ARPA ORDER NO.: 189-1
3P10 Distributed Information Systems

R-1377-ARPA
December 1973

Natural Language, Linquistic Processing, and Speech Understanding: Recent Research and Future Goals

Allen Klinger

A Report prepared for

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

25th Year
Rand
SANTA MONICA, CA. 90406

The research described in this Report was sponsored by the Defense Advanced Research Projects Agency under contract No. DAHC15-73-C-0181. Reports of The Rand Corporation do not necessarily reflect the opinions or policies of the sponsors of Rand research.

ARPA ORDER NO.: 189-1
3P10 Distributed Information Systems

R-1377-ARPA
December 1973

Natural Language, Linquistic Processing, and Speech Understanding: Recent Research and Future Goals

Allen Klinger

A Report prepared for

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

Published by The Rand Corporation

PREFACE

This report is one of a series on voice-data-processing capabilities applied to defense requirements. The work, which is sponsored by the Defense Advanced Research Projects Agency, includes the following:

- An analysis of speech for the human/computer interface.
- A description of military speech and the role of computers.
- A survey of acoustic-signal-processing technology.
- A survey of natural-language and linguistic processing.

The report concerns the last subject. It explains the specialized use of the term "understanding" in computer literature and describes recent research results in programs and languages, deduction, context and grammars, and linguistic (structural) pattern recognition.

The reader who has an awareness of some recent computer software system and knowledge of a programming language such as FORTRAN should find this discussion useful. People with interests in the computer science topics of artificial intelligence, pattern recognition, and man-machine interaction form the main audience for the report. Workers in linguistics, technology assessment, and military command and security applications are also in this group.

An idealized speech-understanding system is discussed as a vehicle for describing possible computer science, mathematical modeling, and language design research. Some R&D applications of the current multi-contractor ARPA speech-understanding research project conclude the report.

SUMMARY

The nature of computer science research related to speech understanding is revealed through discussion of the following topics: artificial intelligence, computer processing of natural-language data, programming languages for problem-solving systems, and structural (contextual, linguistic) pattern recognition. An in-depth survey of recent research results that includes examples of deduction, text recognition, and dialogue by and with computers is followed by discussion of speech data, applications of understanding-software, and suggestions for further research.

There is a wide range of potential applications for computer programs that exhibit "understanding." Such complex software systems involve multilevel decision-making. In text or speech understanding the capability of the system depends on

1. The number of primitive elements and their relative frequencies.
2. The number of grammatical rules restricting allowable combinations of primitives.
3. The number of words combining to effect meaning and the number of associated properties of each word in a vocabulary.
4. The speed of decisionmaking at
   a. The syntactic level (use of item 2).
   b. The semantic level (use of item 3).
5. Accurate knowledge of the elements of speech and their possible combination in natural and artificial language.

CONTENTS

1. INTRODUCTION

This report outlines some recent computer science research results which are being used as building blocks in an extensive program directed toward the development of speech-understanding capability by machine. The results concern question-answering, semantic information, and the syntax (or grammar) of subsets of English. When applied to machine understanding of speech, these results become software techniques with which to build a *capability*. Briefly, that capability includes the following:

1. Consistent correct interpretation of the meaning of the speech. (This may involve computer-controlled actions which respond accurately to the speech or a computer-created query of the user which furthers the ultimate goal via a man-machine dialogue.)

2. Retention of relevant facts regarding the subject of the speech (sometimes called the *world model*) and the existence of an ability to derive and use implications of such facts to make recognition decisions.

3. Inclusion of facts concerning the occurrence of parts of speech, acoustic elements (phonemes), user characteristics, and other grammatical or pragmatic considerations in intermediate program levels, where they can be used to aid in word recognition.

The methods, which are being combined and extended, have been demonstrated in isolation in diverse contexts. Prior developments indicate that a software technology now exists that may demonstrate significant speech-understanding capability in a few years. However, it is a risky business to predict how the current technology will combine to achieve a working speech-understanding system. There are many unforeseen potential problem areas in a software development program of this extensive scope. Hence, this report consists mainly of a survey of the recent research developments in natural language, linguistic processing, and speech recognition with an in-depth emphasis on those topics which bear upon speech-understanding research. Research problems whose

solution would contribute to the development of practical machine speech-understanding capability are suggested. Some numerical estimates regarding the performance of software similar to that being developed for speech understanding are presented.

2. NATURAL-LANGUAGE PROCESSING

The high-complexity problem of machine translation, which dominated the research in this field during the fifties and early sixties, has been replaced by much narrower goals. These concern the ability of the user to employ, and the computer program to respond in, natural language for problem-solving, information retrieval, and other applications. There are examples of natural-language input-output computer programs that make the data in large computer files more accessible to the user. Programs which have this capability are called *conversational, interactive,* or *question-answering*. There are three levels of restrictions on the freedom of the user, and each pertains to a corresponding level of software development. The levels are these (the corresponding non-technical equivalents are given in parentheses):

1. Lexical (allowed vocabulary)
2. Syntactic (allowed phrases or sentences)
3. Semantic (allowed meanings)

Although the concepts are distinct and the parts of natural-language programs which perform operations of each type are specifiable in terms of abstract theory or software implementation (e.g., (1) Word in a table? (2) Word string acceptable grammatically? (3) Word string acceptable logically?), in practice all these levels must operate many times in order to have the computer interpret even a small portion of restricted natural-language input, such as two words. Thus a program which detects "meaning" or logical consequences of word interpretations must perform grammatical operations as well for certain words to determine (i.e., tentatively assign, parse, then test for plausibility via consistency with known facts) their part of speech (noun, verb, adjective, etc.). One way to reduce the combinatorial explosion which results for the number of calls to a given subroutine is to limit the scope of the man-machine discourse. All examples of current natural-language-processing software technology deal with (a) a specialized

vocabulary, and (b) a particular context or set of allowed interpretations, and this is the general meaning of "limited scope of discourse." In terms of the qualitative categorizations of user restrictions, these correspond to lexical and semantic constraints. A parallel fact is that most natural-language software is highly developed at the syntactic level. However, the detection of proper grammatical forms within a sentence is very much a function of the meanings which can be assigned to individual words, and this is provided by programs which do semantic interpretation--frequently by retention of past portions of the discourse. Hence syntactic processing--sometimes called parsing--even with the limited scope or restricted English framework is interdependent with meaning, and this involves the allowed logical relationships among words in the lexicon.

2.1. PROGRAMS AND LANGUAGES

A frequent synonym for "the addition of logical relationships or semantics to syntactical processing" is the term *understanding*. In a survey paper [1], two conversational programs are discussed, ELIZA [2] and [3] and STUDENT [4]. Another program of a similar sort, SIR [5], and some general material on the subject of semantic information and programs which utilize it are found in the introduction to the volume [6] containing the last two references. An extensive discussion of the natural-language software until 1972 is contained in [7, pp. 34-46], and an equally valuable exposition of syntax and meaning, particularly within the framework of the program system developed by Winograd* (which is based on a systematic-grammar theory of language), is also presented there [pp. 16-34]. The material in this section draws heavily on [7], [1], and [8], in that order of importance for the section below.

The early language-understanding systems, BASEBALL [9], ELIZA, and STUDENT, were based on two special formats: one to represent the knowledge they store and one to find meaning in the English input. They discard all input information which cannot be transformed for

*The system is sometimes called "Winograd's block-world". One of the speech-understanding research projects, that at SRI, involves voice extensions of this system.

internal storage. In [1] two of these systems are compared with regard to the amount of "understanding" obtained. ELIZA responds either by transforming a sentence (a more varied form of mimicry) following isolation of a key word or by using a prestored content-free remark. STUDENT translates natural-language "descriptions of algebraic equations..., proceeds to identify the unknowns involved and the relationships which hold between them, and [obtains and solves] a set of equations" [1, p. 85]. Hence ELIZA "understands" only a few words; it transforms these words via a sentence-reassembly rule which throws away other parts of the sentence and adds stock phrases to create the response. STUDENT does more, since it solves the underlying algebraic problem--it "answers questions based on information contained in the input" [4, p. 135]. ELIZA responds but does not understand, since the reply has little to do with the information in the input sentence, but rather serves to keep the person in a dialogue. A similar ability to spout back, which does not involve dealing with underlying meaning, but involves storing a body of text and providing an indexing scheme, followed in the subsequent development of related programs (see [7, p. 35]). This was an approach of limited utility and was replaced by systems which used some formal representation to store limited logical concepts associated with the text. In particular, the program SIR can deduce set relationships among objects described by natural language. It is designed to meet the requirement that "in addition to echoing, upon request, the facts it has been given, a machine which 'understands' must be able to recognize the logical implications of those facts. It also must be able to identify (from a large data store) facts which are relevant to a particular question" [5].

Limited-logic systems are important because they provide methods of representing complex facts encoded in English-language statements so that the facts can be used by computer programs or accessed by a person who did not input the original text-statement of the fact. Such a "second user" may employ a completely different form of language-encoding. Programs of this sort include DEACON [10] and [11] and the early version of CONVERSE [12]. The former could "handle time questions" and used

> a bottom-up analysis method* which allowed questions to be nested. For example, the question 'Who is the commander of the battalion at Fort Fubar?' was handled by first internally answering the question 'What battalion is at Fort Fubar?' The answer was then substituted directly into the original question to make it 'Who is the commander of the 69th battalion?' which the system then answered. [7, p. 37]

CONVERSE contained provisions for allowing even more complex forms of input questions.

Deductive systems are categorized by Winograd [7] as either general or procedural. General systems attempt to remedy deductive deficiencies of limited-logic systems by adding a first-order predicate-calculus theorem-proving capability. Briefly, this means that the computer program can produce all of the logical statements which are reachable by certain procedures from a group of input logical statements (or are consistent with these input statements). However, those that are *inconsistent* with the original statements cannot always be detected. Such deductive systems quickly become impractical as the number of input statements (elementary facts, axioms) becomes larger [6], [7], [16].

In a considerable understatement, Winograd says [7, p. 39] that as the set of axioms becomes large, but "well below the number needed for really understanding natural language, [a theorem prover] becomes bogged down in searching for a proof." Strategy-adding languages such as QA4 [17] and [18] have as their main goal overcoming this explosive growth in the number of logical-statement nodes to be searched as the set of input axioms increases:

> QA4 [seeks] to develop natural, intuitive representations of problems and problem-solving programs. [The user can] blend...procedural and declarative information that includes explicit instructions, intuitive advice, and semantic definitions. [17]

However, currently there is no body of evidence regarding the effectiveness of the programs written in this programming language on problem-solving tasks in general or language understanding in particular. That

---

*See p. 16 of this report.

is, there is a need for an experimental evaluation of the usefulness of the strategies which the language allows for and a determination of whether or not they can be made to bring about efficient search of trees derived from language-understanding problems. (For an example of such an experiment, see Fikes [22]. This paper discusses a problem-solving system for the SRI robot--the program STRIPS, which creates the plan; PLANEX1, which executes the plan; and the test by the theorem-prover QA3.5 of whether or not a number of possible preconditions for an action are "true.")

Procedural deductive systems seek to remedy another defect of "limited logic." Augmenting an existing store of complex information with new subject matter requires a new set of subprograms to deal with the new data. A cascade effect then takes place, where

> each change in a subprogram may affect more of the other subprograms. The structure grows more awkward and difficult to generalize... Finally the system may become too unwieldy for further experimentation. [5, p. 91]

A concomitant difficulty was stated by Minsky [6, p. 18] when he indicated that such programs

> will work best when given exactly the necessary facts, and will bog down inexorably as the information files grow.

Programming systems of two sorts have been developed to deal with this particular difficulty. The first is due to Woods [19], who assumed that semantic primitives existed as LISP subroutines. Winograd [7, p. 40] comments that Woods' data base was highly structured and that had it been more complex, "the same problems of interconnectedness described by Raphael" (see [5]) might have occurred. The second system is the first of several; it is called PLANNER [20], and it allows complex information to be expressed as procedures without requiring user involvement with procedure interaction details. Currently, alternative programming languages of this sort exist at MIT and have been used on various problems. These are a subset of PLANNER called MICRO-PLANNER,

developed by G. J. Sussman, T. Winograd, and E. Charniak, and an alternative approach called CONNIVER (there exists a paper called "Why Conniving Is Better Than Planning" [21]).

While this concludes the survey of programs and languages, it is worth noting that several of the following sections on restricted English, deductive question-answering, and relational models of data contain material which relates to this general subject. In order not to slight these contributions at this point, it is worth noting extensions to CONVERSE [13], [14], and [15]; the "Lunar Sciences Natural Language Information System" [23], [24], and [25]; and the continuing work on REL at Caltech [26] (which continues the approaches of [10] and [11]).

2.2 DEDUCTION AND ENGLISH STATEMENTS

Language understanding depends upon the ability to deduce various facts and relationships from given statements--this has been called "common sense" by J. McCarthy [23, p. 60]. Computer programs have been developed to address pieces of this problem in some cases and the entire problem in others (particular special problem domains). The primary purpose of this section is to explain in more detail the research on portions of the problem. Secondarily, we will point out areas where further development or experiments yielding quantitative measures of performance of existing systems on restricted subsets of English could be of value in bringing about useful computer understanding. As a basis for our discussion we will list a variety of terms which are commonly used in the literature. These generally were coined for text processing [26], where, in addition to machine translation [24], heuristic programming and artificial intelligence]23], [16], [25],* and information retrieval [24], [25], question-answering and the general intent "to bridge the gap between non-programmer users and data-base-oriented systems" [8, p. 6] were foremost in the minds of the designers. Nevertheless, the terms and the basic deductive systems are part of the software technology base, and they are keys to describing the current ARPA speech-understanding research program and its potential domains of

*In order to their relevance to heuristic programming.

applicability. Hence let us consider the definitions of these terms and a brief introduction to the related research.

The first term is most frequently used within the speech-research community, where it takes on a set of special meanings discussed in Section 3. Portions of an understanding program system are called the *front end*, since they perform the function of transforming language input into a form amenable to computer representation. This may be as simple as character-by-character encoding of alphabetic, space marker, and punctuation elements, or as complex as word and phrase detection and encoding. The usual computer science term for such a computer representation is *data structure* [27]. There are many types of data structures; for example, those utilized in DEACON are ring structures [11]. While in principle any structural relationship can be included in a data structure, in practice they tend to represent queue and tree-like associations. There are completely different types of structure inherent in the natural-lanaguage input itself, and these are far more complex--they correspond to the data structure term *list* [27]. The technical term describing language structure deals with the many levels of meaning which can be attached to typical English sentences (see the professors/petition example* below), and the common phrase to describe this is *the deep structure* of the language input. This has a great deal to do with word meanings and even connotations, as well as the context within which the word lies (and this may involve an arbitrary number of previous words). Clearly, deep structure is a highly complex thing, and though it includes meaning, semantics is not an adequate description of what it involved. There are two parts to its complexity: concepts and stored meanings. Winograd [7] discusses the former as follows:

> Language is a process of communication between people, and is inextricably enmeshed in the knowledge that those people have about the world. That knowledge is not a neat collection of definitions and axioms, complete, concise and consistent. Rather it is a collection of concepts designed to manipulate ideas. It is in fact incomplete, highly redundant, and often inconsistent. There is no self-contained

*P. 12.

> set of 'primitives' from which everything else can be defined. Definitions are circular, with the meaning of each concept depending on the other concepts. [7, p. 26]

On the other hand, in discussing a data structure for semantic information processing, Shapiro [28] asserts (emphasis added)

> Perhaps the most important criterion for understanding a language is the ability to relate the information contained in a sentence to knowledge previously acquired. This implies having some kind of memory structure in which the interrelationships of various pieces of knowledge are stored and into which new information may be fitted.... The memory structure in these programs [limited-logic systems] may be regarded as semantic, cognitive, or conceptual structures ...these programs can make statements or answer questions based not only on the individual statements they were previously told, but also on *those interrelationships between concepts* that were built up from separate sentences as information was incorporated into the structure...*the meanings of the terms stored in memory are precisely the totality of the relationships they have with other terms in the memory*. [28, pp. 3, 4]

Within highly restricted English domains, Winograd [7], Plath [8], Woods [33]-[35], and others have succeeded in obtaining useful deep structure from text. However, the limits on vocabulary are serious. In a wider domain of discourse it could lead to total elimination of the naturalness of language. Note that large numbers of possible successor words may result when dealing with a wider vocabulary or domain of discourse; this leads to the previous statement regarding the naturalness of language. (In technical terms, search of large trees makes heavy demands on the theorem-provers of today, which are highly inefficient and extremely heavy consumers of computer time; they may be unable to deduce valid proofs regarding the larger discourse subject.) As Thompson put it [10, p. 354], "English presumably does not prejudge the structural elements that exist among the elements of a universe of discourse." Thus, larger subsets of natural language should yield more and more varied, yet equivalent, deep structures.

Examples are given in [8, p. 28], where the term "underlying structure" is used, of identical tree representations of active and passive versions of two sentences.

a. Does ABC sell widgets?
   Are widgets sold by ABC?

b. ABC sells widgets.
   Widgets are sold by ABC.

The question is represented (in either version) by a tree with nodes labeled according to grammatical part of speech (clause, verb, noun phrase) and either qualitatively (question) or syntactically (past, present, etc.). By representing proper nouns as logical constants (via a node labeled INDEX), a form of data is derived that can be input into a theorem-prover. Winograd [7] uses a similar phrase-structure (deep, underlying) representation (he obtains trees similar to those in [8]; we reproduce some examples in Section 2.3, our discussion of grammar). However, both sets of trees are simple and useful because of the strict context and vocabulary limits.

> In practical terms, we need a transducer that can work with a syntactic analyzer, and produce data which is acceptable to a logical deductive system. [7, p. 28]

The above introduced the role of theorem-proving; the generally accepted term for the programs which utilize theorem-provers in language understanding is *deductive question-answerers*. Indeed the inference rule which is the basis of contemporary theorem-provers is illustrated in [23, p. 61] by a succession of resolutions which deduce a fact rather similar to those sought in language understanding (given "P1 If x is part of v, and if v is part of y, then x is part of y; P2 A finger is part of a hand; P3 A hand is part of an arm; and P4 An arm is part of a man," a proof that "P9 A finger is part of a man" is derived by steps, such as combining P1 and P2 to get "P6 If a hand is part of y, then a finger is part of y.") Further research on *protocols* (texts of computer-man interactions in a deductive question-answering mode) is needed.

This work should evaluate frequencies of computer-generated requests for redefinitions (eliminations of ambiguities in the input) by the user for various types of restricted English (or simply various lexicon sizes). For example, from [28], in order to handle the new input

"The professors signed a petition." Is not true

for which there are three valid deep-structure interpretations:

(a) The professors *didn't* sign a petition.
(b) *The professors* didn't sign a petition.
(c) The professors didn't sign a *petition*.

the computer would need to ask the user, "Is (a), (b), or (c) what you mean?"*

Finally, a separate area of related research, which has involved program development, includes both *relational data files* and *context constraints*. The relational files are essentially ways of including multitudes of facts so that they can be accessed [29]; an earlier work in this area of extending the use of computers to the realm of complex combinations of facts is [30]. It is important to note that the latter reference includes an exposition of *understanding* (pp. 3-10) and a discussion of the related concepts of *language and natural language* (pp. 11-17) which are highly relevant to the current research programs in text and speech understanding. Inadequate understanding can occur in some examples where a relational file may have to be augmented (by computer-query of a user) because the natural-language subset chosen yielded an insufficiently complex set of logical relationships. Thus the following block could result because the program might not have a logical connection between "garage" and "car," but only between "garage" and "house" (the program replies "OK" or "???" to user input sentences):

---

*F. Blackwell pointed out the existence of yet another deep-structure interpretation: "The professors didn't *sign* a petition.

I like Chevrolets.
OK
Chevrolets are economical.
OK
My house has a large garage.
OK
I can get two in.
???

Here, although there was no change of discourse subject, this was not "understood." In limiting discourse, a word may be included, but by not including one of its meanings or associations in a relational file, computer understanding may be blocked. The frequency with which this occurs on given specialized subsets of language should be studied. There is a direct connection to speech here through

> "homophones"--words which sound alike and are also spelled alike, but have different meanings: *fast*--avoid eating; *fast*--fixed color; *fast*--quick, speedy. [52, p. 141]

In order to develop basic knowledge concerning relational files and restricted English there is a need for the kind of protocol studies mentioned above in this context as well. Some extremely interesting protocol examples appear in [7] and have been written by operations of ELIZA [2]. Additional research on protocols has been carried out under the direction of H. Simon at Carnegie-Mellon University, with comparison of human and machine problem-solving the main objective. Also, a small effort on data-base-retrieval protocol research is part of current ARPA speech-understanding program development activities at SDC. However, there seems to be a need for quantitative experimentation with variation of parameters such as lexicon size, computing time, and number of possible logical relationships.

In the area of the quantification of logical relations implicit in word statements there is theoretical material presented in [31] and [32]. There it is shown that some statements ("who did not write _____.") are unanswerable and, in [32], that there is no algorithm which can detect whether a question stated in a zero-one logical form can be answered. Likewise, some experimental work on Woods' "moonrocks data world" is reported:

> "Woods reports that of 111 such requests...for example, '(What samples contain P205?)'...78 percent were handled completely satisfactorily, while another 12 percent failed only due to minor bugs in linguistic coding that were easily found and corrected. The remaining 10 percent failed because of more significant problems in grammatical analysis of semantic interpretation." [8, pp. 12, 13]

However, a carefully controlled set of experiments, with key parameters specified and varied on similar or standardized text, is needed. A seemingly trivial statement on context in natural language will conclude this section (more on context in Section 2.4). Plath [8, p. 16] discusses the role of word order by listing

> [the six] possible permutations of three English words....

(1) a. horses eat hay
    b. horses hay eat
    c. hay horses eat
    d. hay eat horses
    e. eat horses hay
    f. eat hay horses

He concludes that only three seem grammatical (a, c and f), and that they have *different meanings*, so that word order (an elementary form of context) has "an essential role in determining what (if anything) a phrase or a sentence *means*, not just what it is *about*" [8, p. 17].

2.3 CONTEXT AND GRAMMARS

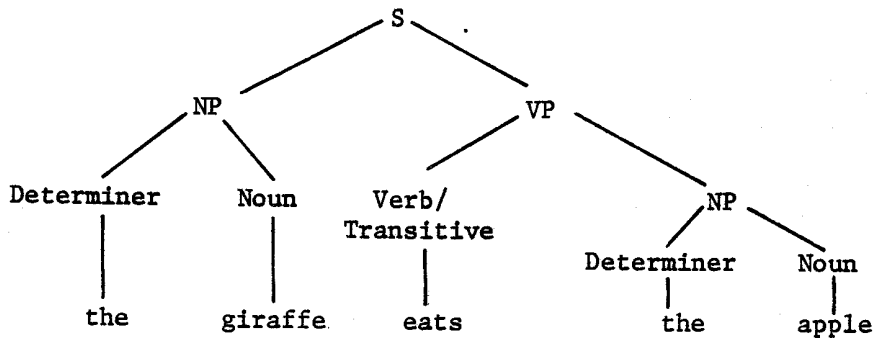
A basic technology for recent developments in text language understanding is the area of grammars or syntactic theory. A *grammar* is a system for making structural decompositions of strings of words; such a decomposition is usually represented as a tree structure and called a *parse*. Parsing programs may reveal whether a word string is valid or invalid (whether or not it is in accord with the given grammar), and examples of such programs and their operation are given in [7] and [8]. We note that both of these references are for restricted English vocabularies, yet in [7] PROGRAMMAR, the parsing program, implements

> (a) grammar...substantially more general than required for the world of toy blocks [which includes] aspects of English [such as] various aspects of tense, modality, quantifiers, and conjunction.... There are still whole areas of syntax which [are] not dealt with, particularly those which are used to convey the reaction, mood, and emphasis of a speaker. [7, p. 47]

Notice that a grammar acts like a pattern-recognizer which interprets a series of primitive elements constituting a whole--a phrase or sentence--and detects whether or not they make an object we allow. A more detailed view of a grammar sees it as a set of *rewriting rules* [16] and [17], and the latter reference gives a simple example (p. 81) of such rules:

1.1 S → NP VP
1.2 NP → DETERMINER NOUN
1.3 VP → VERB/INTRANSITIVE
1.4 VP → VERB/TRANSITIVE NP
1.5 DETERMINER → the
1.6 NOUN → giraffe
1.7 NOUN → apple
1.8 VERB/INTRANSIVE → dreams
1.9 VERB/TRANSITIVE → eats

While the last five are special (and somewhat frivolous), such rules as 1.1 and 1.2, which mean that a sentence (S) consists of a noun phrase (NP) and a verb phrase (VP), respectively, and that a noun phrase consists of a determiner and a noun, allow us to start with a symbol S and produce a sentence such as "The giraffe dreams." (The grammar rules are used to rewrite the intermediate elements (NP, VP, etc.) as they are obtained.) The use of rewriting rules in a manner that begins with a symbol representing a large number of entities, such as a sentence, is called a *top-down* approach. For the above rules, if we apply the list (1.1, 1.2, 1.5, 1.6, 1.4, 1.2, 1.7, 1.5, 1.9) in the given order, we get the following example (taken from [7, p. 81]) of the structural entity called a *parsing tree*:



Clearly, while the given rules can generate many different sentences, such as the last line of the tree, they have limitations. Before turning to these, and a discussion of context, let us point to the essential contribution of grammars to recognition--the introduction of a mechanism for employing *recursion*. That is, the rules may be used over and over. In [16], where the symbol $ is used to indicate an arbitrary string (possibly null), rewriting rules are given and applied to an example of parsing. A string of elementary symbols is operated on recursively by the rules of the grammar to decide whether substrings (and finally the whole string) make up a valid sentence. This approach is the *bottom-up* use of a grammar and it exemplifies the use of recursion. Thus:

$$\$_1 ab \$_2 \rightarrow \$_1 S \$_2$$
$$\$_1 aS \$_2 \rightarrow \$_1 S \$_2$$
$$\$_1 Sb \$_2 \rightarrow \$_1 S \$_2$$
$$\$_1 SS \$_2 \rightarrow \$_1 S \$_2$$

> [are] rules [of] the grammar defining sentences. [16, p. 30]

(That is, the symbol string "ab" located anywhere in a long string of symbols can be rewritten "S.") The example continues with a sequence of productions from the (bottom or elementary symbol) string abaabab:

abaabab
Saabab
SaSab
SSab
SSS
SS
S

As Minsky [37] pointed out, the descriptions generated by such procedures are "arbitrarily complex," yet the mechanism or set of rules is fixed and finite. However, rewriting rules are insufficient as a characterization of English because language has a high degree of context dependence. In other words, there are many sentences which cannot be parsed (recognized, understood by machine) on the basis of a finite set of rewriting rules of the type shown above.

The addition of tests for the presence of a feature in a symbol which is being rewritten by a grammar can be used to direct the productions to be of one type or another. This enables the expansion of similar objects (a question and the corresponding assertion, active and passive voices for the same assertion), such as clauses by *context-sensitive rules.*[*] In [7] Winograd presented 172 different syntactic features used by his PROGRAMMAR grammar and the function PARSE which it calls. Denicoff[**] pointed out that this is for a situation where there are no statements with psychological content and no use of simile. If the psychological meanings are added as in [38], these features would not be enough to describe all the possible meanings of a text drawn from a less artificial source. Indeed, a key problem which grammars seem ill-suited for is the reality that many contexts may be simultaneously valid. This has two aspects: (1) multiple meanings give natural-language communication the richness of overtones and subtleties--poetry carries this to an extreme; (2) the search of parse-trees and the use of semantics (look up related words) depend on a single context--

---

[*] A *context-sensitive rule* is a rewriting which is allowed when certain preconditions are satisfied.

[**] M. Denicoff, Office of Naval Research, Arlington, Va., in discussions with the author, March 1973.

both take geometrically increasing amounts of computing time as the number of contexts grows.

2.4 RELATED RESEARCH

Minsky [37] provided a cogent argument for the use of syntactic representations for patterns. The preceding section discussed grammars, the valid rewriting rules which enable syntactic processing. When more elements are added to a system involving a grammar, the number of trees which can be produced by the grammar's rewriting rules in attempting to verify the validity of (or "recognize") a string grows explosively. There is then the need for a *heuristic*--an *ad hoc* method of reducing the number of possibilities. The introduction of useful heuristics and the development of methods of planning problem solutions are both fundamental research areas in artificial intelligence. The research in this field is discussed in [16] and [23], and among the noteworthy references are McCarthy's 1958 and 1963 papers [6, pp. 403-418], the work of Newell and his colleagues (including the monograph coauthored with Ernst), and Amarel's papers (see [16, pp. 234-244] for detailed citations). Some aspects of this theory have been starting points for the problem-solving languages mentioned in Section 2.1 (PLANNER [21], QA 4 [18]). An approach closely related to the goals of language (and speech) understanding is that due to Feigenbaum and his coworkers on planning and heuristics in the limited-context world of a specialist (see [39], which discusses the problem-solving capability of DENDRAL, a program dealing with organic chemistry). In general, the purpose of the heuristic is to enable the calculation of an evaluation function on the nodes of a tree. This function is used to order these nodes--the successors of the original string produced by the grammar--for selection of the next node for rewriting. (This is discussed in detail in [16]; both the dynamic programming approach used in [44] discussed below, and special fast sequential decoding techniques [51], noted in [16], are related to our discussion of artificial-intelligence terminology.) Languages like PLANNER [20] enable automatic control of the *backtrack process* (the selection of a previously nonrewritten node when the chosen nodes and rewriting rules yield an inconclusive result);

also see [53] and [59] for related developments. Since the purely syntactic aspects of the software are well established, this section concerns the performance of some systems which utilized context (semantics, meaning) as part of the heuristic for reducing the search process. This is in keeping with our earlier observation that lexical and semantic constraints and their interaction with syntactic rules remain as the key research area for both (1) theoretical aspects of artificial intelligence related to "understanding" software systems and, (2) practical speech-understanding capability. As a guide to the material in this section, let us note that at a theoretical level all understanding systems are using some form of structural information to supplement the syntactical or grammatical pattern recognition (see K. S. Fu and P. H. Swain [40]; subsequent work by Fu introduced the important concept of *stochastic grammers* to parallel the complex human pattern-recognition capability). Some structural information is deducible from meaning (semantics), some from words or sounds (lexical information), and some from the limitations imposed by the domain of the patterns (restricted English, rules of games, and pragmatic observations). Supplementary structural information was first used by Vicens [41] as part of his Ph.D. research to build the Vicens-Reddy speech-understanding software system. His advisor, Reddy, has a chess-based speech system [42] in operation (part of the ARPA Speech Understanding Research project). These systems are prototypes of what we expect to be developed in the future. (See Section 3 for a related idealized speech understanding system.) Here we will discuss the use of supplementary structural information in software developed for problem areas other than speech understanding.

Alter [43] suggested the use of syntactic decoding of acoustic data and simulated this by correcting error-containing alphanumeric strings which were to be recognized as valid FORTRAN statements. His examples include complete parsing of a correct seven-character string program statement in 0.489 second.* Striking results in overcoming

*UNIVAC 1219 time; 16,384 eighteen-bit words of core memory; syntax control and assignment statements, but not input-output or specification statements, in FORTRAN.

errors are shown: for the true input FORTRAN statement "RVA=Ø" (zero) which required this time for processing (i.e., string pattern recognition), the program corrected "RVA=Q", "RQA=Ø", and "RVA±Ø" to the desired "RVA=Ø" in 3.65, 1.00, and 1.45 seconds, respectively. It failed to get "QQA=Ø" correctly (it came up with STOP Ø after 4.01 seconds) and took as long as 18.44 seconds to correct "RVAQØ" to "RVA=Ø". Similar results are given for "CONTANUE", "CONINUE", "CONIGUE", and other errors (corrected to "CONTINUE" in the same order of magnitude of processing time as it took to recognize input of the correct word). At about the same time Duda and Hart [44] independently conducted an experimental study on hand-printed FORTRAN programs. Their program, the context-directed analyzer, used compound decision theory. What it did was to

> Compute the confidence of every string of characters of the given length. Bias each string confidence by adding the logarithm of the prior probability of that string. Set the answer equal to the string having the highest biased confidence. [44, p. 1140]

This was implemented as a LISP program, which was run on an SDS-940 computer. The program employed three techniques in addition to compound decision theory. The first technique was syntactic--any FORTRAN-illegal alternative character string was given zero probability. Thus in the five-character string given there as "6" or "G," followed by "O," then "T," then "D" or "O" or "=," and last, "S" or "5," only four FORTRAN statements could be present, among them "GOTO5." The semantic level in FORTRAN was used as another error-resolution technique: facts such as the multiple appearance of variable names, the location of control words "(DIMENSION, IF, etc.) at the beginning of all statements except the arithmetic assignment statement," etc., are "meaning assigning"; an example of the resolution of arithmetic expressions by methods based on similar facts is discussed in detail in [44, pp. 1143, 1144]. As in all semantic-level processing the technique hinged on the successful location of key elements--here the delimiters +, -, / and *, which were used as potential partitioning points in the string. The need for

partitioning arises because there is a combinatorial explosion in the number of possible strings with increasing length (characters). The search method they employed is the third technique of their program. It reduced the search somewhat via

> a modification of dynamic programming [which] while considerably more efficient than the brute force approach and... used frequently in the analyzer implemented,...also suffers from severe combinatorial problems and can be used only on combinatorially simple data structures. *For our* [SDS-940] *computing facilities, the limit of combinatorial complexity for dynamic programming seems to be something on the order of a few thousand combinations; i.e., a string of five or six characters, with about four alternatives for each one.*" [44, p. 1141] (emphasis added)

The importance of the limitations of the combinatorial type are that they are intrinsic to the problem and cannot be overcome completely by improvements in computer hardware technology. Nevertheless, such search-method improvements can increase the string lengths and number of allowable alternatives that could be processed. Furthermore, as the GOTO5 example showed, both syntactic and semantic levels can be used to effect large reductions in the number of candidates (acceptable strings). While interaction was not contemplated in [43] and [44], the understanding technology assumes computer-generated feedback--what Carbonell [49] has called *mixed-initiative discourse* [49, p. 194]. This element of interaction has allowed the LISP program DWIM (for "Do What I Mean") to be a useful working tool for text-input error-correction. This is discussed by Teitelman in [45], [46], and [50]:

> A great deal of effort has been put into making DWIM "smart." Experience with perhaps a dozen different users indicates we have been very successful: DWIM seldom fails to correct an error the user feels it should have, and almost never mistakenly corrects an error. [45, p. 11]

Similarly, a useful limited-discourse user-interactive program which implements syntactic, semantic, and backtrack search software technology is described by Zobrist [48] for chess. The program uses a search

with a maximum look-ahead depth of 20. There are 3000 terms summed in the evaluation function, each with a δ_1 (0 or 1) weight, which depends on whether a board position occurs. Some comparable-depth look-ahead programs would need to examine 500,000 board positions. With advice-taking, i.e., facts input by a chess expert, this is reduced to 15,000 positions and the total of 45 million calculations of δ_1 needed takes 20 seconds of IBM 370/155 time. No natural-language-handling program is as fast, although there is a version of SNOBOL (SPITBALL) which is comparable. Note that the chess program usually takes about 25 seconds to decide a move. It takes only 0.23 microsecond to calculate *one* δ_1, and this is on a machine which needs 1.3 microseconds to do a branch instruction.

The program can do 3000 parses in 10 seconds. Most of the speed is obtained by programming tricks. All masks (templates, patterns) that involve cell 1 are put in the same pile, all that involve cell 2 are put in another pile, and all that involve cell 64 are put in the last pile. Then the sorting is reduced from 12,000 items in one machine-language loop to 64 subloops of 200 items each. This process is repeated by further sorting, by piece type within each cell, the j piles of patterns, j = 1, 2, ..., 64; i.e., those which involve a queen are grouped, those which involve a rook are grouped, etc. This trick is a form of semantic organization of the lexical file. Clearly, it is highly advantageous and enables much more powerful syntactic processing.

The relationship of semantic and syntactic information is a research topic which needs a quantitative theoretical basis. That is, to what extent are parsing and meaning interconnected in restricted-discourse domains? Put another way, if we put in parameters--number of words in a lexicon, number of meanings per word, number of parts of speech per word, probability of word-to-word transformation errors--what is the change in amount of syntactic processing observed as one or more of these is varied? How do we incorporate new words into an established framework? That is, can we quantify the expansion of a lexicon? A nontechnical example of the way we expand our own understanding of words follows from (1) reading, and (2) thinking about the new words in the beginning of *A Clockwork Orange* by Burgess [47] (we italicize those

words which have no natural English meaning outside that which the reader gives them from their context):

> "What's it going to be then, eh?" There was me, that is Alex, and my three *droogs*, that is Pete, Georgie, and Dim, Dim being really dim, and we sat in the *Korova* Milkbar making up our *rassoodocks* what to do with the evening, a flip dark chill winter bastard though dry. The *Korova* Milkbar was a milk-plus *mesto*, and you may, O my brothers, have forgotten what these *mestos* were like, things changing so *skorry* these days and everybody very quick to forget, newspapers not being read much neither. Well, what they sold there was milk plus something else. They had not license for selling liquor, but there was no law yet against prodding some of the new *veshches* which they used to put into the old *moloko*, so you could *peet* it with *vellocet* or *synthemesc* or *drencrom* or one or two other *veshches* which would give you a nice quiet *horrorshow* fifteen minutes admiring *Bog* And All His Holy Angels And Saints in your left shoe with lights bursting all over your *mozg*.

A glossary for the entire Burgess book was compiled by Stanley Edgar Hyman [47, pp. 186-188] from the text, i.e., from word contexts, and from external clues. The following are examples of such clues:

1. Presence of a related word in another lexicon. (Hyman found that many of Burgess' words were of Russian origin.)
2. Conventions such as capitalization of proper nouns.
3. Variations which transform an unknown string of letters into an ordinary word (as in an anagram or a case of spelling backwards).

3. SPEECH, UNDERSTANDING-TECHNOLOGY APPLICATIONS, AND RESEARCH DIRECTIONS

Speech recognition includes a variety of difficult problems and useful application areas; for example,

1. Word-by-word transcription (voice-typewriter, perfect speech recognition).
2. Speaker identification.
3. Speaker verification.
4. Language identification.

However, this section deals exclusively with speech understanding as first demonstrated by Vicens [41], that is, we will not address any of these examples specifically. Speech-understanding systems (SUSs) are motivated by different concerns than found in speech recognition. They should be viewed as extensions of text-understanding and linguistic or structural pattern-recognizing systems into the realm of spoken words, rather than as new solutions to the above difficult problems. This section concerns the nature of current speech-understanding research. In general, we consider SUS software performance evaluation, modeling multilevel decisionmaking, and specialized language design to be research directions which should be explored thoroughly in parallel with the current ARPA speech-understanding research (SUR) program.* We also present two understanding-technology applications, signal processing and password design, and point out the usefulness of the SUR program in extending our capability for learning about natural speech. Ultimate long-range developments of the program may, as in any basic research, result in contributions which are distant

*There is a distinction here between SUS research and the SUR program. "Speech-understanding research" is usually synonymous with the ARPA program. This program began with five contractors engaged in SUS research-development of research prototype speech-understanding software systems. Other contractors are now engaged in speech-understanding research not designed to produce such SUSs.

from the original problem area. To begin this exposition, we will discuss first some specifics of speech-data processing, in general, and SUSs, in particular.

There are several fundamentals that appear in most current research on speech:

1. Variable segmentation of sampled speech data.
2. Acoustic classification of speech segments (e.g., stop, vowel, consonant; phoneme or other speech unit).
3. Lexical (linguistic, word level) comparisons of possible source strings of acoustic elements, sometimes called decoding.

Processing speech data by computer software that involves syntactic, semantic, and contextual information (world model, user model, domain of discourse) has made SUSs capable of

1. Tolerating errors in segmentation and acoustic classification (lexical and semantic processing is used to overcome these errors).
2. Recognizing messages without perfect word recognition.

The virtue of doing speech-data processing by means of natural-language understanding lies in the similarities to what is observed about interactions between people. First, we observe that inaccuracies in human speech recognition at the word or syllable level (listener recognizes a different word than speaker said) are frequently overcome, so that the general drift of what was said is perceived. Second, there are special sets of sounds which are indistinguishable unless nonacoustic information is used--for example, the homonyms "weight" and "wait." Finally, there is evident in speech data the kind of continuous deformation of one signal into another that makes an error-tolerant approach the natural one to use from the outset.

This section presents an idealized SUS, along with some possible research directions which could be explored to determine the ultimate

processing power to be derived from multiple-level understanding-software. Possible applications of understanding-software to speech research, signal processing, and password problems are suggested (each application is actually a potential R&D program with medium-term benefits to government agencies--military and nonmilitary--and industrial users of computers).

3.1 AN IDEALIZED SPEECH-UNDERSTANDING SYSTEM

A computer program that implements an SUS is exemplified by the accompanying flowchart (see Fig. 3.1) and set of functional definitions of program modules (see Table 3.1).*

In operation the idealized SUS consists of (1) files which vary in both contents and length, and (2) probability-computing and ranking programs. The adjustment of the files to the discourse subject is implemented by CONTEXT. This is envisioned as one of three world-model programs: (1) interactive, (2) prestored, or (3) second user.

In an interactive system, user-computer dialogue is presumed. This would take the form of computer-generated speech, "Give your name, rank, and serial number," or "Say the digits zero through nine," followed by the user response. The context is partially a user model (as these examples indicate) and partially a world model. An example of the latter would be a response to the computer query, "Talk about vehicles." Page 13 contains a dialogue fragment with three accepted responses (which could be used to set up a word file) and one unaccepted response; these could follow a computer-generated query of this sort. Clearly, this dialogue allows the establishment of a possible world model; namely, a restricted set of words pertaining to vehicles.

In a noninteractive system a world model is essentially a prestored context, implemented by complicated software which thoroughly describes an interrelated set of facts. Because of the highly variable nature of speech among different speakers and the richness of associations possible with the spoken English language, both our

*The flowchart and table are new; they were designed to illustrate concepts of SUS programs.

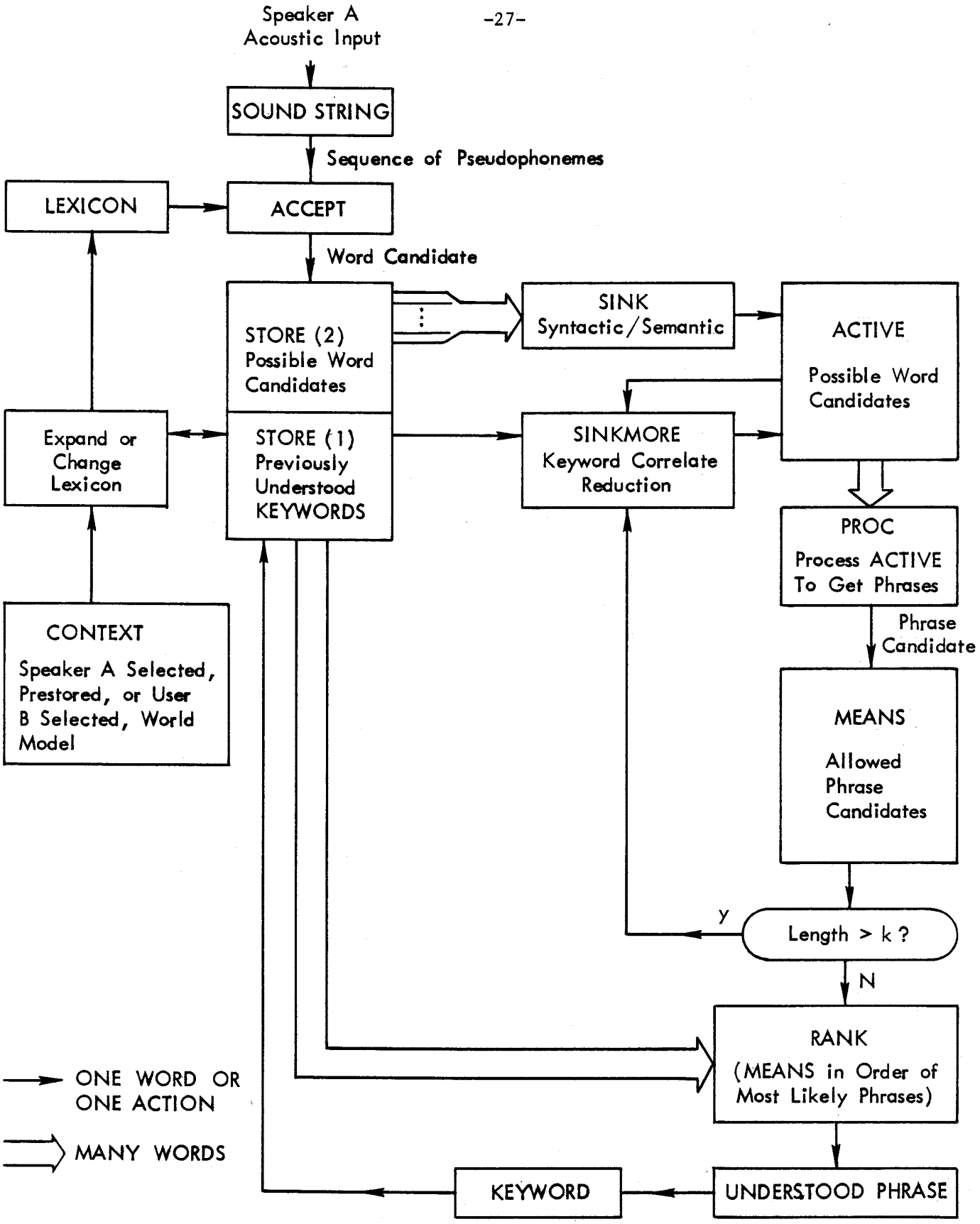


Fig. 3.1 — Flowchart of an idealized speech-understanding program

Table 3.1

FUNCTIONAL DEFINITIONS OF SPEECH-UNDERSTANDING SYSTEM PROGRAM MODULES

Let "ACCEPT"	accept (possibly many) word candidates from a given acoustic-level sequence and a given lexicon.
Let "STORE"	be saved files:
STORE (1)	key words from previous "understood" phrases; and
STORE (2)	current accepted word candidates in a "nonunderstood" string; with the plurality, definiteness, part of speech, tense, etc., kept for each word.
Let "SINK"	delete possible word candidates from STORE (2) based on syntactic/semantic criteria on a flexible length word string to create the file "ACTIVE."
Let "PROC"	process ACTIVE to create a file of candidate phrases "MEANS."

If length of MEANS $>$ k, go to "SINKMORE," otherwise to RANK.

Let "SINKMORE" delete words from ACTIVE based on consistency with STORE (1). Iterate until there are only k phrases in MEANS.

"RANK" is MEANS ordered by using STORE (1) (MEANS in order of most likely phrases).

Top of RANK is "UNDERSTOOD PHRASE." "KEYWORD" is extracted from this phrase and added to STORE (1). This action can expand or change the "LEXICON," the file of words being considered as possible in the current world model. "CONTEXT" defines the world model (realm of discourse). It can be selected by the speaker's responses to prestored questions (i.e., in dialogue with the computer), prestored itself, or input by a nonspeaking user who is investigating the speech record.

idealized SUS and the current ARPA contractor research programs deal only with special and highly restricted contexts. Thus a prestored context can be developed for a limited-vocabulary limited-concept system, and it is possible to store several such world models and call them upon detection of significant key words.

An alternative approach would be to allow a second user of the system to control the sequencing or expansion of prestored world models. A typical example could involve monitored speech, and the second user

could search a given record several times, changing the context for each search.

As the SUS functions, it accepts (ACCEPT) word candidates which correspond to the processed time-sampled acoustic data. We have indicated by SOUND STRING a processor which generates a sequence of "elementary" sounds we call "pseudophonemes" (some authors use "transemes") that possibly correspond to the digitized acoustic data. Both for pseudophonemes and word candidates there may be many possibilities corresponding to a given lower-level source (i.e., digitized acoustic data for pseudophonemes, pseudophoneme string for words). Lists of possible word candidates and previously understood key words (possibly input from CONTEXT) are kept in STORE (1). The outputs of SOUND STRING and ACCEPT contain start/stop time information so that in STORE (2) and in subsequent processing, coherence (correct order of words and sounds) can be established. A word string of flexible length is input to SINK from STORE (2). This program module uses syntactic and semantic criteria to delete word candidates. The output, a reduced set of possible word candidates from STORE (2), is placed in a file called ACTIVE. PROC takes elements from ACTIVE and creates phrase candidates which are stored in MEANS. If a phrase is too long, it is sent to SINKMORE, where it is compared in semantic content with key words from STORE (1). Words which do not correlate with these key words are deleted; only confirmed words are kept in ACTIVE. If a phrase is of acceptable length, it is sent to RANK, a file like MEANS except that the elements are stored in order of their likelihood (probability of occurrence). The top of RANK is called UNDERSTOOD PHRASE, and a KEYWORD is extracted from this phrase and added to STORE (1). Reading in associated words from a dictionary tape can cause an expansion of or a change in the LEXICON.

An overall control program steps the system through the acoustic data and establishes the length of word strings. The control program contains software which computes likelihoods for pseudophoneme and word strings. Comparisons of these likelihoods are used to establish lengths of possible word strings considered by SINK. Several control

functions are implemented by nondeterministic programs* [16]. Searches of large trees are repeatedly conducted, and any of several paths can be taken in that search. For example, continuing with the elementary symbol string abaabab, the rewriting rules (see p. 16 of this report, p. 30 of [16]) give many possible ways to obtain "valid sentence S." The other paths through the search, which could be found by a program nondeterministic in start point, are these:

abaabab		abaabab
abaSab		abaabS
abaSS	and	SaabS
SaSS		SaSS
SaS		SaS
SS		SS
S		S

3.2 UNDERSTANDING-SOFTWARE PERFORMANCE EVALUATION

Performance evaluation of understanding-software concerns measuring the actual operation of such a system. For convenience, we will assume the software operates as the above idealized SUS does. Systematic experimentation combined with monitoring and measurement of the understanding-software is needed. The experimentation data base in some understanding applications could be variations of the test examples (blocks on a table in [7], kinship relationships in [13], etc.). Some measurements which we believe necessary are described below. Before entering into the details of the description (which will depend on Fig. 3.1 and Table 3.1), we will briefly describe the probable result of performance-evaluation analyses of understanding-software.

Relational data files, theorem-provers, and SUSs share the property of working well on small data sets and being "practical-failures"

*A nondeterministic program allows for one of several actions to be executed after reaching certain program states. Choice of which action is taken can depend on a pseudorandom number, a data condition, or a cycling among the alternatives. In this application, nondeterministic control programs can be used to avoid excessive processing at a time when insufficient information is in the files for probable success.

on some problems involving more data (i.e., being overwhelmed by the size of the trees which must be searched to yield a decision). As we will indicate in Section 3.4, observed performance strengths and weaknesses should clarify how to design specialized languages for understanding-technology computer programs. A particular example here could be a limit on the number of possible semantic associations of a word. (In theorem-proving terms, this would restrict the number of resolutions needed.) That is, given a test problem of n entries (e.g., related individuals in the kinship case, words in speech- or text-understanding lexicons), with each entry possessing no more than ℓ syntactic associates, or m semantic associates, careful measurement of the program operation could reveal combinations of large n and ℓ with small m which were successfully processed.

Of interest first are gross measurements about the program and its performance: primary storage required for the program and its data, the number of instructions in the understanding program, and the actual central processing unit (cpu) time needed to understand a spoken message of a given duration, for several samples of the experimentation data base. The detailed measurements of interest concern locality properties of the program: We would like to measure the amount of time spent in a given program module in Fig. 3.1 and time sequences of executions of program instructions. A key to improved designs of understanding-systems will be whether current programs are predominantly executing likelihood ranking, syntactic parsing, or semantic deduction on various test problems. Likewise, it is important for applications to know whether there are groups of instructions which execute (a) rarely and (b) frequently.

For an interactive understanding-system (see p. 26) an important measurement would be the rate of unaccepted input statements (i.e., user input statements with computer responses):

"?"
"I don't understand _____," or
"Do you mean _____ or _____?"

The last two computer replies indicate that key words may be objects *to be defined by the user*. We introduced this notion at the end of Section 2, where the words were implicitly defined by their place in an otherwise understandable text. The potential usefulness of this flexibility in both military and nonmilitary security applications will be discussed in Sections 3.4 and 3.6.

Finally, it would be extremely useful to measure the effectiveness of the system with inputs that possess different degrees of required formatting:

Require: (noun phrase) (verb) (noun phrase)
Require: (three words) (pause)

The measures of effectiveness could involve actual operating time or program simplification effects (the latter could be measured crudely by program length in instructions).

3.3 MODELS OF MULTILEVEL DECISIONMAKING

This section proposes to view understanding technology as a form of multilevel decisionmaking, where software subroutines (theorem-provers for semantic deduction, parsing routines for syntactic acceptability) act as probabilistic building blocks. The assumption we make is a modeling one for understanding possible limitations on the current organization of SUSs in particular. (Of course, any research results obtained from the kind of mathematical model described herein apply as well to text understanding and other nonspeech applications.) An example of some further modeling assumptions is given in the following paragraph.

Let the functional elements of an understanding system be assigned success-probability values for a typical message segment, and let each message segment be given numbers of associates at each level by choices of random numbers. That is, we would first assign p = probability of correct symbol-to-word grouping, q = probability of correct word grammatical classification, and r = probability of correct semantic (word meaning) recognition. Then, independently, we would choose π, σ, ρ,

three random numbers which represent the number of symbols in a word, the number of words in a phrase, and the number of words with common semantic attributes (e.g., Chevrolet, car, garage, house in the example on p. 13). The numbers thus chosen represent required uses of their associated functional element (lexical, syntactic, or semantic processing subroutines). Each use has the assigned success probabilities.

The preceding paragraph gives some assumptions we need for an applied mathematics research study to answer the question, "What is the overall success probability for a multilevel decisionmaking system with given level-by-level success-probability parameters?" Such a study would be useful for delimiting the capability potential of an understanding-technology-based system; upper limits should be found for overall success probability as a function of the level success probabilities or the probability distribution parameters of the random numbers governing the use of each program level. A mathematical model of understanding-technology decisionmaking could have impact on practical SUSs since it would give numerical values for machine understanding as a function of lexical, syntactic, and semantic processing parameters. That is, as the number of words in a lexicon or the number of words with common substrings increases, overall recognition should take more processing time. Modeling this as success probability given a fixed amount of processing time yields a useful parametric representation which could influence the size of the lexicon to be understood.

3.4 LANGUAGE-DESIGN RESEARCH

Many military commands are designed to be intelligible in difficult communication contexts (e.g., a high level of ambient noise). Frequently commands are understood and verified by a response which is artificial. Control systems employ acronyms and other nonstandard words which have limited possibilities of interpretation at other semantic levels. Hence for spoken language a useful research direction would be the design of a specific language for limited-domain discourse which has a high probability of having unique substrings. That is, once a command is spoken, as long as any substantial substring of a

word is recognized (such as three phonemic elements out of six or seven in the word), relatively few possibilities exist in the lexicon with that substring.

In essence the design problem is the dual of the modeling problem discussed in the previous section. What we have exemplified by considering substrings of sounds from a portion of a word is design for the lexical level of recognition. Similar considerations apply to the syntactic and semantic levels. Indeed as [61] (a companion report) notes, there is a military tradition of specialized syntax--a required order for reporting various items. We call this *formatting*. The question, "How much formatting is needed to enable current understanding-software to function at high performance levels (for example, 100 percent overall recognition-success probability) on a given size vocabulary?" is a second language-design research topic.

Third, it would be desirable to know how a vocabulary with meaning variation according to word order or word combinations could be used in combination with understanding software to achieve high rates of accurate message recognition. That is, given constraints on communication channel (time and bandwidth) and computer processing (memory size and central processor time), a limited vocabulary can be designed to communicate more information by using context-dependent meaning. The design of an ambiguity-free, limited-association, limited vocabulary is a language-design research task which should be studied.

The viewpoint taken here is that current artificial languages (e.g., FORTRAN) have not been designed to have a useful relationship of meaning and context (the example from [43], discussed in Section 2.4, and those in [44] show the usefulness of the context and format clues present for eliminating errors in FORTRAN; however, the coding conventions were not chosen to enable the assignment of meaning to a message substring to be deducible from the message context). Military commands traditionally have been designed to have sequences of data which are unambiguously recognizable. It could be highly useful to learn how substrings of words, which are themselves recognizable words in a command language, can be eliminated as candidates for the entire string by contextual processing using meanings. The result should be a command language well-adapted to man-machine communication.

3.5 SIGNAL PROCESSING

Signal processing concerns the computer analysis of numerical time series. Radar and sonar systems, seismic sensors, and pressure transducers are some of the technological devices that generate such data. Usually the time series contains recognizable special shapes (graphs or curves). This section describes how such shapes or *patterns* can be represented and recognized by understanding-technology software. The process of recognition would be accomplished by processing a set of primitive elements (the symbols in the input string to the recognition software) according to certain acceptability rules (a grammar). The main contribution of this section is the view that real phenomena (clutter in radar, reflections from a reef located by sonar, seismic signals deformed by a nearby large mountain range, temperature variations which influence transducer performance) can be dealt with by world-model or semantic-level processing. Hence a signal-processing system could be designed using speech-understanding software.

The literature contains recent information on primitives and grammars for (linguistic or structural) pattern recognition: Reference [55], on linguistic analysis of waveforms, describes techniques for representing patterns in time series by a string of primitive symbols. Reference [40] is an excellent introduction to syntactic (grammatical, linguistic, structural) pattern recognition. Recent texts [56, pp. 216-619, and 57, pp. 426-435] also contain material on this subject. Algorithms have been studied [58, 59] for segmenting time series automatically; hence software is practical for coding signals into a string of primitives like phonemes in speech data.

Some signals are well-formed, and a grammar of acceptable, primitive elements could be defined. Specific target signatures could be prestored in primitive-string form (analogs of words in a lexicon). Special environmental conditions would constitute semantic associations of input strings of primitive symbols. Hence understanding software could be used so that a signal-processing system could deal with (a) specific targets, (b) local obstacles, and (c) variation caused by daily, seasonal, or sporadic climatic changes.

3.6 PASSWORDS AND PRACTICAL CONSIDERATIONS

A possible application of understanding software is in the area of passwords. Dishon* suggested that it could be useful to secure access to time-shared computer-system terminals by requiring the user to speak some phrases. The suggestion is in the realm of speaker identification, yet it provokes the following SUS application question: Can a system which allows a user access by a series of statements, each to be stated only so accurately as to be understood, be (a) as secure as an "exact-input-required" system, and (b) more useful to a potential system user?

The suggestion is that there is a possibility of dialogue where the computer selects the next query based on prestored world and user models and an imperfect input. The sequence of inexact inputs could be easily remembered by an authentic user yet difficult to simulate by an impostor.

Indeed, the password-understanding system described here has additional potential advantages. A time-varying key could direct the discourse subject initiation, for example, by beginning at a different point in a standard situation, such as the first, second, or nth move in a simple chess opening. A user could periodically add new words to his own computer-stored vocabulary (the user model or lexicon) to confuse an intruder. (These could be coinages or strange variations: doof, yenom, etc., as synonyms for food, money, etc.) Note that we have emphasized the understanding aspects of this idea, not the speech aspects. Clearly, it is possible to explore the potential of this concept by a text-input-understanding system.

Some aspects of understanding systems which concern their potential for performance in practice are these:

1. The number of primitive elements and their relative frequencies.
2. The number of grammatical rules which restrict allowable combinations of primitives.

---

*Col. Dr. Dishon, Ministry of Defense Computer Center, Ramat Gan, Tel Aviv, Israel, during discussions there with the author in July 1973.

3. The number of words which combine to effect meaning and the number of associated properties of each word in a vocabulary.
4. The speed of decisionmaking at
   a. The syntactic level (use of item 2).
   b. The semantic level (use of item 3).
5. The complexity of the relationships between words and properties mentioned in item 3.

Cooper* has said "It looks like there is a grammar right down to the phonemes." Pierce [60] has published an excerpt from a highly improbable text, "the 1939 novel *Gadsby*, by Ernest Vincent Wright [which] violates the statistics of English although it violates neither grammar nor sense. Wright's entire novel of more than 50,000 words was written without a single word containing the letter e." Taken together these items indicate that practical SUSs will also depend on the following:

6. Accurate knowledge of the elements of speech and their possible combination in natural and artificial language.

Practical considerations regarding the software itself relate to limits on input data (format, computer-asked questions to initiate a dialogue) and means for combining world-model and user-model information with current speech. That is, syntactic and semantic processors (such as theorem-provers in the latter case) function more rapidly on limited amounts of data. (There is an explosive growth of processing time as the number of logical attributes--meaning, semantics--or as the vocabulary properties--parts of speech a word may be, syntax--increase.) Hence a key to success for understanding-software will be techniques for pruning the size of entries (or number of calls) to syntactic and semantic processors by matching the input stream to the prestored data (the internal model held by the software).

---

*Dr. Franklyn S. Cooper, Haskins Laboratories, New Haven, Conn., in discussions with the author and Dr. A. Hoffman of Rand, March 1973.

4. CONCLUSIONS

Computer programs which exhibit "understanding" are a highly desirable and potentially practical way to present a computer user with a more convenient machine interface. That is, they represent an evolutionary development much like the trend from numeric code (machine, assembly language) to higher-level programming languages (FORTRAN, PL/1). In the development of computer science research, understanding represents a limited step towards a natural interface.

Historically, goals now seen to be overly ambitious--machine translation, natural-language input--have been replaced by related research on limited-vocabulary specified-context man-machine interaction. This research has led to a number of positive developments--working software which accomplishes desired actions or goals from essentially natural computer input--and many of them are mentioned in the framework of Section 2, which surveyed the general field of natural-language processing. However, the use of a keyboard to input text to a computer is a serious constraint on the implications of the term natural language.

Work from such diverse fields as speech recognition and artificial intelligence is summarized in the latter parts of Section 2. The document that best indicates the future possibilities from combining research in speech and artificial intelligence is the final report of a study group chaired by A. Newell [62]. That report established design specifications for computer input or computer dialogue utilizing speech by the human being. These specifications are actually software research and development target goals and they are stated in terms of the size of vocabulary, number of speakers, need for training (speaker reads a predetermined list of words before beginning to use the machine's understanding capability), and many other considerations. In this report we state numerous research subgoals not apparent in the contents or implications of [62]:

1. A need for an experimental evaluation of the usefulness of the search and problem-solving strategies which can be

implemented in artificial-intelligence programming languages.

2. A determination of whether the strategies from artificial-intelligence programming languages are effective for language-understanding problems (i.e., can they bring about efficient search of those trees which result?).
3. The need for development or experiments to yield quantitative measures of performance of existing language-understanding programs (speech and text--nonspeech) on specific restricted subsets of English.
4. The need for research on texts of man-computer interactions as quantitative parameters are varied (e.g., lexicon size, number of possible logical attributes).
5. The development of a quantitative theory relating semantic and syntactic information in restricted-domain limited-vocabulary situations.

A set of related research goals is presented in Section 3. Many of these amplify and some extend the five concepts stated above. Most are presented in a shorter paper entitled "Applications and Goals of Speech Understanding Research" [63], which is based on Section 3. A flowchart of an idealized speech-understanding system is presented and used to state some possible applications of understanding-technology and several potential research goals. Some of these goals can be described as research problems in pattern recognition and computational linguistics, including the following:

1. Models of multilevel decisionmaking
2. Language-design research
3. Signal processing
4. Passwords and practical considerations

To summarize, a new high-technology research area--computer software--has developed over the past two decades. This report describes some of the results achieved over that time period as they

relate to a current research effort to achieve computer understanding of human speech in a limited-vocabulary restricted-domain-of-discourse context. This report contributes several potential applications and possible research goals of the current speech-understanding research program.

REFERENCES

1. Nievergelt, J., and J. C. Farrar, "What Machines Can and Cannot Do," *Computing Surveys, 4*, June 1972, 81-96.

2. Weizenbaum, J., "ELIZA--A Computer Program for the Study of Natural Language Communication Between Man and Machine," *Comm. ACM 9*, January 1966, 36-45.

3. Weizenbaum, J., "Contextual Understanding by Computers," *Comm. ACM 10*, August 1967, 474-480.

4. Bobrow, D. G., "Natural Language Input for a Computer Problem Solving System," in M. Minsky (ed.), *Semantic Information Processing*, MIT Press, Cambridge, Mass., 1968, 135-215.

5. Raphael, B., "SIR: Semantic Information Retrieval," in M. Minsky (ed.), *Semantic Information Processing*, MIT, Press, Cambridge, Mass., 1968, 33-134, 256-266.

6. Minsky, M. (ed.), *Semantic Information Processing*, MIT Press, Cambridge, Mass., 1968.

7. Winograd, T., *Understanding Natural Language*, Academic Press, New York, 1972.

8. Plath, W., "Restricted English as a User Language," IBM T. J. Watson Research Center, Yorktown Heights, New York, 1972.

9. Green, P. F., A. K. Wolf, C. Clomsky, and K. Laugherty, "BASEBALL: An Automatic Question-Answer," in E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963.

10. Thompson, F. B., "English for the Computer," *Proc. FJCC*, Spartan, New York, 1968, 349-356.

11. Craig, J. A., S. Berezner, H. Carney, and C. Longyear, "DEACON: Direct English Access and Control," *Proc. FJCC*, Spartan, New York, 1968, 365-380.

12. Kellogg, C., "A Natural Language Compiler for On-Line Data Management," *Proc. FJCC*, Spartan, New York, 1968, 473-492.

13. Travis, L., C. Kellogg, P. Klahr, *Inferential Question-Answering: Extending Converse*, System Development Corporation, SP-3679, January 31, 1973.

14. Kellogg, C. H., J. Burger, T. Diller, and K. Fogt, "The CONVERSE Natural Language Data Management System: Current Status and Plans," in J. Minker and S. Rosenfeld (eds.), *Proc. Symp. Information Storage and Retrieval*, University of Maryland, College Park, April 1971, 33-46.

15. Kellogg, C. A., *Question-Answering in the Converse System*, System Development Corporation, TM 5015, October 1971.

16. Nilsson, N. J., *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

17. Rulifson, J. F., R. J. Waldinger, and J. A. Derksen, "A Language for Writing Problem-Solving Programs," *Proc. IFIP Congr. 1971* (presented at Ljubljana, Yugoslavia, August 1971).

18. Rulifson, J. F., *QA4 Programming Concepts*, Stanford Research Institute, Artificial Intelligence Group, Technical Note 60, August 1971.

19. Woods, W. A., "Procedural Semantics for a Question-Answering Machine," *Proc. FJCC*, Spartan, New York, 1968, 457-471.

20. Hewitt, C., "A Language for Theorems in Robots," *Proc. Int. Joint Conf. Artificial Intelligence*, Washington, D.C., 1969, 295-301.

21. Sussman, G. J., and D. V. McDermott, "From PLANNER to CONNIVER--A Genetic Approach" ("Why Conniving is Better Than Planning"), *Proc. 1972 FJCC*, AFIPS, Vol. 41, Part II, 1171-1179.

22. Fikes, R. E. "Monitored Execution of Robot Plans Produced by STRIPS," *Proc. IFIP Congr. 1971* (presented at Ljubljana, Yugoslavia, August 1971). Also see R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence, 2*, 1971, 189-208.

23. Slagle, J. R., *Artificial Intelligence: The Heuristic Programming Approach*, McGraw-Hill, New York, 1971.

24. Garvin, P. L. (ed.), *Natural Language and the Computer*, McGraw-Hill, New York, 1963.

25. Sass, M. A., and W. D. Wilkinson (eds.), *Computer Augmentation of Human Reasoning*, Spartan, Washington, D.C., 1965.

26. Martins, G. R., "Dimensions of Text Processing," *Proc. 1972 FJCC*, AFIPS, Vol. 41, Part II, 801-810.

27. Knuth, D., *The Art of Computer Programming*: Vol. I *Fundamental Algorithms*, Chap. 2 "Information Structures," Addison-Wesley, Reading, Mass., 1968.

28. Shapiro, S. C., *The MIND System: A Data Structure for Semantic Information Processing*, The Rand Corporation, R-837-PR, August 1971.

29. Levien, R. E., and M. E. Maron, "A Computer System for Inference Execution and Data Retrieval," *Comm.ACM, 10*, 11, November 1967, 715-721.

30. Kochen, M., D. M. MacKay, M. E. Maron, M. Seriven, and L. Uhr, *Computers and Comprehension*, The Rand Corporation, RM-4065-PR, April 1964.

31. Kuhns, J. L., *Answering Questions by Computers: A Logical Study*, The Rand Corporation, RM-5428-PR, December 1967.

32. Di Paola, R., "The Solvability of the Decision Problem for Classes of Proper Formulas and Related Results," *J. ACM, 20*, January 1973, 112-126.

33. Woods, W. A., and R. M. Kaplan, *The Lunar Sciences Natural Language Information System*, BBN Report 2265, Cambridge, Mass., September 1971.

34. Woods, W. A., *An Experimental Parsing System for Transition Network Grammars*, BBN Report 2362, Cambridge, Mass., May 1972.

35. Woods, W. A., R. M. Kaplan, and B. Nash-Webber, *The Lunar Sciences Natural Language Information System: Final Report*, BBN Report 2378, Cambridge, Mass., June 1972.

36. Dostert, B. H., and F. B. Thompson, *The System of REL English*, California Institute of Technology, REL Report 1, September 1971.

37. Minsky, M., "Steps Toward Artificial Intelligence," *Proc. IRE 49*, January 1961, 8-30. (Reprinted in E. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963, 406-450.)

38. Charniak, E., "Jack and Janet in Search of a Theory of Knowledge," *Proc. Int. Joint Conf. Artificial Intelligence*, Stanford, Calif., 1973.

39. Feigenbaum, E. A., B. G. Buchanan, and J. Lederberg, "On Generality and Problem-Solving: A Case Study Using the DENDRAL Program," in B. Meltzer and D. Michie (eds.), *Machine Intelligence*, Vol. 6, American Elsevier, New York, 1971, 165-190.

40. Fu, K. S., and P. H. Swain, "On Syntactic Pattern Recognition," *Software Engineering*, Vol. 2, J. T. Tou (ed.), Academic Press, New York, 1971.

41. Vicens, "Aspects of Speech Recognition by Computer," Ph.D. Dissertation, Stanford University, April 1969. (Also available U.S. Dept. of Commerce Clearinghouse for Federal Scientific and Technical Information, AD687720.)

42. Reddy, D. R., L. D. Erman, and R. B. Neely, "A Model and a System for Machine Recognition of Speech," *IEEE Trans. Audio Electroacoustics* (to appear). (Also available as ARPA SUR Note: 44, NIC 11622, September 1972.)

43. Alter, R., "Utilization of Contextual Constraints in Automatic Speech Recognition," *IEEE Trans. Audio Electroacoustics*, AU-16, March 6-11, 1968.

44. Duda, R. O., and P. E. Hart, "Experiments in the Recognition of Hand-Printed Text: Part II-Context Analysis," *Proc. FJCC*, Spartan, New York, 1968, 1139-1149.

45. Teitelman, W., "Do What I Mean: The Programmer's Assistant," *Computers and Automation*, April 1972, 8-11.

46. -----, "Toward a Programming Laboratory," *Proc. Int. Joint Conf. Artificial Intelligence*, Washington, D.C., 1969, 8-11.

47. Burgess, A., *A Clockwork Orange*, W. W. Norton & Company Inc., New York, 1963.

48. Zobrist, A. L., "An Advice-Taking Chess Machine," seminar at UCLA, April 1973; also see A. L. Zobrist, and F. R. Carlson, Jr., An Advice-Taking Chess Computer," *Scientific American*, 228, June 1973, 92-105.

49. Carbonell, J. R., "AI in CAI: An Artificial Intelligence Approach to Computer-Assisted Instruction," *IEEE Trans. Man-Machine Systems MMS-11*, December 1970, 190-202.

50. Teitelman, W., D. G. Bobrow, A. K. Hartley, and D. L. Murphy, *BBN-LISP TENEX Reference Manual*, Bolt Beranek and Newman, Cambridge, Mass., 1972.

51. Jelinek, F., "Fast Sequential Decoding Algorithm Using a Stack," *IBM J. Res. Develop, 13*, November 1969, 675-685.

52. Logan, H., and L. Blochman, *Are You Misunderstood?* Wilfred Funk, Inc., New York, 1965.

53. Derksen, J. A., J. F. Rulifson, and R. J. Waldinger, "The QA4 Language Applied to Robot Planning," *Proc. 1972 FJCC*, AFIPS, Vol. 41, Part II, 1181-1192.

54. Feldman, J. A., J. R. Low, D. C. Swinehart, and R. H. Taylor, "Recent Developments in SAIL--An Algol-Based Language for Artificial Intelligence," *Proc. 1972 FJCC*, AFIPS, Vol. 41, Part II, 1193-1202.

55. Pavlidis, T., "Linguistic Analysis of Waveforms," *Software Engineering*, Vol. 2, J. Tou, (ed.), Academic Press, New York, 1971, 203-225.

56. Meisel, W. S., *Computer-Oriented Approaches to Pattern Recognition*, Academic Press, New York, 1972.

57. Duda, R. O., and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York, 1973.

58. Pavlidis, T., "Waveform Segmentation Through Functional Approximation," *IEEE Trans. Comp. C-22*, 1973, 689-697.

59. Pavlidis, T., and G. S. Fang, "A Segmentation Technique for Waveform Classification," *IEEE Trans. Comp. C-21*, 1972, 901-904.

60. Pierce, John, "Communication," *Scientific American*, 227, 1972.

61. Turn, R., A. S. Hoffman, T. Lippiatt, *Potential Military Applications of Speech Understanding Systems*, The Rand Corporation (to be published).

62. Newell, A., et al., *Speech-Understanding Systems: Final Report of a Study Group*, National Technical Information Service, Springfield, Virginia.

63. Klinger, A., *Applications and Goals of Speech Understanding Research*, The Rand Corporation, P-5132, October 1973 (to be published in the proceedings of the IFIP Congress, 1974).