Stanford Heuristic Programming Project
Memo HPP-79-21

Computer Science Department
Report No. STAN-CS-79-754

July 1979

Natural Language Understanding

with contributions by

Anne Gardner, James Davidson, and Terry Winograd

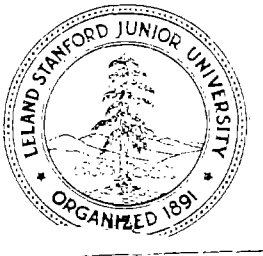
a section of the

Handbook of Artificial Intelligence

edited by

Avron Barr and Edward A. Feigenbaum

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: HPP-79-21 (STAN-CS-79-754)
2. GOVT ACCESSION NO.: HPP-79-21 (9)
3. RECIPIENT'S CATALOG NUMBER: Technical rept.

4. TITLE (and Subtitle): (6) Natural Language Understanding
5. TYPE OF REPORT & PERIOD COVERED: technical, July 1979
6. PERFORMING ORG. REPORT NUMBER: HPP-79-21 (STAN-CS-79-754)

7. AUTHOR(s): (10) Anne Gardner, James Davidson, and Terry Winograd (edited by Avron Barr & Edward A. Feigenbaum)
8. CONTRACT OR GRANT NUMBER(s): (15) MDA 903-77-C-0322, NPHS-RR-00785-06

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Department of Computer Science
Stanford University
Stanford, California 94305 USA

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (12) 102

11. CONTROLLING OFFICE NAME AND ADDRESS: (11)
Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Ave., Arlington, VA 22209

12. REPORT DATE: July 1979
13. NUMBER OF PAGES: 93

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
Mr. Philip Surra, Resident Representative
Office of Naval Research, Durand 165
Stanford University

15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(see reverse side)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 094 120

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Those of us involved in the creation of the Handbook of Artificial Intelligence, both writers and editors, have attempted to make the concepts, methods, tools, and main results of artificial intelligence research accessible to a broad scientific and engineering audience. Currently, AI work is familiar mainly to its practicing specialists and other interested computer scientists. Yet the field is of growing interdisciplinary interest and practical importance. With this book we are trying to build bridges that are easily crossed by engineers, scientists in other fields, and our own computer science colleagues.

In the Handbook we intend to cover the breadth and depth of AI, presenting general overviews of the scientific issues, as well as detailed discussions of particular techniques and important AI systems. Throughout we have tried to keep in mind the reader who is not a specialist in AI.

As the cost of computation continues to fall, new areas of computer applications become potentially viable. For many of these areas, there do not exist mathematical "cores" to structure calculational use of the computer. Such areas will inevitably be served by symbolic models and symbolic inference techniques. Yet those who understand symbolic computation have been speaking largely to themselves for twenty years. We feel that it is urgent for AI to "go public" in the manner intended by the Handbook.

Several other writers have recognized a need for more widespread knowledge of AI and have attempted to help fill the vacuum. Lay reviews, in particular Margaret Boden's Artificial Intelligence and Natural Man, have tried to explain what is important and interesting about AI, and how research in AI progresses through our programs. In addition, there are a few textbooks that attempt to present a more detailed view of selected areas of AI, for the serious student of computer science. But no textbook can hope to describe all of the sub-areas, to present brief explanations of the important ideas and techniques, and to review the forty or fifty most important AI systems.

The Handbook contains several different types of articles. Key AI ideas and techniques are described in core articles (e.g., basic concepts in heuristic search, semantic nets). Important individual AI programs (e.g., SHRDLU) are described in separate articles that indicate, among other things, the designer's goal, the techniques employed, and the reasons why the program is important. Overview articles discuss the problems and approaches in each major area. The overview articles should be particularly useful to those who seek a summary of the underlying issues that motivate AI research.

Eventually the Handbook will contain approximately two hundred articles. We hope that the appearance of this material will stimulate interaction and cooperation with other AI research sites. We look forward to being advised of errors of omission and commission. For a field as fast moving as AI, it is important that its practitioners alert us to important developments, so that future editions will reflect this new material. We intend that the Handbook of Artificial Intelligence be a living and changing reference work.

The articles in this edition of the Handbook were written primarily by graduate students in AI at Stanford University, with assistance from graduate students and AI professionals at other institutions. We wish particularly to acknowledge the help from those at Rutgers University, SRI International, Xerox Palo Alto Research Center, MIT, and the RAND Corporation.

This report, which contains the section of the Handbook on natural language understanding research, has been drafted by numerous Stanford graduate students. Major contributions to revising and editing it have been made by Anne Gardner, James Davidson, and Terry Winograd. Others who contributed to or commented on earlier versions of this section include Jan Aikins, Daniel Bobrow, Rod Brooks, William Clancey, Paul Cohen, Gerard Dechen, Richard Gabriel, Neil Goldman, Norm Haas, Douglas Hofstadter, Andrew Silverman, Phil Smith, Reid Smith, William Van Melle, and David Wilkins.

Natural Language Understanding

with contributions by

Anne Gardner, James Davidson, and Terry Winograd

a section of the

Handbook of Artificial Intelligence

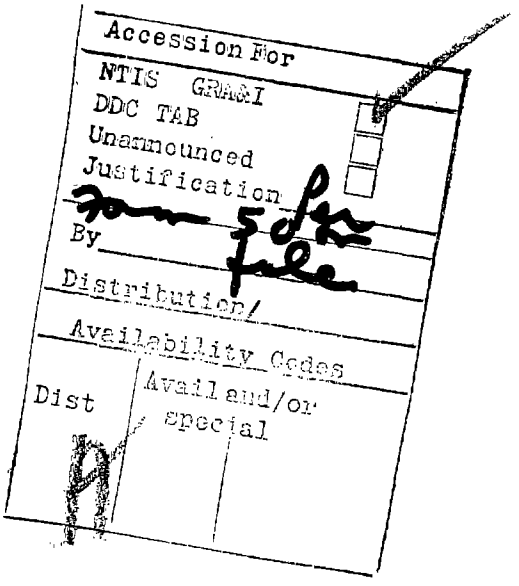
edited by

Avron Barr and Edward A. Feigenbaum

This research was supported by both the Defense Advanced Research Projects Agency (ARPA Order No. 3423, Contract No. MDA 903-77-C-0322) and the National Institutes of Health (Contract No. NIH RR-00785-06). The views and conclusions of this document should not be interpreted as necessarily representing the official policies, either express or implied, of the Defense Advanced Research Projects Agency, the National Institutes of Health, or the United States Government.

Natural Language

Table of Contents:



Foreword

Those of us involved in the creation of the **Handbook of Artificial Intelligence**, both writers and editors, have attempted to make the concepts, methods, tools, and main results of artificial intelligence research accessible to a broad scientific and engineering audience. Currently, AI work is familiar mainly to its practicing specialists and other interested computer scientists. Yet the field is of growing interdisciplinary interest and practical importance. With this book we are trying to build bridges that are easily crossed by engineers, scientists in other fields, and our own computer science colleagues.

In the Handbook we intend to cover the breadth and depth of AI, presenting general overviews of the scientific issues, as well as detailed discussions of particular techniques and important AI systems. Throughout we have tried to keep in mind the reader who is not a specialist in AI.

As the cost of computation continues to fall, new areas of computer applications become potentially viable. For many of these areas, there do not exist mathematical "cores" to structure calculational use of the computer. Such areas will inevitably be served by symbolic models and symbolic inference techniques. Yet those who understand symbolic computation have been speaking largely to themselves for twenty years. We feel that it is urgent for AI to "go public" in the manner intended by the Handbook.

Several other writers have recognized a need for more widespread knowledge of AI and have attempted to help fill the vacuum. Lay reviews, in particular Margaret Boden's **Artificial Intelligence and Natural Man**, have tried to explain what is important and interesting about AI, and how research in AI progresses through our programs. In addition, there are a few textbooks that attempt to present a more detailed view of selected areas of AI, for the serious student of computer science. But no textbook can hope to describe all of the sub-areas, to present brief explanations of the important ideas and techniques, and to review the forty or fifty most important AI systems.

The Handbook contains several different types of articles. Key AI ideas and techniques are described in core articles (e.g., basic concepts in heuristic search, semantic nets). Important individual AI programs (e.g., SHRDLU) are described in separate articles that indicate, among other things, the designer's goal, the techniques employed, and the reasons why the program is important. Overview articles discuss the problems and approaches in each major area. The overview articles should be particularly useful to those who seek a summary of the underlying issues that motivate AI research.

Eventually the Handbook will contain approximately two hundred articles. We hope that the appearance of this material will stimulate interaction and cooperation with other AI research sites. We look forward to being advised of errors of omission and commission. For a field as fast moving as AI, it is important that its practitioners alert us to important developments, so that future editions will reflect this new material. We intend that the **Handbook of Artificial Intelligence** be a living and changing reference work.

The articles in this edition of the Handbook were written primarily by graduate students in AI at Stanford University, with assistance from graduate students and AI professionals at other institutions. We wish particularly to acknowledge the help from those at Rutgers University, SRI International, Xerox Palo Alto Research Center, MIT, and the RAND Corporation.

This report, which contains the section of the Handbook on natural language understanding research, has been drafted by numerous Stanford graduate students. Major contributions to revising and editing it have been made by Anne Gardner, James Davidson, and Terry Winograd. Others who contributed to or commented on earlier versions of this section include Jan Aikins, Daniel Bobrow, Rod Brooks, William Clancey, Paul Cohen, Gerard Dechen, Richard Gabriel, Neil Goldman, Norm Haas, Douglas Hofstadter, Andrew Silverman, Phil Smith, Reid Smith, William Van Melle, and David Wilkins.

Avron Barr
Edward Feigenbaum

Stanford University
July, 1979

Handbook of Artificial Intelligence

Topic Outline

Volumes I and II

Introduction

The Handbook of Artificial Intelligence
Overview of AI Research
History of AI
An Introduction to the AI Literature

Search

Overview
Problem Representation
Search Methods for State Spaces, AND/OR Graphs, and Game Trees
Six Important Search Programs

Representation of Knowledge

Issues and Problems in Representation Theory
Survey of Representation Techniques
Seven Important Representation Schemes

AI Programming Languages

Historical Overview of AI Programming Languages
Comparison of Data Structures and Control Mechanisms in AI Languages
LISP

Natural Language Understanding

Overview - History and Issues
Machine Translation
Grammars
Parsing Techniques
Text Generation Systems
The Early NL Systems
Six Important Natural Language Processing Systems

Speech Understanding Systems

Overview - History and Design Issues
Seven Major Speech Understanding Projects

Applications-oriented AI Research -- Part 1

Overview
TEIRESIAS - Issues in Expert Systems Design
Research on AI Applications in Mathematics (MACSYMA and AM)
Miscellaneous Applications Research

Applications-oriented AI Research -- Part 2: Medicine

Overview of Medical Applications Research
Six Important Medical Systems

Applications-oriented AI Research -- Part 3: Chemistry

Overview of Applications in Chemistry
Applications in Chemical Analysis
The DENDRAL Programs
CRYSALIS
Applications in Organic Synthesis

Applications-oriented AI Research -- Part 4: Education

Historical Overview of AI Research in Educational Applications
Issues in ICAI Systems Design
Seven Important ICAI Systems

Automatic Programming

Overview
Techniques for Program Specification
Approaches to AP
Eight Important AP Systems

The following sections of the Handbook are still in preparation and will appear in the third volume:

Theorem Proving
Vision
Robotics
Information Processing Psychology
Learning and Inductive Inference
Planning and Related Problem-solving Techniques

A. Natural Language Processing Overview

The most common way that human beings communicate is by speaking or writing in one of the "natural" languages, like English, French, or Chinese. Computer programming languages, on the other hand, seem awkward to humans. These "artificial" languages are designed to have a rigid format, or *syntax*, so that a computer program reading and compiling code written in an artificial language can understand what the programmer means. In addition to being structurally simpler than natural languages, the artificial languages can express easily only those concepts that are important in programming: "Do this then do that," "See if such and such is true," etc. The things that can be expressed in a language are referred to as the *semantics* of the language.

The research on understanding natural language described in this section of the Handbook is concerned with programs that deal with the full range of meaning of languages like English. Computers that can understand what people mean when typing (or speaking) English sentences will be easier to use and will fit more naturally into people's lives. In addition, artificial intelligence (AI) research in natural language processing aims to extend our knowledge of the nature of language as a human activity. Programs have been written that are quite successful at understanding somewhat constrained input: the user is limited in either the structural variation of his sentences (syntax constrained by an artificial *grammar*) or in the number of things he can "mean" (in domains with constrained semantics). Some of these programs are adequate for many useful computer-interface tasks and are available commercially. But the fluent use of language as humans use it is still elusive, and natural language (NL) processing is an active area of research in AI.

This article presents a brief sketch of the history of natural language processing research in AI, and it attempts to give some idea of the current state of the art in NL and related research in representing knowledge about the world within the language understanding programs. The next article is a historical sketch of the very earliest ideas about processing language with computers, to achieve *mechanical translation* of one language into another. It is followed by two sections containing technical articles on some of the grammars and *parsing* techniques that AI researchers have used in their programs. Then, after an article on *text generation*, which involves the creation of sentences by the program to express what it wants to say, there are a half dozen articles describing some of the most important NL systems.

Two other sections of the Handbook are especially relevant to NL research. Speech Understanding research attempts to build computer interfaces that actually understand spoken language. Speech and natural language understanding research have been closely linked. Increasingly inseparable from NL research is the study of Knowledge Representation, because AI researchers have come to believe that a very large amount of knowledge about the world is used in even simple dialogue. Research in the representation of knowledge explores ways of making this *world knowledge* accessible to the computer program by "representing" it in internal data structures.

History

Research in *computational linguistics*, the use of computers in the study of language, started in the 1940s, soon after computers became available commercially. The machine's

ability to manipulate symbols was first used to compile lists of word occurrences (word lists) and concordances (their contexts in written texts). Such surface-level machine processing of text was of some value in linguistic research, but it soon became apparent that the computer could perform much more powerful linguistic functions than merely counting and rearranging data.

In 1949, Warren Weaver proposed that computers might be useful for "the solution of the world-wide translation problem" (Weaver, 1949, p. 15). The resulting research effort, called *mechanical translation*, attempted to simulate with a computer the presumed functions of a human translator: looking up each word in a bilingual dictionary; choosing an equivalent word in the output language; and, after processing each sentence, arranging the resulting string of words to fit the output language's word order. Despite the attractive simplicity of the idea, many unforeseen problems arose, both in selecting appropriate word equivalences and in arranging them to produce a sentence in the output language. Article B discusses the history, problems, and current state of research on mechanical translation.

In the 1960s a new group of computer programs was developed that attempted to deal with some of the more complex issues of language that had led to the difficulties in the mechanical translation efforts. These early natural language programs mark the beginning of artificial intelligence work in understanding language. They no longer assume that human communication is a process of word manipulation. Instead, they view human language as a complex cognitive ability involving many different kinds of knowledge: the structure of sentences, the meaning of words, a model of the listener, the rules of conversation, and an extensive shared body of general information about the world. Several of these programs are described briefly in Article F1.

The focus of modern work in natural language processing in AI is "understanding" language. Several different tasks have been used as the criterion for defining what constitutes a demonstration that the program understands a piece of text; these tasks include *paraphrasing*, *question answering*, *mechanical translation*, and *information retrieval*. Many design issues depend on which type of task the program is to perform, but the general approach has been to model human language as a knowledge-based system for processing communications and to create a computer program that serves as a working model of this system.

AI researchers in natural language processing expect their work to lead both to the development of practical, useful language understanding systems and to a better understanding of language and the nature of intelligence. The computer, like the human mind, has the ability to manipulate symbols in complex processes, including processes that involve decision making based on stored knowledge. It is an assumption of the field that the human use of language is a cognitive process of this sort. By developing and testing computer-based models of language processing that approximate human performance, researchers hope to understand better how human language works.

Approaches to NL Processing

Natural language research projects have had diverse goals and used diverse methods, making their categorization somewhat difficult. One coherent scheme, borrowed from Winograd (1972), groups natural language programs according to how they represent and

use knowledge of their subject matter. On this basis, natural language programs can be divided into four historical categories.

The earliest natural language programs sought to achieve only limited results in specific, constrained domains. These programs used ad hoc data structures to represent "knowledge." Programs like BASEBALL, SAD-SAM, STUDENT, and ELIZA (see Article F1) searched their input sentences, which were restricted to simple declarative and interrogative forms, for key words or patterns representing known objects and relationships. Domain-specific rules, called *heuristics*, were used to derive the required information from the key words in the sentence and the knowledge in the database. Though they performed relatively small tasks and avoided or ignored many of the complexities in language, their results and methods were the impetus to dealing with more difficult problems.

The second category can be called *text-based systems*. These programs, such as PROTOSYNTHEX I (Simmons, Klein, & McConlogue, 1964) and the Teachable Language Comprehender, TLC (Quillian, 1969), attempted to expand beyond the limits of a specific domain. The programs dealt with full English text as a base, rather than with key words or phrases. Input text was interpreted as a request to access a structured information store, and a variety of clever methods were used to identify the proper response. Though more general than their predecessors, these programs still failed to deal with the underlying *meaning* of the English language input. They were able to give only responses that had been pre-stored as data--they had no deductive power.

To try to deal with the problem of how to characterize and use the meaning of sentences, a group of programs was developed called *limited logic systems*. In systems like SIR (Raphael, 1968), DEACON (Thompson, 1966), and CONVERSE (Kellogg, 1968), the information in the database is stored in a formal, albeit ad hoc, notation, and mechanisms are provided for translating input sentences into the same form. The function of the formal notation is to attempt to liberate the informational content of the input from the structure of English. The overall goal of these systems was to accept complex input information (e.g., information containing quantifiers and relationships), use it to perform inferences on the database, and thus realize answers to complex questions. Problems, however, arose from the fact that the complexity of the stored information was not really part of the database but was built into the system's routines for manipulating the database. PROTOSYNTHEX II (Simmons, 1966; Simmons, Burger, & Long, 1966, for example, contained statements of the form "A is X" and "X is B" and tried to answer "Is A B?", based on transitivity. The deductive mechanism required for these inferences was embedded in special-purpose subroutines, rather than in the database as a "theorem," and thus was not available to be used to perform more involved inferences, which require a longer chain of reasoning.

Representing Knowledge in NL Programs

The fourth approach to building language understanding programs might be called *knowledge-based systems* and is closely intertwined with current research on the representation of knowledge (see the Knowledge Representation section of the Handbook). Among the most important knowledge representation schemes explored in NL research have been: procedural semantics, semantic networks, case systems, and frame systems.

In the early 1970s, two systems were built that attempted to deal with both syntactic

and semantic problems in a comprehensive way. William Woods's LUNAR system (Article F3) answered questions about the samples of rock brought back from the moon, using a large database provided by the National Aeronautics and Space Agency. It was one of the first programs to attack the problems of English grammar using an *augmented transition network* parser (Article D2). It used a notion of *procedural semantics* in which queries were first converted in a systematic way into a "program" to be executed by the retrieval component. Terry Winograd's SHRDLU system (Article F4) carried on a dialogue with a user in which the system simulated a robot manipulating a set of simple objects on a table top. The naturalness of the dialogue, as well as SHRDLU's apparent reasoning ability, made it particularly influential in the development of AI ideas. These two systems integrate syntactic and semantic analysis with a body of world knowledge about a limited domain, enabling them to deal with more sophisticated aspects of language and discourse than had previously been possible.

Central to these two systems is the representation of knowlege about the world as procedures within the system. The meanings of words and sentences were expressed as programs in a computer language, and the execution of these programs corresponded to reasoning from the meanings. Direct procedural representations are often the most straightforward way to implement the specific reasoning steps needed for a natural language system. Most of the actual working systems that have been developed have made heavy use of specialized procedural representations, to fill in those places where the more *declarative* representation schemes--those where the "knowledge" is encoded in passive data structures that are interpreted by other procedures--are insufficient. (The *procedural/declarative controversy* has been an important focus in the history of AI. See Article Representation.B.)

Perhaps the most influential declarative representation scheme is the *semantic network*. Semantic networks were first proposed by Quillian (1968) as a model for human associative memory. They used the concepts of graph theory, representing words and meanings as a set of linked nodes. By using a systematic set of link types, it was possible to program simple operations (such as following chains of links) that corresponded to drawing inferences. Another important declarative scheme is the use of standard *logic* formulas (Article Representation.C1), which are subject to mathematical rules of deduction for drawing inferences. The advantage of semantic networks over standard logic is that some selected set of the possible inferences can readily be done in a specialized and efficient way. If these correspond to the inferences that people make easily, then the system will be able to do a more natural sort of reasoning than can be easily achieved using formal logical deduction.

Semantic networks have been the basis for a number of systems, including most of the *speech understanding* systems (see Speech Understanding). Recently there has been a good deal of work on formalizing the network notions so that there is a clear correspondence between the graph operations and the formal semantics of the statements represented (see Article Representation.C2).

Case representations extend the basic notions of semantic nets with the idea of a *case frame*, a cluster of the properties of an object or event into a single concept (see Article C4). There have been a large number of variations on this notion, some of which remain close to the linguistic forms. Others such as *conceptual dependency* are based on the notion of *semantic primitives*, the construction of all semantic notions from a small set of "primitive"

concepts. The MARGIE sytem (Article F5), built in the early 1970s by Roger Schank and his students, uses the conceptual dependency representation.

As with semantic networks, the advantage of case representations lies in their focus on clustering relevant sets of relationships into single data structures. The idea of clustering structures in a coherent and efficient way has been carried much further in representation schemes based on the notion of a *frame* (Minsky, 1975; see also Article Representation.C6). Where case representations deal primarily with single sentences or acts, frames are applied to whole situations or complex objects or series of events. In analyzing a sentence, narrative, or dialogue, a language understanding system based on frame representations tries to match the input to prototypes for the objects and events in its domain that are stored in its database.

For example, Roger Schank's SAM system (Article F6) makes use of simple, linear *scripts*, which represent stereotyped sequences of events, to understand simple stories. It assumes that the events being described will fit (roughly) into one of the scripts in its knowledge base, which it then uses to fill in missing pieces in the story. The GUS system (Bobrow et al., 1977) is a prototype travel consultant, carrying on a dialogue to help a person schedule an air trip. It uses frames representing standard trip plans. GUS uses the experimental frame language KRL (Bobrow & Winograd, 1977; see also Article Representation.C6).

The important common element in all of these systems is that the existence of prototype frames makes it possible to use *expectations* in analysis. When a sentence or phrase is input that is ambiguous or underspecified, it can be compared to a description of what would be expected based on the prototype. Assumptions can be made about what was meant, if there is a plausible fit to the expectation. This *expectation-driven* processing seems to be an important aspect of the human use of language, where incomplete or ungrammatical sentences can be understood in appropriate contexts. Research on script- and frame-based systems is the most active area of AI research in natural language understanding at the present time.

The current state-of-the-art in working (non-experimental) NL systems is exemplified by ROBOT (Harris, 1977), LIFER (Hendrix, 1977b), and PHLIQA1 (Landsbergen, 1976).

References

General discussions of natural language processing research in AI can be found in Boden (1977), Wilks (1974), Winograd (1974), Charniak & Wilks (1976), Schank & Abelson (1977), and Winograd (forthcoming). Waltz (1977) contains more than fifty brief summaries of current projects and systems. In addition, many historically important NL systems are described in Feigenbaum & Feldman (1963), Minsky (1968), Rustin (1973), Schank & Colby (1973), and Winograd (1972). COLING (1976), TINLAP-1 (1975), Bobrow & Collins (1975), and TINLAP-2 (1978) are proceedings of recent conferences describing current work in the field.

B. Mechanical Translation

The concept of translation from one language to another by machine is older than the computer itself. According to Yehoshua Bar-Hillel, one of the early investigators in the field, the idea was perhaps first conceived as early as the early 1930s by P. P. Smirnov-Troyansky of the Soviet Union and G. B. Artsouni of France (see Bar-Hillel, 1960, p. 7). Their work apparently never received much attention, lying dormant until a decade later when the climate was much more favorable, due to the recent invention of the digital computer. In certain quarters of the scientific world people imagined--with some justification--that computers would lead to many entirely new and far-reaching ideas about man and--perhaps less justifiably--that computers would help bring about a new world order. In short, there was tremendous excitement over the potential of these new *thinking machines*, as they were quickly dubbed. This was also the time when Claude Shannon was formulating his ideas on information theory, when Norbert Wiener was devising the concept of *cybernetics*, and when Pitts and McCullough were developing their ideas on neural nets and brain function. Moreover, computing had just passed its initial tests, during the war, with flying colors--in such strategic tasks as breaking codes and calculating complicated nuclear cross sections.

It would be well to bear in mind that, when machine translation work began, programming was done by wiring boards and machine language was the only computer language available. Such concepts as arrays and subroutines were still to appear, not to mention pushdown stacks, compiler languages, recursive procedures, and the like. Furthermore, no one had heard of *context-free* and *context-sensitive grammars*, or of *transformational grammars*, or *augmented transition networks*. At the forefront of computational linguistics, the application of the computer to the study of language, were statistical experiments with language, such as compiling matrices of letter frequencies and of transition frequencies between successive letters. Such matrices could be used to produce interesting samples of *pseudo-language*, by producing words from randomly generated letters with the same characteristics as English words. (Also, see the discussion of Yngve's *random text generation* system in Article E).

First Attempts

The real genesis of machine translation dates from a series of discussions between Warren Weaver and A. Donald Booth in 1946. Both men were familiar with the work on code breaking by computers, based on letter-frequency and word-frequency tables. It seemed to them that some of the same methods would be applicable to translation and that the principal obstacle would be incorporating a full dictionary of the two languages. Of course they recognized that simply having a dictionary would not solve all problems. Some of the remaining problems would be the following: (a) Many words have several translations, depending upon context; (b) word orders differ from language to language; and (c) idiomatic expressions cannot be translated word for word but must be translated *in toto*. Nevertheless, it appeared plausible, at the time, that the major problem in translating between two languages was simply that of vocabulary--and so at least a large part of translation seemed mechanizable.

In 1947, Booth and D. H. V. Britten worked out a program for dictionary lookup. This was a full-form dictionary, in that each variant of any basic word (e.g., love, loves, loving, etc.) had to be carried as a separate entry in the dictionary. In 1948, R. H. Richens suggested the addition of rules concerning the inflections of words, so that the redundancy

of the multiple dictionary entries could be eliminated. In 1949, Warren Weaver distributed a memorandum entitled *Translation* to about two hundred of his acquaintances, and a considerable wave of interest ensued. In addition to the idea that all languages have many features in common, three other items from that memorandum are worth repeating. The first is the notion of a *window* through which one can view exactly 2N + 1 words of text; Weaver suggests that when N is sufficiently large, one will be able to determine the unique, correct translation for the word that sits in the middle of the window. He then points out that N may be a function of the word, rather than a constant, and discusses the idea of choosing a value of N such that, say, 95% of all words would be correctly translated 98% of the time. The second is this intriguing statement: "When I look at an article in Russian, I say, *This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.*" This certainly carries to an extreme the concept that source text and translated text "say the same thing." In fact, it leads naturally to the third provocative idea of the memorandum that translating between languages A and B means going from A to an intermediate "universal language," or *interlingua*, that, supposedly, all humans share, and thence to B. This idea, of an intermediate representation of the *semantics* or meaning of an utterance, appears often in modern natural language processing work in AI under the heading *representation of knowledge* (see discussion in the Overview and in the Handbook Section on Knowledge Representation).

After Weaver's memorandum, work sprang up in several centers in the United States. Erwin Reifler conceived the idea of two auxiliary functions to be performed by human beings, those of *pre-editor* and *post-editor*. The pre-editor would prepare the input text to be as free as possible of ambiguities and other sources of difficulty; the post-editor would take the machine-translated text and turn it into grammatical, comprehensible prose.

A 1952 conference produced recommendations to implement a dictionary-lookup program and to work towards the invention, or discovery, of the hypothetical universal language, called *Machinese*, which Weaver had proposed as an intermediate language in mechanical translation.

A. G. Oettinger was one of the first to design a program that carried out word-for-word translation of Russian text into English. A very high percentage of the Russian words had more than one possible translation; so all of them were listed in the output English, enclosed in parentheses. Thus, a sample of English output text read as follows:

> (In, At, Into, To, For, On) (last, latter, new, latest, lowest, worst) (time, tense) for analysis and synthesis relay-contact electrical (circuit, diagram, scheme) parallel-(series, successive, consecutive, consistent) (connection, junction, combination) (with, from) (success, luck) (to be utilize, to be take advantage of) apparatus Boolean algebra. (Oettinger, 1955, p. 55)

A cleaned-up version of this sentence reads: "In recent times Boolean algebra has been successfully employed in the analysis of relay networks of the series-parallel type" (p. 58). Readers of the translated text were expected to discern from the jumble of synonyms what the cleaned-up text really should be. Clearly, there was still a long, long way to go toward mechanical translation.

In the next year or two, most of the effort was directed toward devising ways to handle different endings of inflected words and estimating the size of vocabulary needed for

translations of varying degrees of quality. In 1954 a journal of mechanical translation was founded, called *MT*. Machine translation received considerable public attention when a group from IBM and Georgetown University made grand claims for a program that translated from Russian to English, although this program was not particularly advanced over any others. In any case, machine translation became an "in" thing and groups sprang up in many countries.

Problems Encountered

Early attempts focusing on syntactic information were able to produce only low-quality translation and led eventually to extreme pessimism about the possibility of the endeavor. It has since become clear that high-quality translation systems must in some sense *understand* the input text before they can reconstruct it in a second language. For the first time, it was becoming apparent that much "world knowledge" is used implicitly when human beings translate from one language to another. Bar-Hillel gave as an example the pair of sentences, "The pen is in the box," and "The box is in the pen." Of this example he said, "I now claim that no existing or imaginable program will enable an electronic computer to determine that the word pen" in the second sentence has the meaning "an enclosure where small children can play" (Bar-Hillel, 1960, p. 159). He goes on to remark that, to his amazement, no one had ever pointed out that in language understanding there is a world-modeling process going on in the mind of the listener and that people are constantly making use of this subconscious process to guide their understanding of what is being said. Bar-Hillel continues:

> A translation machine should not only be supplied with a dictionary but also with a universal encyclopedia. This is surely utterly chimerical and hardly deserves any further discussion. . . . We know . . . facts by inferences which we are able to perform . . . instantaneously, and it is clear that they are not, in any serious sense, stored in our memory. Though one could envisage that a machine would be capable of performing the same inferences, there exists so far no serious proposal for a scheme that would make a machine perform such inferences in the same or similar circumstances under which an intelligent human being would perform them. (pp. 160-161)

Bar-Hillel despaired of ever achieving satisfactory machine translation. His sentiments were not universally shared, but in 1966 they came to prevail officially in the so-called ALPAC report (NRC, 1966). This report, made to the National Research Council after a year of study by its Automatic Language Processing Advisory Committee, resulted in the discontinuance of funding for most machine transation projects. The report stated:

> "Machine Translation" presumably means going by algorithm from machine-readable source text to useful target text, without recourse to human translation or editing. In this context, there has been no machine translation of general scientific text, and none is in immediate prospect. (p. 19)

Examples of the output of several MT systems were included in the report; they showed little improvement from the results Oettinger had obtained ten years before. Even with

postediting the output was found to be generally of poorer quality, and sometimes slower and more expensive to obtain, than entirely human translation.

Current Status

The conclusions of the ALPAC report were directed only against funding for MT as a practical tool. Support for computational linguistics, evaluated in terms of its scientific worth rather than its immediate utility, was to be continued. It was also recognized that there had been fundamental changes in the study of linguistics, partly due to cross-fertilization with computational activities.

Both linguistics and computer science have made contributions relevant to the revival of MT research. A signal event was the publication in 1957 of Noam Chomsky's **Syntactic Structures**, in which *transformational grammars* were introduced. This book spurred many new developments in the analysis of syntax. Concurrently, new computer languages and new types of data structures were being explored by computer scientists, leading to the creation (in 1960) of both ALGOL and LISP, with their features of lists, recursion, etc. These languages were the first in a series of languages geared more toward symbol manipulation than "number crunching," as discussed in the AI Programming Languages Section of the Handbook. In artificial intelligence, the 1960s saw considerable progress toward natural language understanding, such as the development of programs that carried on a dialogue of sorts with the user: BASEBALL, SAD-SAM, STUDENT, SIR, etc., which are described in Article F1.

The early 1970s have seen some revival of interest in machine translation, partly because some progress has been made in the internal *representation of knowledge*. The programs of Wilks (Article F2) and Schank (Articles F5 and F6) can both perform translation tasks. They begin by translating input sentences into internal data structures based on *semantic primitives*, which are intended to be "language independent"--elements of meaning that are common to all natural languages. The internal representation can be manipulated relatively easily by procedures that carry out *inferences*; it forms in effect an internal language or *interlingua* for modeling the world. The data structure(s) derived from an input sentence could be considered to be a translation of that sentence into Weaver's *Machinese*. The reverse derivation (i.e., Machinese to, say, French) then is a realization, on some level, of Weaver's idea (see Article E for research on the *generation* of text.)

It is difficult to evaluate the practicality of machine translation. In some applications it is worthwhile to have even a very bad translation, if it can be done by a computer in a much shorter time (or much more cheaply) than by humans. In others (such as the preparation of instruction manuals) it is possible to deal with input texts that use a specially restricted form of the language, thereby making translation easier. There is also the possibility of machine-human interactive translating, in which the output of the computer is used not by the ultimate reader but by someone engaged in producing the final translation. The computer can be used to do sub-tasks like dictionary lookup, or can produce more-or-less complete translations that are then checked and polished by a human post-editor, who perhaps does not know the original language.

At the current time, computers are being used in these ways in a number of translation systems. There is also a renewed interest in fully automatic translation, based on some of

the the techniques for dealing with meaning described below. However, it is not clear whether we are yet ready to reattack the goal of "fully automatic high quality translation." Much current work on language is based on a belief that deep understanding of what is being said is vital to every language use. Applied to translation, this means that we must first have a program that understands a subject before we can translate material about that subject. Since our ability to model large areas of knowledge is still primitive, this places a strong limit on the scope of material we might handle.

References

A brief, popular review of current work in mechanical translation can be found in Wilks (1977a). For the earliest history, see the introduction to Locke & Booth (1955). Later surveys include Bar-Hillel (1960), Josselson (1971), and Hays & Mathias (1976).

See also Bar-Hillel (1964), Booth (1967), NRC (1966), Oettinger (1955), Schank (1975), Weaver (1949), and Wilks (1973).

C. Grammars

A *grammar* of a language is a scheme for specifying the sentences allowed in the language, indicating the rules for combining words into phrases and clauses. In natural language processing programs, the grammar is used in *parsing* to "pick apart" the sentences in the input to the program to help determine their meaning and thus an appropriate response. Several very different types of grammars have been used in NL programs and are described in the articles which follow.

C1. Formal Grammars

One of the more important contributions to the study of language was the theory of *formal languages* introduced by Noam Chomsky in the 1950s. The theory has developed as a mathematical area, not a linguistic one, and has strongly influenced computer science in the design of computer programming languages (artificial languages). Nevertheless, it is useful in connection with *natural language* understanding systems, both as a theoretical and a practical tool.

Definitions

A *formal language* is defined as a (possibly infinite) set of strings of finite length formed from a finite vocabulary of symbols. (For example, the strings might be sentences composed from a vocabulary of words.) The *grammar* of a formal language is specified in terms of the following concepts:

1. The **syntactic categories**, such as <SENTENCE> and <NOUN PHRASE>. These syntactic categories are referred to as *nonterminal symbols* or *variables*. Notationally, the nonterminals of a grammar are often indicated by enclosing the category names in angle brackets, as above.

2. The **terminal symbols** of the language, for example the words in English. The terminal symbols are to be concatenated into strings called *sentences* (if the terminals are words). A language is then just a subset of the set of all the strings that can be formed by combining the terminal symbols in all possible ways. Exactly which subset is permitted in the language is specified by the rewrite rules:

3. The *rewrite rules* or *productions* specify the relationships that exist between certain strings of terminals and nonterminal symbols. Some examples of productions are:

```
   <SENTENCE> -> <NOUN PHRASE> <VERB PHRASE>
<NOUN PHRASE> -> the <NOUN>
      <NOUN> -> dog
      <NOUN> -> cat
<VERB PHRASE> -> runs
```

The first production says that the (non-terminal) symbol <SENTENCE> may be "rewritten" as the symbol <NOUN PHRASE> followed by the symbol <VERB PHRASE>. The second permits <NOUN PHRASE> to be replaced by a string composed of the word the, which is a terminal symbol, followed by the nonterminal <NOUN>. The next two allow <NOUN> to be replaced by

either dog or cat. Since there are sequences of productions permitting <NOUN PHRASE> to be replaced by the dog or the cat, the symbol <NOUN PHRASE> is said to *generate* these two terminal strings. Finally, <VERB PHRASE> can be replaced by the terminal runs.

4. The **start symbol**. One nonterminal is distinguished and called the "sentence" or "start" symbol, typically denoted <SENTENCE> or S. The set of strings of terminals that can be derived from this distinguished symbol, by applying sequences of productions, is called the *language generated by the grammar*. In our simple example grammar, exactly two sentences are generated:

The cat runs.
The dog runs.

The important aspect of defining languages formally, from the point of view of computational linguistics and natural language processing, is that if the structure of the sentences is well understood, then a parsing algorithm for analyzing the input sentences will be relatively easy to write (see Section D1 on parsing).

The Four Types of Formal Grammars

Within the framework outlined above, Chomsky delineated four types of grammars, numbered 0 through 3. The most general class of grammar is type 0, which has no restrictions on the form that rewrite rules can take. For successive grammar types, the form of the rewriting rules allowed is increasingly restricted, and the languages that are generated are correspondingly simpler. The simplest formal languages (types 2 and 3) are, as it turns out, inadequate for describing the complexities of human languages. (See Article C2 for a fuller discussion.) On the other hand, the most general formal languages are difficult to handle computationally. There is an intimate and interesting connection between the theory of formal languages and the theory of computational complexity (see Hopcroft & Ullman, 1969). The following discussion gives a formal account of the different restrictions applied in each of the four grammar types.

Formally, a grammar G is defined by a quadruple (VN, VT, P, S) representing the nonterminals, terminals, productions, and the start symbol, respectively. The symbol V, for "vocabulary," is used to represent the union of the sets VN and VT, which are assumed to have no elements in common. Each production in P is of the form

X -> Y

where X and Y are strings of elements in V, and X is not the empty string.

Type 0. A type-0 grammar is defined as above: a set of productions over a given vocabulary of symbols with no restrictions on the form of the productions. It has been shown that a language can be generated by a type-0 grammar if and only if it can be recognized by a Turing machine; that is, we can build a Turing machine which will halt in an ACCEPT state for exactly those input sentences that can be generated by the language.

Type 1. A type-0 grammar is also of type 1 if the form of the rewrite rules is restricted so that, for each production X -> Y of the grammar, the right-hand side Y contains

at least as many symbols as the left-hand side X. Type-1 grammars are also called *context-sensitive* grammars. An example of a context-sensitive grammar with start symbol S and terminals a, b, and c is the following:

```
 S -> aSBC
 S -> aBC
CB -> BC
aB -> ab
bB -> bb
bC -> bc
cC -> cc
```

The language generated by this grammar is the set of strings abc, aabbcc, aaabbbccc . . . . This language, where each symbol must occur the same number of times and must appear in the right position in the string, cannot be generated by any grammar of a more restricted type (i.e., type 2 or type 3).

An alternate (equivalent) definition for context-sensitive grammars is that the productions must be of the form

uXv -> uYv

where X is a single nonterminal symbol; u and v are arbitrary strings, possibly empty, of elements of V; and Y is a nonempty string over V. It can be shown that this restriction generates the same languages as the first restriction, but this latter definition clarifies the term *context-sensitive*: X may be rewritten as Y only in the context of u and v.

Type 2. *Context-free grammars* or type-2 grammars are grammars in which each production must have only a single non-terminal symbol on its left-hand side. For example, a context-free grammar generating the sentences ab, aabb, aaabbb . . . is:

```
S -> aSb
S -> ab
```

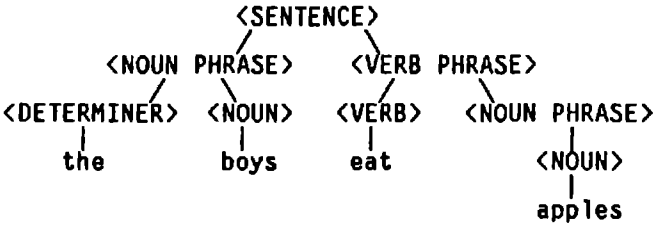
Again, it is not possible to write a context-free grammar for the language composed of the sentences abc, aabbcc, aaabbbccc . . . --having the same number of c's at the end makes the language more complex. The simpler language here, in turn, cannot be generated by a more restricted (type-3) grammar.

An example of a context-free grammar that might be used to generate some sentences in natural language is the following:

```
   <SENTENCE> -> <NOUN PHRASE> <VERB PHRASE>
<NOUN PHRASE> -> <DETERMINER> <NOUN>
<NOUN PHRASE> -> <NOUN>
<VERB PHRASE> -> <VERB> <NOUN PHRASE>
 <DETERMINER> -> the
       <NOUN> -> boys
       <NOUN> -> apples
       <VERB> -> eat
```

In this example, the, boys, apples, and eat are the terminals in the language and <SENTENCE> is the start symbol.

An important property of context-free grammars in their use in NL programs is that every derivation can conveniently be represented as a tree, which can be thought of as displaying the structure of the derived sentence. Using the grammar above, the sentence "the boys eat apples" has the following *derivation tree*:



Of course, "the apples eat boys" is also a legal sentence in this language. Derivation trees can also be used with context-sensitive (type-1) grammars, provided the productions have the alternate form uXv -> uYv, described above. For this reason, context-free and context-sensitive grammars are often called *phrase-structure grammars* (see Chomsky, 1959, pp. 143-144, and Lyons, 1968, p. 236).

Type 3. Finally, if every production is either of the form

X -> aY or X -> a

where X and Y are single variables and a is a single terminal, the grammar is a type-3 or *regular grammar*. For example, a regular grammar can be given to generate the set of strings of one or more as followed by one or more bs (but with no guarantee of an equal number of as and bs):

S -> aS
S -> aT
T -> b
T -> bT

Discussion: Language and Computational Algorithms

Because of the increasingly restricted forms of productions in grammars of type 0, 1, 2, and 3, each type is a proper subset of the type above it in the hierarchy. A corresponding hierarchy exists for formal languages. A language is said to be of type i if it can be generated by a type-i grammar. It can be shown that languages exist that are context-free (type 2) but not regular (type 3); context-sensitive (type 1) but not context-free; and type 0 but not context-sensitive. Examples of the first two have been given above.

For regular and context-free grammars, there are practical parsing algorithms to determine whether or not a given string is an element of the language and, if so, to assign to it a syntactic structure in the form of a derivation tree. Context-free grammars have considerable application to programming languages. Natural languages, however, are not generally context-free (Chomsky, 1963; Postal, 1964), and they also contain features that can be handled more conveniently, if not exclusively, by a more powerful grammar. An example is the requirement that the subject and verb of a sentence be both singular or both

plural. Some of the types of grammars and parsing algorithms that have been explored as more suitable for natural language are discussed in the articles that follow.

References

For a general discussion of the theory of formal grammars and their relation to automata theory, see Hopcroft & Ullman (1969). Their use in NL research is discussed in Winograd (forthcoming).

Also of interest are the works of Chomsky (especially 1956, 1957, and 1959), as well as Lyons (1968), Lyons (1970), and Postal (1964).

C2. Transformational Grammars

The term *transformational grammar* refers to a theory of language introduced by Noam Chomsky in **Syntactic Structures** (1957). In the theory an utterance is characterized as the surface manifestation of a "deeper" structure representing the "meaning" of the sentence. The deep structure can undergo a variety of "transformations" of form (word order, endings, etc.) on its way up, while retaining its essential meaning. The theory assumes that an adequate grammar of a language like English must be a *generative grammar*, that is, that it must be a statement of finite length capable of (a) accounting for the infinite number of possible sentences in the language and (b) assigning to each a structural description that captures the underlying knowledge of the language possessed by an idealized native user. A *formal system of rules* is such a statement; it "can be viewed as a device of some sort for producing the sentences of the language under analysis" (Chomsky, 1957, p. 11). The operation of the device is not intended to reflect the processes by which people actually speak or understand sentences, just as a formal proof in mathematics does not purport to reflect the processes by which the proof was discovered. As a model of abstract knowledge and not of human behavior, generative grammar is said to be concerned with competence, as opposed to performance.

The Inadequacy of Phrase-structure Grammars

Given that a grammar is a generative rule-system, it becomes a central task of linguistic theory to discover what the rules should look like. In **Syntactic Structures** (1957) and elsewhere (see Chomsky, 1963, Postal, 1964), it was shown that English is neither a *regular* nor a *context-free* language. The reason is that those restricted types of grammars (defined in Article C1) cannot generate certain common constructions in everyday English, such as the one using "respectively":

> Arthur, Barry, Charles, and David are the husbands of Jane, Joan, Jill, and Jennifer, respectively.

It was not determined whether a more powerful (i.e., context-sensitive) grammar could be written to generate precisely the sentences of English; rather, such a grammar was rejected for the following reasons.

1. It made the description of English unnecessarily clumsy and complex--for example, in the treatment required for conjunction, auxiliary verbs, and passive sentences.

2. It assigned identical structures (derivation trees) to sentences that are understood differently, as in the pair:

> The picture was painted by a new technique.
> The picture was painted by a new artist.

3. It provided no basis for identifying as similar sentences having different surface structures but much of their "meaning" in common:

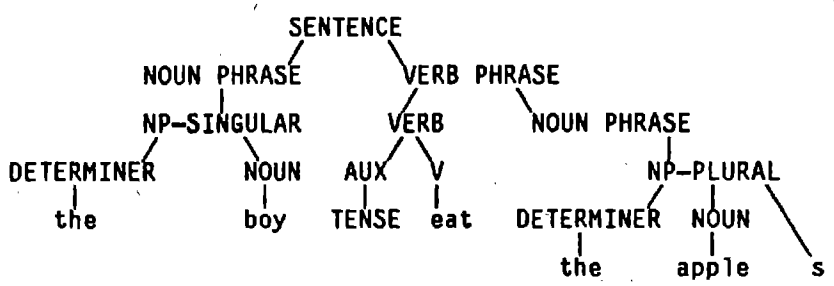
John ate an apple.
Did John eat an apple?
What did John eat?
Who ate an apple?

The failure of phrase-structure grammar to explain such similarities and differences was taken to indicate the need for analysis on a higher level, which transformational grammar provides.

Transformational Rules

In **Syntactic Structures**, Chomsky proposed that grammars should have a tripartite organization. The first part was to be a phrase-structure grammar generating strings of morphemes representing simple, declarative, active sentences, each with an associated phrase marker or derivation tree. Second, there would be a sequence of *transformational rules* rearranging the strings and adding or deleting morphemes to form representations of the full variety of sentences. Finally, a sequence of morphophonemic rules would map each sentence representation to a string of phonemes. Although later work has changed this model of the grammar, as well as the content of the transformational rules, it provides a basis for a simple illustration.

Suppose the phrase-structure grammar is used to produce the following derivation tree:



To generate "the boy ate the apples," one would apply transformations mapping "TENSE + eat" to "eat + PAST"; a morphophonemic rule would then map "eat + PAST" to ate. To derive "the boy eats the apples," the transformational rule used would select present tense and, because the verb follows a singular noun phrase, would map "TENSE + eat" to "eat + s." It is noteworthy that the transformational rule must look at nonterminal nodes in the derivation tree to determine that "the boy" is in fact a singular noun phrase. This example illustrates one respect in which transformational rules are broader than the rules of a phrase-structure grammar.

The transformations mentioned so far are examples of *obligatory transformations*, insuring agreement in number of the subject and the verb. To obtain "the apples were eaten by the

boy," it would be necessary first to apply the optional *passive* transformation, changing a string analyzed as

NOUN-PHRASE-1 + AUX + V + NOUN-PHRASE-2

to

NOUN-PHRASE-2 + (AUX + be) + (en + V) + by + NOUN-PHRASE-1

In other words, this *optional transformation* changes "the boy TENSE eat the apples" to "the apples TENSE be (en eat) by the boy," and then forces agreement of the auxiliary verb with the new plural subject. Further obligatory transformations would yield "the apples be PAST eaten by the boy" (where "be + PAST," as opposed to "be + s + PAST," is ultimately mapped to were). The *ordering* of transformational rules is thus an essential feature of the grammar.

Revisions to the Model

In **Aspects of the Theory of Syntax** (1965), Chomsky made several revisions to the model presented in **Syntactic Structures**. The version outlined in the more recent book has been called the "standard theory" of generative grammar and has served as a common starting-point for further discussion. In the standard theory (as summarized in Chomsky, 1971), sentence generation begins from a context-free grammar generating a sentence structure and is followed by a selection of words for the structure from a *lexicon*. The context-free grammar and lexicon are said to form the *base* of the grammar; their output is called a *deep structure*. A system of transformational rules maps deep structures to *surface structures*; together, the base and transformational parts of the grammar form its *syntactic component*. The sound of a sentence is determined by its surface structure, which is interpreted by the *phonological component* of the grammar; deep structure, interpreted by the *semantic component*, determines sentence meaning. It follows that the application of transformational rules to deep structures must preserve meaning: This was the Katz-Postal hypothesis, which required enlarging the generative capacity of the base and revising many of the transformational rules suggested earlier (Katz & Postal, 1964).

The place of the semantic component in the standard theory has been the major source of current issues. For example, the following pairs of sentences have different meanings, but their deep structures, in the standard theory, are the same.

Not many arrows hit the target.
Many arrows didn't hit the target.

Each of Mary's sons loves his brothers.
His brothers are loved by each of Mary's sons.

Chomsky's response was to revise the standard theory so that both the deep structure of a sentence and its subsequent transformations are input to the semantic component (Chomsky, 1971). He exemplifies the position of *interpretive semantics*, which keeps the syntactic component an autonomous system. The opposing view, called *generative semantics*, is that syntax and semantics cannot be sharply separated and, consequently, that a distinct level of syntactic deep structure does not exist. (This issue is discussed in Charniak & Wilks, 1976.)

There have been a number of developments within the theory of transformational grammar since the work reviewed here, and current debates have called into question many of the basic assumptions about the role of transformations in a grammar. For current discussions of these issues, see Akmajian, Culicover and Wasow (1977) and Bresnan (1978).

References

The classic references here are, of course, Chomsky (1957) and Chomsky (1965). Chomsky (1971) is a shorter and more recent discussion. Culicover, Wasow, & Akmajian (1977) and Bresnan (1978) are the latest word on transformation theory.

Also see Akmajian & Heny (1975), Charniak & Wilks (1976), Chomsky (1956), Chomsky (1959), Chomsky (1963), Harman (1974), Katz & Postal (1964), Lyons (1968), Lyons (1970), Postal (1964), and Steinberg & Jakobovits (1971).

C3. Systemic Grammar

Systemic grammar, developed by Michael Halliday and others at the University of London, is a theory within which linguistic structure as related to the *function* or use of language, often termed *pragmatics*, is studied. According to Halliday (1961, p. 141), an account of linguistic structure that pays no attention to the functional demands we make on language is lacking in perspicacity, since it offers no principles for explaining why the structure is organized one way rather than another. This viewpoint is in contrast to that of *transformational grammar*, which has been concerned with the syntactic structure of an utterance apart from its intended use.

The Functions of Language

Halliday distinguishes three general functions of language, all of which are ordinarily served by every act of speech.

The *ideational function* serves for the expression of content. It says something about the speaker's experience of the world. Analyzing a clause in terms of its ideational function involves asking questions like: What kind of process does the clause describe--an action, a mental process, or a relation? Who is the actor (the logical subject)? Are there other participants in the process, such as goal (direct object) or beneficiary (indirect object)? Are there adverbial phrases expressing circumstances like time and place? The organization of this set of questions is described by what Halliday calls the *transitivity system* of the grammar. (This is related to the ideas of *case grammars* discussed in Article C4.)

The *interpersonal function* relates to the purpose of the utterance. The speaker may be asking a question, answering one, making a request, giving information, or expressing an opinion. The *mood system* of English grammar expresses these possibilities in terms of categories such as statement, question, command, and exclamation.

The *textual function* reflects the need for coherence in language use (e.g., how a given sentence is related to preceding ones). Concepts used for analysis in textual terms include: (1) theme, the element that the speaker chooses to put at the beginning of a clause; and (2) the distinction between what is new in a message and what is given--the latter being the point of contact with what the hearer already knows.

Categories of Systemic Grammar

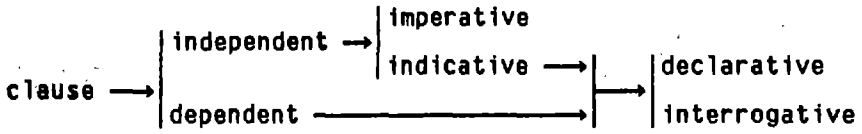
The model of a grammar proposed by Halliday has four primitive categories:

1. The units of language, which form a hierarchy. In English, these are the sentence, clause, group, word, and morpheme. The "rank" of a unit refers to its position in the hierarchy.

2. The structure of units. Each unit is composed of one or more units at the rank below, and each of these components fills a particular role. The English clause, for example, is made up of four groups, which serve as subject, predicator, complement, and adjunct.

3. The classification of units, as determined by the roles to be filled at the level above. The classes of English groups, for instance, are the verbal, which serves as predicator; the nominal, which may be subject or complement; and the adverbial, which fills the adjunct function.

4. The system. A system is a list of choices representing the options available to the speaker. Since some sets of choices are available only if other choices have already been made, the relationship between systems is shown by combining them into networks, as in the simple example below:



The interpretation is that each clause is independent or dependent; if independent, it is either imperative or indicative; and if either indicative or dependent, then it is either declarative or interrogative. In general, system networks can be defined for units of any rank, and entry to a system of choices may be made to depend on any Boolean combination of previous choices.

Conclusion

Systemic grammar views the act of speech as a simultaneous selection from among a large number of interrelated options, which represent the "meaning potential" of the language. If system networks representing these options are suitably combined and carried to enough detail, they provide a way of writing a generative grammar quite distinct from that proposed by *transformational grammar* (see Hudson, 1971, 1976; McCord, 1975; and Self, 1975). Furthermore, this formalism has been found more readily adaptable for use in natural language understanding programs in AI (see especially Winograd's SHRDLU system, Article F4).

References

Halliday (1961) and Halliday (1970b) are the most general original references. Winograd (1972) discusses the application of systemic grammar in an NL program.

Also see Halliday (1967-68), Halliday (1970a), Hudson (1971), Hudson (1976), McCord (1975), McIntosh & Halliday (1966), and Self (1975).

C4. Case Grammars

Case systems, as used both in modern linguistics and in artificial intelligence, are descendants of the concept of *case* that occurs in traditional grammar. Traditionally, the case of a noun was denoted by an inflectional ending indicating the noun's role in the sentence. Latin, for example, has at least six cases: the nominative, accusative, genitive, dative, ablative, and vocative. The rules for case endings make the meaning of a Latin sentence almost independent of word order: The function of a noun depends on its inflection rather than its position in the sentence. Some present-day languages, including Russian and German, have similar inflection systems, but English limits case forms mainly to the personal pronoun, as in I, my, me, and to the possessive ending 's. Case functions for nouns are indicated in English by using word order or by the choice of preposition to precede a noun phrase--as in "of the people, by the people, and for the people."

The examples above describe what have been called "surface" cases; they are aspects of the *surface structure* of the sentence. Case systems that have attracted more recent attention are "deep" cases, proposed by Fillmore (1968) in his paper *The Case for Case*, as a revision to the framework of *transformational grammar*. The central idea is that the proposition embodied in a simple sentence has a deep structure consisting of a verb (the central component) and one or more noun phrases. Each noun phrase is associated with the verb in a particular relationship. These relationships, which Fillmore characterized as "semantically relevant syntactic relationships," are called *cases*. For example, in the sentence

John opened the door with the key ,

John would be the AGENT of the verb opened, the door would be the OBJECT, and the key would be the INSTRUMENT. For the sentence

The door was opened by John with the key ,

the case assignments would be the same, even though the surface structure has changed.

It was important to Fillmore's theory that the number of possible case relationships be small and fixed. Fillmore (1971b) proposed the following cases:

Agent	-- the instigator of the event.
Counter-Agent	-- the force or resistance against which the action is carried out.
Object	-- the entity that moves or changes or whose position or existence is in consideration.
Result	-- the entity that comes into existence as a result of the action.
Instrument	-- the stimulus or immediate physical cause of an event.
Source	-- the place from which something moves.
Goal	-- the place to which something moves.
Experiencer	-- the entity which receives or accepts or experiences or undergoes the effect of an action.

Still another proposal (Fillmore, 1971a) recognizes 9 cases: Agent, Experiencer, Instrument, Object, Source, Goal, Location, Time, and Path.

Verbs were classified according to the cases that could occur with them. The cases for any particular verb formed an ordered set called a *case frame*. For example, the verb "open" was proposed to have the case frame

[OBJECT (INSTRUMENT) (AGENT)]

indicating that the object is obligatory in the deep structure of the sentence, whereas it is permissible to omit the instrument ("John opened the door") or the agent ("The key opened the door"), or both ("The door opened"). Thus, verbs provide *templates* within which the remainder of the sentence can be understood.

The Case for Case

The following are some of the kinds of questions for which case analysis was intended to provide answers:

1. In a sentence that is to contain several noun phrases, what determines which noun phrase should be the subject in the surface structure? Cases are ordered, and the highest ranking case that is present becomes the subject.

2. Since one may say "Mother is baking" or "The pie is baking," what is wrong with "Mother and the pie are baking"? Different cases may not be conjoined.

3. What is the precise relationship between pairs of words like "buy" and "sell" or "teach" and "learn"? They have the same basic meaning but different case frames.

One way of looking at deep cases is to view the verb as a predicate taking an appropriate array of arguments. Fillmore has extended the class of predicates to include other parts of speech, such as nouns and adjectives, as well as verbs. Viewing warm as a predicate, for example, enabled case distinctions to account for the differences among the following sentences:

I am warm.	[experiencer]
This jacket is warm.	[instrument]
Summer is warm.	[time]
The room is warm.	[location]

The Representation of Case Frames

In artificial intelligence programs, such predicates and their arguments can readily be equated to nodes in *semantic networks*; and the case relations, to the kinds of links between

them. Systems making such identifications include those of Simmons (1973), Schank (1975), Schank & Abelson (1977), and Norman & Rumelhart (1975). Semantic nets and related work on *semantic primitives* and *frames* are discussed in the section on Knowledge Representation and in Articles F5 and F6 which describe the MARGIE and SAM systems.

Many other systems using case representations exist. As pointed out in an extensive survey by Bruce (1975), considerable variation exists in both the sets of cases adopted and the ways in which case representation is used. The number of cases used varies from four or five (Schank) to over thirty (Martin). Bruce's proposal on criteria for choosing cases, which departs significantly from Fillmore's original goal of finding a small, fixed set of relationships, is that:

> A case is a relation which is "important" for an event in the context in which it is described. (Bruce, 1975)

Case notation has been used to record various levels of sentence structure. As Fillmore introduced it, within the transformational grammar framework, deep cases were "deep" in the sense that "John opened the door" and "the door was opened by John" were given the same representation. They can also be viewed as relatively superficial, however, in that "John bought a car from Bill" and "Bill sold a car to John" could have distinct representations since they have different verbs. At this level, cases have been used in parsing (Wilks, 1976; Taylor & Rosenberg, 1975); in the representation of English sentences, as opposed to their underlying meanings, as discussed above (Simmons, 1973); and in *text generation* (see Article E).

Systems using case at the deepest level, on the other hand, may represent the meaning of sentences in a way that collapses buy and sell, for example, into a single predicate (Schank, 1975; Norman & Rumelhart, 1975). A typical problem attacked by these systems is *paraphrasing*, where identifying sentences with the same deep structure is the goal. Schank (1975) also requires that all cases be filled, even if the information required was not explicitly given in the sentences represented. Charniak (1975) suggests that the appropriate use of case at this level of representation is in *inferencing*: The "meaning" of a case would then be the set of inferences one could draw about an entity knowing only its case. In the view of some writers, however, the function of case in natural language understanding systems is usually only as a convenient notation (see Charniak, 1975; Welin, 1975).

References

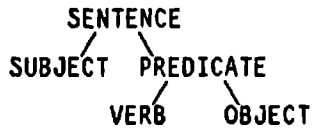
Fillmore (1968) is the classic reference on case grammars. Bruce (1975) is a thorough review of different approaches to case grammar.

Also see Charniak (1975), Fillmore (1971a), Fillmore (1971b), Norman & Rumelhart (1975), Samlowski (1976), Schank (1973), Schank (1975), Schank & Abelson (1977), Simmons (1973), Taylor & Rosenberg (1975), Welin (1975), and Wilks (1976).

D. Parsing

D1. Overview of Parsing Techniques

Parsing is the "delinearization" of linguistic input, that is, the use of syntax and other sources of knowledge to determine the functions of the words in the input sentence in order to create a data structure, like a *derivation tree*, that can be used to get at the "meaning" of the sentence. A parser can be viewed as a *recursive pattern matcher* seeking to map a string of words onto a set of meaningful syntactic patterns. For example, the sentence "John kissed Mary" could be matched to the pattern:



The set of syntactic patterns used is determined by the *grammar* of the input language. (Several types of grammars are described in the articles in Section C.) In theory, by applying a comprehensive grammar, a parser can decide what is and what is not a grammatical sentence and can build up a data structure corresponding to the syntactic structure of any grammatical sentence it finds. All natural language processing computer systems contain a parsing component of some sort, but the practical application of grammars to natural language has proven difficult.

The design of a parser is a complex problem, both in theory and implementation. The first part of the design concerns the specification of the grammar to be used. The rest of the parsing system is concerned with the method of *use* of the grammar, that is, the manner in which strings of words are matched against patterns of the grammar. These considerations run into many of the general questions of computer science and artificial intelligence concerning process control and manipulation of knowledge.

General Issues of Parser Design

The design considerations discussed below overlap; that is, a decision in one dimension affects other design decisions. Taken together they present a picture of the variety of issues involved in natural language parsing.

Uniformity. Parsers may *represent* their knowledge about word meanings, grammar, etc., with a single scheme or with specialized structures for specific tasks. The representation scheme affects the complexity of the system and the application of knowledge during parsing. If rules and processes are based on specialized knowledge of what the input to the parser will contain, it is possible to do things more quickly and efficiently. On the other hand, if one has a simple uniform set of rules and a consistent algorithm for applying them, the job of writing and modifying the language understanding system is greatly simplified, since all the knowledge in the system is uniformly explicated. In general, there is a trade-off between efficiency and uniformity; an algorithm specially designed for only one language can perform more efficiently than one that could uniformly handle any language.

Multiple Sources of Knowledge. Parsing, as originally developed (and still used in programming language compilers), was based purely on syntactic knowledge--knowledge about the *form* of sentences allowed in the language. However, it is possible to design systems in which syntax-based parsing is intermixed with other levels of processing, such as word recognition and use of word meanings. Such methods can alleviate many of the problems of language complexity by bringing more information to bear. Present systems tend toward such intermixed structures, both for effective performance and more psychologically valid modeling of human language understanding (see, for example, Article F4 on SHRDLU and the extensive discussion of multiple sources of knowledge in Article Applications.C3 on the SOPHIE system and the *blackboard* model in the Speech Understanding section).

Precision. Another major trade-off involved in parser design is *precision* vs. *flexibility*. Humans are capable of understanding sentences that are not quite grammatical; even if a person knows that a sentence is "wrong" syntactically, he can often understand it and assign it a structure (and more importantly, a meaning). Some natural language processing systems, such as PARRY (Colby, Weber, & Hilf, 1971) and ELIZA (Article F1) have been designed to incorporate this kind of flexibility. By looking for key words and using loose grammatical criteria, these systems can accept far more sentences than would a precise parser. However, these "knowledge-poor" *flexible* parsers lose many benefits of the more complete analysis possible with a precise system, since they rely on vaguer notions of sentence meaning than a precise system. While they reject less often, flexible systems tend to misinterpret more often. Many systems attempt to use additional *knowledge sources*, especially domain-specific knowledge, to increase flexibility while retaining precision.

Type of structure returned. As mentioned, parsing is the process of assigning structures to sentences. The form of the structure can vary, from a representation that closely resembles the surface structure of the sentence to a deeper representation in which the surface structure has been extensively modified. Which form is chosen depends upon the use to which the parse structure will be put. Currently, most work in natural language favors the deep structure approach.

These four issues--uniformity, multiple knowledge sources, precision, and level of representation--are very general questions and are dealt with in different ways by different systems. In implementing a parser, after settling such general design questions, natural language programmers run up against another set of problems involving specific parsing *strategies*.

Parsing Strategies

Backtracking versus parallel processing. Unfortunately for computational linguists, the elements of natural languages do not always possess unique meanings. For example, in going through a sentence the parser might find a word that could either be a noun or a verb, like "can," or pick up a prepositional phrase that might be modifying any of a number of the other parts of the sentence. These and many other ambiguities in natural languages force the parser to make choices between multiple alternatives as it proceeds through a sentence. Alternatives may be dealt with all at the same time, via *parallel processing*, or one at a time using a form of *backtracking*--backing up to a previous choice-point in the computation and trying again (see Article AI Languages.B3 on *control mechanisms* in AI programming languages). Both of these methods require a significant amount of bookkeeping to keep track of the

multiple possibilities: all the ones being tried, in the case of parallel processing; or all the ones not yet tried, in the case of backtracking. Neither strategy can be said to be innately superior, though the number of alternatives that are actually tried can be significantly reduced when backtracking is guided by "knowledge" about which of the choices are more likely to be correct--called *heuristic* knowledge (see Search.Overview).

Top-down versus bottom-up. In deriving a syntactic structure, a parser can operate from the goals, that is, the set of possible sentence structures (*top-down processing*), or from the words actually in the sentence (*bottom-up processing*). A strictly top-down parser begins by looking at the rules for the desired top-level structure (sentence, clause, etc.); it then looks up rules for the constituents of the top-level structure, and progresses until a complete sentence structure is built up. If this sentence matches the input data, the parse is successfully completed, otherwise, it starts back at the top again, generating another sentence structure. A bottom-up parser looks first for rules in the grammar to combine the words of the input sentence into constituents of larger structures (phrases and clauses), and continues to try to recombine these to show how all the words in the input form a legal sentence in the grammar. Theoretically, both of these strategies arrive at the same final analysis, but the type of work required and the working structures used are quite different. The interaction of top-down and bottom-up process control is a common problem in AI and is not restricted natural language programs (see, for example, the discussion in the Speech.A).

Choosing how to expand or combine. With either a top-down or bottom-up technique, it is necessary to decide how words and constituents will be combined (bottom-up) or expanded (top-down). The two basic methods are to proceed systematically in one direction (normally left to right) or to start anywhere and systematically look at neighboring chunks of increasing size (this method is sometimes called *island driving*). Both these methods will eventually look at all possibilities, but the choice of how to proceed at this level can have a significant effect on the efficiency of the parser. This particular feature is especially relevant to language processing in the presence of input uncertainty, as occurs, for example, in the *speech understanding* systems.

Multiple knowledge sources. As mentioned above, another important design decision that was especially apparent in the speech understanding systems was the effective use of multiple sources of knowledge. Given that there are a number of possibly relevant sets of facts to be used by the parser (phonemic, lexical, syntactic, semantic, etc.), which do you use when?

The issues discussed here under parsing strategies are all questions of *efficiency*. They will not in general affect the final result if computational resources are unlimited, but they will affect the amount of resources expended to reach it.

Actual Parsing Systems

Every natural language processing program deals with these seven issues in its own fashion. Several types of parsers have developed as experience with natural language systems increses.

Template matching. Most of the early NL programs (e.g., SIR, STUDENT, ELIZA) performed "parsing" by matching their input against a series of predefined *templates*--binding

the variables of the template to corresponding pieces of the input string (see Article F1). This approach was successful, up to a point--given a very limited topic of discussion, the form of many of the input sentences could be anticipated by the system's designer who incorporated appropriate templates. However, these systems were *ad hoc* and somewhat inextensible, and the template matching was soon abandoned in favor of more sophisticated methods.

Simple phrase-structure grammar parsers. These parsers make use of a type of *context-free grammar* with various combinations of the parsing techniques mentioned above. The advantage of a phrase-structure grammar is that the structures derived correspond directly to the grammar rules; thus, the subsequent semantic processing is simplified. By using large grammars and skirting linguistic issues that are outside their limitations (such as some types of agreement), a phrase-structure grammar parser can deal with a moderately large subset of English. Phrase-structure grammars are used primarily to produce systems (like SAD-SAM) with useful performance on a limited domain, rather than to explore more difficult language-processing issues.

Transformational grammar parsers. These parsers attempt to extend the notions of *transformational grammar* into a parsing system. Transformational grammar is a much more comprehensive system than phrase-structure grammar, but it loses phrase-structure's direct, rule-to-structure correspondence. Moreover, methods that have been tried, such as analysis by synthesis (building up all possible sentences until one matches the input) and inverse transformations (looking for transformation rules that might have produced the input), have often failed because of *combinatorial explosion*--the proliferation of alternatives the system must examine--and other difficulties with reversing transformations. One of the major attempts to implement a transformational parser was that by Petrick (1973).

Extended grammar parsers. One of the most successful AI approaches to parsing yet developed has been to extend the concept of phrase-structure rules and derivations by adding mechanisms for more complex representations and manipulations of sentences. Methods such as *augmented transition net grammars* (ATNs) and *charts* provide additional resources for the parser to draw on beyond the simple, phrase-structure approach (see Articles D2 and D3). Some of these mechanisms have validity with respect to some linguistic theory, while others are merely computationally expedient. The very successful systems of Woods & Kaplan (1971), Winograd (1972), and Kaplan (1973), as described in the articles in Section F, use extended grammar parsers.

Semantic grammar parsers. Another very successful modification to the traditional phrase structure grammar approach is to change the conception of grammatical classes from the traditional <NOUN>, <VERB>, etc., to classes that are motivated by concepts in the domain being discussed. For instance, such a *semantic grammar* for a system which talks about airline reservations might have grammatical classes like <DESTINATION>, <FLIGHT>, <FLIGHT-TIME>, and so on. The *rewrite rules* used by the parser would describe phrases and clauses in terms of these semantic categories (see Article Applications.C3 for a more complete discussion). The LIFER and SOPHIE systems (Articles F7 and Applications.C3) use semantic grammar parsers (Hendrix, 1977a, and Burton, 1976).

Grammarless parsers. Some NL system designers have abandoned totally the traditional use of grammars for linguistic analysis. Such systems are sometimes referred to as "ad hoc," although they are typically based on some loose theory that happens to fall

outside the scope of standard linguistics. These "grammarless" parsers opt for flexibility in the above-mentioned precision/flexibility trade-off. They are based on special procedures (perhaps centered on individual words rather than syntactic elements) that use semantics-based techniques to build up structures relevant to meaning, and these structures bear little resemblance to the normal structures that result from syntactic parsing. A good example of this approach can be found in the work of Riesbeck (1975).

Conclusion

Recent research in parsing has been directed primarily towards two kinds of simplification: providing simplified systems for dealing with less than full English, and providing simplified underlying mechanisms that bring the computer parsing techniques closer to being a theory of syntax. Systems such as LIFER (Article F7) have been developed which use the basic mechanisms of augmented grammars in a clean and easily programmable way. Systems like these cannot deal with the more difficult problems of syntax, but they can be used quickly and easily to assemble specialized parsers and are likely to be the basis for natural language "front ends" for simple applications.

At the same time, there has been a reevaluation of the fundamental notions of parsing and syntactic structure, viewed from the perspective of programs that understand natural language. Systems such as PARSIFAL (Marcus, 1978) attempt to capture in their design the same kinds of generalizations that linguists and psycholinguists posit as theories of language structure and language use. Emphasis is being directed toward the interaction between the structural facts about syntax and the control structures for implementing the parsing process. The current trend is away from simple methods of applying grammars (as with phrase-structure grammars), toward more "integrated" approaches. In particular, the grammar/strategy dualism mentioned earlier in this article has been progressively weakened by the work of Winograd (1972) and Riesbeck (1975). It appears that any successful attempt to parse natural language must be based upon some more powerful approach than traditional syntactic analysis. Also, parsers are being called upon to handle more "natural" text, including discourse, conversation, and sentence fragments. These involve aspects of language that cannot be easily described in the conventional, grammar-based models.

References

Again, much of this discussion is borrowed from Winograd (forthcoming). Other general surveys include Charniak & Wilks (1976), and Grishman (1976). For examples of recent work, the proceedings of the TINLAP conferences (TINLAP-1, 1975 and TINLAP-2, 1978) are recommended.

D2. Augmented Transition Nets

Augmented transition networks (ATNs) were first developed by William Woods (1970) as a versatile representation of grammars for natural languages. The concept of an ATN evolved from that of a *finite-state transition diagram*, with the addition of tests and "side-effect" actions to each arc, as described below. These additions resulted in the power needed for handling features of English like *embedding* and *agreement* that could not be conveniently captured by regular (or even context-free) grammars. An ATN can thus be viewed as either a grammar formalism or a machine.

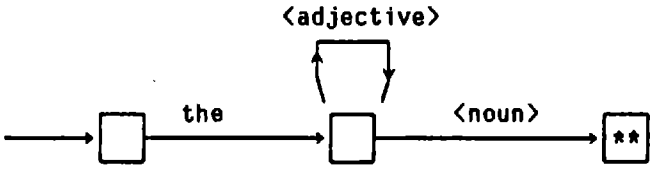
Many current language processors use an ATN-like grammar; in some ways, it may be considered state-of-the-art, at least for actual working systems.

Preliminary Theoretical Concepts

A finite-state transition diagram (FSTD) is a simple theoretical device consisting of a set of states (nodes) with arcs leading from one state to another. One state is designated the START state. The arcs of the FSTD are labeled with the terminals of the grammar (i.e., words of the language), indicating which words must be found in the input to allow the specified transition. A subset of the states is identified as FINAL; the device is said to *accept* a sequence of words if, starting from the START state at the beginning of the sentence, it can reach a FINAL state at the end of the input.

FSTDs can "recognize" only regular or type-3 languages (see the discussion of *formal languages* in Article C1). To recognize a language, a machine must be able to tell whether an arbitrary sentence is part of the language or is not. Regular grammars (those whose rewrite rules are restricted to the form Y -> aX or Y -> a) are the simplest, and FSTDs are only powerful enough to recognize these languages. In other words, it is impossible to build an FSTD that can dependably distinguish the sentences in even a context-free language.

For example, the following FSTD, in which the start state is the left-most node and the final state is labeled **, will accept any sentence that begins with the, ends with a noun, and has an arbitrary number of adjectives in between.

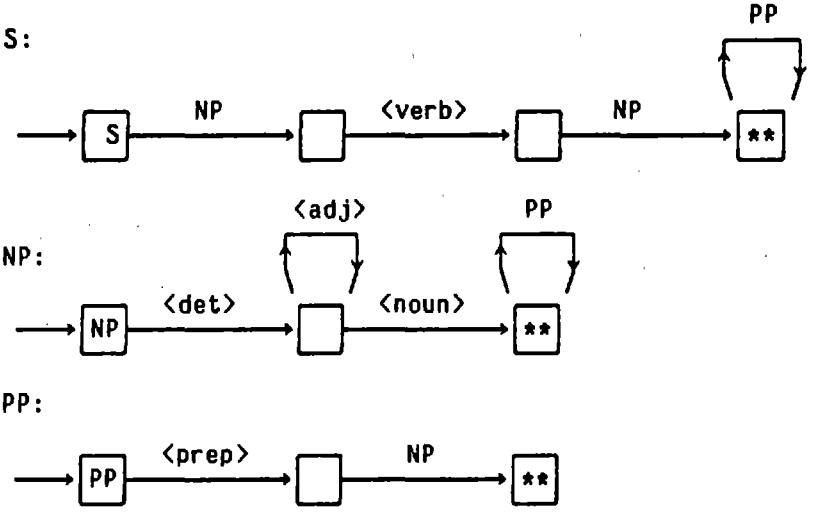


Let's follow through the net with the input sentence "the pretty picture." We start in the START state and proceed along the arc labeled the, because that is the left-most word in the input string. This leaves us in the middle box, with "pretty picture" left as our string to be parsed. After one loop around the adjective arc, we are again at middle node, but this time with the string "picture" remaining. Since this word is a noun, we proceed to the FINAL node, **, and arrive there with no words remaining to be processed. Thus the parse is successful; in other words, our example FSTD accepts this string.

However, regular grammars are inadequate for dealing with the complexity of natural language, as discussed in Article C2. A natural extension to FSTDs, then, is to provide a recursion mechanism that increases their recognition power to handle the more inclusive set of context-free languages. These extended FSTDs are called *recursive transition networks* (RTNs). An RTN is a finite-state transition diagram in which labels of an arc may include not only terminal symbols but also nonterminal symbols that denote the name of another subnetwork to be given temporary control of the parsing process.

An RTN operates similarly to an FSTD. If the label on an arc is a terminal (word or word class), the arc may be taken (as in FSTDs) if the word being scanned matches the label. For example, the word ball would match an arc labeled <noun> but not one labeled <adjective>. Otherwise, if the arc is labeled with a nonterminal symbol, representing a syntactic construct (e.g., PREPOSITIONAL PHRASE) that corresponds to the name of another network, the current state of the parse is put on a stack and control is transferred to the corresponding named subnetwork, which continues to process the sentence, returning control when it finishes or fails.

Whenever an accepting state is reached, control is transferred to the node obtained by "popping the stack" (i.e., returning to the point from which the subnetwork was entered). If an attempt is made to pop an empty stack, and if the last input word was the cause of this attempt, the input string is accepted by the RTN; otherwise, it is rejected. The effect of arcs labeled with names of syntactic constructs is that an arc is followed only if a construction of the corresponding type follows as a phrase in the input string. Consider the following example of an RTN:



Here NP denotes a noun phrase; PP, a prepositional phrase; det, a determiner; prep, a preposition; and adj, an adjective. Accepting nodes are labeled **. If the input string is "The little boy in the swimsuit kicked the red ball," the above network would parse it into the following phrases:

```
  NP:   The little boy in the swimsuit
  PP:   in the swimsuit
  NP:   the swimsuit
Verb:   kicked
  NP:   the red ball
```

Notice that any subnetwork of an RTN may call any other subnetwork, including itself; in the above example, for instance, the prepositional phrase contains a noun phrase. Also notice that an RTN may be nondeterministic in nature; that is, there may be more than one possible arc to be followed at a given point in a parse. Parsing algorithms handle nondeterminism by *parallel processing* of the various alternatives or by trying one and then *backtracking* if it fails. These general parsing issues are discussed in Article D1.

Context-free grammars, however, are still insufficient to handle natural language. The RTNs, then, must be extended, to provide even more parsing power.

ATNs

An augmented transition network (ATN) is an RTN that has been extended in three ways:

1. A set of *registers* has been added; these can be used to store information, such as partially formed *derivation trees*, between jumps to different networks.

2. Arcs, aside from being labeled by word classes or syntactic constructs, can have arbitrary *tests* associated with them that must be satisfied before the arc is taken.

3. Certain *actions* may be "attached" to an arc, to be executed whenever it is taken (usually to modify the data structure returned).

This addition of registers, tests, and actions to the RTNs extends their power to that of Turing machines, thus making ATNs theoretically powerful enough to recognize any language that might be recognized by a computer. ATNs offer a degree of expressiveness and naturalness not found in the Turing machine formalism, and are a useful tool to apply to the analysis of natural language.

The operation of the ATN is similar to that of the RTN except that if an arc has a test then the test is performed first, and the arc is taken only if the test is successful. Also, if an arc has *actions* associated with it, then these operations are performed after following the arc. In this way, by permitting the parsing to be guided by the parse history (via tests on the registers) and by allowing for a rearrangement of the structure of the sentence during the parse (via the actions on the registers), ATNs are capable of building deep structure descriptions of a sentence in an efficient manner. For a well-developed and clear example, the reader is referred to Woods (1970).

Evaluation of ATNs and Results

ATNs serve as an computationally implementable and efficient solution to some of the problems of recognizing and generating natural language. Their computational power provides the capability to embed different kinds of grammars, making them an effective testbed for new ideas. Two of the features of ATNs, the test and the actions on the arcs, make them especially well suited to handling *transformational grammars*. The ability to place arbitrary conditions on the arcs provides context sensitivity, equivalent to the preconditions for applying transformational rules. The capability to rearrange the parse structure, by copying, adding, and deleting components, provides the full power of transformations (see Article C2).

The ATN paradigm has been successfully applied to question answering in limited (closed) domains, such as the LUNAR program, which is described in Article F3. Also, ATNs have been used effectively in a number of *text generation* systems. In addition, the BBN *speech understanding* system, SPEECHLIS, uses an ATN control structure (see Article Speech.B3).

There are limitations to the ATN approach; in particular, the heavy dependence on syntax restricts the ability to handle ungrammatical (although meaningful) utterances. More recent systems (see especially Riesbeck's work, Article F5) are oriented toward meaning rather than structure and can thus accept mildly deviant input.

References

The principal references here are, of course, Woods (1970), Woods & Kaplan (1971), and Woods (1973a). Also see Bobrow & Fraser (1969), Conway (1963), Matuzceck (1972), and Winograd (1976).

D3. The General Syntactic Processor

Ronald Kaplan's (1973) General Syntactic Processor (GSP) is a versatile system for the parsing and generation of strings in natural language. Its data structures are intuitive and the control structures are conceptually straightforward and relatively easy to implement. Yet, by adjusting certain control parameters, GSP can directly emulate several other syntactic processors, including Woods's ATN grammar (Article D2), Kay's MIND parser (Kay, 1973), and Friedman's *text generation* system (Article E).

GSP represents an effort both to synthesize the formal characteristics of different parsing methods and to construct a unifying framework within which to compare them. In this respect, GSP is a "meta-system"--it is not in itself an approach to language processing, but rather it is a system in which various approaches can be described.

Data Structure: Charts

GSP gains much of its power through the use of a single, basic data structure, the *chart*, to represent both the grammar and the input sentence. A chart can be described as a modified *tree*, which is usually defined as a set of nodes that can be partitioned into a root and a set of disjoint subtrees. A tree encodes two sorts of relations between nodes: DOMINANCE, the relation between a parent and daughter node; and PRECEDENCE, the relation between a node and its right-hand sister node. Figure 1 shows a tree representing a particular noun phrase.

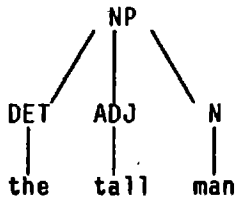


Figure 1. A tree for a noun phrase.

A chart is basically a tree that has been modified in two ways:

1. The arcs of the tree have been rearranged to produce a binary tree, that is, a tree in which each node has at most two dangling nodes (this rearrangement is described by Knuth [1973, p. 333] as the "natural correspondence" between trees and binary trees).

2. The nodes and arcs have been interchanged; what were previously nodes are now arcs, and vice versa.

For example, Figure 2 is the chart representation for the tree of Figure 1:

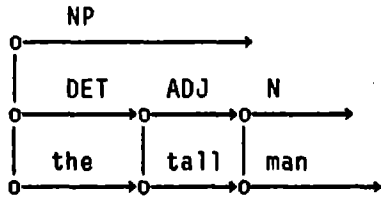


Figure 2. A chart for a noun phrase.

The chart representation has a number of advantages, including ease of access for certain purposes. For example, in Figure 1 there is no direct connection from DET to ADJ. In Figure 2 this connection has been made; that is, the PRECEDENCE relationships have been made explicit, and the DOMINANCE ones have been removed. This explicit encoding of precedence can be helpful in language processing, where the concept of one element following another is a basic relation.

Also, the chart can be used to represent a "string of trees" or "forest"--that is, a set of disjoint trees. For example, Figure 3a shows a string of two disjoint trees, headed by NP and V. Note that these trees cannot be connected, except with a dummy parent node (labeled ?). In Figure 3b, the equivalent chart representation is shown.

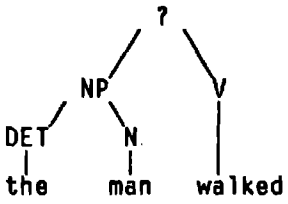


Figure 3a. Two disjoint trees.

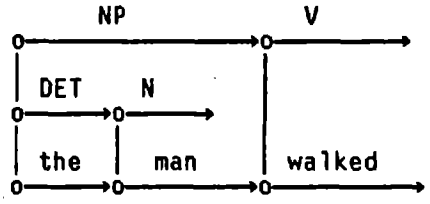


Figure 3b. The equivalent chart.

Finally, the chart provides a representation for multiple interpretations of a given word or phrase, through the use of multiple *edges*. The arcs in a chart are called edges and are labeled with the names of words or grammatical constructs. For example, Figure 4 represents the set of trees for "I saw the log," including the two interpretations for the word saw.

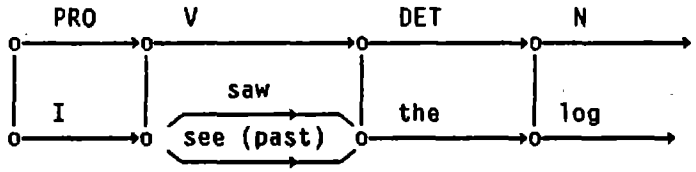


Figure 4. A chart showing multiple interpretations.

The chart allows explicit representation of ambiguous phrases and clauses, as well as of words.

Note that ambiguity could also be represented by distinct trees, one for every possible interpretation of the sentence. However, this approach is inefficient, as it ignores the possibility that certain subparts may have the same meaning in all cases. With the chart representation, these common subparts can be merged.

As defined earlier, the arcs in a chart are called edges and are labeled with the names of words or grammatical constructs. The nodes are called *vertexes*. The chart can be accessed through various functions, which enable one to retrieve specific edges, sets of edges, or vertexes.

At any given moment, the attention of the system is directed to a particular point in the chart called the CHART FOCUS. The focus is described by a set of global variables: EDGE (the current edge), VERTEX (the name of the node from which EDGE leaves), and CHART (the current subchart being considered by the processing strategy). GSP's attention is redirected by changing the values of these variables.

When the chart is initialized, each word in the sentence is represented by an edge in the chart for each category of speech the word can take. Figure 4 is an example of an initial chart configuration, preparatory to parsing. Each analysis procedure that shares the chart is restricted to adding edges, which gives later analyses the ability to modify or ignore earlier possibilities without constraining future interpretations. In this way, the individual syntactic programs remain relatively independent while building on each other's work in a generally *bottom-up* way.

It should be emphasized that the chart is just a data structure and is not directly related to the grammar. It merely serves as the global *blackboard* upon which the various pieces of the grammar operate. We still must specify the sorts of operations that use the chart--that is, the form of the grammar itself.

Data Structure: Grammatical Rules

Grammars for syntactic processing of language can be understood in terms of a network model like Woods's ATN grammar. That is, a grammar is viewed as a series of *states*, with transitions between the states accomplished by following *arcs* (see Article D2).

The grammars encoded by GSP fit this description. What gives GSP its power, however, is the fact that a grammar can be represented in the same way as a chart. That is, we can use the chart manipulation mechanisms, already developed, to operate upon the grammar itself. There is a difference, of course. The chart is merely a passive data store; the grammar contains instructions for: (a) acting on the chart--adding pieces and shifting attention; and (b) acting on the grammar--shifting attention (i.e., moving from one grammar state to another).

Control Structure

To handle the full complexity of grammars, GSP has some extra features. These include:

1. **REGISTERS.** As in ATNs, these are used as pointers to structures.

2. LEVELSTACK. This is a stack used to implement recursion. The chart focus, grammar focus (state), and register list are saved before a recursive call.

3. NDLIST (nondeterminism list). This is a list of choice points in the grammar. Whenever a choice is made, the user can optionally save the current configuration on NDLIST, to allow for backtracking.

4. PROCSTACK. This is a list of suspended processes. GSP allows a *co-routining* facility, under which processes can be suspended and resumed (ATNs have no equivalent to this).

Features like recursion, *backtracking*, and movement of the pointer through the input sentence must all be handled by the user within the general framework provided. This approach can be beneficial, particularly with features such as backtracking: automatic backtracking can be a less-than-desirable feature in a grammar (see the discussion in the AI Programming Languages Section).

Using GSP

Note one facet of the approach outlined: All operations on the grammar and chart must be *explicitly* stated. Thus, GSP has placed much power in the hands of the grammar designer, with a corresponding cost in complexity.

GSP appears to be similar to an ATN, with three extensions:

1. The data structure used is a chart, instead of simply a string of words.

2. The grammar is encoded in the same manner as the chart; thus it is accessible to the system.

3. Processes can be suspended and resumed.

ATNs do not fully demonstrate the power of GSP. Kaplan also used GSP to implement Kay's MIND parser (a context-free, bottom-up system) and Friedman's transformational grammar text-generation system. The latter two made more extensive use of GSP's capabilities, in particular: (a) the possibilities of multiple levels in the chart; (b) the ability to suspend and restart processes; and (c) the ability to rearrange the chart, changing it as necessary. The Kay algorithm, in particular, made extensive use of the ability to modify the chart "on the fly," adding sections as required.

Conclusions and Observations

GSP provides a simple framework within which many language processing systems can be described. It is not intended to be a high-level system that will do many things for the user; rather, it provides a "machine language" for the user to specify whatever operations he wants. GSP's small set of primitive operations seems to be sufficient for representing most desirable features of syntax-based parsing. The clean, uniform structure enables GSP to be used as a tool for comparison (and possibly evaluation) of different systems.

The chart seems to be an effective data structure for representing the syntax of natural language sentences. It provides convenient merging of common subparts (i.e., to prevent re-scanning known components), while permitting representation of various forms of ambiguity. As Kay explained, the function of the chart is to "record hypotheses about the phraseological status of parts of the sentence so that they will be available for use in constructing hypotheses about larger parts at some later time" (Kay, 1973, p. 167).

The backtracking mechanism is very general and thus can be inefficient if used too enthusiastically. Kaplan points out that heuristic ordering of alternatives is possible by altering the function that retrieves configurations from the NDLIST, though compilers should in any case attempt to minimize backtracking.

References

Kaplan (1973) is the principal reference. See also Friedman (1971), Kay (1973), Knuth (1973), and Woods (1970).

E. Text Generation

Text generation is, in a sense, the opposite of natural language understanding by machine--it is the process of constructing text (i.e., phrases, sentences, paragraphs) in a natural language. Although this field has been pursued for fifteen years, few coherent principles have emerged, and the approaches have varied widely. Attempts at generating text have been made with two general research goals: (a) generating random sentences to test a grammar or grammatical theory and (b) converting information from an internal representation into a natural language.

Random Generation

This approach, the random generation of text constrained by the rules of a test grammar, is of limited interest to workers in Artificial Intelligence, being oriented more toward theoretical linguistics than functional natural language processing systems. The objective of implementing a generation system of this sort is to test the descriptive adequacy of the test grammar, as illustrated by the following two systems.

Victor Yngve (1962) was one of the first researchers to attempt English text generation; the work was seen as preliminary to a full program for machine translation (see Article B). Yngve used a generative *context-free grammar* and a random-number generator to produce "grammatical" sentences: The system selected one production randomly from among those that were applicable at each point in the generation process, starting from those productions that "produced" <SENTENCE>, and finally randomly selecting words to fill in the <NOUN>, <VERB>, etc., positions. This is an example of the text produced by the system:

> The water under the wheels in oiled whistles and its polished shiny big and big trains is black.

Joyce Friedman's (1969, 1971) system was designed to test the effectiveness of *transformational grammars* (Article C2). It operated by generating *phrase markers* (derivation trees) and by performing transformations on them until a *surface structure* was generated. The generation was random, but the user could specify an input phrase marker and semantic restrictions between various terminals in order to test specific rules for grammatical validity.

These two systems, while relevant to work in linguistics, are only peripherally related to recent work in Artificial Intelligence. The fundamental emphasis in AI text-generation work has been on the meaning, as opposed to the syntactic form, of language.

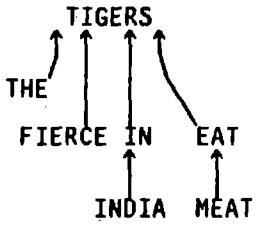
Surface Realization of Meaning

The general goal of text-generation programs in the AI paradigm is to take some internal representation of the "meaning" of a sentence and convert it to surface structure form, i.e. into an appropriate string of words. There has been considerable variety among such systems, reflecting differences both in the type of internal representation used and in the overall purpose for which the text is generated. Representation schemes have included largely syntactic dependency trees, stored generation patterns of different degrees of complexity, and several versions of *semantic nets* (see the Knowledge Representation section

of the Handbook). Purposes have included automatic *paraphrasing* or *mechanical translation* of an input text; providing natural-sounding communication with the user of an interactive program; and simply testing the adequacy of the internal representation.

Sheldon Klein (1965) made a first step beyond the random generation of sentences, by means of a program that attempted to generate a *paraphrase* of a paragraph of text via an internal representation of that text (see also Klein & Simmons, 1963). The program used a type of grammar called *dependency grammar*, a context-free grammar with word-dependency information attached to each production. That is, the right-hand side of each rule in the grammar has a "distinguished symbol"; the "head" of the phrase associated with that rule is the head of the phrase that is associated with the distinguished symbol. All other words that are part of the phrase associated with the production are said to *depend* on this head.

For instance, given the following simple dependency grammar and the sentence "the fierce tigers in India eat meat," Klein's parser would produce both an ordinary phrase-structure derivation tree (see Article C1) and also the dependency tree shown below:

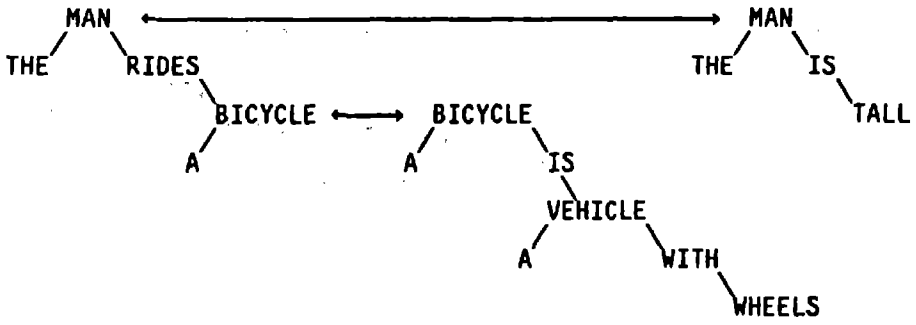


S ← NP* + VP
NP ← DET + ADJ + N* + PP
PP ← PREP* + NOUN
VP ← V* + OBJ

The symbols followed by * are the distinguished symbols in the productions. The dependency trees from the individual sentences of the input paragraph were bound together with "two-way dependency" links between similar nouns. For example, the input paragraph

> The man rides a bicycle. The man is tall. A bicycle is a vehicle with wheels.

would yield the following dependency structure:



One paraphrase generated from the given paragraph was:

> The tall man rides a vehicle with wheels.

The grammar used in generation was similar to the one used for analysis. Rule selection

was random (as in Yngve's method) but with the added constraint that all dependencies among the words that were generated must be derivable from the initial dependency trees. In the example above, vehicle could be generated as the object of rides because vehicle depends on is, is on bicycle, and bicycle on rides. Two restrictions were imposed on the transitivity of dependency relations: Dependency did not cross verbs other than be or prepositions other than of. Thus "the man rides wheels" could not be generated.

The use of dependency trees was expected to insure that the output sentences would "reflect the meaning of the source text" (Klein, 1965, p. 74). A difficulty, however, was that the trees encoded only the crudest of semantic relationships present in the paragraph. In fact, the dependency relationship between words only indicates that some semantic relation exists between them without really specifying the nature of the relationship.

Ross Quillian (1968), in contrast, emphasized the expression of semantic relationships almost to the exclusion of concern for syntactic well-formedness. Quillian did pioneering work in the representation of knowledge (see the Knowledge Representation section of the Handbook}) and was also one of the first to deal with the problems of text generation. His system used a *semantic net* to represent the relations between words, which can be interpreted as their meaning. The task the system was then to perform was to compare two words, that is, find some semantic relation between them, and then to express the comparison in "understandable, though not necessarily grammatically perfect, sentences" (p. 247). For example:

Compare: Plant, Live

Answer: PLANT IS A LIVE STRUCTURE.

This relationship between the two words was discovered as a path in the net between the nodes that represented the words. Although this was a primitive semantic net scheme, many fundamental issues were first raised by Quillian's system.

One important point was that paths in the semantic net did not necessarily correspond to input sentences. Instead, the discovery of paths between two nodes amounted to making *inferences* on the knowledge in memory. For example, another relationship the system found between plant and live was:

> PLANT IS STRUCTURE WHICH GET-FOOD FROM AIR. THIS FOOD IS THING WHICH BEING HAS-TO TAKE INTO ITSELF TO KEEP LIVE.

In order to have found this connection, the system had to discover a connection between PLANT and LIVE, by way of FOOD, that was not directly input.

Although Quillian's semantic net system was limited, it strongly influenced much of the later work in NL and the representation of knowledge in AI (see Article Representation.C2). This influence reflected Quillian's stress on the importance of the semantic versus the surface components of language:

> As a theory, the program implies that a person first has something to say, expressed somehow in his own conceptual terms (which is what a "path" is to the program), and that all his decisions about the

> syntactic form that a generated sentence is to take are then made *in the service of* this intention. (Quillian, 1968, p. 255)

This is a strong statement about language, and this view, of a cognitive process manipulating an internal representation, is perhaps the essence of the AI perspective.

Terry Winograd's *blocks world* program, SHRDLU (1972), contained several text-generation devices. Their function was to enable the system, which is described in Article F4, to answer questions about the state of its table-top domain and certain of the system's internal states.

The basic text-generation techniques used were "fill-in-the-blank" and stored response patterns. For example, if an unfamiliar word was used, SHRDLU responded "I don't know the word . . . ". More complex responses were called for by questions asking why or how an action had been done. For "why", the system answered with "because <event>" or "in order to <event>," where <event> referred to a goal that the program had had when the action was taken. For example, "Why did you clear off that cube?" might be answered by "To put it on a large green cube." The program retrieved the appropriate event from its history list and then used a generation pattern associated with events of that type. For an event of the type "(PUTON OBJ1 OBJ2)," the pattern would be:

(<correct form of to put>, <noun phrase for OBJ1>, ON, <noun phrase for OBJ2>).

Noun phrases in the pattern were generated by associating an English word with every known object; adjectives and relative clauses were added until a unique object (within the domain of discourse) was described.

The stilted text generated by this scheme was moderated by the (heuristic) use of pronouns for noun phrases. For example, if the referent of a noun phrase had been mentioned in the same answer or in the previous one, an appropriate pronoun could be selected for it. SHRDLU's limited domain of discourse allowed it to exhibit surprisingly natural dialogue with such simple techniques.

Simmons and Slocum (1972) developed a natural language system that generated sentences from a *semantic network* representation of knowledge, based on a *case grammar* (see Article C4). The program produced surface structure from the network by means of an *augmented transition net*, adapted for the purpose of generation rather than parsing (see Article D2). The object of the work was to substantiate the claim that "the semantic network adequately represents some important aspect of the meaning of discourse"; if the claim was true, then "the very least requirement" was that "the nets be able to preserve enough information to allow regeneration of the sentences--and some of their syntactic paraphrases--from which the nets were derived" (p. 903).

An illustration of the capabilities of the system is given by the paragraph below, which was initially hand-coded into semantic network notation. (For a later version of the program in which the parsing was done automatically, see Simmons, 1973.)

> John saw Mary wrestling with a bottle at the liquor bar. He went over to help her with it. He drew the cork and they drank champagne together.

The network notation, in simplified form, is indicated by the following representation of "John saw Mary wrestling":

C1	TOKEN	(see)	C3	TOKEN	(wrestle)
	TIME	PAST		TIME	PROGRESSIVE PAST
	DATIVE	C2		AGENT	C4
	OBJECT	C3			
C2	TOKEN	(John)	C4	TOKEN	(Mary)
	NUMBER	SINGULAR		NUMBER	SINGULAR

Here C1, C2, C3, and C4 are nodes in the network representing concepts which are tokens of meanings of "see", "wrestle," "John", and "Mary". PAST and SINGULAR are also nodes. TOKEN, TIME, OBJECT, and the like are types of arcs, or relations.

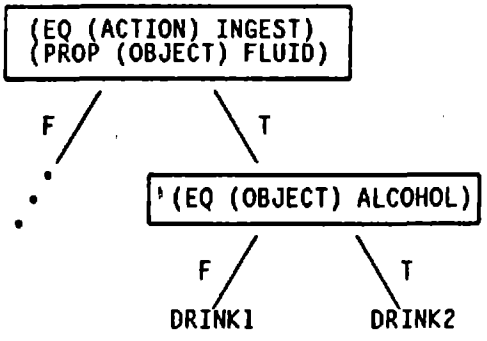
The representation shown was augmented by other relations, attached to verb nodes, such as MOOD (indicative or interrogative), VOICE (active or passive), and information about the relative times of events. Using this representation, the system was able to reconstruct several versions of the original paragraph. One read:

> John saw Mary wrestling with a bottle at the liquor bar. John went over to help her with it before he drew the cork. John and Mary together drank the champagne.

The actual generation was accomplished by an ATN in which the arcs were labelled with the names of relations that might occur in the semantic net. The actual path followed through the ATN--and thus the exact text generated--depended both on which relations were actually present and on which node or nodes were chosen as a starting point.

Wong (1975) has extended this approach, incorporating features to handle extended discourse.

Neil Goldman's (1975) program generates surface structure from a database of *conceptual dependency* networks, as the text-generation part of Roger Schank's MARGIE system, described in Article F5. The conceptual dependency (CD) knowledge representation scheme, discussed further in Article F6 on Schank's SAM system, is based on *semantic primitives* (Article Representation.C5) and is therefore language independent, so the actual word selection for output must be performed by Goldman's text-generation subsystem, called BABEL. This is accomplished by means of a *discrimination net* (a kind of binary decision tree--see Article Information Processing Psychology.C) that operates on a CD network that is to be verbalized. This discrimination net is used to select an appropriate *verb sense* to represent the event specified by the CD. (A verb sense is a meaning of the verb: DRINK, for example has two senses, to drink a fluid and to drink alcohol.) Essentially, there are only a small number of possible verbs that can represent the event, and a set of predicates determines which one to use. For instance, DRINK can be used to describe an INGEST event if the <object> has the property FLUID. The section of the discrimination net that handles DRINK might look like this:



Once a verb sense has been selected, an associated framework is used to generate a case-oriented syntax net, which is a structure similar to the semantic net of Simmons and Slocum. These frameworks include information concerning the form of the net and where in the conceptualization the necessary information is located. After the framework has been filled out, other language-specific functions operate on the syntax net to complete it syntactically with respect to such things as tense, form, mood, and voice. Finally, an ATN is used to generate the surface structure, as in the Simmons and Slocum program.

Yorick Wilks (1973) has developed a program that generates French from a semantic base of *templates* and *paraplates*. This is part of a complete *machine translation* system described in Article F2.

Discussion

The key point is that, as the richness and completeness of the underlying semantic representation of the information has increased, the quality of the resulting paraphrase has improved. Like other areas of AI, the basic problem is to determine exactly what the salient points are and to obtain a good representation of them; progress in generation seems to be closely tied to progress in knowledge representation. Future work in generation will also have to address areas such as *extended discourse*, *stylistics*, etc. In this direction, Clippinger (1975) has looked at psychological mechanisms underlying discourse production, and Perrault, Allen, & Cohen (1978) have studied the planning of speech acts for communication in context.

References

See Clippinger (1975), Friedman (1969), Friedman (1971), Goldman (1975), Klein & Simmons (1963), Klein (1965), Perrault, Allen, & Cohen (1978), Quillian (1968), Quillian (1969), Simmons & Slocum (1972), Simmons (1973), Wilks (1973), Winograd (1972), Wong (1975), and Yngve (1962).

F. Natural Language Processing Systems

F1. Early Natural Language Systems

Early work on machine processing of natural language assumed that the syntactic information in the sentence, along with the meaning of a finite set of words, was sufficient to perform certain language tasks--in particular, answering questions posed in English. Several of these early natural language programs are reviewed here: their techniques, their successes, and their shortcomings. These programs were restricted to dialogues about limited-knowledge domains in simple English and ignored most of the hard grammatical problems in the complex constructions found in unrestricted English. Through work with programs of this genre, it became apparent that people constantly use extensive world-knowledge in processing language and that a computer could not hope to be competent without "understanding" language. These programs bridge the gap between the early *mechanical translation* attempts of the 1950s and current, semantics-based natural language systems (see the Overview Article, Article B, and the Articles on recent NL systems in this section).

SAD-SAM

SAD-SAM (Syntactic Appraiser & Diagrammer - Semantic Analyzing Machine) was programmed by Robert Lindsay (1963a) at Carnegie Institute of Technology in the IPL-V list-processing language (see Article AI Languages.A). The program accepts English sentences about kinship relationships, builds a database, and answers questions about the facts it has stored.

It accepts a vocabulary of Basic English (about 1,700 words) and follows a simple *context-free grammar*. The SAD module parses the input from left to right, builds a syntactic tree structure, and passes this structure on to SAM, which extracts the semantically relevant (kinship-related) information to build the family trees and find answers to questions.

Though the subset of English processed by SAD is quite impressive in volume and complexity of structure, only kinship relations are considered by SAM; all other semantic information is ignored. SAM does not depend on the order of the input for building the family trees; if a first input assigns offspring B and C to X, and offspring D and E to Y, two "family units" will be constructed, but they will be collapsed into one if we learn later that E and C are siblings. (Multiple marriages are illegal.) However, SAM cannot handle certain ambiguities; the sentence "Joe plays in his Aunt Jane's yard" indicates that Jane is either the sister or sister-in-law of Joe's father, but SAM assigns one and only one connection at a time and therefore cannot use the ambiguous information: The structure of the model permits storing definite links but not possible *inferences*.

BASEBALL

Also in the early 1960s, Bert Green and his colleagues at Lincoln Labs wrote a program called BASEBALL (Green et al., 1963), again using the IPL-V programming language. BASEBALL

is essentially an *information retrieval* program, since its database of facts about all of the American League games during one year is not modified by the program. Acceptable input questions from the user must have only one clause, no logical connectives (and, or, not), no comparatives (highest, most), and no facts about sequences of events; and most words must be recognized by the (extensive) dictionary.

The parsing system uses 14 categories of parts of speech and *right-to-left* scanning to structure the input question into functional phrases. Using this structure and the *key-words* found in the question, the input is transformed into a specification list that is the *canonical expression* for the meaning of the question. For example, the question "How many games did the Yankees play in July?" becomes:

```
TEAM = YANKEES
MONTH = JULY
GAMES (number of) = ?
```

The answer is found by searching the database for data items matching the specification list, storing them on a "found" list, and eventually processing and outputting them.

The size of the dictionary and the *heuristics* used for resolving syntactic ambiguities (score can be a verb or a noun; Boston, a place or the name of a team) make BASEBALL able to answer most reasonable English questions about the games stored in the database. However, BASEBALL does not return to the user for clarification of semantic ambiguities. It simply fails to parse the sentence and asks for a rephrasing of the question. Though the database is not modified by the program, BASEBALL exhibits deductive abilities in parsing questions and processing the "found-list."

SIR

SIR (Semantic Information Retrieval) was written in LISP by Bertram Raphael (1968) as part of his thesis research at MIT. It is a prototype "understanding" machine since it can accumulate facts and then make deductions about them in order to answer questions.

SIR accepts a restricted subset of English. It matches sentences against 24 matching patterns of the following sort:

```
* is *
* is part of *
Is * * ?
How many * does * have ?
What is the * of * ?
```

where the elements matched against * are nouns, possibly modified by a "quantifier" such as: a, the, every, each, or a number. The match with a rule triggers the appropriate action in the program. A sample conversation might look like this:

```
Every boy is a person
I UNDERSTAND

A finger is part of a hand
```

```
I UNDERSTAND

There are two hands on each person
I UNDERSTAND

How many fingers does John have ?
THE ABOVE SENTENCE IS AMBIGUOUS BUT I ASSUME (has) MEANS
     (has as parts)
I DON'T KNOW WHETHER FINGER IS PART OF JOHN

John is a boy
I UNDERSTAND

How many fingers does John have ?
THE ABOVE SENTENCE IS AMBIGUOUS BUT I ASSUME (has) MEANS
     (has as parts)
HOW MANY FINGERS PER HAND ?

Every hand has 5 fingers
THE ABOVE SENTENCE IS AMBIGUOUS BUT I ASSUME (has) MEANS
     (has as parts)
THE ANSWER IS 10

Who is President of the United States ?
STATEMENT FORM NOT RECOGNIZED
```

SIR's world is concerned with only a small group of relations between objects, as specified by the 25 matching patterns. Declarative input statements are used to build the model, and the patterns have been chosen to express concepts that commonly occur in human conversation: set inclusion, spatial relationship, etc. The program has certain built-in knowledge (like transitivity of set inclusion) that enables it to answer questions about some facts not explicitly stated during input. SIR can also interact with the user to gather more information or to resolve ambiguities.

STUDENT

STUDENT is another pattern-matching natural language program, written by Daniel Bobrow (1968) as his doctoral research project at MIT. STUDENT is able to read and solve high-school-level algebra story problems like the following:

> If the number of customers Tom gets is twice the square of 20 per cent of the number of advertisements he runs, and the number of advertisements he runs is 46, what is the number of customers Tom gets?

The entire subset of English recognized by STUDENT is derived from the following set of basic patterns:

```
(WHAT ARE * AND *)                          (FIND * AND *)
(WHAT IS *)                                 (* IS MULTIPLIED BY *)
(HOW MANY *1 IS *)                          (* IS DIVIDED BY *)
(HOW MANY * DO * HAVE)                      (* IS *)
(HOW MANY * DOES * HAVE)                    (* (*1/VERB) *1 *)
(FIND *)
(* (*1/VERB) * AS MANY * AS * (*1/VERB) *)
```

A * sign indicates a string of words of any length, *1 indicates one word, and (*1/VERB) means the matching element must be recognized as a verb by the dictionary.

To construct the algebraic equations that will lead to the solution, the problem statement is scanned, first for linguistic forms associated with the equality relation (such as [* IS *]), then for algebraic operators. STUDENT then builds a list of the answers required, the units involved in the problem, and a list of all the variables in the equations. Then STUDENT invokes the SOLVE module with the set of equations and the desired unknowns.

If SOLVE fails, STUDENT applies *heuristics* such as: expanding idioms, identifying two previously "slightly different" variables, or invoking the REMEMBER module that contains special facts like:

```
(FEET IS THE PLURAL OF FOOT)
(ONE HALF ALWAYS MEANS .5)
(SUCCESSFUL CANDIDATES SOMETIMES MEANS STUDENTS WHO
PASSED THE ADMISSTIONS TEST)
(DISTANCE EQUALS SPEED TIMES TIME)
(ONE FOOT EQUALS 12 INCHES)
```

If all else fails, STUDENT can request more information from the user.

Each time a heuristic succeeds in finding a new equation, the augmented set is sent to SOLVE. STUDENT finally prints the solution (using the pattern "variable IS value") or reports its inability to solve. In the example, STUDENT would print

(THE NUMBER OF CUSTOMERS TOM GETS IS 162) .

STUDENT's simple pattern-matching scheme, combined with its small set of well-chosen heuristics, does an impressive job of parsing typical high-school algebra word problems. The program can be tricked easily ("Bill has 3 apples, 2 pears, and one dog" would be parsed into two clauses); but as implemented, it solved the problems submitted as fast as humans could (in one case, several times faster than an MIT graduate student who was not using the special "age problem" heuristics). A slightly modified version was found to be a good model for *human problem solving*, in a study conducted by Paige & Simon (1966).

ELIZA

Perhaps the most famous of these "pattern-matching" natural language programs was ELIZA, written by Joseph Weizenbaum (1966) at MIT. This program assumes the role of a Rogerian or "non-directive" therapist in its dialogue with the user. Although ELIZA's dialogue

with the user appears surprisingly realistic, the program uses no more intelligent mechanisms than those described above. Rather, the nature of therapist-patient dialogue lends itself to simulation in this manner. A sample dialogue might look like this (with ELIZA in caps):

Men are all alike.
IN WHAT WAYS

They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.
I AM SORRY TO HEAR THAT YOU ARE DEPRESSED

It's true. I'm unhappy.
DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY

I need some help, that much seems certain.
WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP

Perhaps I could learn to get along with my mother.
TELL ME MORE ABOUT YOUR FAMILY
. . .
WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER

Bullies.
DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR
BOYFRIEND MADE YOU COME HERE
. . .

ELIZA is programmed in the SLIP language (Symmetric List Processor), developed by Weizenbaum (1963). An offline *script* composed of *keylists* specifies the nature of the input language by specifying for each keyword a set of patterns to look for and the possible transformations associated with each pattern. For example, assuming the keyword is me, a pattern used for matching might be

(0 YOU 0 ME)

and one of the transformation rules associated with this pattern might be

(WHAT MAKES YOU THINK I 3 YOU)

where 0 in the pattern matches any string of words, and 3 in the transformation rule designates the third element matched (in this case, everything between YOU and ME). The input is scanned from left to right for a keyword; a systematic *hashing* procedure very quickly eliminates words that cannot be keywords. ELIZA deletes the part of text preceding a punctuation delimiter if no keyword has been found; otherwise, the part following it (insuring thus that the transformation will be applied to one single phrase or sentence). If

several keywords are found, they are stored in turn in a "keystack" according to the rank of precedence associated with each of them; then the input is matched against each decomposition rule in turn. These patterns can be ordered in the keylist so that the more complex ones are tried first; for the keyword "I" the pattern

(0 I 0 DEPRESSED 0)

is hard to match, but if a match is achieved, the answer can be more spectacular than the transformations for the "general match" pattern

(0 I 0) .

When a match is found, ELIZA generates a response, using the reassembly rules for this decomposition rule in a cyclic manner. If no decomposition rule matches for a given keyword, the keystack is popped and the pattern-matching procedure is repeated for the new keyword. If the keystack is empty, a response like "Please go on," "I see," or "Very interesting" will always do.

Several other tricks--like substituting for keywords in its response, associating keywords with a class or situation (Mother implies family), and remembering these keyword affiliates over the course of the conversation--help enhance the illusion of intelligent dialogue.

Conclusions

None of these early natural language systems dealt with the syntax of language in any sophisticated way. In these early programs, the semantic knowledge needed to respond to the user was implicit in the patterns and the ad hoc rules used for parsing. Modern natural language programs maintain large databases of explicit world-knowledge that they use to assist in parsing the sentence as well as in interpreting it.

References

For general reference, see Boden (1977), for lucid discussions of several of these systems; also, Simmons (1965), Simmons (1970), and Winograd (1974). The collections in Feigenbaum & Feldman (1963) and Minsky (1968) contain much of the original material.

F2. Wilks's Mechanical Translation System

Current work in machine translation of languages is exemplified by Wilks's system (1973), which can produce good French from small English paragraphs. The system is entirely semantics based; that is, no use is made of conventional linguistic syntax in either the analysis or the generation stages. The input English text is first converted to a semantic representation and then converted to the final translated text. (The use of an intermediate representation bears some similarity to the Weaver's idea of *interlingua*, discussed in Article B.) Wilks stresses that his semantic representation is designed for mechanical translation and may not be appropriate for other NL tasks like question answering. The rationale for this is that an explicit representation of the logical implications of a sentence, which is necessary for some tasks, may not be necessary for translation: If the two languages are similar, an appropriate target sentence with the *same* implications can often be found in a more straightforward way.

Wilks's system first fragments the input text into substrings of words; it then matches the fragments against a set of standard *templates*, that is, deep semantic forms that try to pick out the meaning conveyed by the input-text fragments. The output of this stage is a first approximation to a semantic representation of each of these fragments. The system then tries to tie together these representations to produce a more densely connected representation for the complete text. When this process has been completed, the generation of the output text is accomplished by unwinding the interlingual representation using functions that interpret it in the target language.

The interlingual representation is based on *semantic primitives* (see Article Representation.C5) that Wilks calls *elements*. Elements express the entities, states, qualities, and actions about which humans communicate. In the system as reported in Wilks (1973), there were 60 of these elements, which fall into 6 classes, as shown in the following examples.

1. Entities:	MAN (human being), PART (parts of things), STUFF (substances).
2. Cases:	TO (direction), IN (containment).
3. Sorts:	CONT (being a container), THRU (being an aperture).
4. Type indicators:	KIND (being a quality), HOW (being a type of action).
5. Actions:	CAUSE (causes to happen), BE (exists), FLOW (moving as liquids do).

The elements are used to build up "formulas," which each represent one sense of a word. The verb drink, for example, is represented by the following formula:

```
((*ANI SUBJ)
 (((FLOW STUFF) OBJE)
  ((*ANI IN) (((THIS (*ANI (THRU PART))) TO) (BE CAUSE))))) .
```

Drink is thus an action, (BE CAUSE), done by animate subjects, (*ANI SUBJ), to liquids, ((FLOW STUFF) OBJE). It causes the liquid to be in the animate object, (*ANI IN), via a particular aperture of the animate object, ((THIS (*ANI (THRU PART))) TO).

Formulas are understood as expressing preferences rather than absolute requirements. In the formula for drink, for example, it is only a preference that the agent be animate and the object liquid; the system could accept a sentence about cars that drink gasoline. The function of preferences, nevertheless, is to help determine the correct word-senses in the input text. In "John drank a whole pitcher," the preference for a liquid object would select the formula for pitcher as a container of liquid rather than the one for a baseball player.

The system's dictionary contains formulas for all the word-senses paired with *stereotypes* for producing the translated words in the target language.The following is an example of two stereotypes for the word advise (into French):

```
(ADVISE (CONSEILLER A (FN1 FOLK MAN))
        (CONSEILLER (FN2 ACT STATE STUFF)))
```

The two functions, FN1 and FN2, are used to distinguish the two possible constructions in French involving conseiller: *conseiller a* . . . and simply *conseiller* . . . . The first would be used in translating "I advise John to have patience"; the second, for "I advise patience." Functions like these in stereotypes are evaluated by the generation routines. Each function evaluates either to NIL, in which case the stereotype fails, or to words that will appear in the output text. The stereotypes serve the purpose of a *text generation* grammar, providing complex context-sensitive rules where required, without search of a large store of such rules. This is an example of *procedural representation of knowledge* (see Article Representation.C4).

Analysis of an English sentence by the system proceeds in several stages. First the text is separated into fragments, where the fragment boundaries are determined by punctuation marks, conjunctions, prepositions, and so on.

For each word in the fragment, the dictionary may contain several word-sense formulas; therefore one of many possible sequences of formulas must be selected to represent the fragment. For this purpose, the formula sequences are matched against a built-in list of *templates*, which are networks of formulas based on a basic actor-action-object triple called a *bare template*. Examples of such triples are MAN CAUSE THING and MAN DO THING. Special forms of templates are available to match fragments like prepositional phrases. It is assumed that it is possible to build up a finite inventory of bare templates that would be adequate for the analysis of ordinary language. The inventory for the system has been determined empirically and is easily modified.

At the initial stage of template matching, some senses of the words in the fragment can be rejected for failure to match any bare template, but more than one candidate template may remain. For example, if the fragment is "the policeman interrogated the crook," there will still be two possible templates, MAN FORCE MAN and MAN FORCE THING, which take "crook" to be a person and a shepherd's staff, respectively.

At the next stage of the analysis, called expansion, a more detailed matching algorithm is used. The principle is that the template representation chosen for a fragment is the one in which the most preferences are satisfied. In the example, the preference of "interrogate" for an object representing a human being is decisive. The result of this stage is a full template (a network of formulas) for each fragment, in which semantic dependencies among the formulas have been noted. The overall goal of *semantic density*--that is, of maximizing the interdependence of formulas--is one of the key ideas in Wilks's work and produces a good solution to many problems of ambiguity.

In the succeeding stage of analysis, the templates for individual fragments are tied together with higher level dependencies, expressed in terms of *paraplates*, or patterns that span two templates. The use of paraplates is to resolve prepositional or *case* ambiguities (see Article C4). For example, the fragments "he ran the mile" and "in four minutes" would be tied together by a paraplate for the TIMELOCATION case; had the second fragment been "in a plastic bag," a CONTAINMENT case paraplate would have matched instead. A similar technique is used to resolve simple problems of pronoun reference, as in "I bought the wine, sat on a rock, and drank it." In both cases, the chief preference of the system is for semantic density.

Finally, the system uses some commonsense *inference* rules to deal with situations in which more explicit world-knowledge is required to resolve pronoun references than formulas, templates, and paraplates provide. At the completion of this analysis, the input text has been replaced by an interlingual representation with suitable markers, and other information is used by the generation routines in a relatively straightforward manner to produce the final output text.

References

This description of Wilks's work is based primarily on Wilks (1973). Other descriptions include Wilks (1975a), Wilks (1975b), Wilks (1975c), Wilks (1977b), and Wilks (1978).

Also of interest: Charniak & Wilks (1976) and Schank (1975).

F3. LUNAR

LUNAR is an experimental, natural language *information retrieval* system designed by William Woods at BBN (Woods, 1973b; Woods, Kaplan, & Nash-Webber, 1972) to allow geologists to access, compare, and evaluate chemical-analysis data on moon rock and soil composition obtained from the Apollo 11 mission (see Article Applications.F4 for a discussion of AI information retreival systems). The primary goal of the designers was research on the problems involved in building a man-machine interface that would allow communication in ordinary English. A "real-world" application was chosen for two reasons: First, it tends to focus effort on the problems really in need of solution (sometimes this is implicitly avoided in "toy" problems); second, the possibility of producing a system capable of performing a worthwhile task lends some additional impetus to the work.

LUNAR operates by translating a question entered in English into an expression in a formal *query language* (Codd, 1974). The translation is done using an *augmented transition network* parser coupled with a rule-driven semantic interpretation procedure, which is used to guide the analysis of the question. The "query" that results from this analysis is then applied to the database to produce the answer to the request. The query language is a generalization of the *predicate calculus* (Article Representation.C1). Its central feature is a quantifier function that is able to express, in a simple manner, the restrictions placed on a database-retrieval request by the user. This function is used in concert with special enumeration functions for classes of database objects, freeing the quantifier function from explicit dependence on the structure of the database. LUNAR also served as a foundation for the early work done on *speech understanding* at BBN (see Article Speech.B3).

Detailed Description

The following list of requests is indicative of the types of English constructions that can be handled by LUNAR (shown as they would actually be presented to the system):

1. (WHAT IS THE AVERAGE CONCENTRATION OF ALUMINUM IN HIGH ALKALI ROCKS?)
2. (WHAT SAMPLES CONTAIN P205?)
3. (GIVE ME THE MODAL ANALYSES OF P205 IN THOSE SAMPLES)
4. (GIVE ME EU DETERMINATIONS IN SAMPLES WHICH CONTAIN ILM)

LUNAR processes these requests in the following manner:

Syntactic analysis using an augmented transition network parser and *heuristic* information (including semantics) to produce the most likely *derivation tree* for the request;

Semantic interpretation to produce a representation of the meaning of the request in a formal query language; and

Execution of the query language expression on the database to produce the answer to the request.

LUNAR's language processor contains a grammar for a large subset of English, the

semantic rules for interpreting database requests, and a dictionary of approximately 3,500 words. As an indication of the capabilities of the processor, it is able to deal with tense and modality, some anaphoric references and comparatives, restrictive relative clauses, certain adjective modifiers (some of which alter the range of quantification or interpretation of a noun phrase), and embedded complement constructions. Some problems do arise in parsing conjunctive constructions and in resolving ambiguity in the scope of quantifiers. Emphasis has been placed on the types of English constructions actually used by geologists so that the system knows how they habitually refer to the objects in its database.

The Query Language

The formal query language contains three types of objects: "designators," which name classes of objects in the database (including functionally defined objects); "propositions," which are formed from predicates with designators as arguments; and "commands," which initiate actions. Thus, if S10046 is a designator for a particular sample, OLIV is a designator for the mineral olivine, CONTAIN is a predicate, and TEST is a truth-value testing command, then "(TEST (CONTAIN S10046 OLIV))" is a sample expression in the query language. The primary function in the language is the quantifier function FOR, which is used in expressions of the following type:

(FOR QUANT X / CLASS : PX ; QX)

where QUANT is a quantifier like each or every, or a numerical or comparative quantifier; X is a variable of quantification; CLASS determines the class of objects over which the quantification is to range; PX specifies a restriction on the range; and QX is the proposition or command being quantified. FOR is used with enumeration functions that can access the database. Thus, FOR itself is independent of the database structure. As an example (taken from Woods, 1973b), if SEQ is an enumeration function used to enumerate a precomputed list, and if PRINTOUT is a command that prints a representation for the designator given as its argument, then

(FOR EVERY X1 / (SEQ TYPECS) : T ; (PRINTOUT X1))

prints the sample numbers for all type-C samples. In this case there is no restriction on the range of quantification in that PX = T, the universally true proposition.

A fuller example of the operation of LUNAR (simplified slightly from the same source) is shown below.

Request:
(DO ANY SAMPLES HAVE GREATER THAN 13 PERCENT ALUMINUM)

Query Language Translation (after parsing):

(TEST (FOR SOME X1 / (SEQ SAMPLES) : T ; (CONTAIN X1
(NPR* X2 / 'AL203) (GREATERTHAN 13 PCT))))

Response:
YES

LUNAR is perhaps the best operational example of a finely tuned ATN parsing system applied to a real-world problem. Since the system has limited performance goals (i.e., facilitating database inquiry as opposed to holding an interesting conversation), many of the complications inherent in language understanding are avoided.

References

See Codd (1974), Woods (1973b), Woods & Kaplan (1971), and Woods, Kaplan, & Nash-Webber (1972).

F4. SHRDLU

SHRDLU was written by Terry Winograd (1972) as his doctoral research at MIT. It was written in LISP and MICRO-PLANNER, a LISP-based programming language (see Article AI Languages.C2). The design of the system is based on the belief that to understand language, a program must deal in an integrated way with syntax, semantics, and reasoning. The basic viewpoint guiding its implementation is that meanings (of words, phrases, and sentences) can be embodied in procedural structures and that language is a way of activating appropriate procedures within the hearer. Thus, instead of representing knowledge about syntax and meaning as rules in a grammar or as patterns to be matched against the input, Winograd embodied the knowledge in SHRDLU in pieces of executable computer code. For example, the context-free rule saying that a sentence is composed of a noun phrase and a verb phrase,

S -> NP VP

is embodied in the MICRO-PLANNER procedure:

```
(PDEFINE SENTENCE
         (((PARSE NP) NIL FAIL)
          ((PARSE VP) FAIL FAIL RETURN)))  .
```

When called, this program, called SENTENCE, uses independent procedures for parsing a noun phrase followed by a verb phrase. These, in turn, can call other procedures. The process FAILs if the required constituents are not found. With such special *procedural representations* for syntactic, semantic, and reasoning knowledge, SHRDLU was able to achieve unprecedented performance levels in dialogues simulating a *blocks world* robot.

SHRDLU operates within a small "toy" domain so that it can have an extensive model of the structures and processes allowed in the domain. The program simulates the operation of a robot arm that manipulates toy blocks on a table. The system maintains an interactive dialogue with the user: it can accept statements and commands as well as answer questions about the state of its world and the reasons for its actions. The implemented system consists of four basic elements: a parser, a recognition grammar for English, programs for semantic analysis (to change a sentence into a sequence of commands to the robot or into a query of the database), and a problem solver (which knows about how to accomplish tasks in the blocks world).

Each procedure can make any checks on the sentence being parsed, perform any actions, or call on other procedures that may be required to accomplish its goal. For example, the VERB PHRASE procedure called above contains calls to functions that establish verb-subject agreement by searching through the entire derivation tree for other constituents while still being in the middle of parsing the VP. SHRDLU's knowledge base includes a detailed model of the blocks world it manipulates, as well as a simple model of its own reasoning processes, so that it can explain its actions.

Reasoning in SHRDLU

SHRDLU's model of the world and reasoning about it are done in the MICRO-PLANNER

programming language, which facilitates the representation of problem-solving procedures, allowing the user to specify his own heuristics and strategies for a particular domain. Knowledge about the state of the world is translated into MICRO-PLANNER assertions, and manipulative and reasoning knowledge is embodied in MICRO-PLANNER programs. For example, the input sentence "The pyramid is on the table" might be translated into an assertion of the form:

```
(ON PYRAMID TABLE)
```

SHRDLU's problem solver consists of a group of "theorems" about the robot's environment and actions, represented as MICRO-PLANNER procedures. In operation, the theorem prover manipulates the state of the domain by running MICRO-PLANNER programs that perform the actions requested by the user.

The philosophy and implementation of PLANNER are described in the AI Programming Languages section of the Handbook, but a brief discussion here will illustrate its use in SHRDLU. The main idea of PLANNER is to solve problems using specific procedures built into the problem statements themselves, as well as using general problem-solving rules. The advantage of using these problem-specific rules or *heuristics* is that they can radically increase the efficiency of the process. Furthermore, the problem statements are programs and thus can carry out actions in the problem-solving process. Thus, to put one block on another, there might be a MICRO-PLANNER program of the form:

```
(THGOAL (ON ?X ?Y)
        (OR (ON-TOP ?X ?Y)
            (AND (CLEAR-TOP ?X)
                 (CLEAR-TOP ?Y)
                 (PUT-ON ?X ?Y))))
```

This means that, if X is not already on Y, that state can be achieved by clearing off everything that is stacked on top of X (so that the robot can move X), clearing off Y (so that X can be placed on top of Y) and then putting X on Y. The procedure resembles a *predicate calculus* theorem, but there are important differences. The PLANNER procedure is a program, and its operators carry out actions. The THGOAL procedure finds an assertion in the database or proves it using other procedures. AND and OR are logical connectives. The crucial element is that though PLANNER may end up doing a proof, it does so only after checking some conditions that may make the proof trivial, or impossible, and it only performs the proof on relevant arguments, rather than checking all entities in the database as a blind theorem prover might. Moreover, no sharp distinction is drawn between proof by showing that a desired assertion is already true and proof by finding a sequence of actions (manipulating blocks) that will make the assertion true. In addition to the article on PLANNER (AI Languages.C2), the reader is referred to the Knowledge Representation section for a general discussion of these issues.

Grammar, Syntax, and Semantics

SHRDLU's grammar is based on the notion of *systemic grammar*, a system of choice networks that specifies the features of a syntactic unit, how the unit functions, and how it influences other units, discussed in Article C3. Thus, a systemic grammar contains not only the constituent elements of a syntactic group but also higher level features such as mood, tense, and voice.

In order to facilitate the analysis, the parsing process looks for syntactic units that play a major role in meaning, and the semantic programs are organized into groups of procedures that are applicable to a certain type of syntactic unit. In addition, the database definitions contain *semantic markers* that can be used by the syntactic programs to rule out grammatical but semantically incorrect sentences such as "The table picks up blocks." These markers are calls to semantic procedures that check for restrictions, such as that only animate objects pick up things. These semantic programs can also examine the context of discourse to clarify meanings, establish pronoun referents, and initiate other semantically guided parsing functions.

Parsing

To write SHRDLU's parser, Winograd first wrote a programming language, embedded in LISP, which he called PROGRAMMAR. PROGRAMMAR supplies primitive functions for building systemically described syntactic structures. The theory behind PROGRAMMAR is that basic programming methods, such as procedures, iteration, and recursion, are also basic to the cognitive process. Thus, a grammar can be implemented in PROGRAMMAR without additional programming paraphernalia; special syntactic items (such as conjunctions) are dealt with through calls to special procedures. PROGRAMMAR operates basically in a top-down, left-to-right fashion but uses neither a *parallel processing* nor *backtracking* strategy in dealing with multiple alternatives (see Article D1). PROGRAMMAR finds one parsing rather directly, since decisions at choice-points are guided by the semantic procedures. By functionally integrating its knowledge of syntax and semantics, SHRDLU can avoid trying all choices in an ambiguous situation. If the choice made does fail, PROGRAMMAR has primitives for returning to the choice-point with the reasons for the failure and informing the parser of the next best choice based on these reasons. This "directed backup" is far different from PLANNER's automatic backtracking in that the design philosophy of the parser is oriented toward making an original correct choice rather than establishing exhaustive backtracking.

The key to the system's successful operation is the interaction of PLANNER reasoning procedures, semantic analysis, and PROGRAMMAR. All three of these elements examine the input and help direct the parsing process. By making use of this *multiple-source* knowledge and programmed-in "hints" (heuristics), SHRDLU successfully dealt with language issues such as pronouns and referents. The reader is referred to Winograd's **Understanding Natural Language** (1972), pages 8-15, for an illustrative sample dialogue with SHRDLU.

Discussion

SHRDLU was a significant step forward in natural language processing research because of its attempts to combine models of human linguistic and reasoning methods in the language understanding process. Before SHRDLU, most AI language programs were linguistically simple; they used keyword and pattern-oriented grammars. Furthermore, even the more powerful grammar models used by linguists made little use of inference methods and semantic knowledge in the analysis of sentence structure. A union of these two techniques gives SHRDLU impressive results and makes it a more viable theoretical model of human language processing.

SHRDLU does have its problems, however. Like most existing natural language

systems, SHRDLU lacks the ability to handle many of the more complex features of English. Some of the problem areas are agreement, dealing with hypotheses, and handling words such as the and and.

Wilks (1974) has argued that SHRDLU's power does not come from linguistic analysis but from the use of problem-solving methods in a simple, logical, and closed domain (blocks world), thus eliminating the need to face some of the more difficult language issues. It seems doubtful that if SHRDLU were extended to a larger domain, it would be able to deal with these problems. Further, the level at which SHRDLU seeks to simulate the intermixing of knowledge sources typical of human reasoning is embedded in its processes rather than made explicit in its control structure, where it would be most powerful. Lastly, its problem solving is still highly oriented to predicate calculus and limited in its use of inferential and heuristic data (Winograd, 1974, pp. 46-48).

References

Winograd (1972) is the principal reference on SHRDLU. The original version of the thesis is Winograd (1971). A convenient summary is given in Winograd (1973). Boden (1977) also presents a clear and concise discussion of the system.

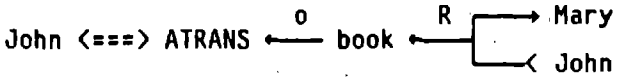
Also of interest are Sussman, Winograd, & Charniak (1970), the MICRO-PLANNER manual; Wilks (1974); Winograd (1974); and Winograd (forthcoming).

F5. MARGIE

MARGIE (Meaning Analysis, Response Generation, and Inference on English) was a program developed by Roger Schank and his students at the Stanford AI Lab (Schank, 1975). Its intent was to provide an intuitive model of the process of natural language understanding. More recent work by Schank and his colleagues at Yale on *story understanding* and *conceptual dependency* theory are described in Article F6 on their SAM and PAM systems.

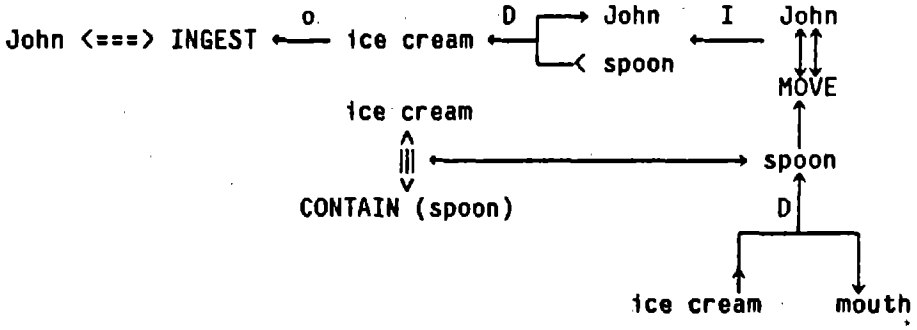
Conceptual Dependency Theory

The central feature of the MARGIE system was the use of a knowledge representation scheme called Conceptual Dependency. Conceptual dependency is intended to represent meaning in a sufficiently deep manner so that all ambiguity is eliminated. Every sentence maps into a canonical form, and any two sentences with the same "meaning" will have the same representation. This goal was approached by designing a graph-structure formalism based on a set of *primitive* concepts. There are 6 basic types of concepts: things, actions, attributes of things, attributes of actions, times, and locations (the first four correspond roughly to nouns, verbs, adjectives, and adverbs). Relations among concepts are called *dependencies*, and there are 15 types of these. Among them are *case* relationships such as those between an act and its object, its direction, or its recipient and donor (see Article C4 on case grammars). Graphically, each type of dependency is denoted with a special arrow symbol (link), and each concept is denoted by a word representing it. For example, "John gives Mary a book" would be expressed as:



where John, book, and Mary are concept nodes. Also, the concept node ATRANS (abstract transfer--i.e., transfer of possession) is one of a small set of primitive verbs (about twelve) from which all actions must be built up. Other primitives include PTRANS (physical transfer--i.e., movement) and PROPEL (apply a force). The complicated, three-pointed arrow labeled R indicates a recipient-donor dependency between Mary and John and the book, since Mary got the book from John. The arrow labeled o indicates an "objective" dependency; that is, the book is the object of the ATRANS, since it is the thing being given. Dependency links may link concepts or other conceptual dependency networks.

Another example, "John eats the ice cream with a spoon," would be represented as:



where the D and I arrows indicate DIRECTION and INSTRUMENT, respectively. Notice that in this example, "mouth" has entered the diagram as part of the conceptualization, even though it was not in the original sentence. This is part of the fundamental difference between conceptual dependency networks and the syntactic tree that a grammar may produce in parsing a sentence. John's mouth as the recipient of the ice cream is inherent in the "meaning" of the sentence, whether it is expressed or not. In fact, the diagram can never be finished, because we could add such details as "John INGESTed the ice cream by TRANSing the ice cream on a spoon to his mouth, by TRANSing the spoon to the ice cream, by GRASPing the spoon, by MOVing his hand to the spoon, by MOVing his hand muscles," and so on. Such an analysis is known to both the speaker and the hearer of the sentence and normally would not need to be expanded. (However, if we were actually designing a robot to perform such an action, we would want access to a more detailed network that would represent the robot's procedural knowledge about eating.)

For some tasks, like *paraphrasing* and *question answering*, this style of representation has a number of advantages over more surface-oriented systems. In particular, sentences like

Shakespeare wrote Hamlet

and

The author of Hamlet was Shakespeare ,

which in some sense have the same meaning, map into the same deep structure. They can thus be seen to be paraphrases of each other. Another important aspect of conceptual dependency theory is its independence from syntax; in contrast with earlier work in the paradigms of *transformational grammar* or *phrase-structure grammar*, a "parse" of a sentence in conceptual dependency bears little relation to the syntactic structure. Schank (1975) also claims that conceptual dependency has a certain amount of psychological validity, in that it reflects intuitive notions of human cognition.

MARGIE

The MARGIE system, programmed in LISP 1.6, was divided into three components. The first, written by Chris Riesbeck, was a *conceptual analyzer*, which took English sentences and converted them into an internal conceptual dependency representation. This was done through a system of "requests," which were similar to *demons* or *production systems*. A request is essentially a piece of code that looks for some surface linguistic construct and takes a specific action if it is found. It consists of a "test condition," to be searched for in the input, and an "action," to be executed if the test is successful. The test might be as specific as a particular word or as general as an entire conceptualization. The action might contain information about: (a) what to look for next in the input, (b) what to do with the input just found, and (c) how to organize the representation. The flexibility of this formalism allows the system to function without depending heavily on syntax, although it is otherwise quite similar to the tests and actions that make *ATN*s such a powerful parsing mechanism.

The middle phase of the system, written by Chuck Rieger, was an inferencer designed to accept a proposition (stated in conceptual dependency) and deduce a large number of facts from the proposition in the current context of the system's memory. The motivation for this component was the assumption that humans "understand" far more from a sentence than

is actually stated. Sixteen types of inferences were identified, including "cause," "effect," "specification," and "function." The inference knowledge was represented in memory in a modified *semantic net*. Inferences were organized into "molecules," for the purpose of applying them. An example of this process might be:

John hit Mary.

from which the system might infer (among many other things):

John was angry with Mary.
Mary might hit John back.
Mary might get hurt.

The module does relatively unrestricted forward inferencing, which tended to produce large numbers of inferences for any given input.

The last part of the system was a *text generation* module written by Neil Goldman. This took an internal conceptual dependency representation and converted it into English-like output, in a two-part process:

1. A *discrimination net* was used to distinguish between different word-senses. This permitted the system to use English-specific contextual criteria for selecting words (especially verbs) to "name" conceptual patterns.

2. An ATN was used to linearize the conceptual dependency representation into a surface-like structure.

The text generation module is also discussed in Article E.

MARGIE ran in two modes: *inference* mode and *paraphrase* mode. In inference mode, it would accept a sentence and attempt to make inferences from that sentence, as described above. In paraphrase mode, it would attempt to restate the sentence in as many equivalent ways as possible. For example, given the input

John killed Mary by choking her.

it might produce the paraphrases

John strangled Mary.
John choked Mary and she died because she was unable to breathe.

Discussion

MARGIE is not, and was not intended to be, a "finished" production-level system. Rather, the goal was to provide a foundation for further work in computational linguistics. Of particular interest in MARGIE was the use of conceptual dependency as an *interlingua*, a language-independent representation scheme for encoding the meaning of sentences. Once the sentence was processed, the surface structure was dropped and all further work was done with the conceptual dependency notation. This method has certain beneficial effects

on the control structure: All interprocess communication can be done through conceptual dependency, without the need to resort to the surface level, although the more subtle information in the surface structure may be lost. Since the intermediate representation is "language-free," it should facilitate translation of the original sentence into another language, as Weaver indicated in his original discussion of *Machinese* (see Article B). As mentioned above, the existence of a unique representation for any fact should also facilitate tasks like paraphrasing and question answering.

References

Conceptual dependency theory and all three parts of the MARGIE system are described in detail in Schank (1975). Since the version described in this article, the theory has evolved considerably, and several new systems have been built using the CD formalisms, all described very well in Schank & Abelson (1977). Other references for MARGIE include Schank (1973) and Schank et al. (1973).

F6. SAM and PAM

Story Understanding

SAM (Script Applier Mechanism) and PAM (Plan Applier Mechanism) are computer programs developed by Roger Schank, Robert Abelson and their students at Yale to demonstrate the use of *scripts* and *plans* in understanding simple stories (Schank et al., 1975; Schank & Abelson, 1977). Most work in natural language understanding prior to 1973 involved parsing individual sentences in isolation; it was thought that text composed of paragraphs could be understood simply as collections of sentences. But just as words are not formed from the unconstrained juxtaposition of morphemes, and sentences are not unconstrained collections of words, so paragraphs and stories are not without structure. The structures of stories have been analyzed (Propp, 1968; Rumelhart, 1975; Thorndyke, 1977), and it is clear that the context provided by these structures facilitates sentence comprehension, just as the context provided by sentence structure facilitates word comprehension (see the Overview; also, the Speech.A article discusses *top-down processing* in speech understanding research). For example, if we have been told in a story that John is very poor, we can expect later sentences to deal with the consequences of John's poverty, or steps he takes to alleviate it.

Different researchers have very different ideas about what constitutes the structure of a story. Some *story grammars* are rather "syntactic"; that is, they describe a story as a collection of parts like setting, characters, goal introduction, and plans, determined by their sequential position in the story rather than by their meaning. The work of Schank and Abelson reported here has a more semantic orientation. They propose an underlying representation of each phrase in a story which is based on a set of *semantic primitives*. This representation, called *conceptual dependency*, is the theoretical basis for more complex story structures such as *scripts*, *plans*, *goals*, and *themes*. The SAM and PAM programs understand stories using these higher level structures. (Article F5 describes the early work on conceptual dependency theory, and Articles Representation.C5 and Representation.C6 discuss related representation schemes.)

Parsing: A Brief Introduction to Conceptual Dependency

Prior to his work with stories, Schank (1973) developed conceptual dependency (CD) for representing the meaning of phrases or sentences. The "basic axiom" of conceptual dependency theory is:

> For any two sentences that are identical in meaning, regardless of language, there should be only one representation of that meaning in CD. (See Schank & Abelson, 1977, p. 11.)

Schank thus allies himself with the early machine translation concept of *interlingua*, or intermediate language (see Articles B and Overview), and has in fact done some *mechanical translation* research in conjunction with the story understanding project. A second important idea is:

> Any information in a sentence that is implicit must be made explicit in

> the representation of the meaning of that sentence. (Schank & Abelson, 1977, p. 11)

This idea is the basis for much of the sophisticated inferential ability of SAM and PAM: We shall see a sense in which the fact that "John ate food" is implicit in the sentence "John went to a restaurant," and how the former sentence can be inferred at the time that the program reads in the latter.

A third important idea is that conceptual dependency representations are made up of a very small number of semantic primitives, which include primitive acts and primitive states (with associated attribute values). Examples of *primitive acts* are:

ACTS:

PTRANS	The transfer of the physical location of an object. For one to "go" is to PTRANS oneself. "Putting" an object somewhere is to PTRANS it to that place.
PROPEL	The application of physical force to an object.
ATRANS	The transfer of an abstract relationship. To "give" is to ATRANS the relationship of possession or ownership.
MTRANS	The transfer of mental information between people or within a person. "Telling" is an MTRANS between people; "seeing" is an MTRANS within a person.
MBUILD	The construction of new information from old. "Imagining," "inferring," and "deciding" are MBUILDs.

In the most recent version of CD theory (1977), Schank and Abelson included 11 of these primitive acts.

Examples of *primitive states* include:

STATES:

Mary HEALTH(-10)	Mary is dead.
John MENTAL STATE(+10)	John is ecstatic.
Vase PHYSICAL STATE(-10)	The vase is broken.

The number of primitive states in conceptual dependency theory is much larger than the number of primitive actions. States and actions can be combined; for example, the sentence

John told Mary that Bill was happy

can be represented as

John MTRANS (Bill BE MENTAL-STATE(5)) to Mary.

An important class of sentences involves *causal chains*, and Schank and Abelson have worked out some rules about causality that apply to conceptual dependency theory. Five important rules are:

1. Actions can result in state changes.
2. States can enable actions.
3. States can disable actions.
4. States (or acts) can initiate mental events.
5. Mental events can be reasons for actions.

These are fundamental pieces of knowledge about the world, and conceptual dependency theory includes a shorthand representation of each (and combinations of some) called *causal links*.

Conceptual dependency representation is, in fact, the interlingua that is produced when SAM or PAM parses sentences. The parser which is used by these programs is an extension of the one developed by Chris Riesbeck (1975) for the MARGIE system (Article F5). As this program encounters words, it translates them into conceptual dependency representation; but, in addition, it makes predictions about what words and linguistic structures (verbs, prepositions, etc.) can be expected to occur and what conceptual dependency structures should be built in that eventuality.

Conceptual dependency is the underlying representation of the meaning of sentences upon which SAM and PAM operate. We turn now to higher level knowledge structures: scripts, plans, goals, and themes. Schank and Abelson make a distinction between scripts and plans that must be clear before the differences between SAM and PAM become apparent.

Scripts

A script is a standardized sequence of events that describes some stereotypical human activity, such as going to a restaurant. Schank and Abelson's assumption is that people know many such scripts and use them to establish the context of events. A script is functionally similar to a *frame* (Minsky, 1975) or a *schema* (Bartlett, 1932; Rumelhart, 1975), in the sense that it can be used to anticipate the events it represents. For example, the RESTAURANT script (see Figure 1) involves going to a restaurant, being seated, consulting the menu, and so on. People who are presented with an abbreviated description of this activity, e.g., the sentence "John went out to dinner," infer from their own knowledge about restaurants that John ordered, ate, and paid for food. Moreover, they anticipate from a sentence which fills part of the script ("John was given a menu") what sort of sentences are likely to follow, e.g., "John ordered the lamb." Scripts attempt to capture the kind of knowledge that people use to make these inferences. (Article Representation.C6 discusses scripts, frames and related representation schemes.)

```
-----------------------------------------------------------------

Players: customer, server, cashier

Props: restaurant, table, menu, food, check, payment, tip

Events:

        1. customer goes to restaurant
        2. customer goes to table
        3. server brings menu
        4. customer orders food
        5. server brings food
        6. customer eats food
        7. server brings check
        8. customer leaves tip for server
        9. customer gives payment to cashier
       10. customer leaves restaurant

Header: event 1

Main concept: event 6

-----------------------------------------------------------------
```

Figure 1. Restaurant Script

Two components of scripts are of special importance. We will discuss later how the *script header* is used by SAM to match scripts to parsed sentences. The second important component is the *main concept* or *goal* of the script. In the restaurant script the goal is to eat food.

The scripts used in SAM grew out of Abelson's (1973) notion of scripts as networks of causal connections. However, they do not depend on explicit causal connections between their events. In hearing or observing events that fit a standard script, one need not analyze the sequence of events in terms of causes, since they can be expected just from knowing that the script applies. The identification of events as filling their slots in the script gives us the intuition of "understanding what happened."

Scripts describe everyday events, but frequently these events (or our relating of them) do not run to completion. For example:

> I went to the restaurant. I had a hamburger.
> Then I bought some groceries.

This story presents several problems for a system like SAM that matches scripts to input sentences. One problem is that the restaurant script is "left dangling" by the introduction of the last sentence. It is not clear to the system whether the restaurant script (a) has terminated, and a new (grocery shopping) script has started; (b) has been distracted by a "fleeting" (one-sentence) grocery script; or (c) is interacting with a new grocery script

(e.g., buying groceries in the restaurant). Another thing that can happen to everyday scripts is that they can be thwarted, as in:

> I went to the gas station to fill up my car.
> But the owner said he was out of gas.

This is called an "obstacle".

Scripts describe rather specific events, and although it is assumed that adults know thousands of them, story comprehension cannot be simply a matter of finding a script to match a story. There are just too many possible stories. Moreover, there are clear cases where people comprehend a story even though it does not give enough information to cause a program to invoke a script, as in

> John needed money. He got a gun and went to a liquor store.

Schank and Abelson point out that even if the program had a script for Robbery, this story offers no basis for invoking it. Nonetheless, people understand John's goals and his intended actions.

> There must be relevant knowledge available to tie together sentences that otherwise have no obvious connection. . . . The problem is that there are a great many stories where the connection cannot be made by the techniques of causal chaining nor by reference to a script. Yet they are obviously connectable. Their connectability comes from these stories' implicit reference to plans. (Schank & Abelson, 1977, p. 75)

Plans

Schank and Abelson introduce *plans* as the means by which goals are accomplished, and they say that understanding *plan-based* stories involves discerning the goals of the actor and the methods by which the actor chooses to fulfill those goals. The distinction between script-based and plan-based stories is very simple: In a script-based story, parts or all of the story correspond to one or more scripts available to the story understander; in a plan-based story, the understander must discern the goals of the main actor and the actions that accomplish those goals. An understander might process the same story by matching it with a script or scripts, or by figuring out the plans that are represented in the story. The difference is that the first method is very specialized, because a script refers to a specific sequence of actions, while plans can be very general because the goals they accomplish are general. For example, in

> John wanted to go to a movie. He walked to the bus-stop.

we understand that John's *immediate* goal (called a *delta-goal* because it brings about a change necessary for accomplishment of the ultimate goal) is to get to the movie theater. *Going somewhere* is a very general goal and does not apply just to going to the movies. In Schank and Abelson's theory, this goal has associated with it a set of *planboxes*, which are standard ways of accomplishing the goal. Planboxes for *going somewhere* include riding an animal, taking public transportation, driving a car, etc.

Obviously, a story understander might have a "go to the movies" script in its repertoire, so that analysis of John's goals would be unnecessary--the system would just "recognize" the situation and retrieve the script. This script would be the standardized intersection of a number of more or less general goals and their associated planboxes. It would be a "routinized plan" made up of a set of general subplans: Go to somewhere (the theater), Purchase something (a ticket), Purchase something (some popcorn), etc.

> A routinized plan can become a script, at least from the planner's personal point of view.
>
> Thus, plans are where scripts come from. They compete for the same role in the understanding process, namely as explanations of sequences of actions that are intended to achieve a goal. (Schank & Abelson, 1977, p. 72)

The process of understanding plan-based stories involves determining the actor's goal, establishing the subgoals (delta- or D-goals) that will lead to the main goal, and matching the actor's actions with planboxes associated with the D-goals. For example, in

John was very thirsty. He hunted for a glass.

we recognize the D-goal of PTRANSing liquid, and the lower level goal (specified in the planbox for PTRANSing liquid) of finding a container to do it with.

Goals and Themes

In story comprehension, goals and subgoals may arise from a number of sources. For example, they may be stated explicitly, as in

John wanted to eat ;

they may be nested in a planbox; or they may arise from *themes*. For example, if a LOVE theme holds between John and Mary, it is reasonable to expect the implicit, mutual goal of protecting each other from harm: "Themes, in other words, contain the background information upon which we base our predictions that an individual will have a certain goal" (Schank & Abelson, 1977, p. 132).

Themes are rather like production systems in their situation-action nature. A theme specifies a set of actors, the situations they may be in, and the actions that will resolve the situation in a way consistent with the theme. The goals of a theme are to accomplish these actions. Schank and Abelson have proposed seven types of goals; we have already considered D-goals. Other examples are:

A- or Achievement-goals.	To desire wealth is to have an A-Money goal.
P- or Preservation-goal.	To protect someone may be a P-Health or P-Mental State goal.
C- or Crisis-goal.	A special case of P-goals, when action is immediately necessary.

The LOVE theme can be stated in terms of some of these goals:

X is the lover; Y is the loved one; Z is another person.

SITUATION		ACTION
Z cause Y harm		A-Health(Y) and possibly cause Z harm
	or	C-Health(Y)
not-Love(Y,X)		A-Love(Y,X)
General goals:		A-Respect(Y) A-Marry(Y) A-Approval(Y)

To summarize the knowledge-structures we have discussed, we note their interrelationships:

Themes give rise to goals.

A plan is understood when its goals are identified and its actions are consistent with the accomplishment of those goals.

Scripts are standardized models of events.

Scripts are specific; plans are general.

Plans originate from scripts.

Plans are ways of representing a person's goals. These goals are implicit in scripts, which represent only the actions.

A script has a header, which is pattern-matched to an input sentence. Plans do not have headers, but each plan is subsumed under a goal.

SAM

Both SAM and PAM accept stories as input; both use an English-to-CD parser to produce an internal representation of the story (in conceptual dependency). Both are able to paraphrase the story and to make intelligent inferences from it. They differ with respect to the processing that goes on after the CD representation has been built.

SAM understands stories by fitting them into one or more scripts. After this match is completed, it makes summaries of the stories. The process of fitting a story into a script has three parts, a PARSER, a memory module (MEMTOK), and the script applier (APPLY). These modules cooperate: The parser generates a CD representation of each sentence, but APPLY gives it a set of Verb-senses to use once a script has been identified. For example, once the restaurant script has been established, APPLY tells the parser that the appropriate sense of the verb "to serve" is "to serve food" rather than, for example, "to serve in the army."

The parser does not make many inferences; thus it does not realize that "it" refers to the hot dog in "The hot dog was burned. It tasted awful." This task is left to MEMTOK. This module takes references to people, places, things, etc., and fills in information about them. It recognizes that the "it" in the sentence above refers to the hot dog, and "instantiates" the "it" node in the CD representation of the second sentence with the "hot dog" node from the first sentence. Similarly, in a story about John, MEMTOK would replace "he" with "John" where appropriate, and would continually update the "John" node as more information became available about him.

The APPLY module has three functions. First, it takes a sentence from the parser and checks whether it matches the current script, a concurrent (interacting) script, or *any* script in the database. If this matching is successful, it makes a set of predictions about likely inputs to follow. Its third task is to instantiate any steps in the current script that were "skipped over" in the story. For example, if the first sentence of a story is "John went to a restaurant," APPLY finds a match with the *script header* of the restaurant script in its database (see Figure 1). APPLY then sets up predictions for seeing the other *events* in the restaurant script in the input. If the next sentence is "John had a hamburger," then APPLY successfully matches this sentence into the restaurant script (event 6). It then assumes events 2-5 happened, and instantiates structures in its CD representation of the story to this effect. Events 7-10 remain as predictions.

When the whole story has been mapped into a CD representation in this manner, the SAM program can produce a summary of the story, or answer questions about it. (See Schank & Abelson, 1977, pp. 190-204, for an annotated sample protocol with the program.) Consistent with the idea of interlingua, SAM can produce summaries in English, Chinese, Russian, Dutch, and Spanish. An example of a SAM paraphrase follows; note the powerful inferences made by instantiating intermediate script steps:

ORIGINAL: John went to a restaurant. He sat down. He got mad. He left.

PARAPHRASE: JOHN WAS HUNGRY. HE DECIDED TO GO TO A RESTAURANT. HE WENT TO ONE. HE SAT DOWN IN A CHAIR. A WAITER DID NOT GO TO THE TABLE. JOHN BECAME UPSET. HE DECIDED HE WAS GOING TO LEAVE THE RESTAURANT. HE LEFT IT.

SAM inferred that John left the restaurant because he did not get any service. The basis for this inference is that in the restaurant script, event 3 represents the waiter coming over to the table after the main actor has been seated. SAM knows that people can get mad if their expectations are not fulfilled, and infers that John's anger results from the nonoccurrence of event 3.

PAM

Wilensky's (1978) PAM system understands stories by determining the goals that are to be achieved in the story and attempting to match the actions of the story with the methods that it knows will achieve the goals. More formally:

> The process of understanding plan-based stories is as follows:

a) Determine the goal,

b) Determine the D-goals that will satisfy that goal,

c) Analyze input conceptualizations for their potential realization of one of the planboxes that are called by one of the determined D-goals. (Schank & Abelson, 1977, p. 76)

PAM utilizes two kinds of knowledge structure in understanding goals: *named plans* and *themes*. A named plan is a set of actions and subgoals for accomplishing a main goal. It is not very different from a script, although the emphasis in named plans is on goals and the means to accomplish them. For example, a script for rescuing a person from a dragon would involve riding to the dragon's lair and slaying it--a sequence of actions--but a named plan would be a list of subgoals (find some way of getting to the lair, find some way of killing the dragon, etc.) and their associated planboxes. When PAM encounters a goal in a story for which it has a named plan, it can make predictions about the D-goals and the actions that will follow. It will look for these D-goals and actions in subsequent inputs. Finding them is equivalent to understanding the story.

Themes provide another source of goals for PAM. Consider the sentences:

a) John wanted to rescue Mary from the dragon.
b) John loves Mary. Mary was stolen away by a dragon.

In both of these cases, PAM will expect John to take actions that are consistent with the goal of rescuing Mary from the dragon, even though this goal was not explicitly mentioned in (b). The source of this goal in (b) is the LOVE theme mentioned above, because in this theme, if another actor tries to cause harm to a loved one, the main actor sets up the goal of Achieving-Health of the loved one and possibly harming the evil party. (It is assumed that the dragon stole Mary in order to hurt her.)

PAM determines the goals of an actor by (a) their explicit mention in the text of the story, (b) establishing them as D-goals for some known goal, or (c) inferring them from a theme mentioned in the story. To understand a story is to "keep track of the goals of each of the characters in a story and to interpret their actions as means of achieving those goals" (Schank & Abelson, 1977, p. 217). The program begins with written English text, converts it into CD representation, and then interprets each sentence in terms of goals (predicting D-goals and actions to accomplish them) or actions themselves (marking the D-goals as "accomplished"). When this process is completed, PAM can summarize the story and answer questions about the goals and actions of the characters.

Summary

Scripts, plans, goals, and themes are knowledge structures built upon conceptual dependency theory. SAM is a program for understanding script-based stories. It matches the input sentences of a story to events in one or more of the scripts in its database. As such, it processes input based on *expectations* it has built up from the scripts. PAM understands plan-based stories by determining the goals of the characters of the story and by interpreting subsequent actions in terms of those goals or subgoals that will achieve

them. A great deal of inference can be required of PAM simply to establish the goals and subgoals of the story from the input text.

Schank and Abelson argue that human story understanding is a mixture of applying known scripts and inferring goals (where no script is available or of obvious applicability). They are experimenting with interactions of SAM and PAM, in particular, with using SAM to handle script-based sub-stories under the control of PAM.

References

The recent book by Schank & Abelson (1977) is the most complete and readable source on both of these systems and on the current state of Conceptual Dependency theory. For the whole truth about PAM, see the doctoral dissertation by Wilensky (1978a).

Also of interest: Abelson (1973), Bartlett (1932), Minsky (1975), Propp (1968), Riesbeck (1975), Rumelhart (1975), Schank (1973), Schank et al. (1975), Thorndyke (1977), and Wilensky (1978b).

F7. LIFER

The natural language systems described in the preceding articles fall into two categories: those built to study natural language processing issues in general and those built with a particular task domain in mind. In contrast, LIFER, built by Gary Hendrix (1977a) as part of the internal research and development program of SRI International, is designed to be an "off-the-shelf" natural language utility available to system builders who want to incorporate an NL front-end interface to improve the usability of their various applications systems. The bare LIFER system is a system for generating natural language interfaces; the interface builder can augment LIFER to fit his particular application, and even the eventual users can tailor the LIFER-supported front-end to meet their individual styles and needs.

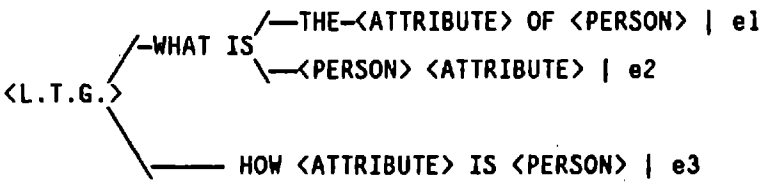
Language Specification and Parsing

The LIFER system has two major components: a set of interactive functions for specifying a language, and a parser. Initially it contains neither a grammar nor the semantics of any language domain. An interface builder uses the language specification functions to define an *application language*, a subset of English that is appropriate for interacting with his application system. The LIFER system then uses this language specification to *interpret* natural language inputs as commands for the application system.

The interface builder specifies the language primarily in terms of grammatical *rewrite rules* (see Article C1). LIFER automatically translates these into transition trees, a simplified form of augmented transition networks (Article D2). Using the transition tree, the parser interprets inputs in the application language. The result is an interpretation in terms of the appropriate routines from the applications system, as specified by the interface builder. The parser attempts to parse an input string top-down and left to right (see Article D1) by nondeterministically tracing down the transition tree whose root node is the start symbol (known as <L.T.G.> for "LIFER top grammar"). For example, suppose the interface builder has specified the following three production rules as part of his application language:

```
<L.T.G.> -> WHAT IS THE <ATTRIBUTE> OF <PERSON> | e1
<L.T.G.> -> WHAT IS <PERSON> <ATTRIBUTE> | e2
<L.T.G.> -> HOW <ATTRIBUTE> IS <PERSON> | e3
```

If an input matches one of these patterns, the corresponding expression (e1, e2, or e3) is evaluated--these are the appropriate *interpretations* that the system is to make for the corresponding input. The transition tree built by the language specification functions would look like this:



Sentences such as:

What is the age of Mary's sister?

How old is Mary's sister?
What is John's height?
How tall is John?

might be parsed using this simple transition tree, depending on how the *nonterminal* symbols or meta-symbols, <ATTRIBUTE> and <PERSON>, are defined. (The interface builder can supply a preprocessing function which is applied to the input string before LIFER attempts to parse it. Typically the preprocessor strips trailing apostrophes and *s*'s so that LIFER sees "John's" as "John".)

During parsing, LIFER starts at the symbol <L.T.G.> and attempts to move toward the expressions to be evaluated at the right. The parser follows a branch only if some portion at the left of the remaining input string can be matched to the first symbol on the branch. Actual words (such as what or of in the above example) can be matched only by themselves. Meta-symbols (such as <ATTRIBUTE> or <PERSON>) can be matched in a number of ways, depending on how the interface builder has defined them:

(a) as a simple set (for example, <PERSON> = the set {Mary, John, Bill});

(b) as a predicate that is applied to the string to test for satisfaction (for example, some meta-symbol used in a piece of grammar to recognize dates might test whether the next string of characters is a string of digits, and thus a number); or

(c) by another transition tree which has this meta-symbol as its root node.

The above example is typical: A large amount of semantic information is embedded in the syntactic description of the application language. JOHN and HEIGHT are not defined as instances of the single meta-symbol <NOUN> as they would be in a more formal grammar, but rather are separated into the semantic categories indicated by the meta-symbols <PERSON> and <ATTRIBUTE>. The technique of embedding such semantic information in the syntax has been referred to as *semantic grammar* (Burton, 1976), and it greatly increases the performance of LIFER's automatic spelling correction, ellipsis, and paraphrase facilities, described below.

Applications

LIFER has been used to build a number of natural language interfaces, including a medical database, a task scheduling and resource allocation system, and a computer-based expert system. The most complex system built with a LIFER interface involved a few man-months of development of the natural language front-end: The LADDER system (Language Access to Distributed Data with Error Recovery) developed at SRI, which provides real-time natural language access to a very large database spread over many smaller databases in computers scattered throughout the United States (Sacerdoti, 1977; Hendrix et al., 1978). Users of the system need have no knowledge of how the data is organized nor where it is stored. More importantly, from the point of view of this article, users do not need to know a data query language: They use English, or rather a subset that is "natural" for the domain of discourse and which is usually understood by the LIFER front-end. The interpretations of the inputs by LIFER are translations into a general database query language, which the rest of

the LADDER system converts to a query of the appropriate databases on the appropriate computers (see Article Applications.F4 on AI in information retrieval systems).

Another interesting system to use a LIFER front-end was the HAWKEYE system (Barrow et al., 1977), also developed at SRI. This is an integrated interactive system for cartography or intelligence, which combines aerial photographs and generic descriptions of objects and situations with the topographical and cultural information found in traditional maps. The user queries the database and invokes image-processing tasks via a LIFER natural language interface. A unique feature of this interface is the combination of natural language and nontextual forms of input. For instance, using a cursor to point to places within an image, the user can ask questions such as "What is this?" and "What is the distance between here and here?" The interpretation of such expressions results in requests for coordinates from the subsystem providing graphical input, which are then handed to subsystems that have access to the coordinates-to-object correspondences.

Human Engineering

LIFER is intended as a system which both facilitates an interface builder in describing an appropriate subset of a language and its interpretation in his system, and also helps a non-expert user to communicate with the application system in whatever language has been defined. For this reason, close attention was paid to the human engineering aspects of LIFER. Experience with the system has shown that, for some applications, users previously unfamiliar with LIFER have been able to create usable natural language interfaces to their systems in a few days. The resulting systems have been directly usable by people whose field of expertise is not computer science.

The interface builder. Unlike PROGRAMMAR (in SHRDLU, Article F4), there is no "compilation" phase during which the language specification is converted into a program. Instead, changes are made incrementally every time a call to the language specification functions is made. Furthermore, it is easy (by typing a prefix character) to intermix statements to be interpreted by the specification functions, statements to be parsed using the partially specified grammar, and statements to be evaluated in the underlying implementation language of LIFER, namely INTERLISP (see Article AI Languages.C1). Thus, the interface builder can define a new rewrite rule for the grammar or write a predicate for some meta-symbol and test it immediately, which leads to a highly interactive style of language definition and debugging. A *grammar editor* allows mistakes to be undone. The ability to intermix language definition with parsing allows the interface user to extend the interface language to personal needs or taste during a session using the application system. This extension can be done either by directly invoking the language specification functions, or, if the interface builder has provided the facility, by typing natural language sentences whose interpretations invoke the same language specification functions.

The interface user. LIFER provides many features to ease the task of the user typing in sentences to be understood by the system. First of all, it provides feedback indicating when LIFER is parsing the input sentence and when the applications software is running. When LIFER fails to parse a sentence, it tries to give the user useful information on how it failed. It tells the user how much of the input was understood and what it was expecting when it got to the point where it could no longer understand. Interactions with the user are numbered, and the user can refer back to a previous question and specify some substitution to be made. For instance:

```
12.     How many minority students took 20 or more units of credit last
        quarter?
PARSED!
        87

13.     Use women for minority in 12
PARSED!
        156
```

Notice the "PARSED!" printed by LIFER to indicate parsing success. This facility can be used to save typing (and more errors), both when similar questions are being asked and when errors in previous inputs are being corrected. The user can simply specify synonyms to be used. For instance:

```
28.     Define Bill like William
```

will cause LIFER to treat the word BILL the same as WILLIAM. LIFER also allows for easy inspection of the language definition, which is useful for both interface builders and sophisticated users.

There are three more sophisticated aspects of LIFER designed to make interactions easier for the user--the spelling correction, ellipsis, and paraphrase mechanisms. Spelling correction is attempted when LIFER fails to parse an input. When the parser is following along a branch of a transition tree and reaches a point where it can go no further, it records its failure in a failure list. If the input is eventually parsed correctly, the failure list is forgotten. However, if no successful parse can be found, the parser goes back to the last (rightmost) fail point and attempts to see if a misspelling has occurred. (Fail points to the left in the sentence are at first assumed not to be caused by spelling errors, since at least one transition using the word must have been successful to get to the fail point further to the right. This is not foolproof, however, and sometimes LIFER will fail on a spelling mistake). The INTERLISP spelling correction facility is used to find candidate words that closely match the spelling of the suspect word. The use of semantically significant syntactic categories (such as <PERSON>) greatly restricts the allowable word susbstitutions and improves the efficiency of the spelling corrector.

While interacting with an applications system, the user may want to carry out many similar tasks (for example, in a database query system, one often asks several questions about the same object). The LIFER system automatically allows the user to type incomplete input fragments and attempts to interpret them in the context of the previous input (i.e., the interface builder need not consider this issue). For instance, the following three questions might be entered successively and understood by LIFER:

```
42.     What is the height of John
43.     the weight
44.     age of Mary's sister
```

If an input fails normal parsing and spelling correction, LIFER tries elliptic processing. Again, because languages defined in LIFER tend to encode semantic information in the syntax definition, similar syntactic structures tend to have similar semantics. Therefore LIFER accepts any input string that is syntactically analogous to any contiguous substring of words

in the last input that parsed without ellipsis. The analogies do not have to be in terms of complete subtrees of the syntactic tree, but they do have to correspond to contiguous words in the previous input. The elliptical processing allows for quite natural and powerful interactions to take place, without any effort from the interface builder.

The paraphrase facility allows users to define new syntactic structures in terms of old structures. The user gives an example of the structure and interpretation desired, and the system builds the most general new syntactic rule allowed by the syntactic rules already known. The similarity between the semantics and syntax is usually sufficient to ensure that a usable syntax rule is generated. The following example assumes that the interface builder has included a rule to interpret the construction shown to invoke a call to the language specification function PARAPHRASE with appropriately bound arguments. After typing

```
63.     Let "Describe John" be a paraphrase of "Print the height, weight
        and age of John"  ,
```

the user could expect the system to understand the requests

```
64.     Describe Mary
65.     Describe the tallest person
66.     Describe Mary's sister
```

even with a fairly simply designed LIFER grammar. (In the context of the earlier examples, this example assumes that "the tallest person" can correspond to the meta-symbol <PERSON>.) The method used to carry out paraphrase (which, as can be seen, is a much more general form of synonymic reference) is quite complex. Basically it invokes the parser to parse the model (the second form of 63) that is already understood. All proper subphrases (i.e., subphrases that are complete expansions of a syntactic category) of the model that also appear in the paraphrase are assumed to play the same role. A new syntactic rule can then be written, and the actions invoked by the model can be appropriately attached to the paraphrase rule.

Conclusions

Although grammars constructed with LIFER may not be as powerful as specially constructed grammars, LIFER demonstrates that useful natural language systems for a wide variety of domains can be built simply and routinely without a large-scale programming effort. Human engineering features and the ability of the naive user to extend the system's capabilities are important issues in the usefulness of the system.

References

Hendrix (1977a), Hendrix (1977b), and Hendrix (1977c) all describe the LIFER system. The LADDER information retrieval application is described in Hendrix et al. (1978) and Sacerdoti (1977). Barrow et al. (1977) describes the HAWKEYE system.

References

Abelson, R. The structure of belief systems. In R. Schank & K. Colby (Eds.), **Computer Models of Thought and Language**. San Francisco: W. H. Freeman, 1973. Pp. 287-339.

Akmajian, A., & Heny, F. **An Introduction to the Principles of Transformational Syntax**. Cambridge: MIT Press, 1975.

Bar-Hillel, Y. The present status of automatic translation of languages. In F. L. Alt (Ed.), **Advances in Computers** (Vol. 1). New York: Academic Press, 1960. Pp. 91-163.

Bar-Hillel, Y. **Language and Information**. Reading, Mass.: Addison-Wesley, 1964.

Barrow, H. G., Bolles, R. C., Garvey, T. D., Kremers, J. H., Lantz, K., Tenenbaum, J. M., & Wolf, H. C. Interactive aids for cartography and photo interpretation. In L. S. Baumann (Ed.), **Image Understanding: Proceedings of a Workshop held at Palo Alto, California, October 20-21, 1977**. Rep. No. SAI-78-656-WA, Science Applications, Inc. Pp. 111-127.

Bartlett, F. C. **Remembering: A Study in Experimental and Social Psychology**. Cambridge: Cambridge University Press, 1932.

Bobrow, D. G. Natural language input for a computer problem-solving system. In M. Minsky (Ed.), **Semantic Information Processing**. Cambridge: MIT Press, 1968. Pp. 146-226.

Bobrow, D. G., & Collins, A. (Eds.). **Representation and Understanding**. New York: Academic Press, 1975.

Bobrow, D. G., & Fraser, B. An augmented state transition network analysis procedure. **IJCAI 1**, 1969, 557-567.

Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., & Winograd, T. GUS, a frame-driven dialog system. **Artificial Intelligence**, 1977, **8**, 155-173.

Bobrow, D. G., & Winograd, T. An overview of KRL, a knowledge representation language. **Cognitive Science**, 1977, **1**, 3-46.

Boden, M. **Artificial Intelligence and Natural Man**. New York: Basic Books, 1977.

Booth, A. D. (Ed.). **Machine Translation**. Amsterdam: North-Holland, 1967.

Bresnan, J. A realistic transformational grammar. In M. Halle, J. Bresnan, & G. A. Miller (Eds.), **Linguistic Theory and Psychological Reality**. Cambridge, Mass.: MIT Press, 1978. Pp. 1-59.

Bruce, B. Case systems for natural language. **Artificial Intelligence**, 1975, **6**, 327-360.

Burton, R. R. **Semantic grammar: An engineering technique for constructing natural language understanding systems.** BBN Rep. 3453, Bolt Beranek & Newman Inc., Cambridge, Mass., December 1976.

Burton, R. R., & Brown, J. S. Toward a natural-language capability for computer-assisted instruction. In H. O'Neil (Ed.), **Procedures for instructional Systems Development.** New York: Academic Press, 1979. Pp. 273-313.

Chafe, W. L. Discourse structure and human knowledge. In R. O. Freedle & J. B. Carroll (Eds.), **Language Comprehension and the Acquisition of Knowledge.** Washington, D. C.: V. H. Winston, 1972. Pp. 41-69.

Charniak, E. **Toward a model of children's story comprehension.** AI TR-266, AI Lab, MIT, 1972.

Charniak, E. **A brief on case.** Rep. 22, Institute for Semantic and Cognitive Studies, Castagnola, Switzerland, 1975.

Charniak, E., & Wilks, Y. **Computational Semantics: An Introduction to Artificial Intelligence and Natural Language Comprehension.** Amsterdam: North-Holland, 1976.

Chomsky, N. Three models for the description of language. **IRE Transactions on Information Theory,** 1956, **2,** 113-124. (Also in R. Luce, R. Bush, & E. Galanter (Eds.), **Readings in Mathematical Psychology** (Vol. 2). New York: John Wiley & Sons, 1965. Pp. 105-124.)

Chomsky, N. **Syntactic Structures.** The Hague: Mouton & Co., 1957.

Chomsky, N. On certain formal properties of grammars. **Information and Control,** 1959, **2,** 137-167. (Also in R. Luce, R. Bush, & E. Galanter (Eds.), **Readings in Mathematical Psychology** (Vol. 2). New York: John Wiley & Sons, 1965. Pp. 125-155.)

Chomsky, N. Formal properties of grammars. In R. Luce, R. Bush, & E. Galanter (Eds.), **Handbook of Mathematical Psychology** (Vol. 2). New York: John Wiley & Sons, 1963. Pp. 323-418.

Chomsky, N. **Aspects of the Theory of Syntax.** Cambridge: MIT Press, 1965.

Chomsky, N. Deep structure, surface structure, and semantic interpretation. In D. Steinberg & L. Jakobovits (Eds.), **Semantics.** Cambridge: Cambridge University Press, 1971. Pp. 183-216.

Clippinger, J. H., Jr. Speaking with many tongues: Some problems in modeling speakers of actual discourse. **TINLAP-1,** 1975. Pp. 78-83.

Codd, E. F. Seven steps to rendezvous with the casual user. In J. W. Klimbie & K. L. Koffeman (Eds.), **Data Base Management.** Amsterdam: North-Holland, 1974. Pp. 179-200.

Cohen, P. R. On knowing what to say: Planning speech acts. Tech. Rep. 118, Department of Computer Science, University of Toronto, January 1978.

Colby, K., Weber, S., & Hilf, F. Artificial paranoia. **Artificial Intelligence**, 1971, 2, 1-25.

COLING76. Preprints of the 6th International Conference on Computational Linguistics. Ottawa, Ontario, Canada, June 1976.

Conway, M. E. Design of a separable transition-diagram compiler. **CACM**, 1963, 6, 396-408.

Culicover, P. W., Wasow, T., & Akmajian, A. **Formal Syntax**. New York: Academic Press, 1977.

Feigenbaum, E., & Feldman, J. (Eds.). **Computers and Thought**. New York: McGraw-Hill, 1963.

Fillmore, C. The case for case. In E. Bach & R. Harms (Eds.), **Universals in Linguistic Theory**. New York: Holt, Rinehart, & Winston, 1968. Pp. 1-88.

Fillmore, C. Some problems for case grammar. In R. J. O'Brien (Ed.), **Report of the Twenty-Second Annual Round Table Meeting on Linguistics and Language Studies**. Monograph Series on Languages and Linguistics, No. 24. Washington: Georgetown University Press, 1971. Pp. 35-56. (a)

Fillmore, C. Types of lexical information. In D. Steinberg & L. Jakobovits (Eds.), **Semantics**. Cambridge: Cambridge University Press, 1971. Pp. 370-392. (b)

Friedman, J. Directed random generation of sentences. **CACM**, 1969, 12, 40-46.

Friedman, J. **A Computer Model of Transformational Grammar**. New York: American Elsevier, 1971.

Goldman, N. Conceptual generation. In R. Schank, **Conceptual Information Processing**. Amsterdam: North-Holland, 1975. Pp. 289-371.

Green, B. F., Jr., Wolf, A. K., Chomsky, C., & Laughery, K. BASEBALL: An automatic question answerer. In E. Feigenbaum & J. Feldman (Eds.), **Computers and Thought**. New York: McGraw-Hill, 1963. Pp. 207-216.

Grishman, R. A survey of syntactic analysis procedures for natural language. **American Journal of Computational Linguistics**, Microfiche 47, 1976.

Halliday, M. A. K. Categories of the theory of grammar. **Word**, 1961, 17, 241-292.

Halliday, M. A. K. Notes on transitivity and theme in English. **Journal of Linguistics**, 1967, 3, 37-81, 199-244; **Journal of Linguistics**, 1968, 4, 179-215.

Halliday, M. A. K. Functional diversity in language as seen from a consideration of modality and mood in English. **Foundations of Language**, 1970, 6, 322-361. (a)

Halliday, M. A. K. Language structure and language function. In J. Lyons (Ed.), **New Horizons in Linguistics**. Harmondsworth: Penguin Books, 1970. Pp. 140-165. (b)

Harman, G. (Ed.). **On Noam Chomsky: Critical Essays.** Garden City, New York: Anchor Books, 1974.

Harris, L. R. ROBOT: A high performance natural language processor for data base query. **SIGART Newsletter,** No. 61, February 1977, pp. 39-40.

Hays, D. G., & Mathias, J. (Eds.). FBIS seminar on machine translation. **American Journal of Computational Linguistics,** Microfiche 46, 1976.

Heidorn, G. E. Automatic programming through natural language dialogue: A survey. **IBM Journal of Research and Development,** 1976, 20, 302-313.

Hendrix, G. G. Human engineering for applied natural language processing. **IJCAI 5,** 1977, 183-191. (a)

Hendrix, G. G. LIFER: A natural language interface facility. **SIGART Newsletter,** No. 61, February 1977, pp. 26-26. (b)

Hendrix, G. G. The LIFER manual: A guide to building practical natural language interfaces. Tech. Note 138, Artificial Intelligence Center, SRI International, February 1977. (c)

Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D. & Slocum, J. Developing a natural language interface to complex data. **ACM Transactions on Database Systems,** 1978, 3, 105-147.

Hendrix, G. G., Thompson, C., & Slocum, J. Language processing via canonical verbs and semantic models. **IJCAI 3,** 1973, 262-269.

Hopcroft, J. E., & Ullman, J. D. **Formal Languages and Their Relation to Automata.** Reading, Mass.: Addison-Wesley, 1969.

Hudson, R. A. **English Complex Sentences: An Introduction to Systemic Grammar.** Amsterdam: North-Holland, 1971.

Hudson, R. A. **Arguments for a Non-transformational Grammar.** Chicago: University of Chicago Press, 1976.

Josselson, H. H. Automatic translation of languages since 1960: A linguist's view. In M. C. Yovits (Ed.), **Advances in Computers** (Vol. 11). New York: Academic Press, 1971. Pp. 1-58.

Kaplan, R. M. A general syntactic processor. In R. Rustin (Ed.), **Natural Language Processing.** New York: Algorithmics Press, 1973. Pp. 193-241.

Katz, J. & Postal, P. **An Integrated Theory of Linguistic Descriptions.** Cambridge: MIT Press, 1964.

Kay, M. The MIND system. In R. Rustin (Ed.), **Natural Language Processing.** New York: Algorithmics Press, 1973. Pp. 155-188.

Kellogg, C. A natural language compiler for on-line data management. **AFIPS Conference Proceedings, 33, 1968 Fall Joint Computer Conference.** Washington: Thompson Book Co., 1968. Pp. 473-492.

Klein, S. Automatic paraphrasing in essay format. **Mechanical Translation,** 1965, **8,** 68-83.

Klein, S., & Simmons., R. F. Syntactic dependence and the computer generation of coherent discourse. **Mechanical Translation,** 1963, **7,** 50-61.

Knuth, D. **The Art of Computer Programming** (Vol. 1): **Fundamental Algorithms** (2nd ed.). Reading, Mass.: Addison-Wesley, 1973.

Landsbergen, S. P. J. Syntax and formal semantics of English in PHLIQA1. In L. Steels (Ed.), **Advances in Natural Language Processing.** Antwerp: University of Antwerp, 1976.

Lindsay, R. K. Inferential memory as the basis of machines which understand natural language. In E. Feigenbaum & J. Feldman (Eds.), **Computers and Thought.** New York: McGraw-Hill, 1963. Pp. 217-233. (a)

Lindsay, R. K. A program for parsing sentences and making inferences about kinship relations. In A. C. Hoggatt & F. E. Balderston (Eds.), **Symposium on Simulation Models: Methodology and Applications to the Behavioral Sciences.** Cincinnati: South-Western Publishing, 1963. Pp. 111-138. (b)

Locke, W. N., & Booth, A. D. (Eds.). **Machine Translation of Languages.** New York: Technology Press of MIT and John Wiley & Sons, 1955.

Lyons, J. **Introduction to Theoretical Linguistics.** London: Cambridge University Press, 1968.

Lyons, J. **Noam Chomsky.** New York: Viking Press, 1970.

Marcus, M. A computational account of some constraints on language. **TINLAP-2,** 1978, pp. 236-246.

Matuzceck, D. **An implementation of the augmented transition network system of Woods.** As revised by J. Slocum. Department of Computer Sciences and CAI Laboratory, University of Texas, Austin, October 1972.

McCord, M. On the form of a systemic grammar. **Journal of Linguistics,** 1975, **11,** 195-212.

McDermott, D. **Assimilation of new information by a natural language-understanding system.** AI TR-291, AI Lab, MIT, February 1974.

McIntosh, A., & Halliday, M. A. K. **Patterns of Language.** Bloomington: Indiana University Press, 1966.

Minsky, M. (Ed.) **Semantic Information Processing.** Cambridge: MIT Press, 1968.

Minsky, M. A framework for representing knowledge. In P. Winston (Ed.), **The Psychology of Computer Vision.** New York: McGraw-Hill, 1975.

Nash-Webber, B. **Semantics and speech understanding.** BBN Rep. 2896, Bolt Beranek & Newman Inc., Cambridge, Mass., October 1974.

National Research Council, Automatic Language Processing Advisory Committee. **Language and Machines: Computers in Translation and Linguistics.** Publication 1416, National Academy of Sciences, National Research Council, Washington, D. C., 1966.

Norman, D., & Rumelhart, D. **Explorations in Cognition.** San Francisco: W. H. Freeman, 1975.

Oettinger, A. G. The design of an automatic Russian-English technical dictionary. In W. N. Locke & A. D. Booth (Eds.), **Machine Translation of Languages.** New York: Technology Press of MIT and John Wiley & Sons, 1955. Pp. 47-65.

Paige, J. M., & Simon, H. A. Cognitive processes in solving algebra word problems. In B. Kleinmuntz (Ed.), **Problem Solving.** New York: John Wiley & Sons, 1966. Pp. 51-119.

Perrault, C. R., Allen, J. F., & Cohen, P. R. Speech acts as a basis for understanding dialogue coherence. **TINLAP-2**, 1978, pp. 125-132.

Petrick, S. R. Transformational analysis. In R. Rustin (Ed.), **Natural Language Processing.** New York: Algorithmics Press, 1973. Pp. 27-41.

Plath, W. REQUEST: A natural language question-answering system. **IBM Journal of Research and Development**, 1976, 20, 326-335.

Postal, P. Limitations of phrase structure grammars. In J. A. Fodor & J. J. Katz, **The Structure of Language.** Englewood Cliffs, N. J.: Prentice-Hall, 1964. Pp. 137-151.

Propp, V. **Morphology of the Folktale.** 2nd edition, translated by L. Scott. Austin: University of Texas Press, 1968.

Quillian, M. R. Semantic memory. In M. Minsky (Ed.), **Semantic Information Processing.** Cambridge: MIT Press, 1968. Pp. 227-270.

Quillian, M. R. The teachable language comprehender: A simulation program and theory of language. **CACM**, 1969, 12, 459-476.

Raphael, B. SIR: A computer program for semantic information retrieval. In M. Minsky (Ed.), **Semantic Information Processing.** Cambridge: MIT Press, 1968. Pp. 33-145.

Rieger, C. Conceptual memory and inference. In R. Schank, **Conceptual Information Processing.** Amsterdam: North-Holland, 1975. Pp. 157-288.

Riesbeck, C. Conceptual analysis. In R. Schank, **Conceptual Information Processing.** Amsterdam: North-Holland, 1975. Pp. 83-156.

Rumelhart, D. Notes on a schema for stories. In D. G. Bobrow & A. Collins (Eds.), **Representation and Understanding.** New York: Academic Press, 1975. Pp. 211-236.

Rustin, R. (Ed.). **Natural Language Processing.** New York: Algorithmics Press, 1973.

Sacerdoti, E. D. Language access to distributed data with error recovery. **IJCAI 5**, 1977, 196-202.

Samlowski, W. Case grammar. In E. Charniak & Y. Wilks (Eds.), **Computational Semantics.** Amsterdam: North-Holland, 1976. Pp. 55-72.

Scha, R. J. H. A formal language for semantic representation. In L. Steels (Ed.), **Advances in Natural Language Processing.** Antwerp: University of Antwerp, 1976.

Schank, R. Identification of conceptualizations underlying natural language. In R. Schank & K. Colby (Eds.), **Computer Models of Thought and Language.** San Francisco: W. H. Freeman, 1973. Pp. 187-247.

Schank, R. **Conceptual Information Processing.** Amsterdam: North-Holland, 1975.

Schank, R., & Abelson, R. P. **Scripts, Plans, Goals, and Understanding.** Hillsdale, N. J.: Lawrence Erlbaum Assoc., 1977.

Schank, R., & Colby, K. (Eds.). **Computer Models of Thought and Language.** San Francisco: W. H. Freeman, 1973.

Schank, R., Goldman, N., Rieger, C., & Riesbeck, C. MARGIE: Memory, analysis, response generation, and inference on English. **IJCAI 3**, 1973, 255-261.

Schank, R., & Yale AI Project. **SAM -- A story understander.** Research Rep. 43, Department of Computer Science, Yale University, August 1975.

Searle, J. **Speech Acts.** Cambridge: Cambridge University Press, 1969.

Self, J. Computer generation of sentences by systemic grammar. **American Journal of Computational Linguistics**, Microfiche 29, 1975.

Simmons, R. F. Answering English questions by computer: A survey. **CACM**, 1965, **8**, 53-70.

Simmons, R. F. Storage and retrieval of aspects of meaning in directed graph structures. **CACM**, 1966, **9**, 211-214.

Simmons, R. F. Natural language question-answering systems: 1969. **CACM**, 1970, **13**, 15-30.

Simmons, R. F. Semantic networks: Their computation and use for understanding English sentences. In R. Schank & K. Colby (Eds.), **Computer Models of Thought and Language.** San Francisco: W. H. Freeman, 1973. Pp. 63-113.

Simmons, R. F., Burger, J. F., & Long, R. E. An approach toward answering English questions from text. **AFIPS Conference Proceedings, 29, 1966 Fall Joint Computer Conference.** Washington: Spartan Books, 1966. Pp. 357-363.

Simmons, R. F., Burger, J. F., & Schwarcz, R. M. A computational model of verbal understanding. **AFIPS Conference Proceedings, 33, 1968 Fall Joint Computer Conference.** Washington: Thompson Book Co., 1968. Pp. 441-456.

Simmons, R. F., Klein, S., & McConlogue, K. Indexing and dependency logic for answering English questions. **American Documentation**, 1964, 15, 196-204.

Simmons, R. F., & Slocum, J. Generating English discourse from semantic networks. **CACM**, 1972, 15, 891-905.

Steinberg, D., & Jakobovits, L. **Semantics**. Cambridge: Cambridge University Press, 1971.

Sussman, G., Winograd, T., & Charniak, E. **MICRO-PLANNER reference manual**, AI Memo 203, AI Lab, MIT, July 1970.

Taylor, B., & Rosenberg, R. S. A case-driven parser for natural language. **American Journal of Computational Linguistics**, Microfiche 31, 1975.

Thompson, F. B. English for the computer. **AFIPS Conference Proceedings, 29, 1966 Fall Joint Computer Conference**. Washington: Spartan Books, 1966. Pp. 349-356.

Thorndyke, P. W. Cognitive structures in comprehension and memory of narrative discourse. **Cognitive Psychology**, 1977, 9, 77-110.

TINLAP-1. Schank, R., & Nash-Webber, B. (Eds.). **Theoretical Issues in Natural Language Processing: An Interdisciplinary Workshop in Computational Linguistics, Psychology, Linguistics, and Artificial Intelligence**. June 1975.

TINLAP-2. Waltz, D. L. (Ed.). **Theoretical Issues in Natural Language Processing-2**. New York: Association for Computing Machinery, 1978.

Waltz, D. L. Natural language interfaces. **SIGART Newsletter**, No. 61, February 1977, pp. 16-64.

Waltz, D. L. **An English Language Question Answering System for a Large Relational Data Base**. In press, 1979.

Weaver, W. Translation (1949). In W. N. Locke & A. D. Booth (Eds.), **Machine Translation of Languages**. New York: Technology Press of MIT and John Wiley & Sons, 1955. Pp. 15-23.

Weizenbaum, J. Symmetric list processor. **CACM**, 1963, 6, 524-544.

Weizenbaum, J. ELIZA--A computer program for the study of natural language communication between man and machine. **CACM**, 1966, 9, 36-45.

Weizenbaum, J. **Computer Power and Human Reason: From Judgment to Calculation**. San Francisco: W. H. Freeman, 1976.

Wellin, C. W. **Semantic networks and case grammar**. Publication 29, Institute of Linguistics, University of Stockholm, May 1975.

Wilensky, R. Understanding goal-based stories. Research Rep. 140, Department of Computer Science, Yale University, September 1978. (a)

Wilensky, R. Why John married Mary: Understanding stories involving recurring goals. **Cognitive Science**, 1978, 2, 235-266. (b)

Wilks, Y. An artificial intelligence approach to machine translation. In R. Schank & K. Colby (Eds.), **Computer Models of Thought and Language**. San Francisco: W. H. Freeman, 1973. Pp. 114-151.

Wilks, Y. Natural language understanding systems within the AI paradigm: A survey and some comparisons. AI Memo 237, Stanford AI Lab, December 1974. (Also in A. Zampolli (Ed.), **Linguistic Structures Processing**. Amsterdam: North-Holland, 1977. Pp. 341-398.)

Wilks, Y. An intelligent analyzer and understander of English. **CACM**, 1975, **18**, 264-274. (a)

Wilks, Y. Preference semantics. In E. L. Keenan (Ed.), **Formal Semantics of Natural Language**. Cambridge: Cambridge Univ. Press, 1975. Pp. 329-348. (b)

Wilks, Y. A preferential, pattern-seeking semantics for natural language inference. **Artificial Intelligence**, 1975, **6**, 53-74. (c)

Wilks, Y. Processing case. **American Journal of Computational Linguistics**, Microfiche 56, 1976.

Wilks, Y. Time flies like an arrow. **New Scientist**, 1977, **76**, 696-698. (a)

Wilks, Y. What sort of taxonomy of causation do we need for language understanding? **Cognitive Science**, 1977, 1, 235-264. (b)

Wilks, Y. Making preferences more active. **Artificial Intelligence**, 1978, **11**, 197-223.

Winograd, T. **Procedures as a representation for data in a computer program for understanding natural language**, AI TR-17, AI Lab, MIT, February 1971.

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

Winograd, T. A procedural model of language understanding. In R. Schank & K. Colby (Eds.), **Computer Models of Thought and Language**. San Francisco: W. H. Freeman, 1973. Pp. 152-186.

Winograd, T. Five lectures on artificial intelligence. AI Memo 246, Stanford AI Lab, September 1974. (Also in A. Zampolli (Ed.), **Linguistic Structures Processing**. Amsterdam: North-Holland, 1977. Pp. 399-520.)

Winograd, T. Parsing natural language via recursive transition net. In R. Yeh (Ed.), **Applied Computation Theory: Analysis, Design, Modeling**. Englewood Cliffs, N.J.: Prentice-Hall, 1976. Pp. 451-467.

Winograd, T. **Language as a Cognitive Process**. Book in preparation, Addison-Wesley, 1980.

Wong, H. K. **Generating English sentences from semantic structures.** Tech. Rep. 84, Department of Computer Science, University of Toronto, August 1975.

Woods, W. A. Transition network grammars for natural language analysis. **CACM, 1970, 13,** 591-606.

Woods, W. A. An experimental parsing system for transition network grammars. In R. Rustin (Ed.), **Natural Language Processing.** New York: Algorithmics Press, 1973. Pp. 111-154. (a)

Woods, W. A. Progress in natural language understanding: An application to lunar geology. **AFIPS Conference Proceedings, 42, 1973 National Computer Conference.** Montvale, N. J.: AFIPS Press, 1973. Pp. 441-450. (b)

Woods, W. A., & Kaplan, R. **The lunar sciences natural language information system.** BBN Rep. 2265, Bolt Beranek & Newman Inc., Cambridge, Mass., 1971.

Woods, W. A., Kaplan, R., & Nash-Webber, B. **The Lunar Sciences Natural Language Information System: Final report.** BBN Rep. 2378, Bolt Beranek & Newman Inc., Cambridge, Mass., June 1972.

Yngve, V. Random generation of English sentences. **1961 International Conference on Machine Translation of Languages and Applied Language Analysis.** National Physical Laboratory, Symposium No. 13. London: Her Majesty's Stationery Office, 1962. Pp. 66-80.

Index