs have attributes giving their numeric value and their grammatical attributes. Determiners are also split into separate groups to simplify the rules: declarative determiners like "the" are in the category DET; question (WH) determiners like "what" are in the category

WHDET; and the indefinite "a" is included as a literal in the noun phrase rule. The categories for adjective (ADJ) and prepositions (PREP) also appear in the lexicon, but are represented by only two words each: "of" and "by" for PREP, "fast" and "long" for ADJ.

The final three lexical categories (N, CLASSIFIER, and NP) are each divided into subcategories. There are two subcategories of NPs: countries (such as "Russia") and pronouns (like "it"). In both cases, the words have attributes similar to those for a noun phrase constructed by the NP composition rule. These attributes include number (singular or plural), case (nominative or accusative), and semantic interpretation. The lexical NP procedure also calls the discourse routines to find possible referents for the pronouns; it blocks use of the pronoun if no referent is found.

Classifiers are prenominal modifiers. There are three subcategories of classifiers in the lexicon: countries (as in "British ships"), type designations (as in "nuclear submarine"), and predicates (as in "patrol sub"). All the classifiers have attributes indicating the kinds of nouns they can modify and their semantic translation.

By far the largest lexical category is N, nouns. There are 10 subcategories of N: units of measure (such as "ton"), parts of ships (such as "reactor"), classes of ships (such as "Nautilus"), individual ships ("Sealion"), companies ("General Dynamics"), countries ("England"), relational measures ("length"), two kinds of ship types ("CGN", "submarine"), and a subcategory of miscellaneous nouns. Members of category N have grammatical attributes like number, semantic attributes such as pointers into a semantic network, and attributes used for both syntax and semantics such as information regarding whether the N is a measure, a unit, or a relation.

3. COMPOSITION RULES

In addition to the twelve lexical categories, the language definition includes ten composition rules. First to be discussed are the three number rules whose phrase structure declarations are given in Figure II-8. SMALLNUMs can be a DIGIT ("one"), a TEEN ("eleven"), a DIGIT followed by the suffix TEEN ("fourteen"), a DIGIT followed by the suffix TY ("seventy"), or a DIGIT TY DIGIT sequence ("sixty four"). The two occurrences of TEEN and DIGIT in the phrase structure are disambiguated by use of numeric suffixes, "1" and "2". Thus, the rule procedure refers to

PHRASE STRUCTURE PARTS OF NUMBER RULES

```
SMALLNUM = TEEN1 | DIGIT1 <"TEEN2 | "TY (DIGIT2)>
CENTI = (SMALLNUM1) ("HUNDRED (("AND) SMALLNUM2))
NUMBER = (CENTI1) ("THOUSAND (("AND) CENTI2))
```

Figure II-8.

the first (leftmost) DIGIT as DIGIT1, and the second, as DIGIT2. The SMALLNUM procedure checks attributes on the digits since some cannot be followed by TEEN or TY ("one" is acceptable, but not "oneteen" or "onety"), some can be followed by TY but not TEEN ("twenty", but not "twenteen"), and some must be followed by either TEEN or TY ("thirteen" or "thirty", but not "thir").

The CENTI rule allows numbers like 2235 to be said in various ways including "twenty two hundred and thirty five". The CENTI procedure blocks cases like 4000 said as "forty hundred" and also computes the numeric value of the CENTI phrase from the values of the constituents. The NUMBER rule can construct number phrases like "two thousand and one". The procedure blocks phrases like 8100 said as "eight thousand hundred" or as "one thousand and seventy one hundred".

In the remainder of this section, the descriptions of rules are typically limited to simple sketches like the preceding ones. However, to give a better indication of how the rules are actually written, one rule procedure, for SMALLNUM, will be discussed in detail. The SMALLNUM rule definition is given in Figure II-9. The procedure body is a conditional statement with four main cases that depend on the constituent structure.

SMALLNUM RULE DEFINITION

```
RULE.DEF SMALLNUM
     SMALLNUM = TEEN1 | DIGIT1 <"TEEN2 | "TY (DIGIT2)>;
     IF HAVE TEEN1 THEN ^ATTRS NUMTYP,NUM FROM TEEN1
     ELSE IF HAVE TEEN2 THEN
       [IF TEEN(DIGIT1) EQ "NO THEN F.REJECT(F.DIGTYP1),
        IF COMPLETE.NODE THEN NUM=NUM(DIGIT1)+10]
     ELSE IF HAVE TY THEN
       [IF TY(DIGIT1) EQ "NO THEN F.REJECT(F.DIGTYP2),
        IF HAVE DIGIT2 THEN
          [IF ALONE(DIGIT2) EQ "NO THEN F.REJECT(F.DIGTYP3),
           IF COMPLETE.NODE THEN
              NUM=NUM(DIGIT1)*10+NUM(DIGIT2)]
        ELSE IF OMIT DIGIT2 THEN
          [NUMTYP="DECADE2,
           IF COMPLETE.NODE THEN NUM=NUM(DIGIT1)*10]]
     ELSE IF HAVE DIGIT1 AND OMITALL(TEEN2,TY) THEN
       [IF ALONE(DIGIT1) EQ "NO THEN F.REJECT(F.DIGTYP4),
        NUM=NUM(DIGIT1)];
```

Figure II-9.

In the first case, the SMALLNUM is a TEEN (TEEN1 in the structure declaration), a number from the lexical category including "ten", "eleven", and "twelve". The attributes NUM and NUMTYP are copied to the SMALLNUM phrase

from the TEEN by the ^ATTRS statement. The second case for SMALLNUM occurs when the suffix TEEN is used (TEEN2 in the structure declaration). In this case, there are two statements to be performed (grouped together by square brackets and separated by a comma). The first statement looks at the TEEN attribute of DIGIT1 and blocks the phrase if the attribute is NO by performing F.REJECT(F.DIGTYP1). This blocks bad DIGIT TEEN sequences such as "oneteen". The second statement tests the flag named "GOMPLETE.NODE" and, if the flag is true, sets the NUM attribute, which gives the numerical value of the phrase, to ten plus the NUM attribute of DIGIT1. (The Executive sets COMPLETE.NODE false when applying a rule with some of the constituents missing or for special tests.)

The third main SMALLNUM case is executed when HAVE TY is true. The first statement checks the TY attribute of DIGIT1 and blocks the phrase if the attribute value is NO (eliminating phrases like "onety"). The second statement is another conditional depending on the phrase structure. If HAVE DIGIT2 is true, two statements are performed: a check that DIGIT2 can occur without a suffix -- that is, ALONE(DIGIT2) is not NO -- and an assignment statement setting the NUM attribute to NUM of DIGIT2 plus ten times NUM of DIGIT1. If HAVE DIGIT2 is false but OMIT DIGIT2 is

true, the NUMTYP attribute is set to DECADE2. This attribute is used in checks in the CENTI rule to block phrases like "forty hundred". The NUM of the SMALLNUM is then set to ten times the NUM of DIGIT1.

The fourth and final case for the SMALLNUM procedure occurs when the phrase has DIGIT1 and omits both suffixes. The ALONE attribute is checked to block the phrase if DIGIT1 needs a suffix (for example, "thir-" is in the lexicon as a digit that needs a suffix). If that test is passed, the NUM attribute is copied from the digit.

As an example composition rule, SMALLNUM accurately reflects the general form, but it is atypically simple. In contrast to the one-third page size of the SMALLNUM rule, the average rule length is about one page, and the biggest rule, for noun phrases, is almost three pages. Consequently, the following discussions are limited to sketches of rules rather than exhaustive, line-by-line documentation.

The phase structure declaration for the noun phrase rule is

```
NP = {"HOW.MANY | <DET | WHDET | "A> (NUMBER)}
        ((CLASSIFIER) N ("PL) (PREPP)).
```

The noun, N, can be optionally preceded by a CLASSIFIER and followed by a plural suffix (PL) and a prepositional phrase

(PREPP). At the front of the noun phrase, there can optionally be "how many" or an optional choice of DET, WHDET, or "a", followed by an optional number. This phrase structure allows many possibilities, some of which must be blocked by the NP procedure. For instance, "a" must be blocked if it occurs without a following number or noun. Other structures are blocked in certain cases depending on the attributes of constituent phrases. For example, the NUMBER cannot be leftmost if it begins with "hundred" or "thousand". On the other hand, if the NUMBER does not start with "hundred" or "thousand", it cannot be preceded by "a".

The NP procedure has a large number of statements related to the head noun, N. Several NP attributes are derived from corresponding N attributes, and many of the restrictions on possible NPs depend on properties of the N. For example, if N refers to an individual such as a particular company, ship, or country, it cannot be preceded by "how many", "a", WHDET, CLASSIFIER, or NUMBER, and the only preceding DET allowed is "the". There are also case semantics checks for the noun with a CLASSIFIER or a PREPP, and number agreement tests for the noun with several other NP constituents.

When the NP is complete, the procedure invokes routines to construct a semantic net representation of its

meaning. If the NP has a definite determiner (like "the" or "that"), there are also calls on discourse routines to look for possible referents. The phrase is rejected if the semantic translation cannot be made or the discourse referents cannot be determined.

The phrase structure part of the verb phrase rule is VP = V <<"SG | "PAST> NP | ("PPL) (PREPP)>. The constituents are verb (V), singular suffix (SG), past tense suffix (PAST), object noun phrase (NP), passive suffix (PPL), and prepositional phrase (PREPP). The VP procedure checks various attributes of the verb and other constituents to block unwanted combinations such as a verb marked as active (like "have") followed by a passive marker (PPL). There are similar syntactic checks regarding tense and number. Case semantics checks ensure that the verb is compatible with the object and the prepositional phrase.

The prepositional phrase rule is one of the simplest. The phrase structure is just PREP NP. The PREPP procedure blocks the phrase if the NP is nominative case (like "we"), or if the preposition and the noun phrase do not go together semantically. The phrase is given a low rating if the NP is marked WH, or if the NP contains a NUMBER and the noun is not a unit or a relation. (For example, "of ten knots" is okay, but "of ten ships" is considered unlikely in view of the expected questions.)

The remaining rules are for the root category, S. The simplest S rule has the phrase structure HOW ADJ BE NP, illustrated by a sentence like "How fast is it?" The rule procedure checks the semantics of the adjective and the noun phrase for compatibility. Phrases are blocked if BE and NP do not agree in number, or if the NP is marked WH or accusative case ("us"). The phrase is given a low rating if the BE is past tense or the NP is indefinite and has a NUMBER. ("How fast are the ten ships?" is okay, but "How fast are ten ships?" is dubious.) The phrase is blocked if the NP is a unit or measure, or if the semantic translation fails for other reasons.

Another relatively simple S rule has the phrase structure S = (DO NP) VP. This rule handles imperatives and questions starting with a DO verb. If DO and NP are present, they must agree in number, the NP must not be marked as WH or accusative case, and the VP must not be imperative. If DO and NP are omitted, the VP must be imperative and must not be marked WH. In either case, the VP must not be marked as singular, past, or passive, and the phrase is blocked if semantic translation fails.

The phrase structures for the last two S rules are

S = BE NP1 {NP2 | VP}, and

S = NP1 <(DO NP2) VP1 | BE {VP2 | NP3 | "THERE"}>.

These rules handle a variety of question types and elliptical sentences. Both make many tests concerning syntax, case semantics, and semantic translation. The procedure for the first rule is about one page long, and the second is about two pages in length.

In summary, the composition rules use the phrase structure declaration to give the basic constituent possibilities and use the procedure to block or downgrade unwanted combinations and upgrade expected ones. The procedures are organized as nested conditionals depending on the constituent structure. Simple syntactic tests are made first, followed by case semantics, and, finally, by semantic translation and discourse processing.

D. THE DEFINITION COMPILER

The Definition Compiler translates ('compiles') a language definition into a form for use by the Executive System. The purpose of compilation is to convert the language definition to an internal representation that can be efficiently used by the Executive in processing utterances. In choosing the internal representation, Executive runtime was the primary consideration, with storage requirements also an important concern. We have put less emphasis on user-features, although the compiler does

allow incremental changes on a rule-by-rule basis. This section describes the internal form of a language definition in detail and sketches the principal Definition Compiler algorithms.

1. CATEGORY RECORDS AND THE LEXICON

The global declarations for a language definition include a list of the categories. In addition to these declared categories, the Compiler creates special one-word lexical categories for each distinct literal used in the composition rules. These special categories are constructed so that the Executive does not have to treat literals as a separate case. For each category, declared or specially created, the Compiler constructs a record containing several components, the most important of which are the following:

* A list of attributes for the category. These are derived from the global declarations section of the language definition.
* A list of rule records for the rules that produce phrases of this category.
* A list of categories that can occur as the leftmost terminal phrase in a phrase of this category. This component, the next one, and the ones like it in the rule records, are used for "lookahead" by the Executive.
* A list of categories that can occur as the rightmost terminal phrase in a phrase of this category.

* A LISP function to set the attributes and factors of terminal phrases of this category. This function is created from the category procedure given in the language definition.

* A list of lexical subcategory structures.

Each lexical subcategory structure is a list containing the name of the subcategory, the default attribute-value pairs for members of the subcategory, and the list of members. Each member is represented by a list with the word, its attribute-value pairs, and a back-pointer to the category record.

A lexical category declaration is compiled in a series of steps. It is first converted into a list structure, and the language CATEGORYFN, if any, is called to modify the definition. The subcategories are then compiled with the global WORDFN and the category WORDFN applied to each word definition before its attributes are stored.

In addition to the words declared in the language definition, the internal lexicon contains items called 'multiword lexical entries', or 'multiwords . These items are treated as single units for acoustic processing but not for linguistic processing. For example, the phrases "of the" and "are the" are among the multiwords used in the speech system. The use of multiwords improves the acoustic performance by providing larger units for testing. However,

the language definition would become excessively complicated and lose linguistic generality if multiwords had to be treated as single words linguistically, so the Compiler and the Executive cooperate to hide the existence of the multiwords from the language definition.

The Executive's treatment of multiwords is described in Chapter III, Section C.4. The Compiler's job is to add them to the lexicon so that a multiword phrase X starting with word A is included in all the lexical subcategories that include A. Thus, whenever the Executive considers A as a candidate word, X will be available for consideration also. Since the Executive can work in both directions in an utterance, the Compiler also adds multiwords Y that end in word B to all subcategories including B. The multiwords in the lexicon are marked to indicated whether they are to be considered in left-to-right tests (such as X) or right-to-left tests (such as Y).

2. RULE RECORDS AND STRUCTURE GRAPHS

The Compiler creates a record for each rule containing, among other things, the following components:

* A graph representing the phrase structure possibilities for the rule.
* A list of categories that can occur as the leftmost terminal phrase in a phrase constructed by this rule.

* A list of categories that can occur as the rightmost terminal phrase in a phrase constructed by this rule.
* A LISP function created from the rule procedure.
* A back-pointer to the category record for this rule.

Rule compilation proceeds in a series of steps: (1) the rule is translated into a list structure, (2) the language RULEFN, if any, is applied to modify the rule definition, (3) the phrase structure graph is created, and (4) the rule procedure is rewritten and compiled as a standard LISP function.

The phrase structure declaration for a rule is translated into an acyclic, directed graph. The arcs are assigned index numbers reflecting their left-to-right order. A bit vector is stored with each arc to indicate the other arcs that are mutually exclusive with it. Redundant NIL arcs are added to the graph so that paths do not need to include two NIL arcs in a row. Finally, the NIL arcs from a point are ordered so that a search for a filled path can stop after considering the first unblocked NIL arc. The following paragraphs describe the creation of the structure graph in greater detail.

The arcs in the graph are labeled with either a category or NIL. Recall that literals are replaced by special one-word categories, so there is no need for a special kind of arc for literals. NIL arcs are introduced to deal with optional elements in the graph.* There is a unique starting, or "leftmost", point in the graph, and a unique ending, or "rightmost", point. A path is a series of arcs A1,...Ak, such that the end point of Ai is the starting point of Ai+1. It is a complete path if the starting point of A1 is the leftmost point in the graph, and the endpoint of Ak is the rightmost point. The category labels along any complete path indicate a valid sequence of constituents for the rule. Figure II-10 shows a phrase structure declaration and its corresponding graph.

A phrase structure graph is stored as a collection of points and arcs. Each point is represented by lists of arcs coming in from the left and arcs going out to the right. The arc lists for each direction from a point are divided into separate sections for category arcs and NIL arcs so that the Executive does not have to test each arc every time it is used to see which kind it is. Each arc is represented by a list containing its starting point, its ending point, its label (a category or NIL), an index

---

* NIL arcs are somewhat like JUMP arcs in an augmented transition network (see Woods, 1970).

A PHRASE STRUCTURE DECLARATION AND
ITS CORRESPONDING GRAPH

S = NP1 <(DO NP2) VP1 | BE {VP2 | NP3 | "THERE}>

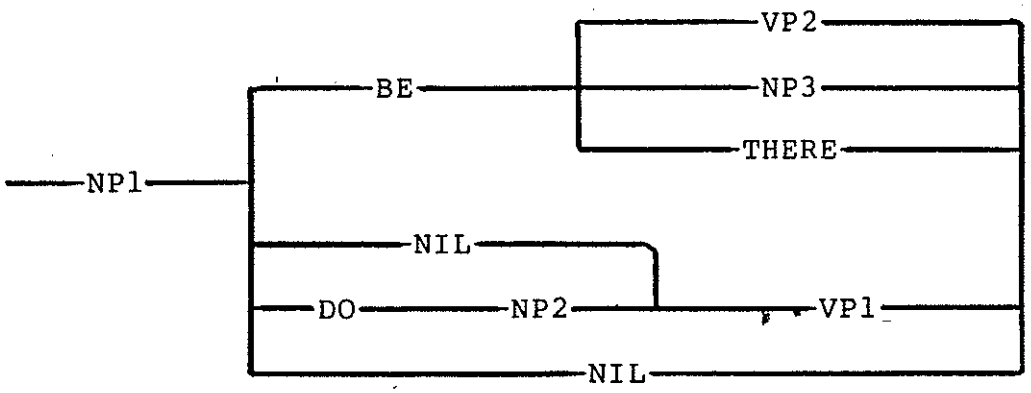

Figure II-10.

number, and a bit vector indicating other arcs in the graph that cannot occur in complete paths that contain this arc (for instance, the BE arc above cannot occur in complete paths with DO, NP2, or VP1 arcs or with either of the NIL arcs). Notice that this representation allows the Executive to search the graph in either direction from any arc or point.

In processing an utterance, the Executive tries to get a series of adjacent phrases corresponding to the category labels on a complete path through the structure graph. As the subphrases of a phrase are acquired, they are stored in a constituent-array for the phrase in the position specified by the index number of the corresponding category

arc. The Executive can check whether it has acquired a constituent for a particular category arc by a simple array reference using the arc index number.

Since the Executive is often concerned with the relative order of constituents, the category arc index numbers are assigned such that if the index of category arc A is less than the index of category arc B, either A is to the left of B or they are mutually exclusive. (Arcs are mutually exclusive if there is no complete path that includes both of them.) As an example, the order of the category arc index numbers in the graph shown above is NP1, BE, VP2, NP3, THERE, DO, NP2, and VP1.

The Executive takes advantage of the ordering of category arcs in many places. For instance, to find the first filled arc (an arc with an acquired constituent for the phrase under consideration) to the right of the arc with index number I, the Executive searches through the constituent-array for the first nonNIL entry at location I+1 or above. The arcs are also stored in an array according to their index numbers, so if a phrase is found in location J of the constituent-array, the same index J can be used to access the corresponding arc in the graph. In this way, the Executive substitutes array scans for graph searches.

The operation of the Executive is simplified further by adding redundant NIL arcs to the graph so that it is never necessary to traverse two NIL arcs in a row. If two points, A and B, in the graph are connected by a path of two or more NIL arcs, but no single NIL arc connects them, the Compiler adds a new one to connect A and B directly. No redundant NIL arcs are added to the graph in Figure II-9, but in other cases, such as the NP rule, many are needed. Figure II-11 shows the NP graph before redundant NIL arcs are added. To this graph, the compiler adds five NIL arcs: (1) from the leftmost point to the point at the right of the NUMBER arc, (2) from the left of the NUMBER to the right of the CLASSIFIER, (3) from the leftmost point to the right of CLASSIFIER, (4) from the left of NUMBER to the rightmost point, and (5) from the left of PL to the rightmost point.

NP GRAPH BEFORE ADDITION OF EXTRA NIL ARCS

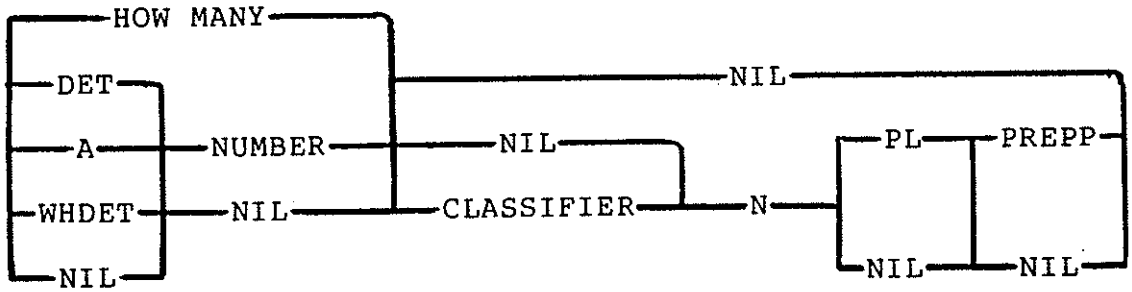

Figure II-11.

The redundant arcs simplify the Executive algorithms by allowing iterative operations to replace recursive searches of arbitrarily long, and perhaps converging, paths of NIL arcs. For example, to check all the categories that can occur immediately to the left of a given constituent, the Executive can fetch the point at the left of the arc for the constituent, check the incoming category arcs, and then for each incoming NIL arc, check the incoming category arcs for the point at the left of it. Because of the presence of redundant NIL arcs, this simple algorithm covers all the possibilities without duplication. For example, to the left of the N arc in the NP graph after the redundant NIL arcs have been added, there is an incoming category arc for CLASSIFIER and three incoming NIL arcs: one to NUMBER and HOW.MANY, a second to DET, WHDET, and A, and a third to the leftmost point in the graph.

The Executive can acquire constituents of a phrase in any order, not just left to right. Consequently, after each category arc is filled, tests are made to see if a complete path has been created. The tests succeed if there is a filled path from the left of the newly filled arc to the leftmost point in the graph and a filled path to the rightmost point. (A path is filled if all of its category arcs are filled.) NIL arcs are used in the search for filled paths if they are not marked as blocked.

The Compiler makes this search more efficient by ordering the list of NIL arcs from each point so the Executive never needs to try more than one of them. The outgoing NIL arcs from a point are ordered such that if arc A precedes arc B in the list, no path from the endpoint of A leads to the endpoint of B. Similarly, the incoming arcs are ordered such that if A precedes B, no path leads from the starting point of B to the starting point of A. Basically, this means putting the 'longest' arcs at the front of the lists. For example, the NIL arcs coming in to the left of the N in the NP graph are ordered such that the first goes to the leftmost point, the second goes to the point at the left of NUMBER, and the third goes to the point at the left of CLASSIFIER.

With the NIL arcs ordered in this way, the search for a filled path only needs to consider the first unblocked NIL arc. If the first one fails to lead to a filled path, none of the following ones can possibly succeed. To prove this, assume to the contrary that A precedes B in a list of outgoing arcs, both are unblocked, a search to the right starting with A fails to lead to a filled path, but a search using B succeeds. The point at the right of B cannot be the rightmost point in the graph, since A leads to the rightmost point, and A is before B in the list. Thus, there must be a

filled category arc immediately following B. However, this contradicts the hypothesis that A is unblocked since no path including A leads to the end of B where the filled arc begins. The proof for incoming NIL arcs is similar.

3. RULE PROCEDURES

After the phrase structure declaration is translated, the Compiler begins work on the rule procedure. The procedure statements dealing with attributes, factors, and constituent structure are rewritten as standard LISP statements that will work in the environment provided by the Executive. Before calling the rule procedure, the Executive sets up an environment containing: (1) the constituent-array for the phrase, (2) the bit vector showing blocked arcs, (3) the array of attribute values, and (4) the array of factor values. The Compiler converts "HAVE constituentname" expressions to constituent-array references using the appropriate category arc index. Similarly, "OMIT constituentname" expressions are converted to a test of the appropriate bit in the bit vector of blocked arcs. In both cases, the Compiler looks up the arc number corresponding to the constituent name, and a macro produces the required code.

References to attributes are converted to attribute-array references using as index the position of the attribute in the category attribute list. Factor references are converted to factor-array references using as index the position of the factor name in the list of factors for the rule. In both cases, the array indexes are constants known at compile-time, so efficient code is produced. For example, references to constituent attributes produce eight PDP-10 instructions to load an item from the constituent-array, and to give UNDEFINED if the item is NIL or, if it is not, get the attribute value from the attribute-array of the constituent.

The statement employed to abort a rule, "F.REJECT(factor)", is converted to a call on a function named F.REJECT with two arguments: the name of the factor and the name of the rule procedure. The F.REJECT function calls an INTERLISP subroutine ("RETFROM") to cause the rule function to return immediately with the value NIL as an indication of failure.

The last step in compiling a rule is to create an empty phrase for it, a phrase with no constituents. The phrase is saved with the rule record and used by the Executive in ways described in the next chapter.

4. LOOKAHEAD INFORMATION

Both category records and rule records contain lists of the categories that can occur as the leftmost or rightmost terminal phrases of that category or rule. This information is used by the Executive to 'look ahead' to avoid unnecessary work on a category or rule whose possible categories for boundary words do not intersect the categories of the word possibilities determined by acoustic tests. For example, before trying to construct a verb phrase starting at a particular location in the input, the Executive checks acoustic results for that location to ensure that the possibilities include at least one word that can occur as the leftmost word in a verb phrase. The next chapter contains more discussion of the use of lookahead by the Executive.

E. DISCUSSION

In this section, we compare our Definition System to some alternative approaches that have been used in previous efforts.

The best known natural language understanding system is undoubtedly Winograd's SHRDLU (Winograd, 1971). The language definition system used in SHRDLU is called

PROGRAMMAR, and, like the other components of SHRDLU, emphasizes a procedural approach to representing knowledge. A PROGRAMMAR program is designed for top-down, left-to-right sentence processing. The structural possibilities for the defined language are encoded in the control structure of the program rather than being declared separately in a form such as phrase structure rules. This method reflects a desire to encode a great deal of special-case knowledge to guide processing as an alternative to relying on a uniform but weak algorithm.

The emphasis on special-case knowledge and close cooperation among different knowledge sources during sentence processing is a major contribution, but the particular method used in PROGRAMMAR makes it difficult to experiment with different overall control strategies. In earlier speech understanding work at SRI, we used a procedural approach in the PROGRAMMAR tradition (Walker, 1973a,b), but that approach was abandoned to allow freer experimentation. Our current approach retains the PROGRAMMAR emphasis on special-case knowledge and close cooperation of knowledge sources, but it eliminates from the language definition the commitment to a particular control strategy. Procedural representation is limited to attribute and factor information; the structural possibilities for the

defined language are declared separately rather than being encoded in the control structure of a program. Thus, our use of augmented phrase structure (APS) rules attempts to keep the most valuable aspects of PROGRAMMAR's procedural representation while eliminating its constraints on system control options.

Another well-known approach is the use of augmented transition networks (ATNs) for language definition (Thorne, Bratley, and Dewar, 1968; Bobrow and Fraser, 1969; Woods, 1970). ATNs are extended versions of finite-state machines of automata theory. The first extension is to allow state transitions to depend on the successful execution of an entire network rather than being limited to testing a single word. By this extension, context-free languages can be handled. The second extension is to allow each arc to have an arbitrary condition associated with it that must be satisfied for the arc to be used in a transition. This extension gives ATNs the theoretical power of Turing machines.

ATNs have been used sucessfully in several large natural language understanding systems (such as LUNAR described in Woods, Kaplan, and Nash-Webber, 1972), and the approach is also used in the BBN speech-understanding system (see papers in the 1974 IEEE Symposium, Erman, 1974b).

However, we prefer an approach based on augmenting phrase-structure rules rather than transition networks. One reason for this is a personal preference for reading and writing rules rather than ATN networks,* but a less subjective reason concerns the relative freedom from control strategy commitments.

Recall that our objection to Winograd's PROGRAMMAR approach centered around its commitment to a particular control strategy. However, as Winograd notes, PROGRAMMAR programs and ATNs "are just two different ways of talking about doing exactly the same thing" (Winograd, 1971, p.201). An ATN is conceptually a description of a nondeterministic machine. To process a sentence, the machine moves through a series of states specified by the structure and conditions of the ATN. The order of transitions is fixed, at least conceptually, by the left-to-right scan of the sentence, so tests and actions associated with arcs at the right of a network make use of information from previous arcs to the left. Although this left-to-right assumption affects the writing of augmentations on the arcs, it is not an absolute barrier to the use of other control strategies. Unlike a

---

* This preference appears to be shared by some users of ATNs. In a recent report, BBN comments that to document its grammar it has used a "semi-BNF" notation "which indicates much more clearly than the grammar listing what sorts of sentences are accepted by the grammar" (Woods et al., 1976a, p.10).

PROGRAMMAR program, an ATN does separate the basic structure (i.e., the network) from the augmentations (the tests and actions associated with the arcs), so it is possible to use ATNs with non-left-to-right control strategies. The method for doing this depends on recognizing augmentations that use contextual information and delaying their execution until the necessary information is available (see Bates, 1975).

Our APS rules avoid control commitments, conceptual or otherwise, by putting the augmentations in a single procedure rather than spreading them over a network; if a test or action uses information from several constituents, it is embedded in conditional statements that check for the relevant structure. The augmentations are thus organized in a way that avoids the left-to-right bias of ATNs. Although that bias can be circumvented, we prefer to use a representation that eliminates it rather than forcing the Executive to try to work around it.

Our preference for APS rules over more procedural methods such as PROGRAMMAR and ATNs is shared by others. Such a preference appears, in fact, in early work on compilers for programming languages. The first programming language compilers were completely procedural; the definition of the language was embedded (lost) in the control structure of the compiler. In reaction to the

obscurity of this method, "syntax-directed compiling" was developed by Irons and others (see, for instance, Irons, 1961; and Cheatham and Sattley, 1964).

The developers of the new method were explicitly concerned with separating the two functions of defining the language and translating it, functions which are merged in procedural approaches (see opening comments in Irons, 1961). Irons used a version of APS rules to state the syntax and semantics of a programming language. Each phrase structure rule had an associated semantic definition to form a 'translation' for a phrase constructed by the rule. The translation was the only attribute of the phrase and was formed from the translations of the constituents. Irons implemented a general translator program to operate on such language definitions and demonstrated the usefulness of the approach by developing an ALGOL 60 compiler.

Irons' technique of using APS rules for programming languages was extended by Knuth in a paper on the "semantics of context-free languages" (Knuth, 1968). Knuth's first extension was to allow an arbitrary number of attributes with each phrase. A set of attribute-defining functions was associated with each phrase structure rule rather than the single translation function of Irons. The second and more significant extension was to allow both 'synthesized' and

'inherited' attributes. Synthesized attributes of a phrase are defined solely in terms of attributes of the constituents of the phrase. Irons' translation attributes and our rule attributes are of this type. Inherited attributes of a phrase are defined by functions associated with phrases that include it as a constituent. In other words, these attributes are 'inherited' from the context rather than being 'synthesized' from information local to the phrase. In this system, there is a danger of circular definitions of attributes (such as inherited attribute A depending on attribute B, which is in turn synthesized by a function with A as an argument), but such circular definitions can at least be detected automatically by an algorithm sketched in Knuth's paper.*

Knuth points out that inherited attributes do not provide greater theoretical power since "synthesized attributes alone are (in principle) sufficient to define any function of a derivation tree" (Knuth, 1968, p.142). However, he claims that in practice the use of both kinds of attributes can lead to important simplifications producing more "natural" definitions. To support this claim, he gives

---

* However, the problem of determining whether the grammar avoids circularity in all possible instances is very difficult computationally. See Jazayeri, Ogden, and Rounds (1975) for a proof that any correct algorithm for solving this problem must require time that grows exponentially with the size of the grammar.

a small language definition that makes use of both synthesized and inherited attributes. The inherited attributes are used for operations such as testing the agreement between the declaration and the use of variables. Knuth comments that "in general, inherited attributes are useful when part of the meaning of some construction is determined by the context in which the construction appears" (Knuth, 1968, p.142).

Although Knuth's discussion is limited to programming languages, his system of inherited and synthesized attributes appears attactive for use in natural language processing. In fact, it has been used for semantic translation in the REQUEST system, which is an experimental question-answering system based on a transformational grammar of English (see Petrick, 1973, 1976). In REQUEST, an input is parsed according to a surface structure context-free grammar, the surface tree is converted to a deep-structure tree by reversed transformations, and the deep-structure tree is mapped into a "logical representation" by Knuth's translation technique, using both synthesized and inherited attributes.

Faced with Knuth's claims supported by the example of REQUEST, we must explain our decision restricting the APS rules to use only synthesized attributes. In this case, as

with our rejection of PROGRAMMAR-like procedural representations, the primary motivation is the desire to free the language definition from features that would excessively constrain the options for the Executive. Knuth states that his approach does not depend on any particular form of syntactic analysis. This is certainly true if the attributes are not to be computed until after a complete derivation tree is constructed, but we cannot afford to force the Executive to find complete context-free parses before drawing on attribute and factor information.

The Executive must be free to use such information during sentence processing to limit and direct its efforts. Furthermore, inherited attributes make it difficult to share a phrase among several competing contexts. Such sharing is particularly important with speech understanding since acoustic uncertainty leads to a large number of alternative contexts. Inherited attributes are context dependent, so they, and all other attributes depending on them, would have to be duplicated for each context. Thus, we have restricted ourselves to using only synthesized attributes because we cannot delay the use of augmentations until a complete parse is found, and we cannot afford to duplicate attribute and factor information for each context. The restriction to synthesized attributes and factors provides important

flexibility in the Executive, and, to date, it has not been an impediment to the development of the SRI language definition.

A variety of computer systems for processing natural language have used some form of APS rules (for example, Sager and Grishman, 1975; Hobbs, 1974; Heidorn, 1975, Pratt, 1973, Landsbergen, 1976). The first was the Linguistic String Parser implemented at New York University under Sager in 1964-1965. The system has been redesigned and reimplemented since then, but it has continued to use a two component grammar: context-free rules defining the broad construction patterns of sentences, and restrictions covering detailed constraints.

There is an emphasis on restrictions (corresponding to our Boolean factors), but the system does allow attributes to be set for nodes in the parse tree. In contrast with our approach, the restrictions for a rule are not organized into a single procedure. Instead, the approach foreshadows ATNs by associating restrictions with particular positions in the rules. As with ATNs, the positioning of restrictions assumes left-to-right sentence processing. For example, in a rule A = B C, a restriction might be positioned between B and C so it would be executed after the B phrase was acquired and before the C was tried. Also, some of the

'restrictions' are really optimizations for the top-down back-up parser, so there is some blurring of the distinction between the language definition and the control strategy for applying the definition. From our standpoint, this blurring and the left-to-right bias caused by positioning restrictions within rules are both shortcomings of the approach. However, the successful application of the approach to produce a grammar of wide scope is evidence for the value of using APS rules for natural language.

Other APS systems for natural language processing have avoided the shortcomings mentioned above. For example, PHLIQA1 uses APS rules each with a single procedure for augmentations to translate from English to the first of several levels of semantic translation (Landsbergen, 1976; Scha, 1976). Our metalanguage was influenced by PHLIQA1 and differs mainly in allowing alternatives and options in structure declarations and in providing for nonBoolean factors in addition to Boolean restrictions. These additions are especially important in a system for speech understanding: the alternatives and options reduce the number of rules and hence decrease the storage requirements of the Executive, and the nonBoolean factors are of use in setting Executive priorities.

To summarize the preceding discussion, our Definition System continues a long-established line of systems using APS rules. We share with earlier developers, such as Irons, the desire to keep the language definition free of control strategy commitments in order to make the definition simpler and to allow greater flexibility in experimenting with different system designs. Our system differs from previous ones in having broader phrase structure declaration capabilities and in allowing nonBoolean factors.

Up to this point the discussion has focused on the metalanguage and the choice of APS rules as a representation. The other major component of the Definition System is the Compiler. The Compiler creates an internal representation of a language definition for use by the Executive in sentence processing. The internal representation has several features that differentiate it from those used in previous systems. The networks representing the phrase structure declarations are reminiscent of ATNs or charts (see papers by Kay, 1973; Kaplan, 1973a) but they are distinguished from those systems by the presence of extra NIL arcs and the ordering of NIL arcs, both changes that contribute to Executive efficiency.

Other distinctive features of the internal representation of the language are also concerned with

efficiency of Executive operations. These features are (1) the use of parallel arrays for structure graph arcs and phrase constituents, with entries ordered to reflect the left-to-right structural possibilities, (2) the use of bit vectors to keep track of blocked arcs and mutually exclusive arcs, (3) the translation of rule procedures into compiled LISP functions employing in-line instructions for efficient operations on attributes and factors and quick tests of constituent structure, and (4) the construction of left and right lookahead information for both rules and categories. The internal representation is tied to the design of the Executive, so further discussion of the representation is deferred to Chapter III.

III THE EXECUTIVE SYSTEM

CONTENTS:

A. INTRODUCTION

The Executive in the speech understanding system has three main responsibilities: (1) it coordinates the work of the other components of the system by calling acoustic processes and applying language definition rules, (2) it assigns priorities to the various tasks in the system, and (3) it organizes hypotheses and results so that information is shared and duplication of effort is avoided. In other words, the Executive carries out the functions of integrating and controlling the system components. Experimental results, to be discussed in Chapter IV, show that the manner in which the Executive performs these functions has a large effect on the overall performance of the system. For example, different techniques for setting priorities result in significant differences in average accuracy and runtime.

The following sections contain (1) a description of the main Executive data structure, called the 'parse net', (2) an overview of the Executive in sufficient detail to allow the reader to understand both the discussion in the last section of this chapter and the experimental results covered in Chapter IV, (3) a complete description of the Executive, and (4) a discussion comparing this approach to several others and sketching its evolution. This chapter presumes

familiarity with the internal representation of the language definition as described in Section D of Chapter II.

B. PARSE NET

The 'parse net' is the principal data structure built by the Executive.* This section describes the form and content of the parse net; following sections describe the procedures that operate on it. Nodes in the parse net are either 'phrases' or 'predictions.' Phrases correspond to words or composition rules from the language definition. Predictions are for particular categories of phrases at particular input locations. 'Terminal' phrases contain a single word and are formed when words are acquired by acoustic tests. 'Nonterminal' phrases are formed when a language definition rule is applied to a set of constituent phrases. If there are no unfilled, unblocked category arcs in the phrase's structure graph, the phrase is called 'complete' (for a description of structure graphs, see Section D.2 in Chapter IV). Otherwise, more constituents can be added, so the phrase is called 'incomplete.' A complete phrase that is formed by adding missing constituents to an incomplete phrase P is called a 'completion' of P. Predictions are made as part of the

---

* The design of the parse net was inspired by Kaplan's multiprocessing consumer-producer approach (Kaplan, 1973b).

process of acquiring constituents to fill category arcs in incomplete, nonterminal phrases. An incomplete phrase is called 'empty' if none of its category arcs are filled. In this terminology, the parse net of predictions and phrases holds intermediate hypotheses and results while completions of empty, root-category phrases are constructed. Such complete root-category phrases with their attributes and factors are called 'interpretations' of the input.

Phrases and predictions have time specifications indicating their position in the input. By analogy with written text, beginning and ending times are referred to as left and right, respectively. The times for terminal phrases are provided as part of the output of the word recognition routines. The times for nonterminal phrases come from the leftmost and rightmost constituents, if those constituents have been acquired for the phrase. An incomplete nonterminal phrase that is missing its boundary constituents has its times either 'fixed' or 'unfixed'. Fixed times for a phrase P are either boundaries of the utterance or times from complete phrases that might be adjacent to completions of P. A fixed right time for a phrase P constrains possible rightmost constituents of P to end at or near the specified position. In contrast, an unfixed right time for a phrase means that there are no

constraints on the ending time of possible rightmost constituents. Similar statements hold for fixed or unfixed left times. Predictions also have fixed or unfixed times that determine which phrases fulfill them. For example, if a prediction has a fixed left time of 50, a phrase fulfilling the prediction must start at or near 50. (The details of how near is near enough are discussed later.)

Other information saved with each phrase includes an array of attributes. For terminal phrases, the attributes come from the lexical entry for the word or are computed by the category procedure. For nonterminal phrases, the rule procedure computes attributes of the phrase from constituent attributes. Terminal phrases have a pointer to the lexical entry that was used to construct them, and nonterminal phrases include a pointer to their composition rule, a list of constituent phrases, and a bit vector showing the blocked arcs in the structure graph.

Each prediction in the parse net is for phrases of a particular category that meet particular time requirements. Stored with the prediction record, in addition to the category and times, are the following lists:

* Instances -- Complete phrases of the predicted category that meet the time requirements. These phrases fulfill the prediction.

* Word sets -- Sets containing words from the predicted category which, if accepted by acoustic tests, can be used to construct terminal phrases fulfilling the prediction.

* Consumers -- Incomplete phrases that can have a phrase that fulfills the prediction added to them as a new constituent. Thus, the phrases on this list can 'consume' instances of the prediction.

* Producers -- Incomplete phrases whose completions could be instances of this prediction. In other words, these phrases can 'produce' instances for the prediction.

A prediction thus serves as an intermediary between two sets of incomplete phrases: consumer phrases that are all missing a constituent of the predicted category at the predicted location in the input, and producer phrases that all might supply the missing constituents. Note that a phrase can be a producer for one prediction and a consumer for another. Thus, 'producers' and 'consumers' are not names for distinct classes of phrases, but instead are names reflecting structural relations in the parse net. The full set of producer-consumer connections in the parse net make explicit the different sentential contexts for each phrase. This contextual information is used by the Executive in setting priorities and in lookahead. These operations and the operations that construct the parse net are sketched in the next section.

C. OVERVIEW OF THE EXECUTIVE

The Executive carries out a series of tasks adding predictions and phrases to the parse net. There are two main types of tasks: the predict task, which operates in a top-down manner, and the word task, which operates in a bottom-up manner.* This section sketches these tasks and briefly describes how the task priorities are established and how the series of tasks is started and stopped. The level of detail in the descriptions is minimal but adequate to provide the reader with the prerequisites for understanding both the discussion in Section E and the experiments reported in Chapter IV. For the reader who wants a detailed description of the Executive, this section provides an introduction that should make the details given later easier to understand.

1. PREDICT TASK AND WORK TASK

Figure III-1 shows the basic outline of the two main types of Executive tasks. The predict task takes incomplete phrases and adds a subnet of predictions and phrases to the parse net. The creation of predictions for categories with lexical entries causes the word task to be

---

* See Aho and Ullman (1972) for a discussion of top-down and bottom-up parsing strategies.

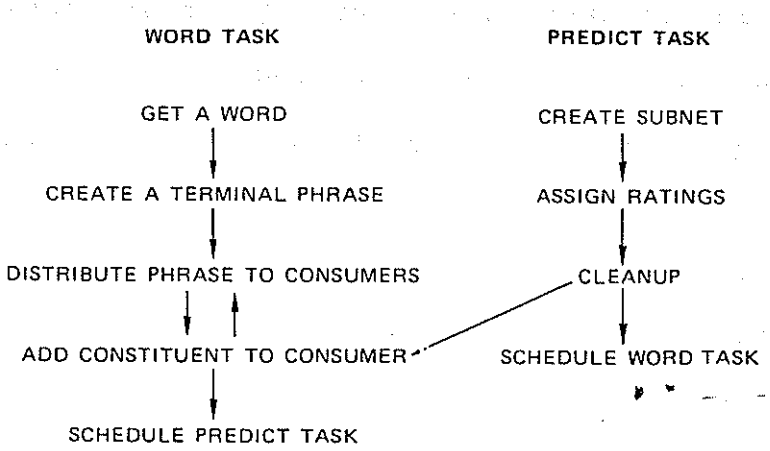

Figure III-1. EXECUTIVE TASKS

scheduled. Performing the word task entails getting an accepted word (one that has passed the acoustic tests), constructing a terminal phrase, and distributing it to consumers in the parse net. Adding the phrase to a consumer can result in a complete phrase P, in which case P is also distributed to consumers, or an incomplete phrase Q, in which case the predict task is scheduled to make predictions for constituents that can be added to Q. The link in Figure III-1 from the cleanup stage of the predict task to the add-constituent-to-consumer operation reflects the possibility of an old prediction with instances acquiring a new consumer.

Both tasks can work either left-to-right through an input or bidirectionally from words selected at arbitrary positions within an utterance. The system is designed to allow constituents of phrases to be added in any order, so experimentation with a variety of control strategies has been possible. Most importantly from the system-control standpoint, each task does a limited amount of processing and then stops after scheduling further operations for later. The scheduling does not specify a particular time for a future operation, but instead gives the operation a certain priority. The operation is performed when it becomes top priority. This organization allows the Executive to control the overall activity of the system by setting task priorities.

We have experimented with two versions of the Executive tasks that differ in where the acoustic tests are performed. In the first version, called 'mapping one at a time', a word is tested as the first step of the word task, and only if the word passes the test is a terminal phrase created. In this method, word priorities are primarily determined by consumer ratings (as described below). With the second method, called 'mapping all at once', all the word tests at a particular input location are performed before predictions are made at that location. The word

priorities are then influenced by the results of the acoustic tests in addition to the consumer ratings. When a word becomes top priority, the word task goes directly to the step of creating a terminal phrase. Mapping all at once causes the system to test more words per location but yields better priorities since experimental results indicate that true hits tend to get higher scores than false alarms. The choice between mapping one at a time or mapping all at once is explored in the experiments reported in the next chapter.

2. SETTING PRIORITIES

The fundamental data for priority setting are the word scores provided by the acoustic mapper and the factors computed by the language definition procedures. Mapper scores indicate how well a word matches the input signal at a particular location in the input. Language definition factors reflect acceptability judgments from syntactic, semantic, and discourse sources of knowledge. The 'score' for a phrase combines mapper scores, language factors, and, for a nonterminal phrase, the scores of its constituents. The score is thus a local, context-free piece of information about how 'good' the phrase is. The score may reflect global data such as a discourse model, but it does not depend on possible sentential contexts for the phrase. In

contrast, the 'rating' of a phrase does depend on the other phrases in which it may be embedded to form a sentence. The rating of a phrase P is intended to provide an estimate of the best score for an interpretation that can be constructed using a completion of P. If P is itself a root category phrase, its score determines its rating directly. Otherwise, the rating for P is determined by reference to the consumers for P in the parse net. (The organization of the Executive guarantees that all non-root-category phrases have at least one consumer.)

We have experimented with two techniques for using the consumer context in setting phrase ratings. In one, called the "merging" method, the rating with respect to a particular consumer is formed by adding the phrase score and the consumer rating. (Whenever possible, ratings are assigned top-down in the parse net so that consumer ratings are directly available for use in this process.) The phrase rating is then the maximum rating with respect to any of its consumers. This method is fast, but it leaves the rating unaffected by the consumer restrictions that are expressed in rule procedures rather than in structure declarations. A phrase may satisfy the structural requirements of a consumer C but still be incompatible with C because of constraints encoded in C's factor statements. For example, if the only

sentential context being considered is "Is it owned by --", the structural requirements will be satisfied by any noun phrase, but semantic factors will restrict the alternatives to possible owners.

The second technique for setting phrase ratings is called the "context checking" method. It takes into account the procedural information in the rules by exploring paths in the parse net and executing the corresponding procedures to gather attribute and factor information. Each producer-consumer path from a phrase P to a root category phrase reflects a way of constructing an interpretation using P. Various paths from P are formed, and the rating for P is its best rating with respect to any of the constructed paths.

To reduce the cost of rating alternatives by the context checking method, a heuristic search is made in the parse net for a near optimal path rather than exhaustively trying all possibilities. The heuristic exploits the fact that, typically, when a phrase is being rated, the higher level phrases that form its context have already been rated. (The parse net is initialized so that a context of previously rated phrases exists even when the system is doing bottom-up processing.) These prior ratings provide important heuristic information. The object is to find the path giving the best score, so the paths with the highest

prior rating are explored first. When a complete path is found, one that leads to a root-category phrase, the score for that path is used as a threshold to prune other paths that have lower expected ratings.

The context checking method takes more computation per rating assignment than the first one, but it produces better phrase ratings since it gathers more information in forming them. Experimental results reported in Chapter IV indicate that the extra effort spent in this method is worthwhile; it leads to better system performance in both accuracy and runtime.

Phrase ratings are used to determine task priorities. The priority of the predict task comes from the highest rating of any phrase scheduled to make predictions. When the predict task is executed, it creates predictions and phrases only at the time and direction (left or right) that are determined by the best phrase scheduled to make predictions. Similarly, the priority of the word task is equal to the highest rating for any predicted word (the word rating is the rating of the terminal phrase that could be constructed from the word). When the word task is performed, it only operates on the highest rated word.

In the case where task priorities are directly determined by ratings, the control strategy is described as 'best-first'. We have experimented with modifying priorities to implement other control strategies in addition to best-first. In particular, we have tried a method we call 'focus by inhibition' in which high scoring words are selected from the best phrases for the predict-task, and tasks that cannot use those words are inhibited by having their priorities lowered. This inhibition is relative rather than absolute; it "discourages" the system from performing certain tasks but does not forbid them.

The selected words are the focus of attention for the system in this method and are described as 'the focus' or as 'in focus'. A phrase conflicts with the focus if it contains a nonfocus word that overlaps some focus word. The tasks that would try to complete such phrases have their priorities lowered. The priority reduction causes the system to be biased against working to complete phrases that conflict with the focus. If a task for a phrase P that is in conflict with the focus manages to overcome the system bias against it to become the task with the highest priority, the system shifts to a new focus by removing the words from focus that conflict with P and adding new words to focus from P.

The technique of focus by inhibition is motivated by a desire to reduce the rapid switching among closely rated alternatives that can happen with a best-first strategy. Switching among incompatible alternatives is reduced with focus by inhibition because the best phrase inhibits its competition and thus keeps the system's attention focused on fulfilling its predictions. The inhibition is a relatively small decrease in priority, so the bias can be overcome. Therefore, focus by inhibition does allow the system to recover from selecting an incorrect word for focus. However, if the system focuses on incorrect words too often, the net effect of the priority changes can be harmful rather than helpful. Experimental results showed that selecting incorrect words was in fact a serious problem for focus by inhibition, and, as a result, overall performance was not improved by this technique. Although this particular attempt to improve performance by adjusting priorities did not succeed, the basic approach still merits further study. As a method for adjusting priorities, it provides simple answers to how, when, and why to focus attention, while still maintaining the completeness of the control strategy. (It does not discard alternatives, it simply revises their priorities.) Better success at selecting hits rather than false alarms for focus could result in a focus by inhibition that improved performance.

3. STARTING THE TASK CYCLE

The Executive starts processing an utterance with an initial parse net already in existence. If the system is using a left-to-right control strategy, the initial net contains: (1) for each root category rule, an empty phrase with times fixed at the beginning and end of the utterance, and (2) for each category that can occur at the left of an input, a prediction with its associated empty producers, all with their left times fixed at the beginning of the utterance and their right times unfixed. Each phrase P in the initial parse net is connected as a consumer to the predictions for categories of phrases that can occur in P as leftmost immediate constituents. The Executive task cycle starts by scheduling the word task for the predictions in the initial net. This task will find a word at the start of the utterance and the interplay of word task and predict task will start.

As an alternative to left-to-right processing, the system can use 'island driving' in which phrases are constructed bidirectionally around 'island' words selected anywhere in the input.* The motivation for island driving

---

* Island-driving is derived from Miller's 'locally organized parsing' based on 'islands of reliability' (see Miller, 1973). For examples of its use in speech understanding systems, see Ritea, 1974 and Bates, 1975.

is that it allows the system to begin processing an utterance where it is most confident that it has found a correct word. It can use that word to provide contextual guidance in processing other parts of the utterance where it is less confident. In contrast, left-to-right processing must start at the beginning of the utterance even if the system is not confident about any of the words there.

For island driving, the initial parse net contains: empty root category phrases with times fixed at the beginning and end of the utterance, and, for each category in the language, a 'monitor' (which is a special kind of prediction) with its associated empty producers, all with both times unfixed. Each phrase P in the initial parse net is connected as a consumer to the monitors for categories of phrases that can be added as immediate constituents of P.

The task cycle starts with the selection of an island word according to a criterion combining the word's mapper score and its estimated likelihood of being a false alarm. The word task is performed for this island word to create a terminal phrase. The terminal phrase is then passed to the distribute-phrase procedure, which is modified for island driving so that if the phrase being distributed does not fulfill any predictions, it is given to the

consumers for the monitor of its category. In general, this operation can construct incomplete phrases that lead to predictions on either side of the island word. After the first island word is distributed, the Executive schedules a task to select a second island word in case the first one fails to lead to highly rated phrases. If this task is performed and starts a second island, it will reschedule itself to try a third in case the second runs into trouble. In this manner, a number of islands can be worked on simultaneously. The effect of island driving on system performance is a topic of Chapter IV, Section D.

4. STOPPING THE TASK CYCLE

After each execution of a task, the Executive checks several parameters to see if it should stop the task cycle. For instance, the Executive keeps track of the amount of storage in use and stops before the available storage is exhausted. Another stopping criterion is the difference between the priority of the best remaining task and the processing time already used for the utterance. The Executive stops if the value of this criterion falls below a certain threshold. The threshold is initialized to a low value, but whenever an interpretation is constructed, the threshold can be raised so that the system will not spend

much more time looking for other interpretations unless the priorities are high. When the Executive decides to stop, it calls the language RESPONSEFN function. This function is also called whenever an interpretation is constructed, and it stores the interpretations and manipulates the priority-minus-processing-time threshold. When the Executive tells the RESPONSEFN that it is time to stop, the function initiates question answering using the highest rated interpretation it has.

This concludes the overview of the Executive. The reader has an option at this point of skipping ahead to the discussion section at the end of this chapter or to Chapter IV (which deals with a series of experiments concerning system performance and the Executive) before going on to the following detailed description of the Executive.

D. DETAILS OF THE EXECUTIVE

The topics covered in this section are the word task, adding a constituent to a consumer, the predict task, multiword lexical entries, and priority setting. To make the following descriptions complete, there is some repetition of information covered in the overview.

1. WORD TASK

The major operations in the word task are to acquire a word that is accepted by the acoustic tests, create a terminal phrase for it, distribute the phrase to consumers, and schedule the predict task for any incomplete phrases that result. Predicted words are organized into 'word sets'. Each word set has a list of words from some lexical subcategory, time specifications like those for a prediction, and priority information. Word sets are typically created during the final step of a predict task, but they are also created at the start of a left-to-right parse.

Like a prediction, a word set has one fixed time and one unfixed time.* The creation of a word set begins by finding the subset of the lexicon that is worth considering at the fixed time. If the system is using the mapping-all-at-once control strategy, this subset contains the words actually accepted by the mapper at or near the time. Otherwise, the subset is created by a special acoustic process called lexical subsetting, which looks at local acoustic features to create a set of words that merit testing by the mapper. In either case, the lexical subset

---

* Word sets for the root-category are exceptional in that both of their times are fixed. The algorithms take care of these as special cases.

is intersected with the set of words in the predicted lexical subcategory to form the entries in the word set.

The word set is then assigned a priority and added to the list of word sets for use in the word task. The word set priority reflects the expected rating of terminal phrases constructed from words in the set. A single rating is computed for the entire set of words. If the system is mapping all at once, the priority of the word set is strongly influenced by the best mapper score for a word in the set. Otherwise, the priority is affected by the estimated false alarm likelihoods for words in the set, so that, other things being equal, the system will try words in an order that is expected to minimize false alarms. The priority of the word task is the highest priority of any word set.

a. GETTING A WORD AND CREATING A TERMINAL PHRASE

When it is performed, the word task begins by selecting a word from the highest priority word set. The selected word is the one with the highest mapper score, if the system is mapping all at once, or the one with the lowest false alarm likelihood, if the system is mapping one at a time. If the system is not using the map-all strategy, the chosen word is now tested by the mapper. If it is

rejected, the word task goes directly to its final stage. In that stage, the priority for further word tests is compared to the priority for other system tasks. If word testing is still the highest priority, the word task is directly reexecuted. Otherwise, it returns control to the top-level Executive procedure.

The next operation in the word task is to create a terminal phrase (see Figure III-2). This operation begins by checking if a terminal phrase for the same word* in the same input location has already been created. For instance, the word might have been accepted as the result of a prediction from the opposite direction (right-to-left instead of left-to-right, say), or it might have been found following a prediction with a slightly different fixed time. If such a terminal phrase exists, the word task does not create a duplicate, but instead, it simply goes to its final stage. Otherwise, the category procedure is called to compute attributes and factors for the word. If the category procedure does not reject the word, and the resulting phrase score is above a certain threshold, the terminal phrase record is constructed. The record holds the word's lexical entry, the times given by

---

* In this discussion, a 'word' is a lexical entry, so if there is a word in category X that happens to have the same spelling or pronunciation as another word in category Y, they are still different words.

CREATE TERMINAL PHRASE

If there is a terminal phrase for the same word and place, quit.

Otherwise, call the category procedure.

Construct the phrase record.

Distribute it to consumers.

Figure III-2.

the mapper, the phrase score, and other information. The phrase is now ready for distribution to consumers.

b. DISTRIBUTING A PHRASE TO CONSUMERS

The procedure for distributing a phrase to consumers is the same for terminal and nonterminal phrases (see Figure III-3). It is called from the procedure that creates terminal phrases and from the procedure that adds a constituent to a consumer to create a complete nonterminal phrase. If the category of the new phrase is the root category of the language, the phrase is passed to the language RESPONSEFN. This function saves the phrase for possible use in question answering and adjusts the Executive stopping parameters. If the category of the new phrase is not the root category, all predictions for the category are checked to see if the new phrase satisfies their time

DISTRIBUTE A PHRASE TO CONSUMERS

If it is a root-category phrase, give it to the RESPONSEFN.

For each prediction fulfilled by the phrase,

Record the phrase as an instance of the prediction, and

Add the phrase to each consumer of the prediction.

Figure III-3.

constraints. For example, if the phrase is an NP starting at location 35 and ending at location 55, it satisfies an NP prediction with left time 35 and right unfixed, but it does not satisfy an NP prediction with left unfixed and right at 180.

The actual algorithm for checking times takes two times as its input and decides whether or not they are compatible. If a time is unfixed, it is compatible with any other time. Two fixed times are compatible if the gap between them is not too large. In the simulation experiments described in the next chapter, the allowed gap was given by a parameter -- the standard size was 0.05 seconds, but this was varied in one of the experiments. With real rather than simulated acoustic processing, syllable boundary information is used in the time check -- the gap between two times must not contain an entire syllable. This test eliminates the obviously bad cases and

leaves the more difficult ones to be handled by phrase mapping. Phrase mapping looks at a pair of words that have been accepted individually to see if they are acceptable as a sequence. Phrase mapping is done in the add-constituent operation when phrases are put together to form larger phrases (by a procedure discussed below).

For each prediction that the new phrase fulfills, the phrase is added to the prediction's instances list and then given to the prediction's consumers. The instances list is maintained so that consumers arriving later can make use of previously constructed instances. The operation of giving the phrase to a consumer is the source of nonterminal phrases that have one or more constituents.

2. ADDING A CONSTITUENT TO A CONSUMER

The add-constituent procedure (Figure III-4) performs preliminary tests to ensure that the phrase and its consumer are compatible with respect to times, phrase mapping, and lookahead. The time checks in the distribute-phrase procedure described above ensure that the phrase satisfies the times of the consumer's prediction; the time checks for the add-constituent procedure are more detailed and take into account other constituents of the consumer. If the preliminary tests succeed, the procedure goes on to try creating both complete and incomplete phrases.

ADD CONSTITUENT TO CONSUMER

If the preliminary tests fail, quit.

Try to create a complete phrase,
and distribute it if successful.

Try to create an incomplete phrase, and if successful then
assign its rating and schedule the predict task.

Figure III-4.

a. PRELIMINARY TESTS

The preliminary tests in the add-constituent procedure stop it from trying to add a constituent to a consumer that is incompatible with respect to times, phrase mapping, or lookahead. Even if the constituent and the consumer are actually compatible, the tests still provide useful information by blocking arcs in the structure graph to reduce the number of possibilities that must be considered in later operations.

The first tests concern the time constraints on the new constituent imposed by the time specifications of the consumer and its old constituents (see Figure III-5). The tests to the left of the new constituent will be described; similar tests are performed to the right. Let the index number for the constituent category arc be I. If

PART 1 OF PRELIMINARY ADD-CONSTITUENT TESTS

If find a neighboring constituent to the left,

(1) quit if the neighbor is not to the left in the input,

(2) if the neighbor must be adjacent,
phrase map and quit if the test fails,

else if the neighbor optionally can be adjacent,
phrase map and
block the NIL arc if the test fails.

Otherwise, if the consumer phrase left time is fixed and the prediction left time is unfixed, then

if the new constituent must be leftmost,
check the left times and quit if the test fails

else if the new constituent can be leftmost,
check the left times and
block the NIL arc if the test fails.

Do the same tests to the right of the new constituent.

Figure III-5.

the new constituent has a left neighbor constituent in the consumer, the neighbor can be found by scanning through the consumer's constituent-array, starting at position I-1 and going down to position 1 looking for the first nonNIL entry (recall that entries are ordered from left to right in increasing positions of the array). If there is a neighbor, the following tests are made to ensure that it is time compatible with the new constituent. As an initial check, the left neighbor must really be somewhat to the left of the new constituent. If neither of the left or right times of

the new constituent is to the right of the corresponding time of the left neighbor, the preliminary tests fail.

The next test depends on the structure graph relation between the new constituent and its left neighbor. If the consumer's structure graph indicates that the two phrases must be immediately adjacent (i.e., the right point of the left arc is the left point of the right arc), the rightmost word in the left phrase and the leftmost word in the right phrase are passed to the phrase mapping procedure. The job of the phrase mapper is to deal with coarticulation effects at word junctions, and if it rejects the pair of words, the add-constituent procedure terminates without adding the phrase to the consumer. Alternatively, the structure graph for the consumer may indicate that the phrases do not have to be adjacent, but that they can optionally be adjacent if an unblocked NIL arc is used. In this case, phrase mapping is performed, and if it fails, the NIL arc is blocked to record that the phrases cannot in fact be adjacent. The last alternative is that the structure graph does not allow the neighbors to be immediately adjacent. No phrase mapping is done in this case.

If there is not a left neighbor for the new constituent, but the consumer's left time is fixed, a time check may be made like the time checks to see if a phrase

satisfies a prediction. However, if the consumer and the constituent were brought together by a prediction with a fixed left time, a time check is not necessary here since it would duplicate the check made when the phrase was added as an instance for the prediction. In case the prediction's left time is not fixed, but the consumer's left time is (which can happen if the consumer is a root category phrase), and the new phrase can be the leftmost constituent, a time check is made with the constituent's left time and the consumer's left time. If the test fails and the constituent must be leftmost, the add-constituent procedure terminates. If it can be leftmost by the use of an unblocked NIL arc and the time test fails, the NIL arc is blocked. These tests, and similar ones regarding the right side of the new constituent, ensure the acceptability of the times and word junctions between the new constituent and its consumer.

The second group of preliminary add-constituent tests look at the lexical subsets adjacent to the new constituent to block various arcs in the consumer (see Figure III-6). The lexical subsets are determined by acoustic tests and indicate the words that may be found in the utterance at the specified location and direction. For example, if the constituent starts at location 70 and

PART 2 OF PRELIMINARY ADD-CONSTITUENT TESTS

For each arc coming in to the left of the new constituent,
block the arc if it is inconsistent with the lookahead.

If all arcs to left are blocked, quit.

Do similar tests to the right of the new constituent.

Block any arcs that can no longer be in a complete path.

Figure III-6.

ends at 105, the subset to its left will contain words that can end around 70, and the subset to its right will contain words that can start around 105. Each category in the language has a precomputed list of possible leftmost and rightmost terminal phrase categories. These lists are used to block arcs that are inconsistent with the lookahead provided by the lexical subsets.

b. CREATE COMPLETE NONTERMINAL PHRASE

The procedure for constructing a complete nonterminal phrase (Figure III-7) starts with a test for a complete, filled path through the structure graph. If filled paths do not exist both from the right of the new constituent to the rightmost point in the graph and from its left to the leftmost point, the complete-phrase procedure terminates. The next test ensures that the new complete

COMPLETE-PHRASE PROCEDURE

Check for a complete, filled path in the structure graph,
    and quit if there is none.

Check consumers regarding lookahead next to the new
    constituent, and suspend construction
    if all consumers are blocked.

Check if the same phrase already exists,
    and quit if it does.

Call the rule procedure, and
    if it rejects the phrase or gives it a
        subthreshold score, quit.

Create a phrase record.

Distribute the new phrase to consumers.

Figure III-7.

phrase will be compatible with its consumer context with respect to time constraints and lookahead. The details of this test are discussed below. If the test fails, construction of the new phrase is suspended pending the arrival of a new, compatible consumer. Assuming the test succeeds, the procedure checks whether it has already constructed a complete phrase using the same rule and constituents. If so, it terminates. The same phrase can be arrived at in different ways when the system uses control strategies that do not fix the order of acquisition of constituents. For example, with island driving, a phrase could be built both from the left and from the right. This

test is necessary, therefore, to make sure that duplicate phrases are not constructed.

The rule procedure is then executed to produce values for the attributes and factors, and the results are saved for future equivalence tests. If the rule procedure rejects the new phrase, or the phrase score is below a certain threshold, the complete-phrase procedure terminates. Otherwise, a record is constructed holding a pointer to the rule, the constituent list, the array of attributes, the score, and other information about the phrase. The newly made phrase is then distributed to consumers by the procedure described earlier.

A major step in the complete-phrase procedure is the consumer-lookahead test. This test is similar to the lookahead tests performed in the preliminary add-constituent stage, but it considers lookahead with respect to the consumer context of the new phrase being constructed rather than within the consumer that is getting a new constituent. The purpose of this check is to prevent construction of complete phrases that cannot be used in the existing context of consumers. Building a complete phrase can require expensive semantic and discourse operations, so the system tries to discover an incompatibility before the phrase is constructed.

The consumer-lookahead test looks both to the left and to the right of the new phrase, but the tests are similar, so only the right side tests are described in detail. The first step is to locate the arc in the consumer's structure graph that the new phrase would fill. (Remember that the new phrase we are talking about is the phrase being constructed by the complete-phrase procedure, and the consumer now being discussed is a potential consumer of this new complete phrase.) The procedure looks to the right of the arc for either: (1) an unfilled, unblocked category arc that is not eliminated by lookahead, (2) a filled category arc with a constituent whose left time is compatible with the right time of the new phrase, or (3) an unblocked NIL arc that leads to a category arc satisfying case (1) or case (2).* In any of the three cases, the phrase and the consumer are compatible to the right.

If none of the cases succeeds, the consumer C may still be compatible with the phrase P if P can be C's rightmost constituent and the phrase resulting from adding P to C would be all right. P can be rightmost in C if its arc ends at the rightmost point in C's graph or if an unblocked

---

* We could have added phrase mapping in case (2) as a further test of consumer compatibility, but because phrase mapping is a relatively expensive operation in our system, we decided that the extra sensitivity at this point would not justify the cost.

NIL arc connects it to the rightmost point. If P can be rightmost, and the right boundary of C is fixed and compatible with the right boundary of P, P and C are compatible to the right. If P can be rightmost and the right boundary of C is unfixed, the consumer-lookahead procedure is called recursively to check consumers of C. For example, if C is of category X, the procedure checks consumers of C, which may have phrase structures such as W=X Y. In this particular case, the lookahead to the right of P would have to be compatible with a category Y phrase for the consumer-lookahead test to succeed.

The consumer-lookahead procedure can thus go through an arbitrarily long path of intermediate consumers before getting to one that satisfies the lookahead requirements. Because of the structure of the parse net, these paths of consumers can form a network rather than a tree. To deal with this convergence, the procedure adds each prediction to a queue when the prediction's consumers are first reached. If the search returns to a prediction that is already on the queue, the consumers for that prediction are not checked again.

c. CREATE PARTIALLY FILLED NONTERMINAL PHRASE

If the preliminary add-constituent tests indicate that there will be at least one unblocked, unfilled category arc remaining after the constituent is added to the consumer, the procedure to create a partially filled nonterminal phrase is called (see Figure III-8). The procedure first checks if it has already made an incomplete phrase for the same combination of rule, constituents, and time specifications. If it has, it stops rather than creating a duplicate. Otherwise, it calls the rule procedure to compute the attribute and factor values.* If the rule rejects the phrase, or the score is subthreshold, the incomplete-phrase procedure quits.

Next, a phrase record is constructed which has a pointer to the rule, the constituent list, the attribute array, the score, and other information. If the consumer that was used in this operation is a producer for some prediction (as will be the case unless it is a root-category consumer), the new phrase is also added as a producer for that prediction. The new phrase is then

---

* We do not bother here to look for equivalent phrases as we did in the complete-phrase procedure because semantic translation and discourse processing are only done for complete phrases, and consequently the cost of recalculating the attributes and factors of an incomplete phrase is much less than the cost for a complete phrase.

CREATE AN INCOMPLETE PHRASE

Check for an unfilled, unblocked category arc;
if none, quit.

Check if a phrase for the same rule, constituents, and time specifications already exists; if so, quit.

Call the rule procedure.

If it rejects the rule or gives a subthreshold score, quit.

Create a phrase record.

Calculate the phrase rating.

If the rating is above threshold, add the new phrase to the predict sets.

Figure III-8.

assigned a rating using the techniques described in Sections C.2 and D.5.c. If the rating is above a certain threshold, the phrase is added to the phrases scheduled to make predictions for missing constituents.

The incomplete phrases scheduled to make predictions are organized into 'predict sets'. Each predict set contains incomplete phrases that can make predictions at a particular time in a particular direction. To illustrate, if a phrase has a constituent ending at position 75 and needs to acquire a constituent to the right of that one, the phrase could be in a predict set for time equal to 75 and direction equal to RIGHT. The priority of a predict set is

initially set to the highest rating of any phrase in the set, but it can be modified in accordance with various strategies for focusing the system's attention. The priority of the predict task is the highest priority of any predict set.

A single incomplete phrase can be in several predict sets if it can make predictions at different locations and directions. For example, if the phrase structure calls for three constituents, A B C, and only the B phrase has been acquired, the phrase can be in a predict set for time equal to the left time of B and direction equal LEFT (to predict A), and also in a predict set for time equal to the right of B and direction equal RIGHT (to predict C).

The algorithm for adding an incomplete phrase to predict sets looks for all possible predictions that the phrase can make such that either the left or right time of the prediction is fixed.* For the purposes of this algorithm, a point in the structure graph is called 'open for predictions to the right' if there is an unfilled,

---

* We decided against having predictions with both times fixed. This choice sacrifices the ability to make word tests with both times fixed (such tests might succeed where tests with only one time fixed would fail), but it simplifies the word task and the predict task as well as reducing storage requirements.

unblocked category arc going out from the point to the right, or there is an unblocked NIL arc going out from the point that leads to an unfilled, unblocked category arc. First, if the left time of the phrase is fixed and the leftmost point in the graph is open for predictions to the right, the phrase is added to the predict set for time equal to the left of the phrase and direction, RIGHT. Then, for each acquired constituent, if the right point of the constituent arc is open for predictions to the right, the phrase is added to the predict set for time equal to the right of the constituent and direction, RIGHT. If the Executive is using a control strategy such as island driving that allows right-to-left predictions, a similar set of operations is performed to add the phrase to predict sets for direction equal LEFT.

3. PREDICT TASK

The predict task is divided into three main stages. In the first, a subnet of predictions and phrases is created for the highest priority predict set. The entire subnet is filled out at once: the predictions for the phrases in the predict set, the producer phrases for those predictions, the predictions for the new phrases, and so on. The second stage consists of going through the subnet

assigning phrase ratings: whenever possible, all the consumers for a phrase are given ratings before the phrase rating is calculated. During the final stage, miscellaneous 'cleanup' operations are performed such as creating word sets for new predictions, revising priorities for old word sets, and adding old phrases to new consumers.

The predict task is organized in this manner to reduce the processing time for rating phrases and setting priorities. The rating for a phrase is recalculated when its consumer context changes, and the rating calculation procedure uses the ratings of consumers, so the predict task is designed to provide all of the consumers and finish giving them ratings before calculating the rating for a phrase.

a. CREATE SUBNET

The create-subnet procedure, which performs the first stage of the predict task, is outlined in Figure III-9. While it creates the subnet, the procedure also sets up several queues for use in later stages. For the rating stage, it creates a queue, called PRQ, holding the predictions in the subnet that have up-to-date ratings for all of their consumers, and a second queue, called WPRQ, holding the other subnet predictions. For the cleanup

```
CREATE SUBNET PROCEDURE

Select the best predict set.

Get the lookahead information.

For each phrase P in the predict set

     For each prediction PR made by P

          If PR is newly created.
               Add it to PRQ and fill out its subnet

          Otherwise

               If it is not on PRQ or WPRQ, add it to PRQ

               Traverse its subnet

               If it has instances,
                    Add P as consumer for it to INSTLIST.
```

Figure III-9.

stage, it creates LEXQ with the new predictions that need to have word sets made for them, LEXQ2 with old predictions that need to have the priorities for their word sets revised, INSTLIST holding new consumers for old predictions that have nonempty instances-lists, and OTHERINSTLIST holding consumers that provide a new context for phrases suspended in the complete-phrase procedure because of consumer-lookahead failure.

The create-subnet stage of the predict task begins by removing the highest priority entry from the list of predict sets. This entry will determine the predictions

to be made in the rest of the task. The time and direction from the selected predict set are passed to the lexical subsetting procedure to retrieve the lookahead set of terminal categories. (Often, the subset will have already been computed for lookahead during an earlier operation, so the stored results can simply be reused.) This information is used to block the creation of predictions and phrases that are incompatible with the lookahead.

Next, the subnet is filled out for each phrase in the predict set.* As an illustration of this process, assume the predict set time is 55 and the direction is RIGHT. Then, for each predict set phrase P, the procedure finds the structure graph point at the right of the rightmost constituent that has a right time of 55, if such a constituent exists. Otherwise, P must have its left time fixed at 55, and the leftmost point in the graph is used in the following operation. For each unblocked, unfilled category arc A leading out of the selected point or connected to it by an unblocked NIL arc, find or create a prediction PR for the category of arc A, left time fixed at 55, and right time unfixed. If PR was newly created for P, the subnet below PR is filled out by a procedure described

---

* We do all of the phrases in the predict set at once rather than just doing the highest priority phrase. This is because we want to decrease the recalculation of priorities that is caused by changes in consumer contexts.

below, and PR is added to PRQ to record that all of its consumers have up-to-date ratings. Alternatively, if PR existed previously, then (1) it is put on PRQ unless it is already on a queue, (2) its subnet is traversed by a procedure described below, and (3) if PR has instances, P as a consumer for PR is added to INSTLIST.

The procedure used to fill in the subnet below a prediction (Figure III-10) begins by checking if the prediction is for a category with lexical entries. If it is, the prediction is added to LEXQ for further processing during the cleanup stage. The next step is to put a mark on the prediction so that if a later consumer arrives during this stage of the predict task, the subnet will not be reprocessed. The mark will be removed before the predict task is over.

Then, in the case of a left-to-right prediction, for each rule that can produce a phrase of the predicted category, if the possible leftmost terminal categories for the rule intersect the lookahead categories, an empty phrase P' is created for the rule with the same times as the prediction. The phrase is connected to the prediction as a producer, and then predictions are made in the following manner for the possible leftmost constituents of the phrase. For each category arc that can be leftmost

FILL-OUT-SUBNET PROCEDURE

To fill out subnet for prediction PR with new consumer P:

If the category of PR has lexical entries, add PR to LEXQ.

Mark PR.

For each of the category rules

    Create an empty phrase P' as a producer for PR

    For each prediction PR' made by P'

        Add PR' to WPRQ and, if necessary,
            remove it from PRQ.

        If PR' is newly created, fill out its subnet

        Else traverse its subnet, and

            If PR' has instances, add P'
                (as a consumer for PR') to INSTLIST.

Figure III-10.

and is okay with respect to the lookahead, a prediction PR' is found or created, and the phrase is added as a consumer for it. The prediction is added to WPRQ and removed from PRQ if it was there. This operation records the fact that the prediction now has at least one consumer whose rating is not set. If PR' was created by this operation, the subnet below it is filled in by calling this procedure recursively. Otherwise, the subnet below the prediction is traversed according to the procedure described in the next paragraph. If it has instances, P' as a consumer for it is added to

INSTLIST. Similar operations are performed to fill in right-to-left predictions.

The procedure for traversing the subnet below a preexisting prediction (Figure III-11) starts by checking for suspended construct-complete-phrase operations. If there are any, the consumer that led to the current traversal is added to OTHERINSTLIST for special processing during the cleanup stage. The procedure then checks to see if the prediction is marked as already traversed. If so, the procedure returns. If not, it marks the prediction and continues. The mark is checked after the (possible) addition to INSTLIST rather than before in order to take care of cases in which more than one consumer changes for an old prediction. If the prediction has word sets, it is added to LEXQ2 so that the word set priorities will be revised to reflect their new consumer context. Finally, for each producer phrase P for this prediction, each of the predictions made by P is put on WPRQ if it is not already there and traversed by calling this procedure recursively.

At the completion of the first stage of the predict task, the subnet of predictions and phrases has been created for all the phrases in the highest priority predict set. The queues PRQ and WPRQ have been created for the rating stage, the queues LEXQ and LEXQ2 are ready for use in

```
TRAVERSE SUBNET PROCEDURE

To traverse the subnet for a prediction PR
      with a new consumer P:

If PR has suspended construct-complete-phrase operations,
    add P (as a consumer for PR) to OTHERINSTLIST.

If PR is marked, quit.

Mark PR.

If PR has word sets, add it to LEXQ2.

For each prediction PR' by a producer for PR

     Traverse subnet of PR' and put PR' on WPRQ.
```

Figure III-11.

creating or revising word sets, and INSTLIST and OTHERINSTLIST hold consumers that may lead to the creation of new phrases.

To illustrate the create-subnet procedure, assume that the highest priority predict set has a time equal to the right boundary of the utterance, a direction equal to LEFT, and a single phrase P1 which has a structure declaration S=BE NP (see Figure III-12). The only prediction to be made by P1 is for an NP ending at the right of the utterance. Assume that this prediction has not been created by a previous predict task (if it had been, the traverse-subnet procedure would be called rather than the fill-out-subnet procedure). The prediction PR1 for the NP

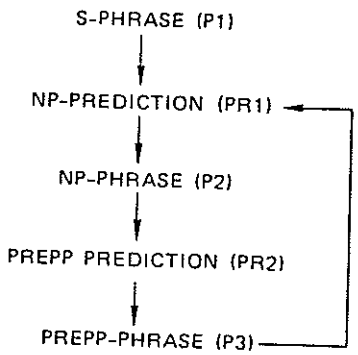

Figure III-12. NP-PREPP PARSE NET LOOP

is created, added to PRQ, and passed to the fill-out-subnet procedure. The NP category has lexical entries such as "it", so PR1 is added to LEXQ for further processing in the cleanup stage. The prediction is then marked so that it will not be reprocessed. An empty NP phrase P2 is created as a producer for PR1, and it makes a prediction PR2 for a PREPP at the end of the utterance and adds PR2 to WPRQ. (Other predictions would be made by P2, but for simplicity we only consider PR2.)

Again assuming PR2 did not exist before, it is passed recursively to the fill-out-subnet procedure. The PREPP category does not have lexical entries, so PR2 is not added to LEXQ. PR2 is marked, and a producer P3 is created for it. P3 makes a prediction for a noun phrase at the end of the utterance, but that is prediction PR1, which was

already created. Therefore, PR1 is moved to WPRQ from PRQ to record that it has a consumer without an up-to-date rating. Then, PR1 is passed to the traverse subnet procedure which quits after noticing that PR1 is already marked. Assuming for simplicity that no other predictions or phrases are formed, the first stage ends with LEXQ holding PR1, WPRQ holding PR1 and PR2, and the other queues empty.

b. ASSIGN RATINGS

The second stage of the predict task takes care of assigning ratings to phrases in the subnet. The algorithm for calculating the rating of a particular phrase is sketched in Section C.2 above and is described in detail in section D.5.c below. This section deals with the procedure that goes through the subnet visiting the phrases and calling the rating procedure for each one.

In the simplest case, a single top-down pass is made through the subnet, but the operation can be more complicated if there are producer-consumer loops. A top-down pass is desirable because the calculation of ratings takes advantage of the ratings for the consumers for the phrase. The predictions placed on PRQ during the first stage have up-to-date ratings for their consumers, so the

producers for those predictions are ready to have their ratings calculated. When a phrase gets its rating, each of its predictions is checked, and, if all of the prediction's consumers now have up-to-date ratings, the prediction is moved to PRQ from WPRQ.

If there are no loops in the subnet, repeated removal of predictions from PRQ followed by processing of their producers in the above manner, will eventually provide ratings for all the subnet phrases in the desired top-down way. If there is a loop in the net, it is not possible to assign ratings in a strictly top-down manner since phrases in the loop are consumers for themselves. When there is a loop, the algorithm goes as top-down as possible and makes a second pass to recalculate ratings in certain cases. The following paragraphs describe the algorithm in detail and illustrate it by continuing with the example from the previous section.

The first pass of the rating stage is an iterative operation that continues until both PRQ and WPRQ are empty (see Figure III-13). When both queues are empty at the start of an iteration, the second pass begins. If PRQ is not empty, the first prediction on it is removed and set aside for processing. If PRQ is empty, but WPRQ is not, the subnet has a loop that includes the WPRQ predictions.

```
PASS 1 OF RATING ASSIGNMENT

If PRQ and WPRQ are both empty, go to pass 2.

Pick a prediction from PRQ, or,
     If PRQ is empty, pick one from WPRQ that has at
   least one rated consumer.

For each producer P of the selected prediction

     Assign a rating to P.

     For each prediction PR made by P

          If PR is on WPRQ, then,
               If all consumers of PR now have ratings,
                    Move PR to PRQ

          Else if the rating for P is higher than the
                    rating of any other PR consumer
                    Then add PR to PASS2Q
```

Figure III-13.

However, at least one of the WPRQ predictions must have one or more consumers that are not involved in the loop. If this were not the case, the predictions on WPRQ would not be reachable from the rest of the parse net, but they are in fact reachable from at least one of the phrases in the current predict set. A prediction on WPRQ with at least one rated consumer is removed and set aside for processing.

The processing of the selected prediction is the same whether the prediction is from PRQ or WPRQ. Each producer phrase P for it is given a rating, and then for each prediction PR by P, (1) if PR is on WPRQ, then if all

the consumers of PR now have up-to-date ratings, PR is moved to PRQ, else (2) PR is a prediction from a producer-consumer loop, and if the rating just calculated for its 'looping' consumer is higher than the ratings of all of its other consumers, PR is added to PASS2Q for further processing in the second pass. At this point, control goes back to the start of the first pass instructions to check again whether PRQ and WPRQ are empty.

The second pass is an iterative operation that continues until PASS2Q is empty (see Figure III-14). If PASS2Q is not empty, a prediction is removed from it and marked so that it will be recognized if the second pass returns to it. Each producer phrase for the prediction has its rating recalculated, and, if its rating changes, each prediction made by the phrase is added to PASS2Q if the prediction is not marked and the other consumers for P have lower ratings than the rerated consumer. When the second pass terminates, all the phrases in the subnet have up-to-date priorities.

To illustrate this stage of the predict task, we continue the previous example (see Figure III-12). Recall that the first stage for the example ended with PRQ empty, and PR1 and PR2 on WPRQ. At the start of the second stage, PR1 is selected from WPRQ since it has a rated

```
PASS 2 OF RATING ASSIGNMENT

If PASS2Q is empty, quit.
Pick a prediction from PASS2Q.
Mark it.
For each producer P of the selected prediction
     Recalculate the rating for P.
     If the rating changed.
          For each prediction PR by P not marked
               If P is the highest rated consumer for PR,
                    Add PR to PASS2Q.
```

Figure III-14.

consumer (P1) and PR2 does not. PR1's producer P2 has its rating assigned. The only prediction for P2 is PR2. PR2 is on WPRQ and now has all of its consumers rated, so it is moved to PRQ. Thus, the first iteration ends with PR2 on PRQ, P2 with a rating, and WPRQ empty.

The second iteration starts by removing PR2 from PRQ and then assigning a rating to its producer P3. The only prediction by P3 is PR1, which is not on WPRQ. Assuming that the rating of P3 is higher than the rating for P1, PR1 is added to PASS2Q. The second iteration ends with PR1 on PASS2Q, P2 and P3 with ratings, and both PRQ and WPRQ empty. At this point, the second pass begins by removing

PR1 from PASS2Q, marking it, and recalculating the rating for P2. Assuming that the recalculation produces a higher rating for P2, its prediction PR2 is added to PASS2Q. Thus, the first iteration of pass two ends with PR1 marked, P2 rerated, and PR2 on PASS2Q. The second iteration begins by removing PR2 from PASS2Q, marking it, and recalculating the rating for P3. The prediction made by P3 is marked, so it is not processed further and the second pass ends.

c. CLEANUP

The final stage of the predict task performs various cleanup operations. It erases marks which may have been left by the previous stage. Predictions and phrases needing erasures are chained together through their records, so the erasing can be done quickly. LEXQ holds new predictions for categories with lexical entries. For each prediction on LEXQ, word sets are created in the manner described earlier (see Section D.1). LEXQ2 holds old predictions with word sets that now have a modified consumer context. Each such word set has its priority recalculated. INSTLIST contains new consumers for old predictions that have been previously satisfied. Each phrase on the prediction's instances-list is added to the new consumer by the add-constituent procedure. OTHERINSTLIST contains

consumers providing a new context for old predictions with suspended construct-complete-phrase operations. The consumers on OTHERINSTLIST may be new consumers that were created during the first stage or old consumers that had an addition made to their consumer context during the first stage. The traverse-subnet procedure takes care of both cases. During the cleanup stage, for each suspended operation with a new or modified consumer P, the consumer-lookahead test is performed. If the test succeeds, the construction of the phrase goes ahead. Otherwise, it remains suspended.

After the cleanup stage, one cycle of the predict task is complete. However, there may still be other predict sets waiting to be processed. If there are, and the highest priority for them is higher than any of the tasks currently scheduled, the predict task goes ahead for another cycle. Otherwise, it schedules itself at the priority of the best predict set and returns control to the top-level Executive procedure.

4. MULTIWORD LEXICAL ENTRIES

Small words like "of", "are", "a", and "the" cause difficulties for a speech understanding system because they are often poorly articulated or reduced to a short duration

remnant. In order to minimize false rejections for such words, the acoustic mapper must use very loose criteria in testing them. However, the loose criteria lead to a high false alarm rate. This creates a problem because the language definition often allows several small words to occur in a series. To reduce the bad effects of small words, we have introduced a mechanism allowing a series of small words to be processed acoustically as a single unit. With this approach, a series of words is combined to produce a larger 'multiword lexical entry', or 'multiword', which can be tested more reliably. Small words are tested individually only for contexts that do not put other small words beside them. A set of over 50 multiwords was used in the experimental tests of the system. They included phrases such as "is the", "of a", and "what are the".

This method does not eliminate the small-word problems, but preliminary results suggest that it does reduce them. Following the introduction of multiwords, the mapper criteria for small words were tightened to reduce their false alarm rate. Data collected on the mapper performance (see the discussion in Chapter IV) included two cases in which small words were incorrectly rejected, perhaps because of this tightening. However, in both cases the missed words were part of a series of small words, and

the multiword for the series matched well enough to be accepted in spite of the fact that one of its component words was rejected when mapped alone. Thus, the sentences could be understood correctly although one of the (small) words was incorrectly rejected. More tests are required to assess the effects of multiwords, but these initial results suggest that they may provide a means of raising the accuracy of acoustic processing.

The multiwords are treated as single units for acoustics, but not for linguistic processing. The language definition would be badly distorted by an attempt to incorporate multiwords directly. Instead, the Executive breaks multiwords into their individual parts, and multiword entries like "is a" are analyzed by the linguistic knowledge sources as two separate words although recognized by the mapper as one. When a multiword with individual words W1,...,Wn is accepted from position P1 to position P2 in the input, the Executive creates n-1 new 'pseudo-positions', X1,...,Xn-1, which are distinct from positions in the actual input. The multiword is then analyzed as a series of regular words: W1 from P1 to X1, W2 from X2 to X3,..., and Wn from Xn-1 to P2.*

---

* This method is similar to the Kay and Kaplan method of representing the input as a 'chart' (see Kay, 1967, 1973).

The pseudo-positions must be treated as special cases in some of the Executive algorithms. The time compatibility tests that are used by the distribute-phrase and add-constituent procedures look for pseudo-positions. If both times being tested are fixed and one is a pseudo-position, the other must be the same pseudo-position or the times are incompatible. The phrase mapping procedure accepts without testing two adjacent words from the same multiword phrase. However, if the words for phrase mapping are not from the same multiword, either word that is from a multiword is replaced by the entire multiword before phrase mapping takes place. For example, if called with the pair "the" and "ship" for phrase mapping and "the" comes from the multiword "is the", the phrase mapping is done using "is the ship". If the lexical subsetting procedure is called with a pseudo-position, it does not do any acoustic processing, but instead looks in a table of accepted multiwords to find the word from the multiword phrase at the given position. Thus, from the W1...Wn example, the subset to the right of X1 contains just W2.

Multiwords are standard sequences of words from the language, so there is a danger of wasted effort expended analyzing both a multiword and the individual words composing it. For instance, if the multiword phrase

"of the" is accepted from position 35 to position 55, the mapper may also accept "of" from 35 to 45 and "the" from 45 to 55. The Executive has two ways of blocking series of small words that duplicate a multiword. First, the phrase mapping routine rejects pairs that together form a multiword. Thus, a phrase whose rightmost word is "of" cannot be put to the left of a phrase whose leftmost word is "the" unless the "of the" comes from a single multiword accepted by the mapper. The second block on small words that duplicate a multiword occurs in the word task. If "of" is the only word accepted ending at position 45, "the" will be excluded from the candidate words starting at 45 because "of the" is one of the system multiwords. The exclusion is not permanent; it will stop if another word ending at 45 is later found and leads to a new execution of the predict task at position 45 and direction equal RIGHT. The exclusion is removed as part of the word set revisions during the cleanup phase of the predict task.

Recall that the Definition Compiler adds the multiwords to the lexicon so that they are available for the word task algorithms. Multiwords beginning with word W are added to all lexical subcategories containing W and are marked for use in left-to-right word sets. Similarly, multiwords ending with W are added also, and are marked for

use in right-to-left word sets. The procedure that creates word sets eliminates multiwords that are marked for the wrong direction. The remaining multiwords are treated like standard words until they reach the procedure to create terminal phrases. This procedure recognizes the accepted word as a multiword, creates pseudo-positions for it, and then creates a terminal phrase for one of the standard words in the phrase If the multiword lexical entry was marked for left-to-right use, the terminal phrase is constructed for the leftmost word in the multiword. Otherwise, the terminal phrase is created for the rightmost word. The terminal phrase is distributed and added to consumers in the standard manner. The resulting incomplete phrases lead to predictions at the pseudo-position within the multiword. The standard parse net structure and task algorithms are used in analyzing the multiword, and the analysis benefits from the precise lookahead information available at the pseudo-positions.

In summary, multiword lexical entries appear to offer a way to improve acoustic accuracy and, by the methods outlined above, have been integrated into the system in a simple manner without requiring any changes in the language definition.

5. PRIORITY SETTING

Newell and Simon (1976) conclude their 1975 Turing Award Lecture by stating:

> For all physical symbol systems, condemned as we are to serial search of the problem environment, the critical question is always: What to do next? (p.126)

In our system, this critical question is answered according to task priorities, thereby replacing the original question by another difficult one: How to set priorities? The approach we have taken is to base priorities on phrase ratings that combine results from acoustic mapping and language factors. The predict task priority is the highest rating for an incomplete phrase waiting to make a prediction. The word-task priority is the highest rating for a terminal phrase from the current word candidates. The priorities are initially derived from ratings, but they can be modified according to various strategies such as focus by inhibition.

a. FACTORS

Phrase ratings are derived from phrase scores, and phrase scores are in turn derived from language

factors and mapper factors. The language factors are defined in lexical-category procedures and rule procedures. Many of the language factors are Boolean (i.e., two-valued: true or false). Such factors block the construction of a phrase if the phrase would have certain undesirable properties. If the properties are not present, the Boolean factor allows the phrase to be constructed and does not affect the phrase score. The remaining language factors are non-Boolean. They raise or lower the phrase score according to the results of various tests and are used to 'tune' the language definition (as discussed in Robinson, 1975b). For example, if questions are expected to be the predominant form of input to the system, non-Boolean factors can be added to raise scores for questions relative to other kinds of sentences. Such factors would bias the system toward looking for ways to interpret utterances as questions.

Boolean language factors are implemented as conditional calls on the F.REJECT procedure, which causes the construction of a phrase to be immediately terminated. The non-Boolean factors are implemented as variables assigned integer values in the range 0 to 100 or pairs of integers, <WEIGHT TOTAL>, such that TOTAL divided by WEIGHT is between 0 and 100. A factor value that is a single integer J is equivalent to the pair <1 J>. The factor

WEIGHTs reflect relative importance; a factor with WEIGHT equal 2 has twice the effect on the phrase score as a factor with WEIGHT equal 1.

Mapper factors are derived from mapper scores and thus are based on acoustic, phonetic, and phonological judgments. Mapper factors for individual words and multiword lexical entries directly contribute to the scores of terminal phrases. Similarly, phrase mapping factors contribute to the scores of nonterminal phrases. A nonterminal phrase with N constituents has N-1 phrase junctures and N-1 phrase mapping factors.

b. PHRASE SCORES

The score for a phrase combines factors to form a WEIGHT and TOTAL pair that can itself be used like a factor value. For a terminal phrase, the score combines language factors from the lexical-category procedure and the mapper factor for the word. For a nonterminal phrase, the score combines factors from the rule procedure, phrase-mapping factors, and constituent scores. In most cases, the score is simply a vector sum of the factors: the WEIGHT is the sum of the factor WEIGHTs, and the TOTAL is the sum of the factor TOTALs. However, the vector sum method does not allow a very low factor to pull the score down as much as it

should. A possible solution would be to use a geometric mean instead (the geometric mean of N numbers is the Nth root of their product). We have taken a different course and treat low factors as special cases; if the quotient of TOTAL divided by WEIGHT for the factor is below a certain threshold by X percent, the TOTAL for the score is reduced by X percent. Ideally, this ad hoc method would be replaced by a scoring technique based on greater information about the performance characteristics of the system components (perhaps like the probabilistic scoring described by Klovstad in Woods et al., 1975a, pp.33-39).

c. PHRASE RATINGS

The rating for a root-category phrase comes directly from its score. The rating for a nonroot phrase is derived by operations that look at the consumer context of the phrase. As discussed in the preceding overview, we have experimented with two techniques for calculating phrase ratings. The two methods for assigning nonroot phrase ratings are 'context checking', which entails the execution of consumer rule-procedures to gather factor information, and 'merging', which does not.

Both methods take advantage of the fact that the system is designed to assign ratings to the consumers of

a phrase P before assigning a rating to P itself. Define the 'rating-score' of the consumer to be the WEIGHT and TOTAL pair forming the score used in calculating the consumer rating. Also, let the 'merged rating-score' of P and a consumer C be the vector sum of the score of P and the rating-score of C. The rating of P with respect to the consumer C is determined from the merged rating-score of P and C. The rating of P, as calculated by merging, is the highest rating of P with respect to any of its consumers. (Recall that there is always at least one consumer for a nonroot phrase.)

The merging method is fast and simple. Moreover, it reflects both the quality of the phrase and the quality of the consumers of the phrase. However, it does not indicate whether the phrase is really compatible with its consumers. The phrase may have attributes that will cause any complete phrase built from it to be rejected by the consumers. There is nothing to be gained from working on such a phrase, but the merging method will not give it a low rating if its score is good and the consumers' ratings are good. The context-checking method is designed to avoid this defect.

To start with a simple example, assume that the phrase P has a single consumer C, and C is a root-

category phrase. The context-checking method assigns a rating to P by creating a 'virtual' phrase C' from C and P. Virtual phrases are only created during context checking and are deleted immediately afterward with all storage reclaimed. By assumption, C' is a root category phrase, so the score of C' is used to determine the rating of P.

If C is not root category, but has a single consumer D which is, the context-checking method adds C' to D to form another virtual phrase D', and the score of D' determines the rating for P. In general, any path leading from P to a root-category consumer can be used to establish a rating. The phrase rating could be defined as the maximum rating with respect to any consumer path leading to a root-category phrase. However, there may be many such paths, and they would all have to be tried in order to guarantee finding the best one. The cost of such an exhaustive exploration of the consumer context could overshadow the potential benefits, so the system instead uses heuristic methods to find a near-optimal path without necessarily considering all of the paths.

The details of the context-checking algorithm for setting phrase ratings are as follows. If the phrase P to be given a rating is a root-category phrase, its rating is calculated directly from its score. Otherwise, a lower

bound for the rating is initialized and the consumers of P are scheduled for the creation of virtual phrases. The initial lower bound is zero if P is being rated for the first time. In case P is having its rating recalculated because of some addition to its consumer context, the lower bound is set to its prior rating. Each consumer C is scheduled at a priority based on the rating determined by merging the score of P with the rating-score of C. (Note that the scheduling referred to here is internal to the rating algorithm; it is not part of the Executive task scheduling.)

After the lower bound is initialized and the consumers scheduled, the path growing begins. It continues until there are no more extensions scheduled or the highest priority is lower than the lower bound. At each step in the growing, the highest priority extension is performed. A scheduled extension consists of a consumer C and a (possibly empty) path of virtual phrases. Let X be the most recently added virtual phrase in the path or P if the path is empty. In either case, C is a consumer for X, and the attributes and score of X are available for use in constructing a new virtual phrase, C'. If C' is rejected by the rule procedure or if its score is subthreshold, this step in the path growing is terminated. Otherwise, if C' is root category,

its score is used to update the lower bound. If C' is not root category, extensions are scheduled for each of its consumers at a priority based on the rating determined by merging the consumer's rating-score and the score of C'. The consumer is not scheduled if its priority is less than the lower bound, or if it already appears twice in the path leading to C' (thus blocking paths from going around consumer-producer loops more than once).

In practice, the merging method and the context-checking method give similar results for phrases that have no conflicts with their consumer contexts. However, when a phrase has attributes that would cause completions of it to be rejected or downgraded by consumers, the context-checking method gives it a lower rating than the merging method. If the rule procedures are a major source of evaluative knowledge, context-checking can produce more useful ratings and hence better priorities. However, merging is a faster method and can produce results that are as useful if the rule procedures provide only limited constraints. In our system, the rule procedures are in fact an important source of information, and experimental results (Chapter IV, Section D) show that the use of the context-

checking method has good effects on performance.*

d. RELATION TO EXECUTIVE TASKS

Ratings are calculated at several points in the Executive tasks. When an incomplete phrase is created in the add-constituent operation, it is inserted in the parse net to establish its consumer context, and then its rating is calculated. If its rating is above a certain threshold, the phrase is added to the predict sets. Otherwise, the phrase will not be allowed to make any predictions unless its consumer context changes to raise its rating above the threshold.

During the second stage of the predict task, new phrases get initial ratings, and old phrases with modified consumer contexts get revised ratings. In the case of a revised rating, if the phrase had been kept out of predict sets previously, but now has an above threshold rating, it is added to the predict sets and becomes eligible to make predictions. Otherwise, if the phrase is a member

---

* The system actually uses a mix of the methods: context-checking for partially completed nonterminal phrases, and merging for empty ones. This mix is preferred to pure context-checking because empty phrases have so few attributes established that they are almost always compatible with their context. By mixing the methods, the costlier checking is only done in cases where it is more likely to help.

of a predict set and was or is now the best phrase in the set, the priority of that set is updated to reflect the new phrase rating. These revisions ensure that a phrase that is given a low initial rating has a chance to get a higher rating later if its consumer context changes. The revisions also keep the priorities in step with the ratings.

Like predict sets, word sets have priorities that depend on ratings. These ratings are calculated as part of the cleanup stage of the predict task. They are initially calculated when the word set is created and are revised when the consumer context changes. The consumers that are used in setting a word set rating come from the prediction that caused the creation of the word set.

The details of setting word set ratings depend on whether the system is mapping all at once and also on the number of words in the set. When mapping all at once, the words in the set have mapper scores and corresponding mapper factors. The word set is assigned a score equal to the mapping factor with the highest Q. The rating of the word set is then determined by the merging method.

If the system is not doing mapping all at once, the word set rating is calculated either by merging

(using a default score for the word set) or by context-checking. The context-checking makes virtual phrases using a default score and the default attributes of the word set lexical subcategory. The defaults are shared by all the words in the subcategory, so only one rating needs to be calculated for all the words in the word set. The savings resulting from this are the principal motivation for having lexical subcategories. The merging method is employed if the system is using a no-context-checking control strategy, or if there are few words in the word set (three or less, in the current implementation).

The use of merging when there are few words in the word set represents a trade-off between processing time for context checking and processing time for acoustic tests. If the context checks are performed and result in a low rating, the system may avoid doing unnecessary acoustic processing. If merging is used and results in an inflated rating, extra acoustic tests may be done. The larger the word set, the more likely it is that the cost of the context checking will be offset by savings in acoustic processing since the context checking is done only once for the entire set but the acoustic tests are repeated for each word.

6. ADJUSTING PRIORITIES AND FOCUS BY INHIBITION

Phrase ratings determine initial priorities, but priorities can be adjusted according to a variety of strategies. If no adjustments are made, the system will work in a 'best-first' manner strictly following the ratings in moving from one task to another. Looking at the series of tasks performed by such a best-first method, one often observes the system shifting its activity among competing possibilities at a high rate. Such observations led to an experiment in adjusting priorities in hopes of making the system 'focus attention' better. The method of adjusting priorities is called 'focus by inhibition' because it affects the system focus of attention by inhibiting work on certain alternatives.

To explain the motivatation for focus by inhibition, assume, contrary to fact, that the system could find a word X in the input that was guaranteed not to be a false alarm. Having found X, the system would not want to waste time on any task that would try to replace it. In other words, the system should block word tasks that would produce words overlapping X and block predict tasks for phrases containing words that overlap X. In this way, the system would avoid wasting its effort on tasks that were in conflict with a word that was sure to be correct. In actual

practice, the system is only relatively sure of words rather than being absolutely sure. Instead of being sure that X is correct, the system will have a certain confidence that X is correct. If it is highly confident in X, it will be reluctant to perform conflicting tasks, but it will not absolutely refuse to perform them because X may be a false alarm. Focus by inhibition implements this reluctance as a reduction in priority of conflicting tasks. The priority reduction causes a system bias, but it is a bias that can be overcome -- the task can still become top priority in spite of having its priority reduced.

In the implementation of focus by inhibition, the focus is represented by a set of time-compatible words (or multiwords) from the input. Two words are time-compatible if they do not overlap or have a small enough overlap that they could still be in the same phrase. Words are time-incompatible if they are not time-compatible -- in other words, if they overlap so much that they cannot be in the same phrase. A phrase conflicts with focus if any word in the phrase is time-incompatible with some word that is in focus. The phrase priority is initialized to its rating and is lowered if the phrase conflicts with focus. The percentage amount by which the priority is lowered depends on the 'strength' of the focus word. The focus strength

goes up according to the mapper score for the word and the ratings of the phrases putting the word in focus. The strength decreases acccording to the likelihood that the word is a false alarm.

At the start of the predict task, the top priority phrase P is selected from the top priority predict set. It is possible that P became top priority in spite of being in conflict with some focus word W. If so, W is removed from focus and any task inhibited because of conflict with W has its priority raised to its preconflict level. Next, the predict task checks for a conflict between P and the current focus set. (Even if P just caused the removal of a focus word, P may still be in conflict with a different focus word.) If P conflicts with a focus word X, the priority of P is set to its rating reduced according to the inhibition strength of X, and the predict task goes to the rescheduling stage without making any predictions. If there is no conflict, the words contained in P are added to the focus set, and the normal predict task begins. These operations at the start of the predict task take care of adding words to focus, removing them, and adjusting prediction priorities.

The word task also has operations for focus by inhibition. A word set conflicts with a focus word X if any

word from the set would be time-incompatible with X. For example, if X begins at position 40 and the word set left time is also 40, there is a conflict. In case of a conflict, the word set priority is lowered according to the strength of the focus word. At the start of the word task, the highest priority word set WS is selected. It may have become the highest priority in spite of a conflict with a focus word X. If so, WS is marked as 'immune' to inhibition by X. The word X is not removed from focus; however, if WS leads to the acquisition of a good word, a high priority prediction will remove X from focus later. Next, the word task checks for a conflict between WS and some focus word Y to which WS is not immune. If there is such a Y, the priority of WS is revised and the word-task is rescheduled. Otherwise, the normal word-task operations begin.

No other changes are involved in implementing focus by inhibition. The structure of the system makes it easy to try different methods for adjusting priorities, but, as mentioned earlier, the methods must be justified experimentally. This particular technique did not improve system performance, but perhaps a related approach will. The strength of the method is that it provides simple answers to how, when, and why to focus attention, while maintaining the completeness of the system control strategy.

The weakness of this particular attempt is its inability to focus primarily on hits rather than on false alarms; perhaps greater selectivity in adding words to focus would produce a focus by inhibition with a positive effect on system performance (evidence suggesting that this may be so is given in Chapter IV, Section D.3.)

E. DISCUSSION

This section compares our framework for speech understanding to some others and sketches the evolution of our system over the last four years. The purpose is to clarify further our system design rather than to give a complete review of the literature (for a recent survey of speech recognition research, see Reddy, 1976). We consider systems developed at Carnegie-Mellon University (CMU), at Bolt Beranek and Newman (BBN), and earlier at SRI.

1. CMU: HARPY AND HEARSAY-II

Two lines of speech research have been carried out at CMU in the last few years. One is based on the use of a simple dynamic programming model with all system knowledge represented in a state transition network. The other is based on a more complex design using multiple, cooperating knowledge sources. Both lines of research have been very

productive, and there has been significant cross-fertilization between them. We discuss the most recent (and most successful) systems in each line: HARPY in the dynamic programming line and HEARSAY-II in the multiple knowledge source line.

The HARPY system has the best performance statistics of any existing system for understanding connected speech (Lowerre, 1976). The design for HARPY evolved from a comparative study of two previous CMU systems, Hearsay-I (see Erman, 1974a) and Dragon (see Baker, 1975). HARPY uses a state transition network to represent all possible pronunciations of all legal input language sentences. The network thus embodies the entire system knowledge of syntax, word pronunciations, and interword coarticulation effects. To process an utterance, HARPY looks for the path through the network that best matches the input. The search proceeds left-to-right, segment by segment across the utterance. At each segment, all paths are discarded that are more than a threshold amount worse than the best path. This heuristic means that the system is not guaranteed to find the optimal path (the best one may be dropped if it starts out poorly), but the threshold for dropping paths is chosen so that in practice the optimal path is found in nearly all cases. Compared with an

exhaustive search of all paths, there is a small sacrifice in accuracy for a large improvement in processing time.

The performance achieved is impressive -- on a 1011 word vocabulary, HARPY got 92% correct of a test set of 100 sentences with an average of a little over 20 seconds of processing per second of speech on a 1.3 million instructions per second computer (DEC KL-10; these results were reported at a system demonstration at CMU on September 8, 1976). Because of its one-pass search strategy, HARPY has a low variance in its speed, which is also an important feature for a useful system. Finally, in contrast to most other speech-understanding systems, HARPY is conceptually simple and has been well-studied. For example, there is a good understanding of how its recognition time depends on parameters such as the number of samples in the input, the number of phonetic-classification templates, and the size of the recognition network.

HARPY achieves good performance with simple means, but it does so by sacrificing generality. With its reliance on a finite-state network for syntax, HARPY can only deal with very restricted input languages. The network representation also makes it difficult to deal with dynamic changes such as take place in natural connected discourse. For example. the use of anaphora and ellipsis depends on the

preceding sentences, and what is acceptable in one context may not be in another.

The performance of HARPY appears to be largely dependent on its use of an extremely constrained input language. The average branching factor, the number of alternatives at each word, is about 9.5 for the grammar with which HARPY achieved the results mentioned above. Preliminary tests show a drop in accuracy to about 85% when the branching factor is increased to 25 (see Goodman et al., 1976). Based on measurements of our SRI language definition, we feel that a more natural input language with a 1000+ word vocabulary could easily have a branching factor well over 200 — so these results suggest that HARPY may be limited to very restricted languages unless there is a big improvement in acoustic accuracy. In addition, HARPY's strictly left-to-right search for a complete path through its network prevents it from dealing with incomplete sentences or inputs that in any way deviate from the predefined grammar.

In summary, on the limited tasks HARPY was designed for, it does very well. However, it does not represent or claim to be a general technique for speech understanding with a wide range of input languages.

Undoubtedly, applications exist where a carefully constrained, artificial input language can be useful, and in these applications, HARPY offers a viable approach. For more general applications that may require a closer approximation to natural language input, the simple approach used by HARPY seems unlikely to succeed.

Another system developed at CMU, Hearsay-II (HS-II), has taken a more general approach (see Lesser et al., 1975). HS-II is based on many interesting design concepts. Perhaps the most distinctive is the representation of knowledge as self-activating, asychronous, parallel processes that communicate with each other through a global data structure, called the 'blackboard'. There is an emphasis throughout HS-II on generality and uniformity in representation and control. The same approach is proposed for all levels of the system from signal processing to semantics.

This search for generality is reminiscent of CMU efforts in other research areas such as problem solving. In fact, Lesser et al suggest viewing HS-II as a production system that is executed asynchronously.* They go on to state that in this uniform framework there are to be many small knowledge sources, each independent of the others.

---

* See Newell (1973) for a discussion of production systems.

Knowledge source communication is to be through the generation and modification of globally accessible hypotheses on the blackboard. Changes on the blackboard are also to control the activation of knowledge sources. Each knowledge source is to have a precondition that is a description of some partial state of the blackboard. The knowledge source process can be run when the precondition is satisfied. The blackboard modifications made by one knowledge source trigger other knowledge sources by satisfying their preconditions. In addition to preconditions, each knowledge source has a specification of the kinds of changes it makes (information used in scheduling knowledge source activations), and a program which accomplishes those changes. There are to be no separate data structures or state information for the individual knowledge sources; everything is to be done uniformly by means of the blackboard.

In the following discussion, we comment first on the HS-II goals and compare them to our system design. Later, we consider how close the actual HS-II system that was demonstrated in September 1976, approached the design goals given in the 1974 paper.

HS-II as described above is an elegant and ambitious system. It is more general than HARPY, for

instance, in that it does not force all knowledge into a state transition network representation. Also, it can hope to deal with sentence fragments that do not fit the predefined language. It is not limited to a left-to-right search and can build up phrases anywhere in the input in any order. HS-II and our system share the features just mentioned, but, in spite of that similarity, there are significant design contrasts between HS-II and our system with respect to system integration and control.

First, unlike HS-II, we are not trying to use a uniform approach throughout the system for representing partial results and for controlling knowledge source operations. We feel it is very unlikely that a uniform method can produce satisfactory results for disparate operations such as semantic interpretation and acoustic labeling. Second, we emphasize explicit, direct knowledge source interactions through the procedural parts of rules rather than trying to keep all knowledge source interactions indirect through a global data structure. It is clear that some of the interactions must be indirect to allow the Executive to determine dynamically the order in which tasks will be performed as it searches for an interpretation. This flexibility is necessary because of the inefficiencies associated with a fixed order of search (see Paxton, 1975, for further discussion of this point).

We provide for indirect interactions by means of the phrases in the parse net and achieve modularity at the level of rules in the language definition. However, within a rule, the knowledge source interactions are explicitly stated in the procedure. There are several reasons for such a mix of direct and indirect interactions. One major consideration is to encourage interactions by providing a low-cost mode of communication. We feel that there is a large potential for mutual guidance that would not be realized if all knowledge source communication was indirect; the cost of modifying the global data structure and triggering the relevant preconditions would inhibit the interactions. In addition to being more efficient, direct interactions are often simpler to specify than the indirect ones. The simplicity further encourages the development of knowledge sources that cooperate closely.

Another consideration is the overhead associated with scheduling and activating tasks. If every knowledge source interaction had this overhead, the system performance would suffer. Moreover, efficiency considerations would limit the algorithms for determining task priorities to simple operations such as merging local scores. By reducing the number of tasks, we are able to use a more complex scheduling algorithm and still keep the total scheduling

cost small relative to the time spent actually performing the tasks. Tasks in our system are thus substantial operations such as making entire sets of predictions or performing the acoustic processing needed to look for words at a particular location in the input. This relatively large task size is another difference between our system and HS-II as described in the 1974 paper. We also have a few large knowledge sources where they call for many small ones, and we have separate data structures for special use by the different knowledge sources rather than enforcing the uniform use of a single global data structure.

In September 1976, a version of HS-II was demonstrated at CMU that was somewhat different in design from the description given in the 1974 paper. (The 1976 system is briefly described in Reddy et al., 1976). The 1976 HS-II has a few large knowledge sources rather than many small ones. One component does parameter extraction, acoustic analysis, segmentation, and labeling. Another does bottom-up word recognition based on syllables. A third is a word verification process using HARPY techniques. This verifier checks words found by the bottom-up recognizer or words predicted by the syntactic component. A fourth knowledge source process is a word-pair adjacency tester. It looks at the speech data in a word-pair gap or overlap

and decides if the pair is acceptable or not on the basis of the phonetic spellings and various word juncture rules. A fifth knowledge source process is the word-sequence hypothesizer, which provides multiword seeds for an island driving control strategy. This process uses a bit matrix indicating allowable word-pairs in the language to ensure that the words in the multiword island are at least pairwise grammatically acceptable. It also calls the word-pair adjacency tester to make sure that the words in the island can be adjacent acoustically. The sixth and final knowledge source process is the parser. It has a template grammar for use in parsing word sequences, predicting words that can be syntactically adjacent to the ends of the sequence, and constructing larger sequences when predicted words are verified.

The performance of HS-II on the same 1011 word vocabulary and input language as used in the tests of HARPY was 84% sentence accuracy (versus 92% for HARPY) and processing times of about 2 to 20 times longer than HARPY (results reported at a system demonstration at CMU on September 8, 1976). The poorer performance relative to HARPY is probably a result of the very restricted input language used for the tests; with such a language, HARPY's fast, simple techniques are adequate and give better

accuracy by considering more alternatives before making a choice.

In many ways, the 1976 HS-II moved away from the description given in the 1974 paper and closer to our system design. As mentioned above, the demonstrated HS-II has a few large knowledge sources rather than many small ones. It makes use of direct, explicit knowledge source interactions: the word verifier is called directly from the bottom-up word recognizer and the word junction tester is called from the word sequence hypothesizer. The knowledge sources maintain private data structures and state information rather than always using the blackboard, presumably because of the inefficiencies of doing everything in a uniform representation structure. As in our system, data-directed invocation is used for higher level processes but not for lower level ones. For example, the word verifier uses a HARPY control structure rather than the event-driven technique. Thus, the system uses varying control strategies rather than always using a precondition-activation scheme, again presumably because of the inefficiencies of the uniform method.

All of the above changes bring HS-II closer in design to our system; however, differences still remain. For example, HS-II has adopted a multiword seed technique

for island driving that apparently improves its ability to get started correctly. We have not tried that method, but it could be added to our system easily and might result in better island driving performance for us also. However, the technique may depend on having a restricted grammar to make the pairwise grammaticality tests sufficiently restrictive; with our more general language, such a test would be much less useful since so many word-pairs are possible.

A more significant difference is the emphasis HS-II has placed on parallel processing. The results of future experiments with HS-II on the special C.mmp multiprocessor system will be interesting as an indication of whether or not the potential parallelism can be effectively utilized (see Fennell and Lesser, 1975). On the other hand, we have placed greater emphasis on natural input languages than CMU has. This emphasis has led us to develop special techniques such as context checking in setting phrase ratings. To the extent that the systems have converged, it indicates a growing consensus regarding system architecture for general speech understanding; the differences between the systems reflect the different research goals of the projects, such as parallel processing in the case of HS-II and natural language processing in our case.

2. BBN: SPEECHLIS AND HWIM

SPEECHLIS and HWIM do not represent parallel speech projects in the way HARPY and HEARSAY-II do. Rather, the names refer to systems developed during two relatively distinct phases in the speech understanding research at BBN. In the first phase, from 1971 until 1975, BBN produced SPEECHLIS. In the second phase, 1975 to 1976, the BBN system was changed significantly in both its components and its overall organization. The changes merited the adoption of a new name, HWIM. The two systems are particularly different with respect to system integration and control, so it is appropriate for us to discuss both of them.

The SPEECHLIS system has an interesting design that evolved through the use of 'incremental simulation' combining computer programs and human simulators [see Woods and Makhoul, 1974, regarding incremental simulation; see the BBN papers in the 1974 IEEE Proceedings (Erman, 1974b) regarding SPEECHLIS]. In processing an utterance, SPEECHLIS starts with acoustic-phonetic analysis to produce a segment lattice representing all of the alternative segmentations of the utterance and the alternative phonetic identities of the segments. A lexical retrieval component then searches through the segment lattice for good matches for words of three or more phonemes. Such matches are added to a word

lattice. A semantic component constructs sets of nonoverlapping words from the lattice by selecting semantically related words. These word sets, and information regarding their syntactic and semantic analysis, are called 'theories.' When semantic association can add no more words to a theory, the theory is passed to a syntactic component, called SPARSER -- "speech parser" (see Bates, 1975). SPARSER postulates grammatical structures for the words in the theory and proposes words to fill gaps between the words. The SPEECHLIS control component keeps track of different theories, proposals, and events (such as the retrieval of a word satisfying some proposal), and decides what to do next on the basis of a weighted sum of scores from lexical retrieval, syntax, and semantics.

SPARSER in particular is worth discussing in more detail. It uses an augmented transition network (ATN) formalism and a grammar that was derived from earlier work at BBN (see Woods, Kaplan, and Nash-Webber, 1972). SPARSER is activated by the control component to process a set of nonoverlapping words, each contiguous word sequence being called an 'island.' The parser's job is to create parse paths through the islands and to predict syntactically acceptable words to fill the gaps between them.

Each island is processed left-to-right. The first step is to find all the places in the grammar where the leftmost word of the island can occur. This and similar operations make use of precompiled grammatical indexes. The second step is to find all the transitions that lead to the arc for the first word but do not use the previous word of input (i.e., JUMP transitions). The grammar states that are reachable from the left of the first word by these lead-in transitions are used in making predictions for words to the left of the island. The third and last step in parsing an island is for paths to be extended through the island in preparation for making predictions at the right end. The parse paths are extended to the right in a best-first manner according to scores that reflect the likelihood of the arcs. If no paths can be found through the island, the theory is rejected. Otherwise, the ends of the paths determine states for predictions to the right of the island. (Notice that predictions on the right correspond to the end of a path leading completely through the island, but predictions on the left are constrained only by the leftmost word.)

After all the islands in the theory have been processed, SPARSER looks for predictions to fill the gaps. In particular, it looks for small gaps that could be filled by a single word. If a prediction is made from both sides

of such a gap, a proposal is made for the lexical retrieval component to search for the predicted words. While SPARSER is making proposals, it can return to an island and try to extend more paths through it to get more predictions on the right. This is done in hopes of producing a common prediction to fill a gap.

The ability to use pairs of islands to make predictions to fill gaps is one of the noteworthy features of SPARSER. Miller's LPARS (see Miller, 1973) also made predictions to fill gaps between islands, but it used an exhaustive search strategy and so was probably impractical for large vocabularies and grammars. SPARSER appears to have been the first to provide a combinatorially feasible control strategy for multi-island processing. Another significant feature of SPARSER is the large amount of merging of alternatives resulting in the sharing of information among different theories. For instance, only one instance of a particular state and input-location configuration is ever created and it is shared by all theories.

However, there are problems in the SPARSER design. First, there is not enough communication with other components, especially semantics. For example, SPARSER's proposals are not constrained by semantic information and so

they may lead to wasted effort looking for words that are acceptable syntactically but bad semantically. Second, the use of multi-island theories probably costs more than it is worth. There are no inter-island consistency checks; the only interaction among islands is in making predictions at gaps that are small enough to be filled by a single word. Even when there is such an interaction, it is simply to look for common predictions, and failure to find one does not disqualify the theory since the gap might be filled by more than one word. The overall result is that there is relatively little gain from having multi-island theories. In contrast to the small gain from having multi-island theories, the costs are significant. The same island may occur in many different theories, and, although SPARSER does share information among theories, there is a nontrivial overhead for reprocessing an island in each theory.*

A final and especially serious problem with SPARSER is its failure to make good use of the available grammatical constraints to limit its operation. We mentioned above that predictions on the left of islands depend only on the leftmost word and are therefore not taking advantage of possible constraints provided by the

---

* The time for reprocessing is about one-third the time for initial processing; Bates (1975, p.111) reports 16.5 CPU seconds for reprocessing a theory that took 47.5 seconds to process originally.

other words in the island. Another failure to use the available information relates to the restrictions on arcs. The grammar operations on arcs are explicitly divided into local ones, which depend only on the word or constituent for the particular arc, and context sensitive ones, which depend on information from some other arc. SPARSER performs local operations when it creates arc transitions, but it does not do any context sensitive operations until it has a complete path through a network allowing a constituent to be formed. When such a complete path is acquired, SPARSER goes through the path left-to-right executing the context sensitive operations, thus making sure that the required information is available when it is needed. This is a simple solution to the problem of context sensitive operations, but it fails to prevent SPARSER from working on partial paths that are bad syntactically. Bates acknowledges this problem and comments to the effect that the parser could take context into account more easily if the grammar had less of a left-to-right orientation (Bates, 1975, p.190). This comment supports our decision to use a language definition representation that is more neutral than ATNs with respect to control strategies.

SPEECHLIS has other problems in addition to those mentioned above regarding SPARSER. First, the use of

semantic associations does not seem to be sufficiently selective to help in producing a few good starting theories for the system. In a limited task domain, most words will be semantically related, so almost any set of words found by lexical retrieval will be accepted by semantics.* If lexical retrieval is very accurate and only finds a few outstanding matches, the poor selectivity of semantics will not hurt, but semantic association will be an unnecessary step. However, if lexical retrieval produces many words that match equally well, semantic association seems likely to swamp the system with theories. Given the problems with acoustic recognition, the latter alternative seems more probable. A second problem with SPEECHLIS is the generally poor communication among knowledge sources. We mentioned that syntax makes proposals that may be bad semantically. The converse is also true; semantics makes proposals that may be bad syntactically. In general, any component in SPEECHLIS is free to make proposals, but there is inadequate communication to ensure that such proposals are mutually satisfactory. There is a need for closer cooperation among the components.

---

* However, Nash-Webber and Bruce suggest that in the lunar rocks task domain used in SPEECHLIS, the possible semantic relationships among the entities were so limited that this was not a problem (Woods et al., 1976b, p.42). Apparently it became a problem when the task domain was changed to travel budgets (see comments in Woods et al., 1975b, p.44).

The performance of SPEECHLIS seems to support these criticisms. There has been no report of systematic tests of its performance, but comments appear in various publications (especially Nash-Webber and Bruce in Woods et al., 1976b; also in Woods et al., 1975b). The processing time required by SPEECHLIS may have prevented extensive testing. For example, SPARSER reportedly can take over 40 seconds of processing on a single theory if much bottom up processing is necessary (Bates, 1975, p.111). Also, storage demands apparently made it impossible to run tests to completion in many cases. There is a comment (in Woods et al., 1975b, p.44) that, as a result of space problems during the semantic association phrase, the system was unable to complete processing any utterances. (The comment referred to tests with the travel budget task, which may have had worse space problems for semantic association than the previous lunar rocks task.) By 1976, SPEECHLIS was replaced at BBN by a new system called HWIM ("Hear What I Mean"). In the change, the control strategy was replaced, the semantic association component was dropped, and SPARSER was dropped.

The HWIM system is in several respects a reaction to the problems of SPEECHLIS. For instance, it attacks the problems related to coordinating knowledge sources by embedding much of its syntactic, semantic, and pragmatic

knowledge in a transition network representation. The result is called a 'pragmatic grammar' and has much in common with the 'task oriented grammars' of HARPY and Hearsay-II. General categories like noun phrase and verb phrase are replaced by task specific ones like 'meetings', 'trips', and 'expenses' (Woods et al., 1976b, p.54). The grammar has more states and arcs than the more general SPEECHLIS grammar because general categories have been split into special ones and also because many word arcs have been added. Like the CMU template grammars, large portions of the BBN pragmatic grammar would have to be rewritten for a different task domain (as they admit; Woods, et al., 1975b, p.23).

Another problem in SPEECHLIS was SPARSER's failure to use context sensitive restrictions until it had a complete constituent. HWIM has a partial solution to this problem based on the specification with each arc operation of the scope of its context dependency. For example, if arc A has a test that depends on a feature set by an action on arc B, the test on A is marked to show that its scope includes B. The HWIM parser executes context sensitive actions as soon as the parse path covers the necessary scope.

This method is an improvement over SPARSER, but it still is not as thorough in its use of contextual information as our technique of exploring the sentential context to set ratings. HWIM is better than SPARSER because it does not wait to complete a constituent before testing the relations among its subphrases. However, HWIM does wait until it has acquired complete phrases for the arcs in the scope of the operation, and often it is not necessary to wait that long. The tests often depend on phrase attributes that are determined long before the phrase is complete. To use an example from our system, the number attribute of a noun phrase can frequently be inferred from its determiner without waiting for the entire phrase to be built. If the sentential context calls for a singular noun phrase but the determiner indicates a plural one, our method of context checking recognizes the inconsistency when it sets the rating for the incomplete noun phrase, whereas the HWIM method would not notice the conflict until it had a complete noun phrase and tried to use it in the sentence phrase.

HWIM depends on the existence of an efficient and effective lexical retrieval component. In providing one, Klovstad has produced a particularly interesting part of the system (see Klovstad in Woods et al., 1976b). Significant features of the lexical retrieval program include the following:

* It finds the n best matching words without requiring exhaustive testing of all the words in the vocabulary. (The processing time varies with the log of the vocabulary size rather than linearly.)

* It can take advantage of syntactic predictions to constrain its search for matches.

* It makes effective use of phonological word boundary rules by precomputing their possible effects. This technique eliminates the need to make very loose judgments at word boundaries to compensate for possible coarticulation effects.

The internal dictionary for lexical retrieval is stored as a tree structure merging common phonetic sequences. A tree with initial sequences merged is used for left-to-right searches; another tree merging final sequences is used for right-to-left searches. Word boundary rules are applied to the trees to reflect the possible coarticulation effects. To allow selective retrieval according to syntactic category, each node in the dictionary tree is tagged with a bit mask showing the categories of words in that branch.

The lookup procedure matches paths through the dictionary tree against phonemes in the segment lattice. When a path reaches the end of a word spelling, a match has been found. The matching operation allows for 'merges' and 'splits' to take care of possible segmentation errors. In a merge, a single phoneme accounts for two adjacent acoustic

segments. In a split, two adjacent phonemes map onto a single acoustic segment. Path scoring penalizes matches that require splits or merges. Paths are eliminated from consideration by three operations: (1) syntactic selection -- if no words in the selected syntactic categories are reachable by the path, eliminate it; (2) forward pruning -- if the path is judged unlikely to produce words with scores better than some threshold, eliminate it; and (3) finite memory -- if the path score falls relative to the others so that the path is no longer among the k best paths (where k is a parameter), eliminate it.

The lexical retrieval component was tested on 99 sentences with a dictionary of 702 words. Directed to return up to 15 matches per sentence, it selected an average of 9.48 distinct matches per sentence. Overall, 23.7% of the words returned were correct, and in about 70% of the sentences the best match was correct. In only about 4% of the sentences did the lexical retrieval process fail to find at least one correct word. The processing times for these unanchored scans is reported to be about one CPU minute for a 2.5 second utterance (Nash-Webber and Bruce, in Woods et al., 1976b, p.46). They also report that to do a search at a given position in the segment lattice with a 448-word vocabulary takes an average of 6.83 CPU seconds (Woods et

al., 1976b, p.49). We are especially interested in the lexical retrieval component because it may present an efficient alternative to our map-all-words-at-once control strategy.

The HWIM island driving control strategy makes heavy use of the lexical retrieval component. HWIM starts processing an utterance by creating a segment lattice. Lexical retrieval then does an unanchored scan to find up to 15 of the best matching words. A one word theory is created from the best match, and the other matches are reserved for later use if the first one runs into trouble. There is no attempt to use semantic associations to form multiword theories as was done in SPEECHLIS. Instead, syntax is immediately called with the theory to make predictions to extend it. New theories are formed by adding words to old ones or by starting another one-word theory using one of the original words from lexical retrieval. There is little sharing of information among islands. One exception occurs if two islands 'collide' by growing together. In this case, the words from the two islands are joined together in a single operation. Other than this, the different islands do not interact, and there is none of the merging of work on common subparts that we have in our parse net or that SPARSER had in its shared configurations and transitions.

By giving up sharing, the parsing becomes simpler, but there are combinatorial problems if a large number of islands must be considered.

In the areas of scoring and priority setting, HWIM is more sophisticated than SPEECHLIS. HWIM uses experimentally determined statistical scoring. The use of probabilities provides a uniform scoring technique so that scores from different sources can be compared and combined in a theoretically sound manner.* On the basis of these scores, HWIM uses a 'shortfall density' technique for setting priorities. This technique depends on having a lexical retrieval component that can find the best matching words spanning the input. The shortfall method begins by finding the best spanning words and uses them to set an upper bound score for each segment of speech. The 'shortfall score' for an island is the difference between the sum of the upper bounds for segments in the island and the actual scores for words in the island. (Scores are additive in HWIM.) The 'shortfall density' for an island is its shortfall score divided by its length. The priority for working on an island is determined by the island's shortfall density: the smaller the shortfall density, the higher the priority.

---

* The change to probabilistic scoring began during the end of the SPEECHLIS period at BBN (see Klovstad in Woods et al., 1975a, pp.33-39).

The control strategy using shortfall density priorities is guaranteed to find the optimal interpretation as its first spanning island. In the terminology of heuristic search, this is an 'admissible' strategy (see Hart, Nilsson, and Raphael, 1968). However, the price of admissibility for this method appears to be a relatively breadth-first search. BBN has explored a number of variations to try to get better performance, and, as of September 1976, their favorite variant is one that sacrifices the guarantee of finding the optimal island first. It restricts the island seeds to be near the left end of the utterance. After a seed word is selected, the island is first extended to the left boundary of the utterance and then to the right. This method results in primarily left-to-right processing, so there is less chance of duplicating effort by creating the same island in different ways. It avoids the potential problems of a strictly left-to-right method because it can jump over the start of the sentence to pick a seed word. This compromise may offer a way around some of the problems with island driving that we report in the next chapter.

To summarize, HWIM has introduced a variety of interesting design concepts, particularly in the areas of lexical retrieval, island parsing, probabilistic scoring,

and priority setting. However, most of these concepts still need to be thoroughly tested to determine their effect on the performance of the system. In our own work, we have discovered that intuitively appealing design features can sometimes have distressing effects on actual performance.

We conclude the discussion of HWIM with some comments regarding the use of a 'pragmatic' grammar. Nash-Webber and Bruce state that the overall performance of HWIM is improved by fusing syntactic, semantic, and pragmatic knowledge sources, and that the improvement "comes from being able to constrain as soon and as tightly as possible the acceptable ways in which a given theory can be extended" (Woods et al., 1976b, p.53). They go on to say that "a naive conception of KS [knowledge source] interaction, which assumes that if communication channels exist, they will be used effectively, is wrong, at least in terms of currently realizable systems of HWIM's size and complexity" (Woods et al., 1976b, p.55). On this basis, they propose a principle that they dub "WORK TOGETHER":

> If it is found that one must frequently consider simultaneously information from several KSs, then the activity of those KSs should be tightly coupled. (p.55)

We agree that tight coupling of knowledge sources is important; we have emphasized that fact in the design of our systems since we began working on speech understanding (see, for example, comments in Walker, 1973). We are willing to believe that pragmatic grammar improves HWIM's performance (although Nash-Webber and Bruce offer no evidence supporting the claim). It may also be true that the improvement comes from the source they suggest -- namely, from being able to apply constraints as soon and as tightly as possible (again, no evidence is given). An alternative explanation might be that the pragmatic grammar improves performance by greatly reducing the generality of the input language: fewer choices, so better results. However, if Nash-Webber and Bruce are correct in their claim that the improvement is caused by better use of constraints, then the improved performance can be attained without resorting to a pragmatic grammar. In our system, explicit checking of the sentential context as part of setting ratings leads to early and effective use of constraints from all relevant sources of knowledge. Constraints are actually applied earlier by our technique than in HWIM since we do not require completing constituents before considering their interrelations.

3. EARLIER SRI SYSTEMS

This section gives a brief sketch of the evolution of our system design and mentions some of the techniques that were tried and then modified or discarded. The emphasis of the sketch is on unpublished material, but we also review work discussed more fully in earlier publications.

Our 1972 system (see Paxton and Robinson, 1973) tried to minimize the demands on acoustics by restricting the acoustic processing to testing words that had been hypothesized on the basis of other knowledge. The hope was that by hypothesizing words in roughly the order of their likelihood in a particular context, we could reduce the average number of errors in acoustic recognition. To achieve flexibility in ordering hypotheses, the system used a best-first strategy rather than depth-first with backtracking. The language definition was written in a procedural style following Winograd (1971; in fact, an earlier version of our system had been constructed by making modifications to Winograd's SHRDLU; see Walker 1973a,b). There was some effort to share information among different attempts to parse an input, but sharing was limited to successes rather than failures or partial results. Rating of alternatives was done by associating priority functions

with alternatives at choice points. The original intention was to implement the system using 'spaghetti stacks' (see Bobrow and Wegbreit, 1973), but since that facility was not available in time, an interpreter was written instead.

During 1973 we increased the amount of interparse cooperation in the system by introducing a limited version of Kaplan's producer-consumer scheme (Paxton, 1975; also see Kaplan, 1973b). It was restricted to a single level of producers; in other words, the producers could not be consumers too. Also, the context dependency within producers was restricted to the lexical level. Global control was still embedded in the procedural grammar. Performance results for this system are reported in Walker (1974, 1975). With a 54 word vocabulary, the system got 44 of 71 sentences correct (62%).

In 1974 we began working with SDC on a joint project. This change led to a major redesign influenced by SDC's previous work (see Barnett, 1973; Ritea, 1975) and the new Hearsay-II design (Lesser et al., 1974). The SDC parser was able to work top-down, bottom-up, and bidirectionally, and we decided to provide that much flexibility in our new system also. This decision required a change in the language definition methodology and resulted in the adoption of augmented phrase structure rules. From HS-II, we got the

idea of distinguishing between ratings (reflecting 'goodness') and priorities (reflecting 'importance'). However, we still thought of focus of attention in terms of suspending and reawakening processes and had trouble specifying why, when, and how, such operations should take place. By the end of the year, we replaced the ideas of suspending and reawakening by the idea of changing priorities and designed focus by inhibition in essentially its current form. The original version used entire phrases for focus, but when that resulted in a great number of incorrect phrases in focus, we shifted to focusing on words selected from phrases. Also during this period, discussions with Barnett at SDC led to the ideas of phrase mapping and lexical subsetting, the former to deal with coarticulation effects and the latter to give the system more guidance from acoustics.

By late 1974, a system was implemented based on augmented phrase structure rules and a general producer-consumer structure for parsing (see sections II and III in Walker et al., 1975). However, the rules did not allow options or alternatives, and the attributes and factors were defined by simple lists of assignment statements. The producer-consumer structure for the parse net took care of cooperation among different attempts to parse an utterance,

but it increased the problem of using contextual restrictions in setting priorities. From the beginning, we felt we had to take the context into account, but our first effort was a failure. Rather than searching up the consumer tree as we do now, we tried passing restrictions down the tree from consumers to producers. To illustrate, let C be a consumer phrase with an acquired constituent A and a missing constituent B. A factor expression for C is in general a function F of attributes of A and B. A new function F' can be formed by fixing the arguments of F that depend on A to the actual values from A; thus, F' depends on attributes of B only and can be passed down to B-producers as a restriction from C. Similarly, B-producers can further modify the arguments of F' to yield restrictions to pass down to their producers. This method was implemented and debugged, but as soon as it was tried on tests simulating speech input, its extravagant use of storage for propagating restrictions showed that it would have to be replaced.

The system design steadily evolved during 1975. The tree search context checking technique was developed for setting ratings. The first version searched the entire consumer tree to find the best path; later, the current technique was developed to do a heuristic search for a near optimal path. The system had a relatively small task size;

for example, the prediction task added only the immediate producers for the highest priority phrase rather than adding the entire subnet of producers. The small task size was intended to give a great deal of control over the operation of the system; it did that, but it also caused the system to spend over half of its time calculating and recalculating priorities. By adopting a larger task size, the scheduling overhead has now been reduced to about 12% of the Executive processing time (Chapter IV, Section G).

Another problem was the lack of alternatives and options in rules. Without them, it was impossible to merge related rules. For instance, there were 10 S rules, 5 of which had initial NPs. There were five empty S phrase consumers to consider when setting ratings for initial NPs, and whenever an initial NP was found, five incomplete S phrases had to be constructed. After we redesigned the language definition using rules with alternatives and options, there was only one S rule with an initial NP. The use of alternatives and options in rules complicates the Executive, but it provides an important reduction in the number of rules and hence in the processing and storage requirements of the system.

Also during 1975, we added multiword lexical entries, lookahead, word task scheduling by lexical

subcategories rather than by individual words, and gap/overlap testing by syllables rather than by fixed duration. To improve efficiency, we began to use a preconstructed initial parse net and preconstructed empty phrases. The monitor subnet for island driving was added as more than just an efficiency measure; it provides the necessary consumer context for island driving so that all consumer paths lead to a root-category phrase.

In late 1975 and early 1976, we put together a version of our system on the SDC IBM 370/145.* Before thorough testing could be done, SDC's computer was removed for administrative reasons, and our joint effort with SDC came to an end. However, enough information about their acoustic processing routines was collected so that we could test our parts of the system on our own computer by using a simulation. The results of those simulation experiments are reported in Chapter IV.

---

* Ann Robinson was primarily responsible for transferring the system from our PDP-10 to the SDC IBM/370.

IV EXPERIMENTAL STUDIES

CONTENTS:

A. INTRODUCTION

This chapter discusses a series of experiments concerning our framework for speech-understanding. The experiments use a simulation of the acoustic processing component which is described in Section B. Sections C through E discuss the experiments. In Experiment 1, the main experiment of the series, the speech system is tested on a set of 60 sentences for all combinations of four control strategy design choices. The best configuration from Experiment 1 is tested again in the second experiment, allowing different sizes of gaps and overlaps between words

in the simulated acoustic processing. The third experiment studies the performance of the two most promising system configurations from Experiment 1, while varying vocabulary size and acoustic processing accuracy. Section F deals with detailed measurements of the Executive's performance for the best version of the system on Experiment 1. Section G reviews the conclusions drawn from the experiements and suggests directions for further study. This chapter assumes familiarity with the Executive System, at least to the level of detail given in the overview (Section C) in Chapter III.

B. MAPPER SIMULATION

The experimental results reported in this chapter are based on measurements of the system performance using a simulated version of the acoustic processing component called the 'mapper' (described in Chapter I, Section B, and in Bernstein, 1975). To review, the mapper carries out acoustic tests: given a predicted word at a certain location in the input, the mapper gives it a score between 0 and 100, indicating how well it matches the input (100 indicates a perfect match), and determines its beginning and ending boundaries rounded to the nearest 0.05 second. If the score is above a certain threshold (45 for the studies reported here), the word is accepted for possible use in

interpretations of the input. Words accepted by the mapper are either 'hits', words really in the input sentence, or 'false alarms', words accepted although not in the input.

There were compelling reasons for doing simulation experiments rather than testing the system with the actual mapper. First, it would have been impossible to do extensive testing with the actual mapper -- the time required would have been too great, both because of increased processing time and because of increased memory demands (leading to large delays for 'page swapping' by the time-sharing system). By using a simulation, we were able to do a large number of tests because the processing and memory demands of the mapper were eliminated. Also, we were able to study certain control strategies, such as mapping all at once, which would be too slow for use with the actual mapper, but which otherwise have good effects on system performance and suggest new system designs.

Another reason for doing simulation experiments was that our access to the actual mapper was cut off when the SDC speech-project computer was removed in March 1976. (This reason is certainly not a scientific one, but it illustrates the fact that not all decisions in research are determined by scientific considerations.) Luckily, there was a week between the time that the mapper started working

well enough to be tested and the time that the SDC computer was removed. During that week the data used in simulating the mapper were collected. The experimental studies were subsequently carried out on the SRI computer.

The mapper simulation used in the experiments reproduces the observed mapper performance statistics for the false-alarm rate, the hit scores, and the particular false-alarm words and their scores. Because of insufficient data, we cannot construct a simulation that reproduces more complex statistics such as the co-occurrence of various hits and false alarms or possible dependence of scores on the position within the utterance. Although we have no reason to believe that these more complex statistics are particularly important in determining the system performance, we cannot say with confidence that they are not, and, consequently, we do not claim that the observed performance levels in the following simulation tests are precise estimates of the system's performance with a real mapper. In view of this, the experiments are designed to emphasize comparisons between performance levels rather than basing judgments on absolute performance levels. For example, to judge some design feature F, we look at the difference in performance with F versus without it, rather than simply reporting the absolute performance observed with F.

Such comparative experiments are often useful as a way to judge the effects of a design feature, but they are especially important when simulating a major component of the system. For instance, if some property of the actual mapper is not reflected in the simulation, that lack may affect the performance levels of the versions of the system with and without some design feature. However, as long as both versions are affected in roughly the same way, conclusions about the design feature drawn from significant differences in performance in the simulation tests will probably be valid regarding performance with the actual mapper. In short, the simulation experiments should be good for making design decisions based on comparative judgments, but the absolute performance levels in the experiments must be taken with a grain of salt.

Data for the simulation were collected by calling the mapper for all of the words in the vocabulary at the start of an utterance and then at each position where a previously accepted word ended (independent of whether the previous word was a hit or a false alarm). This procedure resulted in testing the entire vocabulary at an average of about 16 out of the 20 possible positions per second of speech (recall that word boundaries are rounded to multiples of 0.05 second, so there are 20 possible ending positions per

second).* Overall, tests were made at 160 positions in 11 test utterances. For the 305-word vocabulary used in the following experiments, the mapper had 48 hits and 1564 false alarms. The false alarms were distributed throughout the vocabulary [229 of the 305 words (75%) were falsely accepted at least once], with small words like "a" and "the" each accounting for approximately 30 false alarms.

The false-alarm rate for the mapper was determined by counting the number of false alarms that fell entirely within a section of the input. For the 305-word vocabulary, the average rate was 114 false alarms per second of speech. Since there were about 3 hits per second of speech, this rate indicates that the mapper produced an average of almost 40 false alarms for each hit. Figure IV-1 summarizes the results for three vocabulary sizes.

As compensation for the high false-alarm rate, there were no 'misses' (cases in which the mapper failed to accept

---

*This procedure was originally designed to record the results of all mapper calls that might be made in a left-to-right parse. The intention was to use this information in place of the actual mapper in tests of the entire system. However, technical and administrative difficulties made it impossible to gather enough information to satisfy the original goal. If the original goal had been to provide data for a simulation of the mapper, the mapper would have simply been tested on the entire vocabulary across each utterance at 0.05-second intervals. The change in goals may have resulted in missing some potential false alarms because of the untested positions where no word ended.

MAPPER PERFORMANCE

	VOCABULARY SIZE (words)		
	305	451	823
False Alarm (FA) Rate	114	142	360
Mean FA Score	59	58	58
Total FAs in sample	1564	2026	4989
Words with FAs	229	326	654
Words with no FAs	76	125	169

Figure IV-1.

a correct word), and the mean hit score was higher than the mean false alarm score (73.5 versus 59.4), although both score distributions spread over the entire range, from near 100 down to the threshold of 45. The score distributions are shown in Figure IV-2.

The cumulative percentage distributions for the scores are shown in Figure IV-3. Note that a threshold of 55 instead of 45 would eliminate 45% of the false alarms but only 6% of the hits. We are not suggesting such an increase in the threshold, but it illustrates the extent to which the false alarms are found at low scores.

To simulate the performance of the mapper on a particular sentence, the words of the sentence were first assigned lengths in seconds of speech. Each word was then assigned a score picked from the hit scores actually

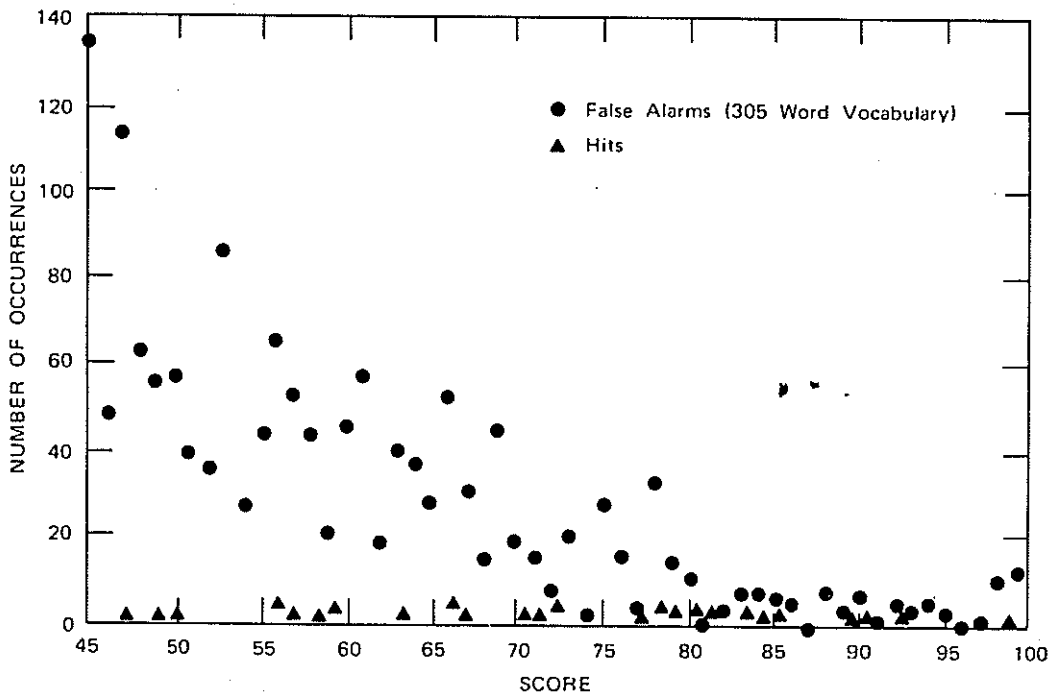

Figure IV-2. SCORE DISTRIBUTIONS FOR FALSE ALARMS AND HITS

produced by the mapper (each of the actual hits had an equal likelihood of having its score selected). The words were concatenated to determine the length of the utterance, and the length was multiplied by the false alarm rate to give the total number of false alarms to be simulated. The simulated false alarms were selected from the false alarms actually produced by the mapper and then assigned a position in the sentence. Each of the actual false alarms and each

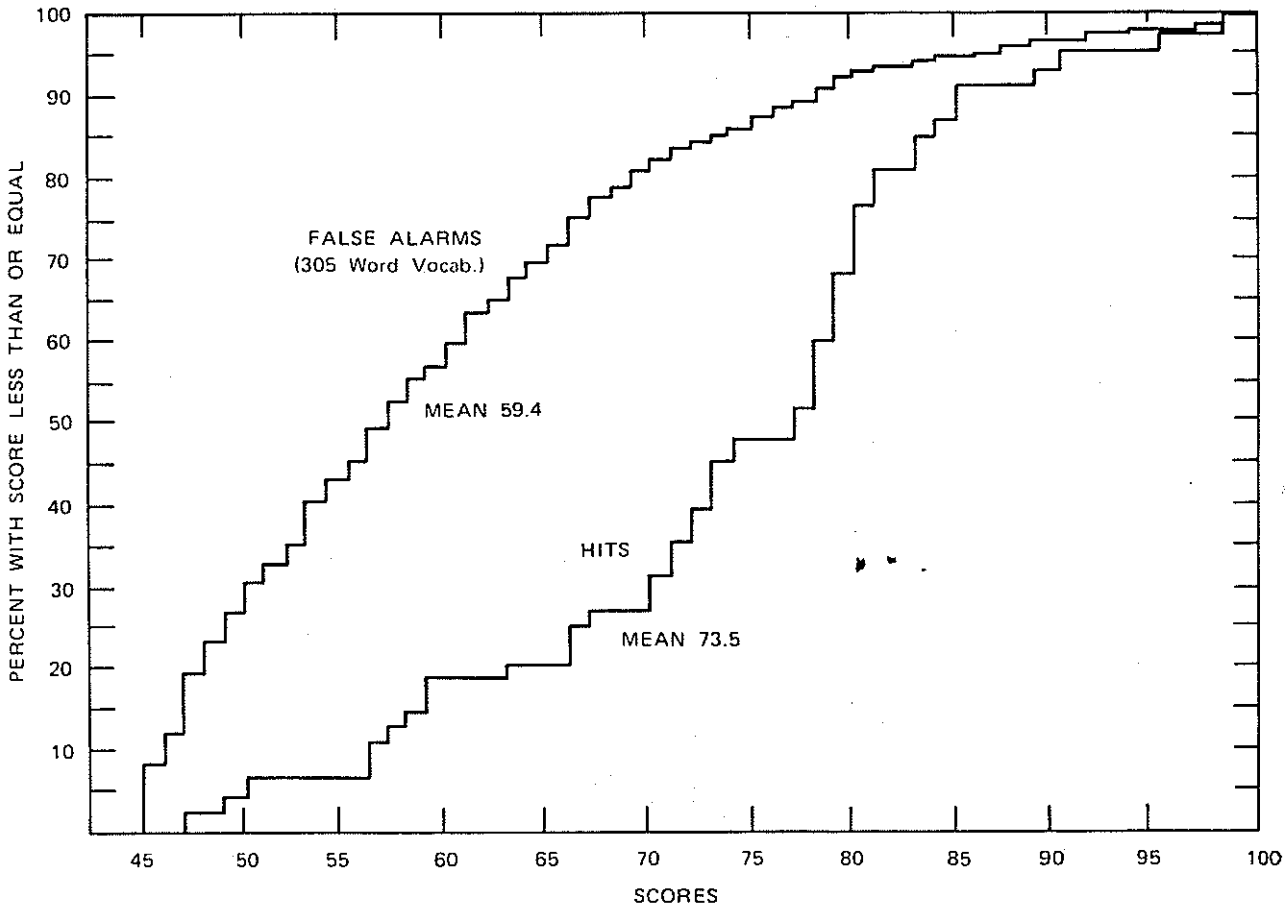

Figure IV-3. DISTRIBUTIONS OF HIT AND FALSE-ALARM SCORES

of the positions in the simulated utterance had an equal chance of being selected, except for a check to ensure that the word was not already at the chosen position.

In computing the simulated processing time for the mapper in later experiments, we used figures of 0.30-second processing per word tested, 1.0 second per position for

lexical subsetting,* and 10.0 seconds per second of speech in the sentence if 'island driving' was being simulated (see discussion of Experiment 1). These timing figures are based on rough measurements of the mapper running on an IBM System/370 Model 145.

C. EXPERIMENT 1 -- CONTROL STRATEGY DESIGN CHOICES

In this experiment, the performance of the speech system was measured on a set of 60 test sentences, while varying four major control-strategy design choices. The sentences covered a wide range of vocabulary and included questions, commands, and elliptical sentences (see Section H at the end of this chapter for a list of the test sentences). There were 10 sentences at each length of simulated speech ranging from 0.8 to 2.3 seconds at intervals of 0.3 second. The sentences averaged 5.9 words in length, with a maximum of 9 words. The choices used as experimental variables are described briefly in Figure IV-4 and extensively in Chapter III.

Recall that the lexical subsetting component uses local, robust acoustic cues to select a subset of the lexicon for further testing at a particular input location. To simulate this component, the system creates a subset containing the simulated hits and false alarms at or near the specified location and then adds randomly selected words to increase the size of the subset to a specified value. For the 351-word vocabulary, the subset size was set at 50 reflecting the expected performance of a well-tuned version of the lexical subsetting component.

EXPERIMENTAL VARIABLES

Island Drive or Not -- Going in both directions from arbitrary starting points in the input versus proceeding strictly left-to-right from the beginning: Island driving allows the system to use words that match well anywhere in an input and to build up an interpretation around them. Left-to-right processing is simpler and less flexible but may still be more accurate and efficient than island driving.

Map All or One -- Testing all the words at once at a given location versus trying them one at a time and delaying further testing when a good match is found: Mapping all at once lets the system know the best candidates from the acoustics and reduces the chances of following a false path. Mapping one at a time avoids exhaustive testing and will be more efficient than mapping all at once if the system does not encounter too many false alarms.

Context Checks -- Taking into account the restrictions of the possible sentential contexts as part of setting priorities versus ignoring the contextual restrictions except for use in eliminating already formed structures: Context checking should give more information for setting priorities and should lead to better predictions. However, the checks can be expensive and therefore may not lead to an overall improvement in performance.

Focus by Inhibition -- Focusing the system on selected alternatives by inhibiting competition versus employing an unbiased best-first strategy: Focus by inhibition allows the system to concentrate on a particular set of potential interpretations rather than switching among a large number of alternatives. However, if the focus of attention is too often wrong, the net effect may be harmful to system performance.

Figure IV-4.

All combinations of the four control-strategy variables were tested on the 60 sentences. This experimental design allows us to compare the 16 combinations of control choices and to evaluate, by analysis of variance, the main effects and interactions of the control strategy variables. The main effect of a variable is the change in performance it produces, averaged over all the possibilities for the other variables. The interaction of two variables tells whether the effect of one variable is the same for all possibilities of the other. The interaction of three variables tells whether the interaction of two of them is the same for all possibilities of the third, and so on.

Analysis of variance is a statistical technique for computing the probability that the observed effects or interactions are really caused by the experimental variables, rather than the result of random variation (see e.g., Winer, 1971; also, see Cox, 1958, for an introduction to experimental design). In other words, this method aids in evaluating results of experiments influenced by substantial random factors. In our case, the random factors include the random choices of false alarms and hit scores in simulating the mapper, and the selection of a particular sample of sentences from the much larger population of possible sentences. The statistical results for a main

effect or interaction are given in a form such as "F(1,5)=6.9, p < .05." This means that the F ratio (a statistic for comparing variances) for the effect or interaction has 1 and 5 degrees of freedom and has a value equal to 6.9. This in turn implies that the probability is less than .05 that the observed effect or interaction was caused by random variation alone. If the probability is given by itself in the following discussion, it is based on the these values: F(1,5)>=16.3 for p < .01, F(1,5)>=6.6 for p < .05, and F(1,5)>=4.1 for p < .10.

The most important performance measures for the system are accuracy (the percentage of sentences for which the correct sequence of words is found) and runtime (the computation required by the system, including simulated acoustic processing time). For these measures, the control strategy variables had large, significant effects. Before discussing the effects, we need to introduce a notation for naming the experimental designs. The capital letter "F" will refer to focus by inhibition, lower case "f" to no focus by inhibition; "C" stands for context checks, "c" for no context checks; "M" for map all at once, "m" for not map all at once; "I" for island driving, and "i" for no island driving. As an example of this notation, "fCMi" refers to the system that does not use focus by inhibition, does use

context checks, does map all at once, and does not island drive. Using this notation, Figure IV-5 shows the accuracy and runtime of the 16 experimental systems.

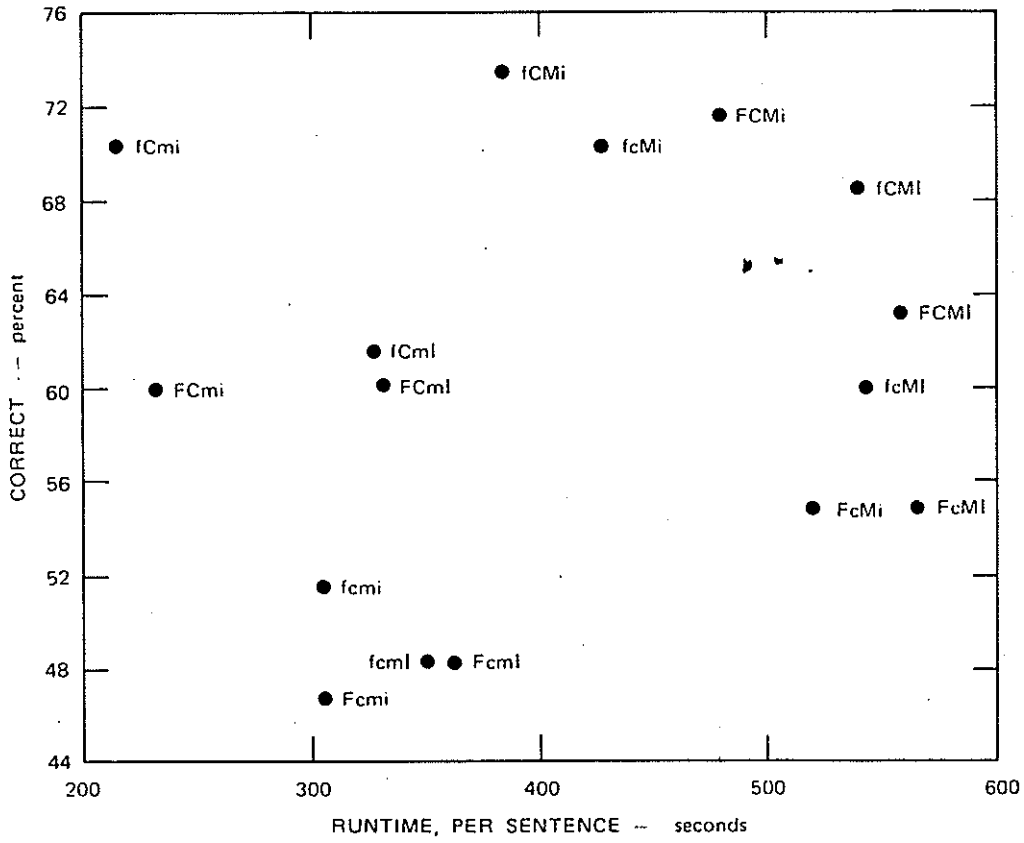

Figure IV-5. ACCURACY AND RUNTIME OF THE 16 DESIGNS

Notice the range of values for both measures, from 46.7% to 73.3% for accuracy, and from 221 to 559 seconds processing per sentence for runtime. These wide ranges confirm the importance of control strategy in determining system performance. With respect to the individual control

variables, comparing the C-systems to the corresponding c-systems shows that context checks for priority setting result in better accuracy and faster runtimes (see Figure IV-6). Similar comparisons show that mapping all at once improves accuracy but increases runtime (see Figure IV-7), while focus by inhibition and island driving both reduce the accuracy and increase the runtime (see Figure IV-8 and Figure IV-9). In the remainder of this section, we discuss these effects and propose explanations for them.

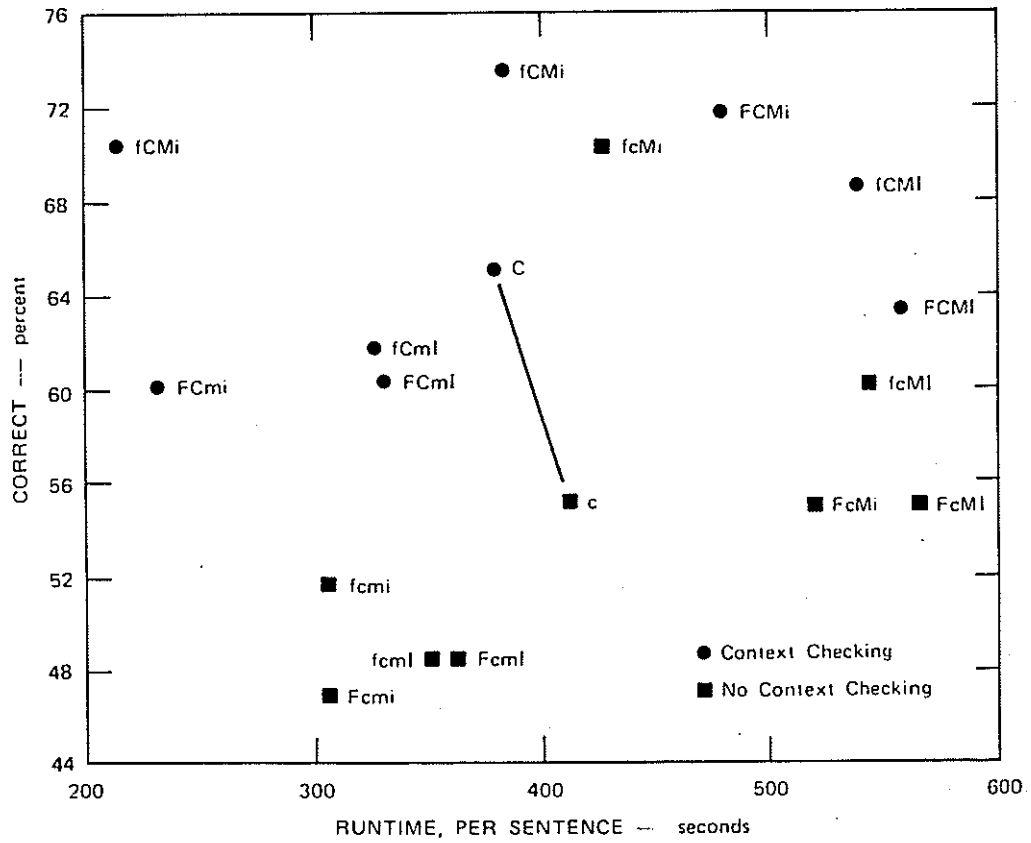

Figure IV-6. CONTEXT CHECKING -- MAIN EFFECTS

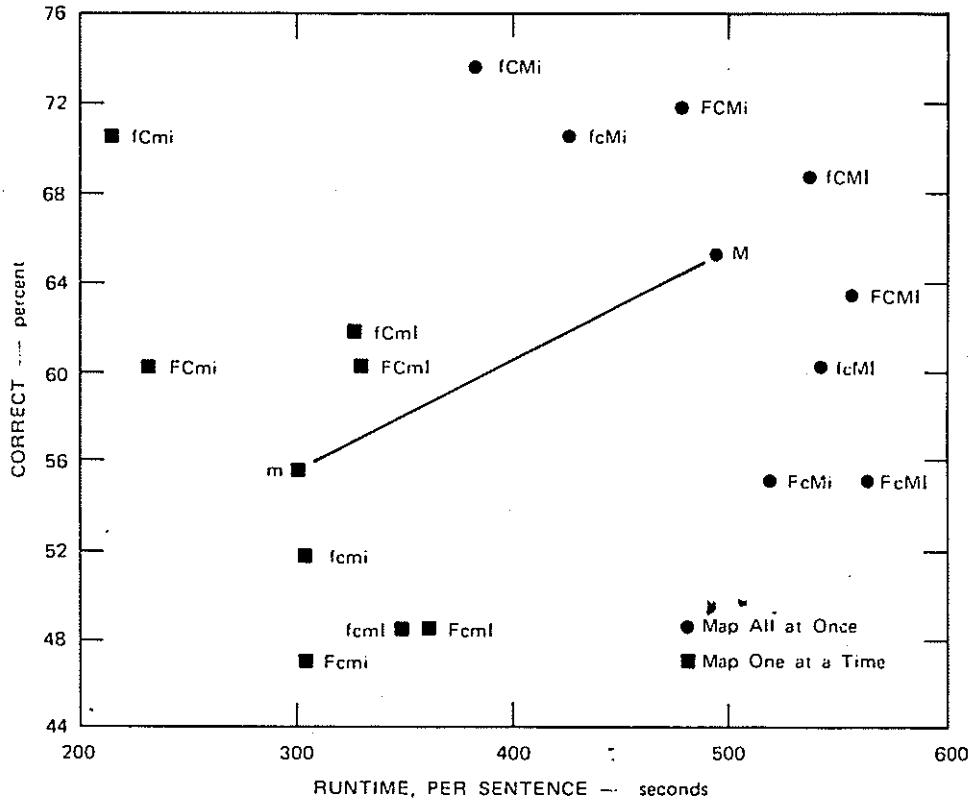

FIGURE IV-7 MAPPING ALL AT ONCE --- MAIN EFFECTS

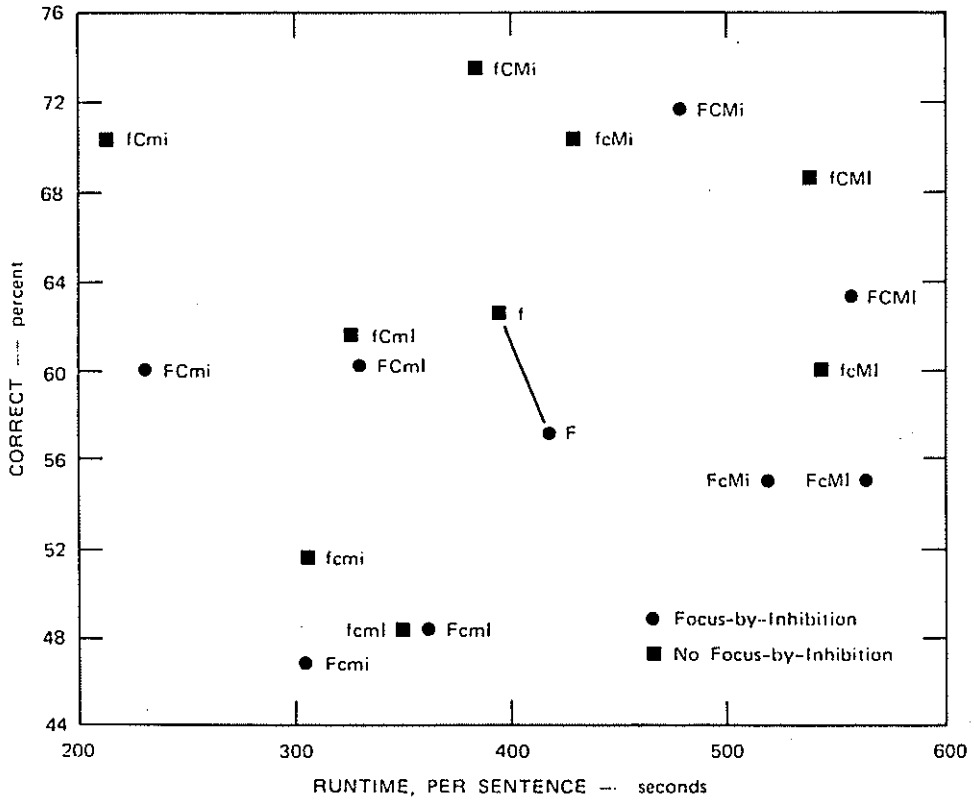

FIGURE IV-8 FOCUS-BY-INHIBITION --- MAIN EFFECTS

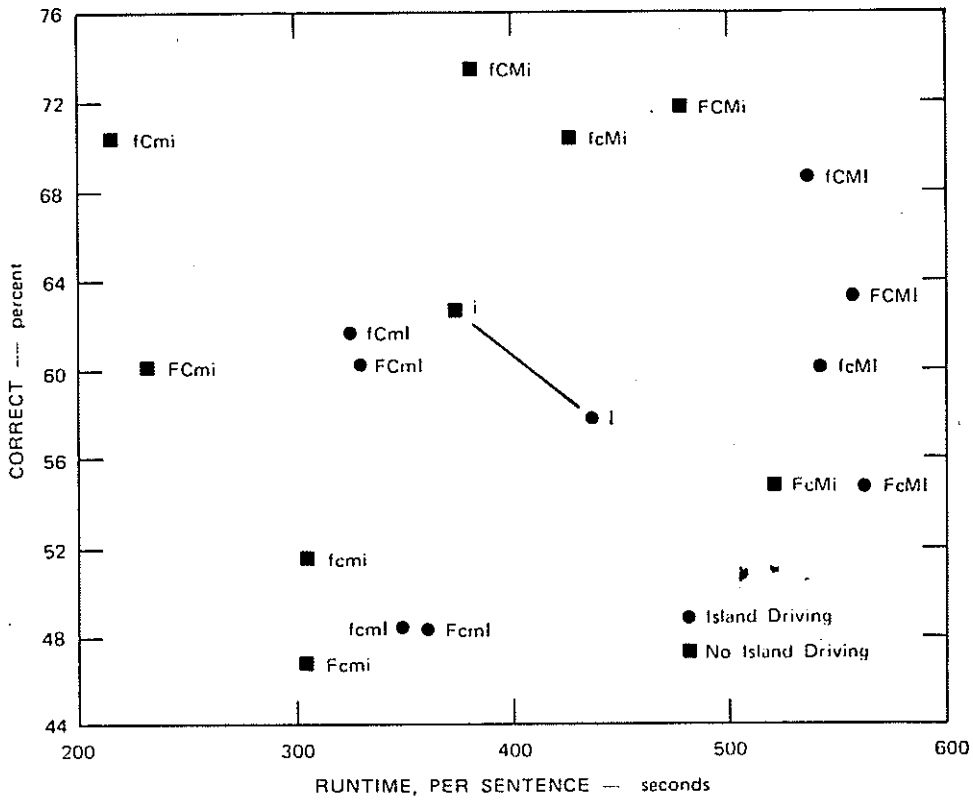

Figure IV-9. ISLAND DRIVING -- MAIN EFFECTS

1. ACCURACY

Figure IV-10 shows the effect of the control variables on accuracy. For the purposes of analysis of variance, we pooled the results on each set of 10 sentences of equal length to get six accuracy measures per system. The interaction with length was then used as the error term for calculating statistical significance.

As previously mentioned, context checking and mapping all at once both improve accuracy, while focus by inhibition and island driving make it worse. The island

MAIN EFFECTS OF VARIABLES ON PERCENT CORRECT

	WITH	WITHOUT	DIFFERENCE	
F	57.5	62.9	-5.4	*
C	66.0	54.4	11.6	*
M	64.6	55.8	8.8	*
I	58.1	62.3	-4.2	

* $p < 0.05$

Figure IV-10.

driving effect was not significant statistically because of a large interaction with sentence length. For the long sentences, 1.7 to 2.3 seconds, island driving decreased accuracy by 15.8%, but for the short ones, 0.8 to 1.4 seconds, it actually increased accuracy by 7.5% (see Figure IV-11). There was a significant interaction ($p < 0.05$) between focus by inhibition and island driving. As shown in Figure IV-12, the bad effect of island driving is less with focus, and the bad effect of focus is less with island driving. To explain this collection of results we must first consider how accuracy is influenced by control strategy.

The control strategy affects accuracy indirectly: all the strategies are 'complete' in the sense that they only reorder, and never eliminate, alternatives. If there were no false alarms, all the systems would get 100% of the

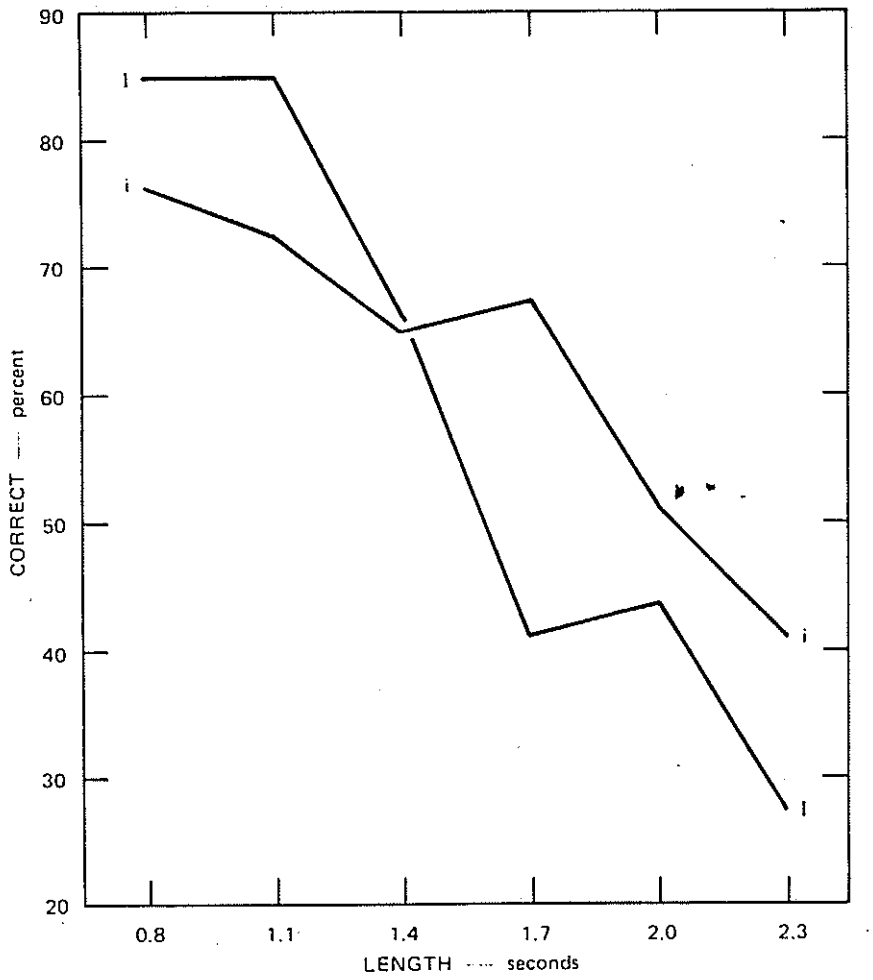

Figure IV-11. ACCURACY VERSUS LENGTH FOR ISLAND DRIVING

test sentences correct. Even with false alarms, the strategies would get an equal percent correct, if all the possible alternatives could be tried before the system picked an interpretation. Errors would only occur when false alarms had high enough scores to displace hits in the

FOCUS AND ISLAND-DRIVING INTERACTION

	I	i	I-i
F	56.7	58.3	-1.6
f	59.6	66.3	-6.7
F-f	-2.9	-8.0	

(percent correct)

Figure IV-12.

highest rated interpretations. However, in the actual system, the large number of alternatives makes it impossible to consider all of them in the space and time available. As a result, the order in which the alternatives are considered can affect the accuracy, and so can the demands on space and time. Control strategy thus affects accuracy indirectly by reordering alternatives and by modifying space and time requirements. To explain the accuracy effects, we must look at these other factors.

In this experiment, the storage limit had an important influence on accuracy. In the 960 tests (60 sentences times 16 systems), 578 (60.2%) were correct and 382 (39.8%) were wrong. Of the errors, 175 (46%) had an incorrect interpretation, while 207 (54%) had no interpretation at all. Since the systems could potentially get the correct answer, and no time limit was imposed until at least one interpretation had been found, all of the 207

sentences with no interpretation were a result of running out of storage.

The storage limit used in the tests was based on the number of phrases constructed. When the total reached 500, the system would stop trying new alternatives and, if any interpretation had been found, pick the highest rated interpretation as its answer. The average number of phrases constructed was 204 nonterminal and 63 terminal. The system with the best accuracy, fCMi, had the lowest average (113 nonterminals and 45 terminals), while the system with the worst accuracy, Fcmi, had one of the highest averages (260 nonterminals and 68 terminals). Overall, there was a strong negative correlation (-.93) between system accuracy and average number of phrases constructed (see Figure IV-13); the accuracy drops by about 1% for an increase of 6 phrases in the average storage requirements.

Figure IV-14 shows the effects of the control variables on the number of phrases. The pattern is the same as for accuracy; context checking and mapping all at once have good effects, while focus by inhibition and island driving have bad effects. Again, because of a large interaction with length, the island driving effect is not significant statistically. There are significant interactions, $p < 0.05$, between focus by inhibition and

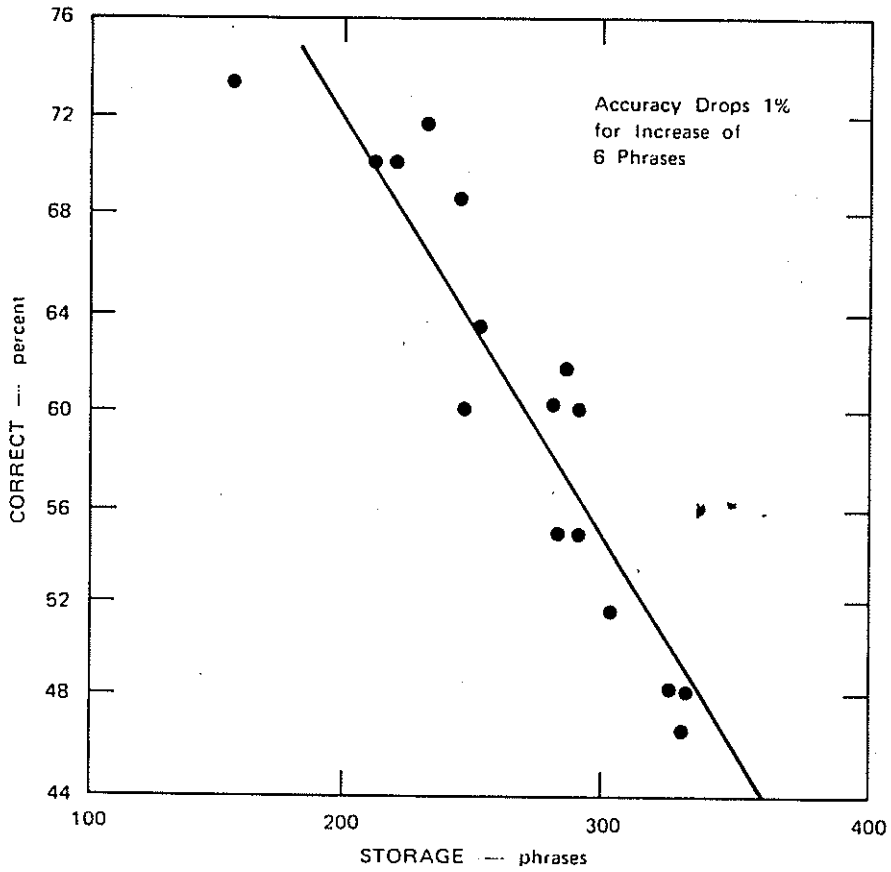

Figure IV-13. STORAGE AND ACCURACY FOR THE 16 SYSTEMS

MAIN EFFECTS ON STORAGE

	WITH	WITHOUT	DIFFERENCE	
F	281	253	28	*
C	240	294	-54	**
M	244	290	-46	**
I	287	247	40	

(number of phrases)

** $p < .01$ * $p < .05$

Figure IV-14.

FOCUS AND ISLAND-DRIVING

	I	i	I-i
F	290	272	18
f	284	222	62
F-f	6	50	

(number of phrases)

Figure IV-15.

island driving for storage, as seen in Figure IV-15, and between context checking and mapping all at once, as seen in Figure IV-16.

CONTEXT AND MAP-ALL INTERACTION

	M	m	M-m
C	221	259	-38
c	267	322	-55
C-c	-46	-63	

(number of phrases)

Figure IV-16.

The beneficial effects of mapping all at once are caused by a reduction in the proportion of false alarm terminal phrases. Mapping all at once significantly reduces the proportion of terminal phrases that are false alarms -- from 88.0% to 85.7%, $p < .01$. The false terminal proportion is in turn significantly correlated with the number of phrases (.72) and the accuracy (-.75). When the words are

all mapped at once at a given location, the system is able to take advantage of the difference in false alarm and hit score distributions to reduce the likelihood of constructing false terminal phrases. Notice that a relatively small change in false terminal percentage has a large effect on system performance.

Surprisingly, context checking also results in a significant reduction in the false terminal percentage -- from 87.5% to 86.2%, $p < .01$. This reduction may be evidence that context checking is giving lower priority to looking for words adjacent to false alarms than it gives to looking next to hits. This change could affect the false terminal likelihood, since there is always a hit adjacent to a hit, while false alarms often have nothing but other false alarms next to them. In addition to its effect on false terminals, context checking may also be improving the storage requirements and accuracy by generally improving the priority setting, thereby reducing the likelihood of following false paths.

Focus by inhibition slightly increases the proportion of false alarm terminal phrases (from 86.3% to 87.3%), but this increase is not statistically significant. The explanation of the ill effects of focus is essentially the converse of the explanation of the effects of context

checking. Context checking makes performance better by improving priorities, while focus by inhibition makes it worse by distorting priorities. Focus too often changes priorities to bias the system in favor of a false alarm instead of a hit. In the F-systems, there was an average of 3.5 hits put in focus per sentence compared to 12.9 false alarms. Focus conflicts changed priorities in favor of a false alarm 112 times per sentence and in favor of a hit, only 15 times per sentence. Thus the priorities, and the system performance, were better with the unbiased best-first strategy than with focus by inhibition.

Island driving did not affect the false terminal proportion, but it did have bad effects on storage and accuracy for the longer sentences. To get a sentence correct, island driving must start at least one island with a hit. If all the seeds -- words selected to start islands -- are false alarms, the sentence will not be interpreted correctly. The overall average was 3.7 false alarm seeds per sentence and 0.9 hit seeds. There were one or more hit seeds in 82% of the tests using island driving. The bad effects of island driving on long sentences was not caused by an increase in the number of false alarm seeds. The average rank of the first hit in the sequence of words for use in forming islands was 4.8, and the rank did not

increase with sentence length. (The correlation between rank and length was .04). For sentences 1.7 seconds or longer, instead of an increase in the number of seeds necessary to get a hit, there was an increase in the amount of storage consumed per island. Perhaps the greater length allowed islands to grow in both directions, whereas in shorter sentences the sentence boundaries blocked one direction or the other.

The interaction of focus by inhibition and island driving can be explained as the result of the storage limit. The limit put a ceiling on the size of the possible combined effect. The limit was reached frequently, and, thus, the combined effect was less than the sum of the individual effects. Similarly, the interaction between context checking and mapping all at once is a result of overlapping good effects, which consequently fail to add. The same interaction pattern for context checking and mapping all at once appears in false terminal proportion, $p < .05$, and in accuracy, $F=4.00$ versus $F(1,5)=4.06$ for $p < .10$.

We now turn to a brief analysis of the sentences that got one or more interpretations but were incorrect because their highest rated interpretation was wrong. As mentioned previously, this happened in 175 tests. In 109 of these (62%), the chosen interpretation was reasonable

linguistically but contained incorrect words. In 10 tests (6%), the chosen interpretation could have been eliminated by a better (i.e., more restrictive) language definition ("Was feet one builder of the Farragut?" is an example from these 10). Finally, 56 of the errors (32%) were harmless, in that the system would probably produce the same answer as if it had found the correct sequence of words (e.g., "What reactor does it have?" instead of "What reactors does it have?" is a typical example of these harmless errors). If the harmless errors are counted as correct in calculating the accuracy, the most accurate system, fCMi, increases in percent correct from 73.3% to 81.7%, and the average accuracy for all the systems goes up about 5.8%.

The accuracy effects have been explained in terms of storage requirements, proportions of false terminal phrases, and priorities. The important role of the storage limit raises the question of whether the accuracy effects would have disappeared if more storage had been available. We believe that the effects would have been smaller but still important. The effects on the proportion of false terminal phrases would remain, as would the effects on priorities. A smaller percentage of false terminals and better priorities will cause the system to find the correct interpretation sooner, and, even if the storage limit were

relaxed, the limit on runtime would remain to penalize systems that were slow to find the right answer. The effects of control strategy choices on accuracy would only vanish if space and runtime limitations were both removed.

2. RUNTIME

The system runtime is another important performance measure. Here, we will use the phrase 'total runtime' to refer to the simulated acoustic processing, plus the actual processing time (on a DEC PDP KA-10) for the Executive and the semantic components. The Executive time is mainly spent setting priorities and parsing. The semantics time is used in constructing semantic translations and in dealing with anaphoric references and ellipsis. The reported total runtime does not include acoustic preprocessing or question answering, since neither is affected by the experimental variables.* We report results only for total, Executive, and acoustic times; semantic times are not reported, both because they are redundant given the other three measures, and because they are relatively small in comparison to the others. In analysis

---

* Acoustic preprocessing takes about 160 CPU seconds per second of speech on a PDP 11/40 (according to personal communication with Iris Kameny at SDC). The time for question answering varies greatly with the complexity of the request.

of variance of the runtimes, interaction with length was used as the error term.

The main effects of the control variables on total runtime are given in Figure IV-17. All the variables except context checking increase the runtime. Dividing the sentences into a short group (0.8 to 1.4 seconds) and a long group (1.7 to 2.3 seconds) shows that island driving has a much worse effect on runtime for long than for short sentences. For short sentences, island driving increased the mean runtime from 262 to 290 seconds, a difference of 28. For long sentences, the increase was from 457 to 598 seconds, a difference of 141. Recall that for long sentences island driving also had worse effects on accuracy and storage.

MAIN EFFECTS ON TOTAL RUNTIME

	WITH	WITHOUT	DIFFERENCE	
F	417	386	31	*
C	383	421	-38	**
M	498	305	193	**
I	444	359	85	#

(seconds per sentence)

* $p < .05$ ** $p < .01$ # $p < .10$

Figure IV-17.

EFFECTS ON EXECUTIVE RUNTIME

	WITH	WITHOUT	DIFFERENCE	
F	120	106	14	**
C	109	117	-8	#
M	90	135	-45	**
I	127	98	29	#

(seconds per sentence)

** p < .01 # p < .10

Figure IV-18.

EFFECTS ON ACOUSTIC RUNTIME

	WITH	WITHOUT	DIFFERENCE	
F	276	260	16	#
C	254	282	-28	**
M	389	147	242	**
I	295	241	54	#

(seconds per sentence)

** p < .01 # p < .10

Figure IV-19.

Figure IV-18 and Figure IV-19 show the main effects on Executive runtime and acoustic runtime, respectively. In both cases, context checks decrease the runtime, while focus by inhibition and island driving increase it. Mapping all at once improves the Executive runtime but leads to a huge increase in acoustic processing time. As usual, examination of the results according to

sentence length shows that island driving is worse for longer sentences. The average Executive and acoustic times together account for 95% of the average total, so, as mentioned previously, we do not report separate effects for semantics.

Analysis of variance for total, Executive, and acoustic runtimes reveals a significant interaction between context checking and mapping all at once ($p < .01$ for total and acoustics; $p < .05$ for Executive). For total and acoustic runtime, the good effect of context checking was reduced when words were mapped all at once at each location, and the increase in runtime caused by mapping all at once was greater when also context checking. For Executive runtime, both context checking and mapping all at once had good effects, and there was actually a synergistic relation; context checking helped more when mapping all at once, and vice versa.

The runtime results follow basically the same pattern as the accuracy and storage results. Focus by inhibition and island driving have bad effects, with worse results from island driving for longer sentences, while context checking has consistently good effects. Mapping all at once has a good effect on Executive runtime, but, unfortunately, it causes large increases in acoustic and

total runtimes. The only inconsistency with the previous pattern of effects for accuracy and storage is the bad effect of mapping all at once on the acoustic runtime. This fact is explained by pointing out that the mapper was designed for checking words one at a time and, in the simulation, does not accumulate or share information to make subsequent tests more efficient. Finally, it is noteworthy that the extra effort for context checking resulted in a net decrease in processing time. For example, fCMi spent an average of 6.3 seconds more per sentence doing extra processing for context checks, but it was still 41 seconds per sentence faster than fcMi.

The runtime figures above are in units of seconds used to process a sentence. A more common unit for runtime is seconds per second of speech. This is a reasonable scale if the runtime can be approximated by a linear function of sentence length with a zero intercept. Both assumptions, linearity and zero intercept, are consistent with our data. No significant nonlinearity was found by an F test of the variance of the mean for each length about the regression line, relative to the combined variance of the sentences within a given length (for instance, the data for fCMi gave $F=1.37$ versus $F(4,54)=1.41$ for $p < .25$). Moreover, the 95% confidence interval for the intercept of the regression line

included the origin. With this justification, we used zero-intercept linear regression to calculate the processing times per second of speech and their 95% confidence intervals.

The results for the fastest system, fCmi, were 141, plus or minus 14 seconds processing per second of speech for total runtime; 66, plus or minus 9 for Executive runtime; and 63, plus or minus 7 for acoustic runtime. The results for the most accurate system, fCMi, were 247, plus or minus 21 for total runtime; 34, plus or minus 6 for Executive runtime; and 205, plus or minus 14 for acoustic runtime. Thus, for fCMi, 83% of the total runtime slope comes from acoustic processing, 14% from the Executive, and the remaining 3% from the semantics (Figure IV-20). Clearly, the best approach to improving fCMi runtime is to redesign the mapper for mapping all at once. The potential is large for sharing work to improve efficiency in the mapper, since the data show that fCMi is mapping all the words at an average of 13 out of the 20 possible positions per second of speech.

3. REVIEW OF THE EFFECTS

The only control strategy choice with mixed effects in Experiment 1 is whether or not to map all at

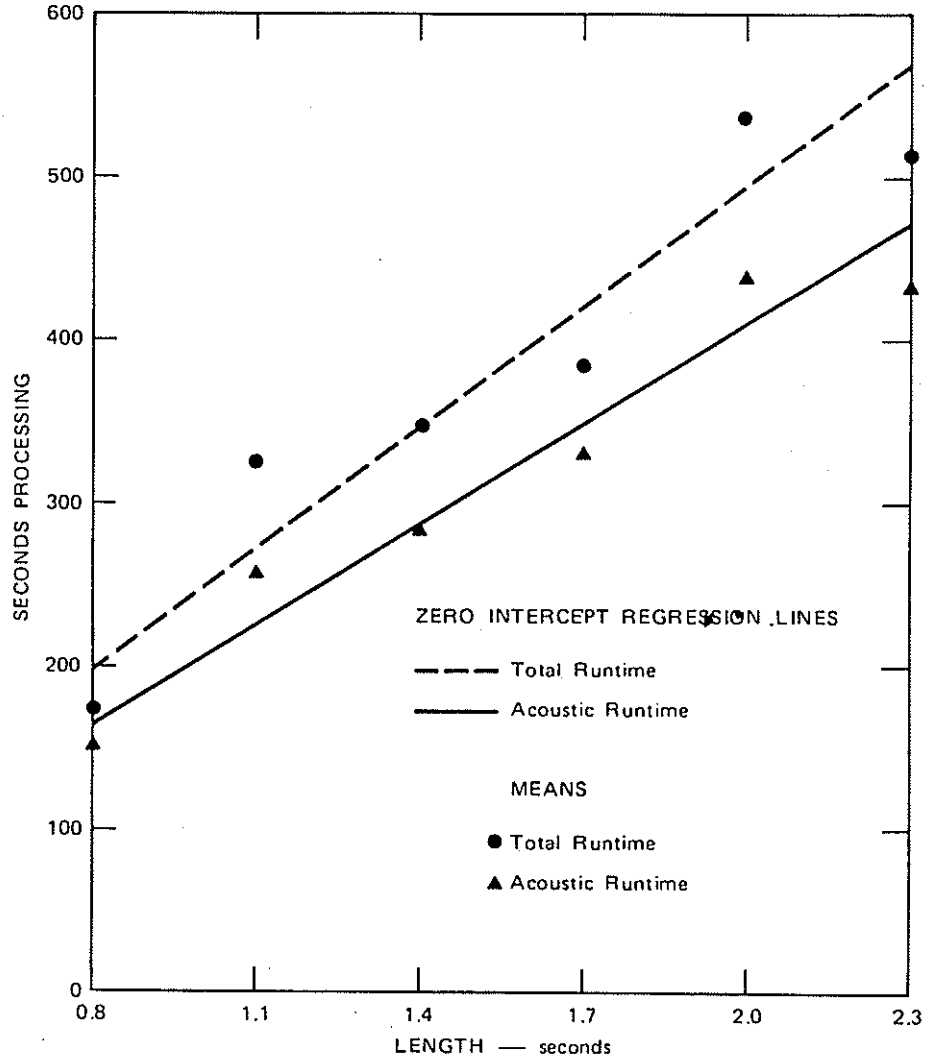

Figure IV-20. MEAN fCMi RUNTIMES

once. Mapping all at once improves accuracy and Executive runtime, but at a large cost in acoustic and total runtime. Redesign of the mapper, perhaps along the lines of the BBN lexical retrieval component (see Klovstad in Woods et al., 1976b), could undoubtedly resolve this choice in favor of mapping all at once. For example, just cutting the acoustic processing in half would make the fCMi system about as fast as the fCmi system. The choice, whether to map all or not, is explored further in Experiment 3.

The overall effects of island driving were bad, and they were particularly bad for longer sentences. Island driving was hurt by false alarm seeds, especially when the sentence was long enough for the islands to grow in both directions. Perhaps island driving can be modified to overcome this problem. For example, a multiword seed technique like Hearsay-II's (see Chapter III, Section E.2) might reduce the number of false alarm seeds, and a restriction like the one used by BBN to keep seeds near the start of the utterance might reduce the storage needed per island. Another alternative would be to pick seeds anywhere in the utterance but to restrict them to growing in one direction to an utterance boundary before allowing them to grow in the other direction. Island driving did give higher accuracy for the shorter sentences and correctly answered some of the sentences that fCMi missed (see Section F), so further effort is justified to look for versions of island driving that have good effects for long sentences as well as for short ones.

The effects of focus by inhibition were bad on all measures. The cause of the bad effects was too much focusing on false alarms, so we have tried a modified version that is much more conservative about which words to put in focus. It uses the false-alarm likelihood estimates

as a primary criterion in selecting words for focus. The modified focus method was tested on the 60 utterances using the FCMi system, which was the best of the original focus systems. The results are shown in Figure IV-21. The modification greatly reduced the number of false alarms in focus and improved the FCMi performance of all measures. In fact, the modified-FCMi is the most accurate of all the systems tested; it correctly answered all the sentences that fCMi did, plus one more. However, it is still slightly worse than fCMi in storage and runtime. The differences between fCMi and the modified-FCMi are small (because so few words are put in focus by the modified technique), but they suggest that focus by inhibition might have significant good effects if further effort was devoted to tuning the algorithm for selecting focus words.

Context checking had uniformly good effects. For both accuracy and runtime, it was worth the extra effort to get better priority setting. This result clearly depends on the fact that we put a large amount of the system's knowledge into the rule procedures of the language definition rather than into the structural declarations. It would be interesting to repeat these tests with different language definitions that had the same linguistic scope but put more information into the structure and less in the procedures.

EFFECTS OF MODIFIED FOCUS BY INHIBITION

	new-FCMi	old-FCMi	fCMi
Words in Focus			
Hits	1.3	3.9	0.0
False Alarms	1.6	8.2	0.0
Focus Conflicts			
Hits	4.0	13.3	0.0
False Alarms	12.8	75.8	0.0
Raw Accuracy, %	75.0	71.7	73.3
Forgiving, %	81.7	76.7	81.7
Runtime (sec/sent)	392	477	385
Executive "	60	83	53
Acoustic "	321	377	320
Storage (phrases)	165	231	158

Figure IV-21.

D. EXPERIMENT 2 -- GAPS AND OVERLAPS

The mapper performance data described in Section B do not aid us in simulating the mapper when it is called on to test whether two words accepted individually are also acceptable as a contiguous pair. Such tests, referred to as 'phrase mapping', are necessary whenever words and phrases are combined to form larger units. In the simulation of the mapper, we replaced phrase mapping by a simple threshold test; we allowed gaps and overlaps of up to 0.05 second of speech, but rejected those that were larger. Experiment 2 tests the effect of different values of the gap-overlap parameter on the performance of the fCMi system from

EFFECTS OF GAP-OVERLAP PARAMETER

	GAP-OVERLAP SIZE		
	0.00	0.05	0.10
Raw accuracy, %	96.7	73.3	48.3
Forgiving accuracy, %	98.3	81.7	58.3
False terminal, %	58.2	83.2	89.1
Number of nonterminals	31	113	217
Total runtime (sec/sec-speech)	140	247	333
Executive runtime "	10	34	69
Acoustic runtime "	128	205	243

Figure IV-22.

Experiment 1. Figure IV-22 gives the results for a variety of measures with gap-overlap sizes of 0.00, 0.05, and 0.10 second.

The performance is much better for 0.00 and much worse for 0.10 second of gap or overlap. The observed distribution of gaps and overlaps is shown in Figure IV-23. Notice that a technique using a simple threshold on the size of gaps and overlaps would not be acceptable in practice; the threshold would have to be at least 15 centiseconds, and the data reported in Figure IV-22 suggest that the resulting performance would be terrible. This is strong evidence for the importance of special acoustic tests to verify word-pair junctions. Such tests can lead to a large reduction in the average hit rank and, consequently, to significant improvements in both accuracy and runtime.

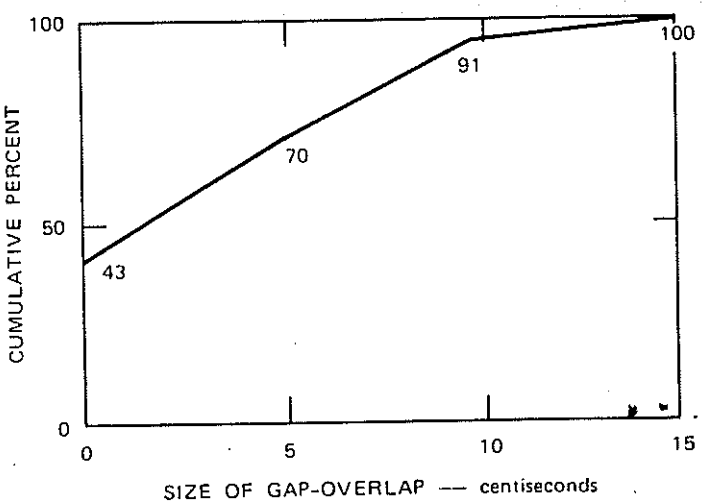

Figure IV-23. OBSERVED DISTRIBUTION OF GAPS AND OVERLAPS

E. EXPERIMENT 3 -- BIGGER VOCABULARY AND BETTER ACOUSTICS

Experiment 3 studies the effects of increased vocabulary size and improved acoustic-processing accuracy. As test systems, we use fCMi and fCmi from Experiment 1. These are the best systems for accuracy and speed, respectively, and they also give us more information about the map-all control strategy choice. Thus there are three experimental variables: vocabulary size, acoustic accuracy, and map-all. Data for two of the eight combinations, map-

all or not for smaller vocabulary and regular acoustic accuracy, come from Experiment 1. For Experiment 3, the other six combinations were tested to provide a complete set of data for analysis of the effects of the variables.

The large vocabulary is a 451-word superset of the 305-word vocabulary used in the other experiments. The mapper performance data showed that, with the 451-word vocabulary, the mapper made 2026 false alarms and had a false alarm rate of 142 false alarms per second of speech (compared with 114 for the 305-word vocabulary). Using this information, the mapper performance was simulated for the large vocabulary on the same set of 60 test sentences.

Improved acoustic-processing accuracy was simulated by a 7% downward stretch of the false alarm score distribution, while leaving the hit scores unchanged. In other words, a false alarm score X, in the range 45 to 100, was replaced by 1.07X-7. If the result was below the threshold of 45, the false alarm was eliminated. This process reduced the number of false alarms for the 305-word vocabulary from 1564 to 1204, and for the 451-word vocabulary, from 2026 to 1541. Because the subthreshold scores were eliminated, the simulated improvement left the average false alarm score almost unchanged: for the 305-word vocabulary, it went from 59.4 to 60.2, and for the 451-word vocabulary, it went from

58.2 to 58.8. We feel that an improvement in acoustic accuracy of the magnitude simulated here could have been achieved by careful tuning of the mapper.

Figure IV-24 records the accuracy results using the notation "M" for tests with mapping all at once, "m" for those without, "A" for systems with improved acoustic accuracy, "a" for those without, "V" for systems with increased vocabulary, and "v" for those without. Improved acoustics raises fCMi accuracy from 73.3% to 85.0%, or from 81.7% to 95.0% if harmless errors are forgiven. However, if vocabulary size is also increased, accuracy drops slightly from 73.3% to 71.6%. Thus, in this experiment, a 7% improvement in acoustic accuracy almost compensates for a 48% increase in vocabulary. Comparison of the M-results to the m-results shows that map-all consistently helps accuracy.

ACCURACY RESULTS

	AMv	Amv	aMv	AMV	amv	AmV	aMV	amV
Raw, %	85.0	78.3	73.3	71.6	70.0	68.3	68.3	53.3
Forgiving, %	95.0	85.0	81.7	78.3	76.7	76.7	75.0	58.3

Figure IV-24.

The main effects on accuracy and several other measures are given in Figure IV-25. Improved acoustics leads to

MAIN EFFECTS OF ACOUSTICS, VOCABULARY, AND MAP-ALL

		WITH	WITHOUT	DIFFERENCE	
Raw Accuracy (percent)					
	A	75.8	66.3	9.5	**
	V	65.4	76.7	-11.3	#
	M	74.6	67.5	7.1	*
Phrases (total number terminal and nonterminal)					
	A	155	208	-53	**
	V	204	159	45	**
	M	156	206	-51	**
False Terminals (percent)					
	A	80.6	85.9	-5.3	**
	V	84.3	82.1	2.2	
	M	81.7	84.8	-3.1	**
Total Runtime (seconds/sentence)					
	A	266	320	-54	**
	V	312	275	37	*
	M	383	204	179	**
Acoustic Runtime (seconds/sentence)					
	A	187	213	-26	**
	V	205	195	10	
	M	315	84	231	**
Executive Runtime (seconds/sentence)					
	A	66	89	-23	**
	V	88	67	21	**
	M	55	101	-46	**

** $p < .01$ * $p < .05$ # $p < .10$

Figure IV-25.

big gains in accuracy, storage, and runtime. Increased vocabulary makes performance worse, but at least the system does not collapse. As in Experiment 1, mapping all at once improves everything except acoustic and total runtimes.

Vocabulary size and mapping all at once interacted significantly for acoustic runtime ($p < .05$). Figure

VOCABULARY AND MAP-ALL INTERACTION FOR ACOUSTIC RUNTIME

	M	m	M-m
V	335	75	260
v	296	94	202
V-v	39	-19	

(seconds/sentence)

Figure IV-26.

IV-26 shows that the increase caused by map-all is greater for the bigger vocabulary, and, surprisingly, that the increase in vocabulary size leads to a reduction in processing, if the system is not mapping all at once. The latter effect is presumably the result of an increased number of false alarms making it easier to form complete (but wrong) interpretations.

Mapping all at once also interacted significantly with acoustics for acoustic runtime ($p < .01$), total runtime ($p < .01$), and false terminal percentage ($p < .05$). All cases were similar to the one shown in Figure IV-27. There was a synergistic interaction causing mapping all at once to be more effective with better acoustics, and vice versa. This result is readily explained since map-all is designed to take advantage of the difference between false-alarm and hit-score distributions, and improving the acoustics enhances that difference by reducing the number of high scoring false alarms.

ACOUSTICS AND MAP-ALL INTERACTION FOR FALSE TERMINALS

	M	m	M-m
A	78.6	82.6	-4.0
a	84.8	87.0	-2.2
A-a	-6.2	-4.4	

(percent)

Figure IV-27.

In addition to the main tests for Experiment 3, we also ran another test to study the effect of improved acoustics on a system using island driving. The best island driving system from Experiment 1 was fCMI. When tested on the 305-word vocabulary with 7% simulated improvement in acoustics, fCMI gained in accuracy from 68.3% to 78.3%. It was still below the non-island driving fCMi, and the gap between them remained large. (Recall that fCMi went from 73.3% to 85.0%.) Thus, improvements in acoustics of the size considered here appear to be inadequate to solve the problems with island driving.

In summary, this experiment has given us information about how badly the system is hurt by increased vocabulary, and how much it is helped by improved acoustics. With respect to the control-strategy design choices, further evidence appeared in favor of mapping all at once, and against the current version of island driving.

F. DETAILED MEASUREMENTS OF EXECUTIVE OPERATION

This section gives a detailed breakdown of the statistics for the most accurate of the Experiment 1 systems, fCMi. Based on the performance for fCMi on the 60 test sentences, we report information regarding the composition of the parse net, the effects of lookahead, the performance of the context-checking procedures, the processing time for the major Executive procedures, and the breakdown of the accuracy results according to the existence and relative scores of correct and incorrect interpretations. Because of its level of detail, this section presupposes familiarity with the description of the Executive given in Section D of Chapter III.

Figure IV-28 shows the average composition of the parse net at the end of processing an utterance. Notice that there are more consumer-to-prediction links than there are predictions (78 versus 61), so there is some sharing taking place. To estimate the amount of sharing, we computed what the total size would be if the parse net were a tree (instead of a network) and all of the shared structures were duplicated.* The average number of phrases plus predictions was 220 in the actual parse net; expanding

---

* The fCMi system works left-to-right and there is no left-recursion in the rules of the language, so the parse net does not have loops and can be converted to a finite tree.

COMPOSITION OF THE PARSE NET

133	Nonterminal phrases per sentence 38 complete, 50 partial, 25 empty
45	Terminal phrases per sentence
61	Predictions per sentence
78	Consumer-to-prediction links per sentence

Figure IV-28.

the net into a tree increased the average to 404, an 83% increase. Thus, while only 17 out of the 78 consumer-links (22% of them) went to another consumer's prediction, the overall savings were quite large.

Whereas Figure IV-28 shows the number of phrases and predictions that were actually constructed, Figure IV-29 shows how many were blocked for various reasons. There were 42.3 phrases per sentence rejected by language factor statements, and of these, syntactic factors, which are usually tested first because they are less expensive computationally, accounted for over 90%. The preliminary tests in the add-constituent procedure (times, phrase mapping, and lookahead) blocked 5.1 phrases per sentence. In 17.8 cases per sentence, the same terminal or nonterminal phrase already existed, so the construction of a duplicate

BLOCKING OF PHRASES AND PREDICTION

42.3	Factor Rejections	
	38.3	(90.3%) by syntactic factors
	1.5	(3.5%) by case-grammar factors
	0.3	(0.7%) by semantic translation factors
	2.3	(5.4%) by discourse factors
5.1	Add-Constituent Preliminary Tests	
	3.1	in Part 1
	2.0	in Part 2
17.8	Same Phrase Already Existed	
	4.5	Nonterminal
	13.3	Terminal
31.4	Phrases and Predictions Blocked by Lookahead in the Predict Task	
96.6	Total (per sentence)	

Figure IV-29.

was blocked.* The final type of blocking is lookahead in the predict task, which accounted for 31.4 blocked phrases and predictions per sentence.

The data in Figure IV-29 show that the lookahead mechanism is providing a substantial constraint, so it is of interest to compare the performance of the system with and without lookahead (see Figure IV-30). Lookahead has good effects on accuracy, storage, and Executive runtime, but not

---

It is possible to have such duplication even with a left-to-right control strategy because of the looseness in the time constraints. For example, a word starting at position 45 can be found by predictions 40, 45, and 50, leading to two blocked duplicates.

EFFECTS OF LOOKAHEAD

		WITH	WITHOUT	DIFFERENCE
Raw Accuracy, %		73.3	71.7	1.6
Forgiving, %		81.7	76.7	5.0
Total Runtime	(sec/sent)	385	287	98
Exec Runtime	(sec/sent)	53	64	-11
Acoustic Runtime	(sec/sent)	320	208	112
Storage (phrases)		158	192	-34

Figure IV-30.

on acoustic runtime or total runtime.* The bad effects result from an increase in the number of places per sentence where words were tested -- up from 13 without lookahead, to 20 with. Lookahead is causing the system to 'peek' at places that without lookahead it would simply ignore. The extra testing done with lookahead would be less expensive with a redesigned mapper, but the cost undoubtedly would not decrease enough to compensate for the existing difference.* Lookahead appears to help accuracy somewhat, so rather than simply discard it, we feel that further effort is called for to design a version of the system that uses lookahead in a more efficient way.

---

* Based on the data in Figure IV-30 the acoustic runtime would have to drop to one-eighth of its current level to cause the total runtimes to be the same (subtract seven-eighths of the acoustic runtime from the total runtime: with lookahead, 385-320*7/8=105; without lookahead, 287-208*7/8=105).

Unlike lookahead, context checking has uniformly good effects. As mentioned previously, an average of about 6.3 seconds of processing per sentence was spent doing context checking, and this effort resulted in a net decrease of 41 seconds per sentence in the total runtime (based on comparision with the results for the best system without context-checking (fcMi) -- the savings were 19.4 seconds in Executive processing, 17.3 seconds in acoustics, and 4.2 in semantics). There was an average of 50 rating assignments made by context checking per sentence, with the construction of 78 virtual phrases and 28 complete consumer paths (average length of a complete path, 1.5 virtual phrases). The rejection of a virtual phrase by rule factors caused 35 paths to be terminated per sentence. Ten paths per sentence were blocked by the heuristic search procedure because their priority was less than the established lower bound. Note that the number of rating assigments by context checking equals the number of partial nonterminal phrases (given in Figure IV-28). Thus, there was no recalculation of ratings for partial nonterminal phrases after the first assignment. Also, since there were only 28 complete paths, at most 28 of the partial nonterminals were allowed to make predictions.* The ones without a complete consumer path received a rating

---

* It may have been fewer than 28 if some rating assignments created more than one complete path.

of zero and were not added to predict-sets. This result helps to explain the value of context checking; about half of the partial nonterminals that were all right with respect to local tests and fit their consumers' structural requirements were rejected by their consumers' factor statements and, therefore, were given a zero rating.

The processing time for context checking appears to be well spent, but how was the time spent for the rest of the Executive procedures? Figure IV-31 shows the total time per sentence, the number of calls per sentence, the time per call, and the percentage of Executive runtime, for the major Executive routines. Notice that the combined rule procedure execution time for complete, incomplete, and virtual phrases is only 12.1% of the Executive processing. The time for the rule procedure with a complete phrase is about 78 milliseconds, while with an incomplete or virtual phrase the time is only 24 milliseconds. Given the size of the rule procedures and the capabilities of the INTERLISP compiler and the PDP KA-10, these times probably cannot be improved significantly. The create-subnet procedure stands out as taking the most time per call, 279 milliseconds on the average. However, it is a complex operation and that amount of time does not seem unreasonable for it. Perhaps the main conclusion to be drawn from the timing data in Figure IV-3

TIMING BREAKDOWN

FUNCTION	TOTAL-TIME (seconds)	CALLS	PER-CALL (msecs)	PERCENT-OF EXECUTIVE
Word Task				
Create-word-set	6.1	48	127	10.8
Get-a-word	1.5	18	83	2.7
Create-terminal	3.9	70	56	6.9
Distribute-phrase	1.5	65	23	2.7
Add-constituent				
Top-level	2.5	112	22	4.4
Prelim-part-1	1.5	112	13	2.7
Prelim-part-2	4.2	109	39	7.5
Complete-phrase	6.0	107	56	10.7
Rule procedure	3.2	41	78	5.7
Consumer-checks	3.3	156	21	5.9
Incomplete-phrase	3.8	71	54	6.7
Rule procedure	1.7	71	24	3.0
Add-predict-sets	1.1	35	31	2.0
Predict-task				
Create-Subnet	5.3	19	279	9.4
Assign-Ratings	1.6	19	84	2.8
Cleanup	0.5	19	26	0.9
Context-checking				
Virtual-phrase	3.4	78	44	6.0
Rule procedure	1.9	78	24	3.4
Search	1.7	50	34	3.0
Top level	1.6	1	1584	2.8
		(per sentence)		

Figure IV-31.

is that big improvements in processing time will not come from discovering and correcting implementation oversights in the Executive -- instead, nontrivial design innovations will be needed.

The final measurements to be discussed deal with the accuracy of the fCMi system. Figure IV-32 shows the

ACCURACY BREAKDOWN

```
32 times only got correct interpretation
 8 times only got incorrect interpretation
 3 times got no interpretation
12 times correct score better than bad score
 2 times correct score same as bad score
 3 times correct score worse than bad score
---------
60 total -- 44 correct and 16 errors.
```

Figure IV-32.

accuracy breakdown in terms of the existence and relative scores of correct and incorrect interpretations. Overall, fCMi got 44 sentences correct and missed 16. Of the 16 errors, five were 'harmless'. Three of the harmless errors consisted of leaving out a plural morpheme. These accounted for all of the cases in which an interpretation was found but had a worse score than an incorrect interpretation that was also found. The other two harmless errors were among the cases in which only incorrect interpretations were found. In one, the system picked an interpretation containing "has" instead of a plural morpheme followed by "have". In the other, a singular verb suffix was accepted instead of a past tense suffix. In both cases, the incorrect interpretation had a higher score than would have been given to the correct interpretation (if it had been found). Thus, because of high scoring false alarms, the optimal solution (the interpretation with the highest score possible) was not the correct solution.

The two cases having correct and incorrect interpretations with equal scores were caused by the presence of false alarms that could not be rejected by linguistic considerations alone. The three cases getting no interpretation all had a low scoring word in either the first or second position (mapper scores of 59 or less), and in two of the cases, island driving (by the fCMI system) succeeded in finding the correct answer.

The eight cases in which only an incorrect interpretation was found can be divided into three categories: forgiven errors, optimal but not correct, and suboptimal. As mentioned previously, two of the eight with no correct interpretation were forgiven errors. Three were the result of finding an optimal interpretation that was not correct. [Surprisingly, island driving (fCMI) got one of these correct by stopping with a suboptimal, correct interpretation.] The final three were the result of stopping with a suboptimal, incorrect interpretation. In these last three, the correct interpretation started with either a bad score (56) or a small word ("how" or "the"). In each case, island driving (fCMI) got the correct answer.

Of the 16 sentences that fCMi missed, five were forgivable acoustic errors, six were correctly interpreted by an island driving system (fCMI), four were the result of

finding optimal but incorrect interpretations, and one had so many attractive false paths that it could not be handled within the storage limits by any of the systems. These results indicate that a different control-strategy might have correctly answered at least five sentences more than fCMi did: three for which fCMi picked a suboptimal interpretation and two for which fCMi found no interpretation although fCMI found the correct one. Such an improved strategy would have an 81.7% accuracy (90.0% forgiving), with 'nonforgiven' errors traceable to either acoustics (five cases) or storage limits (one case). This result gives a rough upper bound for improvements by modifying the control strategy versus modifying the acoustics. Of the 16 errors by fCMi, five were the result of the control-strategy failing to find the optimal interpretation, and 11 were the result of acoustic errors -- but five of the 11 acoustic errors could be forgiven.

G. DISCUSSION

Reviewing the series of experiments, the first studied the effects on system performance of four control-strategy design choices. Focus by inhibition and island driving had bad effects, while context checks for priority setting had good effects. Mapping all at once had good effects on everything except acoustic and total runtime, and these bad effects could probably be eliminated by redesign of the mapper. The second experiment varied the size of allowed gaps and overlaps between words and showed the potential value of special acoustics tests to verify word-pair junctions. The third experiment gave quantitative measures of how badly the system is hurt by increased vocabulary, and how much it is helped by improved acoustic accuracy. The experiment also provided more information about the control choices. The final study considered detailed measurements of the Executive performance and provided insights into the use of time and storage and the kinds of errors made by the system.

Overall, the series of experiments gives a better understanding of the system performance and suggests new possibilities for further research. With respect to methodology, the results indicate that experimentation using analysis of variance is a useful technique for analyzing

complex computer systems (as suggested by Newell, 1975). It is a technique that has been widely used in other areas of science and technology, but it has seen almost no use in computer science.*

There are two principal ways to improve future experimental studies of this type. The first is to develop a large corpus of sentences reflecting actual use of the system. Test sentences would then be randomly selected from the corpus. This technique would relieve the experimenter from the drudgery of formulating test sentences and, more importantly, would avoid the danger of sample bias. It is important to have a sample of test sentences that reflects the actual population of inputs not only with respect to the different kinds of constructions used, but also with respect to their frequency of use. For example, in our tests there were equal numbers of sentences at each length. This made it simple to use length as an experimental factor, but it may have produced somewhat unrealistic results.

The second experimental improvement would be to test the effects of varying the style of the language definition. For example, while keeping the scope of the language constant, a greater amount of the linguistic knowledge could

---

* However, Newell (1975) mentions that Gillogly at CMU has applied analysis of variance to the study of a chess program.

be encoded in the rule structures rather than in the rule procedures. Such an experiment would provide information about the performance effects of the different representational styles and would indicate the extent to which the effects of design features such as context-checking for assigning ratings are dependent on a particular style of language definition. In drawing conclusions from such an experiment, we would be generalizing both over the population of input sentences and over the population of language definition styles. This two-way generalization introduces some statistical complications (see Clark 1973), but it is still a useful experimental design and could provide important information.

H. TEST SENTENCES

The 60 sentences listed below were used in the control strategy experiments. The sentences are grouped according to their simulated length in seconds of speech. Processing for all of the sentences assumed a dialog context in which the preceding utterance was "What is the speed of the Batfish?". For example, the test utterance "Submerged displacement?" was interpreted by the system as meaning "What is the submerged displacement of the Batfish?".

2.3 seconds

Is the size of the Hammerhead 2000 tons?
Was Portsmouth Naval Shipyard the builder of the Seadragon?
Which subs have a length of 300 feet?
How many subs did Puget Sound Naval Yard manufacture?
What engine was manufactured by General Dynamics?
Whose ships did the Electric Boat Company construct?
Which destroyers were constructed by Cammell Laird Company?
Which AGFF did H. M. Dockyard construct?
How many diesel frigates does Great Britain have?
Did Bethlehem Steel Company manufacture a cruiser?

2.0 seconds

Were the Lafayettes built by Todd Pacific Shipyards?
Which countries have conventional submarines?
Which frigates were built by Newport News Shipyard?
Does the Swordfish have a speed of 30 knots?
What is the surface displacement of the Queenfish?
What categories of submarines are there?
Did Vickers Armstrongs Limited construct the Olympus?
Does the United States own that submarine?
What standard displacement does the Seahorse have?
What training submarines does England own?

1.7 seconds

Who constructed the English cruisers?
How many frigates are owned by the U.S.?
How many cruisers does England own?
How many classes of subs are there?
Do Resolutions have two reactors?
Is Britain the owner of the Conqueror?
Is the Renown a British submarine?
How many patrol submarines are there?
How many countries have CGNs?
Name the owners of aircraft carriers.

1.4 seconds

What country owns the Superb?
Who was the builder of the Jack?
Was its builder Avondale Shipyards?
Do we have ten diesel carriers?
How fast are the Graybacks?
What reactors does it have?
Was it built by Norfolk Navy Yard?
How many turbines do Brookes have?
Which nuclear carriers do we own?
Print the draft of the Scamp.

1.1 seconds

List the CHGs.
How many Darters are there?
Is it a research sub?
Who is the owner of it?
The speed of the Onslaught?
How fast is the Trout?
How many CGs are there?
Speed of the Bluefish?
What engines are there?
Is a CV a submarine?

0.8 seconds

Submerged displacement?
How long is it?
Whose ship is it?
Is it owned by us?
The Constellation?
Its surface speed?
Who constructed it?
What is its size?
What displacement?
Displacement of it?

V EXTENSIONS AND REVISIONS

Contents:

The previous chapters describe the results we have achieved in developing and testing a framework for speech understanding. This chapter outlines some possible extensions and revisions to that framework. We consider changes both to improve performance in speech understanding and to adapt the framework for use with text input. The main emphasis is on metalanguage extensions.

A. EXECUTIVE SYSTEM

Several possibilities for revising or extending the Executive have been mentioned in the previous chapters. First, the scoring should be made less ad hoc by basing it on the results of further tests. Second, the focus-by-inhibition technique deserves more experimentation to determine whether, by careful selection of words for focus, it can provide a significant improvement in system performance. Third, different versions of island-driving should be tested to see if the bad effects on long sentences can be reversed. Among possible modifications are: restricting island seeds to the start of the utterance; restricting island growth to a boundary of the utterance before growth starts in the opposite direction; and picking seeds of more than one word. Fourth, new word-mapping techniques are needed that can take advantage of the difference in score distributions for hits and false alarms without incurring the bad effects on runtime of mapping all at once. Fifth, the use of the lookahead mechanism should be revised so that it does not cause such a large increase in acoustic processing times.

Other worthwhile extensions to the Executive include provisions for user-defined additions to the vocabulary and syntax of the language (as provided in REL; Thompson 1969)

and special facilities for dealing with elliptical constructions. In the former, the Executive must be able to convert a user input into a rule or word definition that is then passed to the Definition Compiler for incorporation into the grammar. This capability gives users a simple method for introducing idiosyncratic terminology.

The extensions for elliptic constructions are necessary so that the composition rules will not be required to provide explicitly for the possible sentence fragments that can appear in a dialog. Rather than our current approach of having the composition rules account for the fragments, a separate mechanism would search for a way to embed the fragments in a complete sentence using contextual information. For example, after the question "What is the speed of the Kitty Hawk?" the elliptical utterance "location" would be interpreted by embedding it in the previous question to produce "What is the location of the Kitty Hawk?" It would not be necessary for the language definition to allow "location" to be parsed as a sentence.*

The addition of special mechanisms for intersentential ellipsis is only one example of an extension designed to

---

* Hendrix has implemented a simple version of this technique (see Hendrix 1976). The XEROX PARC language understanding system GUS also includes some special mechanisms for dealing with elliptical sentences (Winograd, personal communication).

make the framework linguistically more adequate. Further extensions motivated by linguistic considerations will be suggested in the following section.

B. DEFINITION SYSTEM

Various modifications would increase the expressive power of the metalanguage to make it better able to deal with complex forms of natural language. In particular, the following paragraphs discuss possible changes in the metalanguage for composition rules, suggest possible uses for redundancy rules, and propose a new type of 'dynamic' rule.

1. COMPOSITION RULES

Some minor modifications to the Definition System would simplify the development of composition rules. In the current system, a rule must be completely recompiled whenever it is changed in any way. Also, structural data (such as the rule graph and the lookahead tables) are recomputed even if the actual changes are limited to the rule procedure. These shortcomings could be eliminated by modifying the syntax of the rule procedures so that LISP could directly interpret or compile them (without requiring modifications by the Definition Compiler) and by separating

the rule structure declaration from the rule procedure declaration.

Straight LISP rule procedures would speed up the test-edit-retry debugging cycle, and separating the structure declaration from the rest of the rule would ensure that the structural information was recomputed only when necessary. The Definition Compiler would operate exclusively on the structures (since the procedures would be LISP from the start). These modifications were not made for the speech understanding system because they lead to increased time and storage requirements during parsing. However, the greater user convenience would justify the cost for text processing.

2. RULE PROCEDURES

In the current metalanguage, the rule procedures set attributes only for the phrase being constructed and not for its constituents or for dominating phrases. Also, the procedures use information only about attributes of constituents and not about attributes of the sentential context. The motivation for these restrictions was discussed in Chapter II, Section E, where we contrasted our method with that of Knuth (1968). Recall that Knuth's approach uses both synthesized and inherited attributes so

that information can be passed down the parse tree as easily as up. In contrast, we have limited ourselves to using only synthesized attributes because we cannot delay the use of augmentations until a complete parse is found, and we cannot afford to duplicate attribute and factor information for each context.

However, we recognize the value of providing inherited attributes and propose to allow for them by following Pratt (1973) in adding a second type of rule procedure for use after a complete parse tree has been formed. The current rule procedures would then be split in two parts: a 'constructor' to be applied when the phrase is first built, and a 'translator' to be applied after the phrase is used in a complete parse of the input. The constructor procedures would be limited in their use of contextual information (like the current rule procedures); the translators would have access to the entire sentential context. The constructors would set only local attributes; the translators would be able to set attributes anywhere in the parse tree. The order of evaluation for the constructors would be bottom-up, but the translators would operate top-down so that the context could easily influence the computation. In practice, the operations performed by the constructors would probably be limited to simple checks

such as various syntactic agreement tests and semantic compatibility tests based on some form of case grammar (Fillmore 1968). The translators would perform the more complex operations that typically require contextual control -- semantic translation being the prime example.

Compared to our current method, the new approach would simplify the treatment of many constructions by allowing access to the full sentential context. A major example is intrasentential anaphora, cases like the following in which a pronoun is coreferential with another phrase in the same sentence:

Does the bucket have a hole in it?

Because of the lack of necessary contextual information, the current system does not allow any form of intrasentential anaphora; with the new organization, it would be straightforward to treat this important process in a general way.

The new method also shares the benefits claimed by Woods for a top-down, recursive approach to semantic translation (see Woods 1967, 1968). Woods' system for semantics has been used in large question-answering systems (such as Woods, Kaplan, Nash-Webber 1972) and involves first forming a complete parse tree and then making a recursive,

top-down application of a set of semantic rules written in a "pattern=>action" format, where the "pattern" describes a phrase structure fragment and the "action" specifies the interpretation. A major difference between Woods' method and the method of associating translation procedures with composition rules is the relation between syntax and semantics. We prefer a close coupling between the two, while Woods' method leads to a loose coupling. In Woods' system, as in many others (e.g., Winograd's SHRDLU -- Winograd 1971), primarily syntactic procedures produce a structure that is then passed to a separate set of semantic translation procedures. There is no explicit connection between the semantic rules and the routines that construct their input. Consequently, it is difficult to ensure that the syntax and semantics are coordinated in their coverage; in particular, the semantic rules may fail to account for certain meaningful structures produced by the syntactic routines.*

---

* The problem of ensuring the coordination of the translation process and the syntax can become particularly severe if the semantic rules take on an ad hoc character by referring to collections of particular lexical items. For example, in Woods' sample set of rules (Woods 1967, Appendix A, Part II) the 36 S-rules each refer to an average of three lexical items. If the translation rules did not refer to particular lexical items, it might be easier to ensure that the rules cover all the cases and remain unaffected by additions to the vocabulary.

The coordination problem should be easier with the proposed organization because of the one-to-one correspondence between procedures for constructing phrases and those for translating them. Such close coupling between syntax and semantics is of course not a new idea. It is in part responsible for the elegance of Montague Grammar systems in formal linguistics (see Montague 1974; Partee 1975) and has its basis in logic and metamathematics going back to Frege. However, this particular version does have some advantages even over the Montague Grammar type of organization. The top-down approach makes it easier to deal with effects of context on translation such as those discussed by Carden (1973). He argues against the hypothesis that disambiguation is the basic mechanism for semantic translation by showing sentences that have readings in context not among the possible readings in isolation. He cites dialects in which the relative scopes of negation and quantification change when certain modifiers are added. For example, "All the boys didn't arrive" may switch from a "for all not" reading to a "not for all" reading when the tag "did they?" is added.

Hintikka (1975) makes a similar point based on changes in the meaning of "any."

1a. Any member can afford to contribute.

1b. If any member can afford to contribute, I'll be greatly surprised.

In sentence 1a, "any" is a universal quantifier, but when embedded in sentence 1b it becomes an existential quantifier. Hintikka concludes that "we must give up the Fregean principle that the meaning (semantical representation) of a complex expression is a function of its constituent parts, plus the way they are put together"(p.43). In other words, semantic translation cannot be a strictly bottom-up process.

Montague Grammar tries to avoid these problems while keeping a bottom-up translation method. The key to this approach is the use of substitution for bound variables as part of the syntactic derivation of sentences. Thus, sentence 1b would be derived by using the noun phrase "any member" to bind the variable "X" in "If X can afford to contribute, I'll be greatly surprised." The existential interpretation follows from the logical equivalence of $(x)(P(x)\Rightarrow Q)$ and $((Ex)P(x))\Rightarrow Q$.

This method salvages bottom-up translation, but it leads to other difficulties. In particular, Wasow (1975) has presented evidence that syntactic rules require pronouns to be specified as such throughout a derivation rather than

being produced by substitution for a bound variable. Wasow's arguments are too lengthy to reproduce here, but his basic conclusion is that bound variables may be a useful semantic representation but should be kept out of the syntax. The proposed technique of associating both bottom-up constructors and top-down translators with composition rules avoids the problems Wasow pointed out and appears to be capable of handling a wide range of natural language in a simple and elegant manner.

3. NULL RULES

Another addition to the metalanguage is required to simplify the treatment of constructions involving apparent movement of constituents in the parse tree. An important example of these constructions are so-called WH questions such as those listed below:

2. What can you see?
3. How big is it?
4. How long ago did you see it?
5. How much does it weigh?

Analysis of WH questions shows that they are in part distinguished by an "extra" phrase in initial position and, in the remainder of the sentence, a missing constituent of

the same category as the extra one (see, for example, the discussion of WH fronting in Akmajian and Heny 1975). These features combine to make it appear as if the WH constituent has been moved from its original position to a new one at the front of the sentence. In fact, transformational grammarians deal with the formation of questions by a rule that actually performs the apparent movement, thereby capturing in a simple manner the correspondence between the extra constituent and the missing one. Nontransformational grammarians take a less literal view of the movement involved in question formation and must find other means to establish a connection between the WH phrase and its apparent source.

Woods-style ATNs deal with the movement problem by special HOLD commands and VIR arcs (Woods 1970). A VIR arc in an ATN can be traversed if a phrase of a specified category has been placed on a 'hold list' by a prior HOLD command. The phrase is removed from the hold list when it is used by a VIR arc transition. To handle WH questions, the interrogative phrase is explicitly checked for and, if found, added to the hold list. VIR arcs are placed in the network wherever a constituent might be missing because of question formation. In this manner, the ATN captures the correspondence between the WH constituent and its source

location since exactly one VIR arc must be used for each held phrase.

However, as Kaplan pointed out (Kaplan 1974), VIR arcs scattered over the network do not adequately reflect the generality of the question-formation process. To state more concisely the fact that questioned phrases can come from almost anywhere, Kaplan replaces multiple VIR arcs by a single RETRIEVE arc crossing the entire network for a category that can be questioned. Like VIR arcs, RETRIEVE arcs can be followed if an appropriate phrase is on the hold list. In comparison with phrase structure rules, VIR arcs are like an option allowing certain constituents to be missing in special circumstances, namely when a phrase of the same category is on the hold list. RETRIEVE arcs are like a constrained form of null rules, rules that allow a nonterminal to have zero constituents.

In our current metalanguage, we do not have special mechanisms to deal with operations such as WH question formation. Constituents that can be questioned are simply specified as optional, and rule procedures must provide for establishing the correspondence between the initial WH phrase and its source location. This lack in the metalanguage could be tolerated for the simple language definition of the speech understanding system, but it must

be remedied if the system is used for more ambitious definitions. In line with the preceeding discussion, we propose to deal with constructions involving apparent movement by adding a facility for null rules with their application constrained by a mechanism similar to the ATN hold list.

The null-rule-and-hold-list facility may also be useful with constructions involving apparent deletion. Comparative clauses provide an example. In a series of papers, Bresnan has convincingly argued for a rule of Comparative Deletion (see Bresnan 1973, 1975, 1976a). This rule has been used in the following sentences to remove a constituent from the location marked by "-."

6. It costs more than it weighs -.
7. He told more lies than I've ever listened to -.
8. Are you as good at listening as you are - at talking?
9. Eat as much of this as you eat - of that.

There is both syntactic and semantic evidence that a phrase is missing in the comparative clauses. First, the clauses lack a constituent syntactically required in similar independent sentences:

10. * It weighs.

11. * I've listened to.

12. * You are at talking.

13. * You eat of that.

Second, the semantic interpretation of the clauses requires restoring the deleted material by using information from the head of the comparative. The following examples indicate the restored material. They also illustrate the fact that the deleted constituent must start with a measure phrase but may contain other material identical to part of the comparative head:

14. It costs more than it weighs [x-much].

15. He told more lies than I've ever listened to [[x-many] lies].

16. Are you as good at listening as you are [[x-much] good] at talking?

17. Eat as much of this as you eat [x-much] of that.

The null-rule-and-hold-list facility could prove useful in parsing this kind of comparative clause. At the start of the clause, the alternatives for the deleted constituent (which depend on the category of the comparative's head) would be added to the hold list. Inside the clause, a null rule of the appropriate category would

apply in place of the missing constituent. Because of the options for deletion, the hold list must indicate that more than one kind of category can be missing. The deletion can remove either the entire compared constituent (of the same category as the comparative head), or it can remove just an initial measure phrase, leaving the remainder of the compared constituent intact. The latter case is referred to as "subdeletion" in Bresnan's papers. Just as Bresnan treats subdeletion as a variety of comparative deletion, we treat both cases together by allowing the hold list to contain a specification for a set of possible deletion categories. The following sentences (taken from Bresnan 1975) illustrate the contrast between subdeletion and full deletion of the entire compared constituent:

18. They have many more enemies than we have -.

19. They have many more enemies than we have - friends.

In sentence 18, the entire compared noun phrase has been deleted; in sentence 19, only a measure phrase at the front of the compared noun phrase has been deleted.

The role of the null phrase in comparatives is analogous to its role in movements such as question formation and relative clause formation. It fills a slot to

satisfy the syntax and provides a binding site for semantic translation. This functional similarity and the processing similarity (the use of the hold list) may help to explain Bresnan's observations that comparative deletion and the unbounded movement rules behave identically with respect to many conditions that limit the applicabilty of syntactic rules (discussed in Bresnan 1975). It would also be interesting to look for psychological evidence to support the use of the hold-list approach to treating comparatives. In particular, Kaplan's experiments (Kaplan 1974) on the transient processing load and the hold list could be repeated using comparatives rather than relative clauses. We would expect to find that comparatives are similar to relative clauses in creating an increased processing load during the time that the hold list is in use.

Comparative clauses are complex structures, and the hold-list technique sketched above is not a complete solution. For example, in some cases ellipsis may eliminate the site of comparative deletion, and in other cases the "clause" is only a single phrase:

20. We have more friends than he does.

21. We have more friends than enemies.

In spite of these complications, the use of the hold list mechanism appears to offer at least part of a solution to dealing with a construction that has not been treated satisfactorily in previous computational efforts.

4. REDUNDANCY RULES

As indicated in Chapter II, Section B.3, the current metalanguage provides for LISP functions that can operate on rule and category definitions before compilation, but for a variety of reasons this capability has not been used. The principle reasons for this are the relative simplicity of the current definition, the lack of special metalanguage facilities for stating redundancies, and the inability to create new composition rules as part of the effect of the expressed generalization.

As the language definition becomes more extensive in scope, redundancy rules will become increasingly important as a way of stating in a single place a generalization with wide-spread implications for the language. There will be both practical and theoretical motivation for using redundancy rules: the practical motivation will come from the need to reduce the complexity of the definition to a manageable level, and the theoretical motivation will come from the desire to capture significant

generalizations about the object language. The importance of the latter consideration is indicated by the following passage from Peters (1973):

> The linguistic inadequacy of these models [such as phrase structure grammar] was established not by showing that some language lay beyond their weak generative power, the set of languages they generate, [footnote deleted] but by demonstrating that they are forced to state in many different rules what is, in effect, a single generalization about English (or another natural language) and that they are incapable of assigning to generated sentences structural descriptions which correctly match their degree of ambiguity, relationships of similarity to other sentences, etc. (p.372).

The redundancy rules will be "syntactic" or "lexical" according to the kind of definition they affect. Lexical redundancy rules will be used to express regularities and relationships in the lexicon. In some cases, the rules will add features to existing entries, and in other cases the rules will cause the creation of new entries. As an example of the former, nouns marked -COUNT could be marked -PL, meaning not plural, by means of a lexical redundancy rule. As an example of the creation of new entries, nouns of the form adjective+ness, such as "redness," could be derived from adjectives by a lexical redundancy rule.

In addition to their use in constructing the lexicon, the word-formation rules may be useful in recognizing new words. The rules would thus serve a dual role: as redundancy rules in compiling the lexicon and as low level composition rules during parsing. For further discussion of the use of lexical rules in expressing grammatical regularities, see Jackendoff 1975, Aronoff 1976, and Bresnan 1976b.

Number agreement between the subject and predicate of sentences is a simple example of a generalization that could be treated by a syntactic redundancy rule. Number agreement is a single fact about English that is not captured by scattered tests appearing in various S-rules of the grammar. It can be expressed in a more satisfactory manner by a redundancy rule modifying the S-rules in order to add the agreement tests.

An example of the addition of new composition rules by a redundancy rule is provided by WH question formation. WH questions can be formed by fronting noun-, adjective-, adverb-, and quantity-phrases that begin with an interrogative (our discussion of WH phrases follows Bresnan 1976a). In each case, the phrase is fronted to the same position, has the same effect with respect to subject-auxiliary inversion, and is constrained by the same restrictions on possible source locations.

The uniformity underlying the different types of WH questions can be expressed by having a single redundancy rule that creates the necessary composition rules.* For each category fronted in a WH question, the WH redundancy rule will cause an S-rule to be created with that category occurring initially. The procedures for the new S-rules will be derived from those for the original S-rules; additions will be made to indicate that the fronted constituent must begin with an interrogative and must bind a null phrase of the same category somewhere else in the sentence. (This involves putting the phrase on the hold list.) The redundancy rule will also cause the addition of null rules for the constituent unless such rules already exist. The same procedure will be invoked in all the new S-rules to establish the connection between the fronted WH phrase and the corresponding null phrase.

Readers familiar with transformational grammar may note an analogy between transformations and our redundancy rules; they serve to capture similar generalizations, but a major difference is that transformations apply to phrase markers in generating sentences of the language, whereas redundancy rules apply to basic definitions in creating a final version of the grammar for use in subsequent

---

* The composition rules would make use of the hold-list mechanism described above.

processing. This difference is important in the computational efficiency of the processing model and its possible psychological reality. If transformations can be replaced by lexical and syntactic redundancy rules, it may be easier to relate the resulting model of grammar to psycholinguistic models of language use. For evidence of interest in nontransformational analyses in formal linguistics, see Freiden 1975, Schachter 1976, and Bresnan 1976b.

Another aspect of the facility for redundancy rules concerns the techniques for specifying syntactic categories. Redundancy rules may apply to several categories of constituents. This was illustrated in the case of WH question formation, and it is also the case for other rules as well (see Bresnan 1976a). Simple enumeration is an unsatisfactory method of indicating the domain of a rule if there is a common property of the categories that could be used instead. Chomsky (1970) suggests replacing category symbols such as NP by sets of category features so that cross-categorial rules can be stated using features instead of lists of alternatives. The feature approach allows "natural" sets of categories to be specified by means of a submatrix of features. (For further discussion of the value of this approach in syntax, see Bresnan 1976a and Jackendoff 1974).

In order to explore the use of category features, the metalanguage should include facilities for declaring features and their possible values; it should also allow feature sets to be used in place of category symbols in structure specifications in redundancy rules and composition rules. In pattern matching for redundancy rules, a feature set would match any category that is a superset of it. A composition rule with a structure declaration using an incomplete set of features would be treated as an abbreviation for all the rules that could be formed by completing the feature set.

5. DYNAMIC RULES

It may prove necessary to add a more complex type of rule to the metalanguage for certain constructions that appear to involve the deletion of a nonconstituent. Examples of these constructions (from Hankamer 1973) are given below with the deleted material shown in brackets:

22. (Gapping) Max is believed to have tried to seem helpless, and Harvey [is believed to have tried to seem] irresponsible.

23. (Comparative Ellipsis) Joe was admired by Sue more that [Joe was admired] by Martha.

24. (Intersentential Ellipsis) What has Harry done?-- [Harry has] incriminated himself.

25. (Sluicing) Sally's coming to the party, but I don't know who [Sally's coming to the party] with.

To this list can be added cases of conjunction reduction such as the following example taken from Woods (1973):

26. (Conjunction Reduction) John drove his car through and completely demolished a plate glass window.

These constructions can show up in so many places and take so many forms that it seems futile to try to enumerate the possibilities through prespecified structural descriptions as the current type of composition rules would require. The use of the standard, static composition rule for these constructions appears to lead to unsatisfying, ad hoc compromises. Avoiding the use of nonconstituent deletion would be preferable both to simplify the parsing process and to restrict the power of the metalanguage. However, lacking alternative analyses for the constructions illustrated above, we must consider the possibility of introducing a new, dynamic type of rule that can generate structural possibilities during a parse. A discussion of some previous efforts to deal with similar problems will give a feeling for what we are proposing.

There have been two main experiments in computational linguistics related to constructions that appear to involve the deletion of a nonconstituent: the treatment of conjunctions in the Linguistic String Parser developed by Sager and her colleagues (Sager 1973; Grishman 1973; Raze 1976) and in the experimental facility developed by Woods (1973).

The Linguistic String Parser (LSP) operates with a grammar composed of two principal components: a set of context-free rules and a set of restrictions. The restrictions are procedures expressing conditions of well-formedness that must be met by a parse tree produced by the context-free rules in order for the tree to be accepted as a correct analysis. Using this approach, the LSP group has deleveloped a grammar based on a linguistic theory of string analysis (Harris 1962).

With respect to conjunctions, the linguistic string theory claims that "and" may combine strings of sister constituents. In other words, if there is a rule X = E1 ... En, then a conjunction may occur after Ei, followed by a string of constituents Ej...Ei, with j between 1 and i inclusive. This theoretical claim is reflected in the primary LSP approach to handling conjunctions. As an example, assume that "and" is encountered after filling in

Ei in the rule mentioned above. A special mechanism interrupts the parser when this happens and modifies the rule to read X = El...Ei SP-AND Ei+l...En, where SP-AND = "and" Q-CONJ. After this interrupt, the parser continues its depth-first analysis using the modified rule. When the Q-CONJ is reached, a special restriction mechanism generates a definition for Q-CONJ that depends on the place of the original interrupt. In this instance, the definition would include as alternatives all the constituent strings Ej...Ei for j from 1 to i.

This approach is clearly superior to prespecifying the possible conjunctions, as Sager and her colleagues point out. Moreover, it offers a reasonably efficient means of handling both the simple cases of single constituent conjunction and more complex cases such as sentence 27 in which the adverb-verb string of constituents is duplicated.

27. You totally baffle and utterly amaze me.

Unfortunately, not all conjunctions have the form assumed by this method. In cases illustrated by sentences 28 and 29, the conjunctions are not strings of sister constituents according to typical phrase structure analyses of English.

28. (Right Node Raising) Charlie believes in and stands up for his right to read Playboy.

29. (Gapping) Dick likes the stories, and Pete, the pictures.

To deal with such cases, the LSP group has added a second special mechanism allowing the parser to skip over a 'zeroed' element and to delay executing restrictions that refer to the omitted element until it can be filled in by linking it to some other subtree in the sentence. This takes care of sentences 28 and 29 by assuming a zeroed element after "in" and "Pete," respectively. However, sentences 30 and 31 illustrate cases in which the parser would have to skip over several omitted constituents at different levels in the parse tree, an ability not claimed for LSP.

30. Tom could have easily been shot, and Dan, arrested.

31. Harry wants to marry a ballerina, and Marty, an actress.

A final problem with this mechanism is its relation to the first special mechanism for conjunction. Many sentences handled by the first technique also fall within the scope of the second, which leads to an excess of parses. Perhaps a version of the second, zeroed constituent

approach extended to deal with cases like sentences 30 and 31 would make the first method entirely redundant so that it could be dropped.

Woods (1973) reports a conjunction facility, SYSCONJ, developed as part of an ATN parsing system. SYSCONJ deals with sentences like several of those above, in which fragments are conjoined in a single clause, with shared material factored out to the left and right. More precisely, SYSCONJ handles inputs of the form Y-A-B-and-C-D-Z, where Y,A,B,C,D, and Z are strings of words not necessarily forming constituents. The strings A-B-D and A-C-D must both form constituents of some category X, with A and D analyzed and interpreted in the same way in each. The interpretation produced by SYSCONJ corresponds to Y-A-B-D-and-A-C-D-Z. Referring back to sentence 27 as an example, we see that the category X is 'S' (sentence), and the substrings are as follows: Y=empty, A="you", B="totally baffle", C="utterly amaze", D="me", and Z=empty. When rewritten as Y-A-B-D-and-A-C-D-Z, this sentence becomes "You totally baffle me and you utterly amaze me." In general, any of the strings Y, A, D, or Z may be empty, but both B and C must be non-empty. These options allow SYSCONJ to deal with a wide variety of conjoined structures although some forms are not covered (such as sentence 29 and others formed by gapping).

The following description of the SYSCONJ algorithm is somewhat idealized in order to avoid reference to implementation details. Recall that the task of the algorithm is to find the correct analysis of the input as a sequence Y-A-B-and-C-D-Z. SYSCONJ's first step upon encountering the word "and" is to pick a place to start the string B. The choice is nondeterministic in the sense that if it fails to produce an acceptable result, it can be retracted and another selection can be made in its place. The system then begins to parse Y-A concatenated with C-D-Z (the breaks between Y and A, C and D, and D and Z have not yet been determined, of course). Whenever a constituent is formed that includes words from both sides of the "and," the words from the left are used as A and the ones from the right as C-D. This leaves only the break between C and D to be specified. SYSCONJ nondeterministically picks a place to break C and D and tries to parse Y-A-B-D in the same way it just finished parsing Y-A-C-D. If this action succeeds, then the two parses, one for Y-A-B-D and the other for Y-A-C-D, are merged to finish parsing Z as if the input had been Y-A-B-D-and-A-C-D-Z.

Despite the combinatorial problems, SYSCONJ manages to handle an impressive range of conjunctions and is important as a pioneering effort in this difficult area.

Some of its shortcomings, such as the lack of semantic guidance, are pointed out by Woods himself. Beyond these, when viewed as part of a general framework for dealing with natural language, the main shortcoming of Woods' conjunction facility is that it is a special facility intricately embedded in his parser. Conjunction reduction, comparative ellipis, and the like, should be dealt with through general mechanisms and as part of the language definition rather than by intricate, special-purpose modifications to the parsing algorithm. Woods' experiment is important as an attempt to treat the process of conjunction reduction in a computational setting, but it does not address either the problem of stating such processes in a linguistically reasonable way or of allowing several such processes to coexist in a single system.

To summarize, we have indicated the need for a metalanguage extension providing mechanisms for dynamic rules to handle constructions such as comparative ellipsis and conjunction reduction that appear to involve deletion of a nonconstituent. The necessary mechanisms are far from clear at this time, but they may include transformations such as those used informally in the preceeding description of the SYSCONJ algorithm.* Like other rules in our

---

* Such a transformational approach would have precedents in the "powerful parser" of Kay (1967) and in the REQUEST system of Petrick (1973) and Plath (1973), as well as in Woods' SYSCONJ.

metalanguage, the dynamic rules would include nonsyntactic and non-Boolean judgments. Thus, we would be able to explore the effects of various knowledge sources in reducing the combinatorial problems, and also to consider more subtle effects such as the role of parallel structure. Most importantly, the dynamic rules would use general mechanisms in the framework; they would not be obscurely embedded in the parser.

6. CONCLUSION

In this section, we have outlined a series of metalanguage extensions of increasing complexity: simple changes in format to help the development of composition rules; introduction of a new type of rule procedure to allow better use of context; addition of a null-rule-and-hold-list mechanism for constructions involving movement or deletion of a constituent; development of redundancy rules to allow lexical and syntactic generalities to be stated concisely; and provision for dynamic rules to deal with constructions that appear to result from the deletion of a nonconstituent.

Underlying and motivating these proposals is a conviction that computational linguistics can be successful in developing a metalanguage to express the true and significant generalizations about English or other natural

languages in a form allowing grammars to be compiled for use in an efficient processing system. According to this view, computational linguistics is a branch of computer science concerned with algorithms and representations for defining and processing natural language. Although a part of computer science, computational linguistics has, or should have, strong ties with other areas such as linguistics and with psycholinguistics. We will be pleased if the work described in the previous chapters and the possibilities sketched in this one contribute to stimulating interest in computational linguistics and to strengthening its ties with related fields.

VI REFERENCES

Aho, A. V., and Ullman, J. The Theory of Parsing, Translation, and Compiling. Volume 1. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

Akmajian, Adrian, and Heny, Frank. An Introduction to the Principles of Transformational Syntax. MIT Press, Cambridge, Mass., 1975.

Aronoff, Mark. Word Formation in Generative Grammar. MIT Press, Cambridge, Mass., 1976.

Baker, James K. The DRAGON System -- An Overview. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975, ASSP-23, 24-29.

Barnett, Jeffrey A. INFIX LISP for SDC IBM 370 Users. TM-4310, System Development Corporation, Santa Monica, California, May 1973.

Barnett, Jeffrey A. Speech Understanding System Overview. TM-5732, System Development Corporation, Santa Monica, California, August 1976.

Bates, Madeleine. Syntactic Analysis in a Speech Understanding System. BBN Report 3116, Bolt Beranek and Newman, Cambridge, Massachusetts, August 1975.

Bernstein, Morton I. Interactive Systems Research: Final Report. TM-5243/004, System Development Corporation, Santa Monica, California, 1975.

Bobrow, Daniel G., and Fraser, J. Bruce. An Augmented State Transition Network Analysis Procedure. Proceedings of the International Joint Conference on Artificial Intelligence, Washington, D.C., 7-9 May 1969. Edited by Donald E. Walker and Lewis M. Norton. The MITRE Corporation, Bedford, Massachusetts, 1969, 557-568.

Bobrow, Daniel G., and Wegbreit, Ben. A Model and Stack Implementation of Multiple Environments. Communications of the ACM, 1973, 16, 591-603.

Bresnan, Joan W. Syntax of the Comparative Clause Construction in English. Linguistic Inquiry, Vol. 4, 1973, pp.275-343.

Bresnan, Joan W. Comparative Deletion and Constraints on Transformations. Linguistic Analysis. I.I 1975, pp.25-74.

Bresnan, Joan W. On the Form and Functioning of Transformations. Linguistic Inquiry 7:1, 1976 (a), pp.3-40.

Bresnan, Joan W. Toward a Realistic Model of Transformational Grammar. unpublished paper presented at the MIT-AT&T Convocation of Communications at MIT. 1976 (b).

Carden, Guy. Disambiguation, Favored Readings, and Variable Rules. In: C.N. Bailey and R.W. Shuy (eds.), New Ways of Analyzing Variation in English. Georgetown Univ. Press, Washington D.C., 1973.

Celce-Murcia, Marianne. Verb Paradigms for Sentence Recognition. American Journal of Computational Linguistics, Microfiche 38, 1976.

Chomsky, N. Remarks on Nominalization. In: R. Jacobs and P.S. Rosenbaum (eds.), Readings in English Transformational Grammar. Ginn, Waltham, Mass., 1970.

Cheatham, Thomas E., and Sattley, Kirk. Syntax Directed Compiling. AFIPS Conference Proceedings: Spring Joint Computer Conference, 1964, 25, 31-57.

Clark, H. H. The language-as-fixed-effect fallacy: A critique of language statistics in psychological research. Journal of Verbal Learning and Verbal Behavior. 1973. Pp. 335-359.

Cox, D. R. Planning of Experiments. John Wiley, New York, 1958.

Deutsch [Grosz], Barbara G. The Structure of Task-Oriented Dialogs. Contributed Papers, IEEE Symposium on Speech Recognition. Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974. Edited by Lee D. Erman. IEEE, New York, 1974, 250-254.

Deutsch [Grosz], Barbara G. Establishing Context in Task-Oriented Dialogs. American Journal of Computational Linguistics, 1975, 4, Microfiche 35.

Erman, Lee D. An Environment and System for Machine Understanding of Connected Speech. Ph.D. Thesis, Stanford University, Stanford California, 1974. (Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania.) (a)

Erman, Lee D., (Ed.). Contributed Papers, IEEE Symposium on Speech Recognition. Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974. IEEE, New York, 1974, 250-254. (b)

Fennell, Richard D., and Lesser, Victor R. Parallelism in AI Problem Solving: A Case Study of Hearsay II. Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1975.

Fillmore, Charles J. The Case for Case. In: Universals in Linguistic Theory. Edited by Emmon Bach and Robert T. Harms. Holt, Rinehart and Winston, 1968. Pp. 1-88.

Freiden, Robert. The Analysis of Passives. Language 51:2, 1975, pp.384-405.

Goodman, R. Gary. Analysis of Languages for Man-Machine Communication. Ph.D. Thesis, Stanford University, Stanford, California, 1976. (Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania.)

Goodman, R. Gary, Lowerre, Bruce T., and Reddy, D. Raj. Effects of Branching Factor and Vocabulary Size on Performance (Abstract). In: Speech Understanding Systems: Summary of Results of the Five-Year Research Effort. By Reddy, D. Raj, et al. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, September 1976.

Grishman, Ralph. Implementation of the String Parser of English. In: Randall Rustin (ed.), Natural Language Processing. Algorithmics Press, New York, 1973, pp.89-109.

Grosz, Barbara. The Representation and Use of Focus in Dialog Understanding. University of California at Berkeley, Ph.D. dissertation, 1977.

Hankamer, Jorge. Unacceptable Ambiguity. Linguistic Inquiry 4:1, 1973, pp.17-68.

Harris, Zellig. String Analysis of Sentence Structure. Mouton, 1962.

Hart, Peter E., Nilsson, Nils J., and Raphael, Bertram. A Formal Basis for the Heuristic Determination of Minimal Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 1968, SSC-4, 100-107.

Heidorn, George E. Augmented Phrase Structure Grammars. In: Theoretical Issues in Natural Language Processing. Edited by Roger C. Schank and Bonnie Nash-Webber. Center for Applied Linguistics, Arlington, Virginia, 1975, 1-5.

Hendrix, Gary G. Semantic Processing for Speech Understanding. American Journal of Computational Linguistics, 1975, 4, Microfiche 34. ».(c).

Hendrix, Gary G. LIFER. unpublished SRI document. Menlo Park, Cal., 1976.

Hintikka, Jaakko. On the Limitations of Generative Grammar. manuscript distributed in Philosophy of Language Seminar, Stanford Univ., 1975.

Hobbs, Jerry R. A Metalanguage for Expressing Grammatical Restrictions in Nodal Spans Parsing of Natural Language. Courant Computer Science Report No. 2, New York University, New York, 1974.

Irons, E. T. A Syntax Directed Compiler for ALGOL 60. Communications of the ACM, 1961, 4, 51-55.

Jackendoff, Ray. Introduction to the X-bar Convention. available from Indiana Univ. Linguistics Club, Bloomington, Indiana, 1974.

Jackendoff, Ray. Morphological and Semantic Regularities in the Lexicon. Language. Vol. 51, 1975.

Jazayeri, Mehdi, Ogden, William F., and Rounds, William C. The Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars. Communications of the ACM, 1975, 18, 697-706.

Kaplan, Ronald M. A General Syntactic Processor. In: Natural Language Processing. Edited by Randall Rustin. Algorithmics Press, New York, 1973a. Pp. 193-241.

Kaplan, Ronald M. A Multi-Processing Approach to Natural Language. Proceedings, National Computer Conference, New

York, New York, 4-8 June 1973b. Volume 42. AFIPS Press, New Jersey, 1973, 435-440.

Kaplan, Ronald. M. Transient Processing Load in Relative Clauses. unpublished Harvard doctoral dissertation. 1974.

Kay, Martin. Experiments With a Powerful Parser. RM-5452-PR, The RAND Corporation, Santa Monica, California, 1967.

Kay, Martin. The Mind System. In: Natural Language Processing. Edited by Randall Rustin. Algorithmics Press, New York, 1973. Pp. 153-188.

Knuth, Donald E. Semantics of Context-Free Languages. Mathematical Systems Theory, 1968, 2,, 127-145.

Landsbergen, S. P. Jan. Syntax and Formal Semantics of English in PHLIQA1. In: COLING 76, Preprints of the 6th International Conference on Computational Linguistics, Ottawa, Ontario, Canada, 28 June - 2 July 1976. No. 21.

Lesser, Victor R., Fennell, Richard D., Erman, Lee D., and Reddy, D. Raj. Organization of the Hearsay II Speech Understanding System. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975, ASSP-23, 11-24.

Lowerre, Bruce T. The HARPY Speech Recognition System. Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1976.

Miller, Perry L. A Locally Organized Parser for Spoken Input. Technical Report 503, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, Massachusetts, May 1973.

Montague, Richard. The Proper Treatment of Quantification in Ordinary English. In: Richmond Thomanson (ed.) Formal Philosophy: Selected Papers of Richard Montague. Yale Univ. Press, New Haven, Conn., 1974.

Newell, Allen, et al. Speech Understanding Systems. North-Holland Publishing Company, Amsterdam, 1973.

Newell, Allen. Production Systems: Models of Control Structures. In: Visual Information Processing. Edited by W. C. Chase. Academic Press, New York, 1973. Pp. 463-526.

Newell, Allen. A Tutorial on Speech Understanding Systems. In: Speech Recognition: Invited Papers of the 1974 IEEE Symposium. Edited by D. Raj Reddy. Academic Press, New York, 1975. Pp. 3-54.

Newell, Allen, and Simon, Herbert A. Computer Science as Empirical Inquiry: Symbols and Search. 1975 ACM Turing Award Lecture. Communications of the ACM, 1976, 19, 113-126.

Partee, Barbara. Montague Grammar and Transformational Grammar. Linguistic Inquiry 6:2, 1975, pp.203-300.

Paxton, William H. A Best-First Parser. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975, ASSP-23, 426-432.

Paxton, William H. A Framework for Language Understanding. In: COLING 76, Preprints of the 6th International Conference on Computational Linguistics, Ottawa, Ontario, Canada, 28 June - 2 July 1976. No. 14.

Paxton, William H., and Robinson, Ann E. A Parser for a Speech Understanding System. Advance Papers, International Joint Conference on Artificial Intelligence, Stanford, California, 20-23 August 1973. Stanford Research Institute, Menlo Park, California, 1973, 216-222.

Paxton, William H., and Robinson, Ann E. System Integration and Control in a Speech Understanding System. American Journal of Computational Linguistics, 1975, 4, Microfiche 34.

Peters, P. Stanley. On Restricting Deletion Transformations. In: Gross, Halle, Schutzenberger (eds.) The Formal Analysis of Natural Language. Mouton, 1973.

Petrick, Stanley R. Semantic Interpretation in the Request System. In: Computational and Mathematical Linguistics, Proceedings of the International Conference on Computational Linguistics, Volume I. Edited by Antonio Zampolli. Casa Editrice Leo S. Olschki, Firenze, 1973.

Petrick, Stanley R. On Natural Language Based Computer Systems. IBM Journal of Research and Development, 1976, 20, 314-325.

Plath, Warren J. Transformational Grammar and Transformational Parsing in the REQUEST System. In:

Computational and Mathematical Linguistics, Proceedings of the International Conference on Computational Linguistics, Vol. II, Antonio Zampolli (ed.), Firenze, 1973.

Pratt, Vaughn R. A Linguistics Oriented Programming Language. Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford Univ., Calif., 1973.

Pratt, Vaughn R. LINGOL - A Progress Report. Advance Papers, International Joint Conference on Artificial Intelligence, Tbilisi, Georgian SSR, 3-8 September 1975, 422-428.

Raze, Carol. A Computational Treatment of Coordinate Conjunctions. AJCL Microfiche 52, 1976.

Reddy, D. Raj. Speech Recognition by Machine: A Review. Proceedings of the IEEE, 1976, 64, 501-531.

Reddy, D. Raj, et al. Speech Understanding Systems: Summary of Results of the Five-Year Research Effort. Department of Computer Science. Carnegie-Mellon University, Pittsburgh, Pennsylvania, September 1976.

Ritea, H. Barry. Automatic Speech Understanding Systems. Proceedings of the 11th Annual IEEE Computer Society Conference, Washington, D.C., September 1975.

Ritea, H. Barry. A Voice-Controlled Data Management System. Contributed Papers, IEEE Symposium on Speech Recognition. Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974. Edited by Lee D. Erman. IEEE, New York, 1974, 28-31.

Robinson, Jane J. Performance Grammars. In: Speech Recognition: Invited Papers of the 1974 IEEE Symposium. Edited by D. Raj Reddy. Academic Press, New York, 1975. Pp. 401-427. (a)

Robinson, Jane J. A Tuneable Performance Grammar. American Journal of Computational Linguistics, 1975, 4, Microfiche 34. (b)

Sager, Naomi. The String Parser for Scientific Literature. In: Randall Rustin (ed.), Natural Language Processing, Algorithmics Press, New York, 1973.

Sager, Naomi, and Grishman, Ralph. The Restriction Language for Computer Grammars. Communications of the ACM, 1975, 18, 390-400.

Scha, Remko J. H. Semantic Types in PHLIQA1. In: COLING 76, Preprints of the 6th International Conference on Computational Linguistics, Ottawa, Ontario, Canada, 28 June - 2 July 1976. No. 10.

Schachter, Paul. A Nontransformational Account of Gerundive Nominals in English. Linguistic Inquiry 7:2, 1976, pp.205-241.

Slocum, Jonathan. Speech Generation from Semantic Nets. American Journal of Computational Linguistics, 1975, 4, Microfiche 33.

Thompson, F.B, Lockemann, P.C., Dosert, B., and Deverill, R.S.. REL: A Rapidly Extensible Language System. Proceedings of the National ACM Conference, San Francisco, 1969.

Thorne, James P., Bratley, Paul, and Dewar, Haimish. The Syntactic Analysis of English by Machine. In: Machine Intelligence 3. Edited by Donald Michie. Edinburgh University Press, Edinburgh, 1968.

Walker, Donald E. Speech Understanding Research. Annual Report, Project 1526, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, February 1973. (a)

Walker, Donald E. Speech Understanding Through Syntactic and Semantic Analysis. Advance Papers, International Joint Conference on Artificial Intelligence, Stanford, California, 20-23 August 1973. Stanford Research Institute, Menlo Park, California, 1973, 208-215. (b)

Walker, Donald E. Speech Understanding Research. Annual Report, Project 1526, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, May 1974.

Walker, Donald E. The SRI Speech Understanding System. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975, ASSP-23, 397-416.

Walker, Donald E., Paxton, William H., Robinson, Jane J., Hendrix, Gary G., Deutsch, Barbara G., and Robinson, Ann E. Speech Understanding Research. Annual

Report, Project 3804, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, June 1975.

Walker, Donald E., Paxton, William H., Hendrix, Gary G., Grosz, Barbara J., Fikes, Richard E., Slocum, Jonathan, Robinson, Ann E., and Robinson, Jane J. Speech Understanding Research. Annual Report, Project 4762, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, October 1976.

Wasow, Thomas. Anaphoric Pronouns and Bound Variables. Language 51:2, 1975.

Winer, B. J. Statistical Principles in Experimental Design. Second Edition. McGraw-Hill, New York, 1971.

Winograd, Terry. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Report MAC-TR-84, Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1971. [Published as Understanding Natural Language, Academic Press, New York, 1972.]

Woods, William A. Semantics for a Question-Answering System. Ph.D. thesis, Rep.NSF-19, Aiken Computation Lab., Harvard U., Cambridge, Mass., 1967.

Woods, William A. Procedural Semantics for a Question-Answering Machine. AFIPS Conference Proceedings, Vol. 33, 1968.

Woods, William A. Transition Network Grammars for Natural Language Analysis. Communications of the ACM, 1970, 13, 591-606.

Woods, William A. An Experimental Parsing System for Transition Network Grammars. In: Randall Rustin (ed.), Natural Language Processing, Algorithmics Press, New York, 1973.

Woods, William A., Kaplan, Ronald M., and Nash-Webber, Bonnie. The Lunar Sciences Natural Language Information System. Final Report. BBN Report 2378, Bolt Beranek and Newman, Cambridge, Massachusetts, 1972.

Woods, William A., and Makhoul, John. Mechanical Inference Problems in Continuous Speech Understanding. Artificial Intelligence, 1974, 5, 73-91.

Woods, William A., Schwartz, Richard M., Klovstad, John W., Cook, Craig C., Wolf, Jared J., Bates, Madeleine A., Nash-Webber, Bonnie L., Bruce, Bertram C., and Zue, Victor W. Speech Understanding Systems. Quarterly Technical Progress Report No. 1. BBN Report 3018, Bolt Beranek and Newman, Cambridge, Massachusetts, February 1975. (a)

Woods, William A., M. Bates, G. Brown, B. Bruce, C. Cook, L. Gould, J. Klovstad, J. Makhoul, B. Nash-Webber, R. Schwartz, J. Wolf, and V. Zue Speech Understanding Systems. Annual Technical Progress Report. BBN Report 3188, Bolt Beranek and Newman, Cambridge, Massachusetts, October 1975. (b)

Woods, William A., M. Bates, G. Brown, B. Bruce, C. Cook, L. Gould, J. Klovstad, J. Makhoul, B. Nash-Webber, R. Schwartz, J. Wolf, and V. Zue Speech Understanding Systems. Quarterly Technical Progress Report No. 5. BBN Report 3240, Bolt Beranek and Newman, Cambridge, Massachusetts, January 1976. (a)

Woods, William A., M. Bates, G. Brown, B. Bruce, C. Cook, J. Klovstad, B. Nash-Webber, R. Schwartz, J. Wolf, and V. Zue Speech Understanding Systems. Quarterly Technical Progress Report No. 6. BBN Report 3303, Bolt Beranek and Newman, Cambridge, Massachusetts, April 1976. (b)