American Journal of Computational Linguistics *Microfiche 61*

TOWARD A "NATURAL" LANGUAGE QUESTION-ANSWERING FACILITY

BILL D. MAXWELL AND FRANCIS D. TUGGLE

Department of Computer Science
University of Kansas
Lawrence 66045

Maxwell is also associated with the Computation Center
Tuggle is also associated with the School of Business

SUMMARY

This study describes the structure, implementation and potential of a simple computer program that understands and answers questions in a humanoid manner. An emphasis has been placed on the creation of an extendible memory structure--one capable of supporting conversation in normal, unrestricted English on a variety of topics. An attempt has also been made to find procedures that can easily and accurately determine the meaning of input text. A parser using a combination of syntax and semantics has been developed for understanding YES-NO questions, in particular, DO-type (DO, DID, DOES, etc.) questions. A third and major emphasis has been on the development of procedures to allow the program to converse easily and "naturally" with a human. This general goal has been met by developing procedures that generate answers to DO-questions in a manner similar to the way a person might answer them.

TABLE OF CONTENTS

1. INTRODUCTION

There are a number of extant computer programs which interact intelligently with human interrogators, but all do so in a way we characterize as "unnatural". By "unnatural," we mean that they converse in a way significantly unlike two normal adult humans do. For example, Winograd's SHRDLU [19] seems child-like, and it shares with the Woods' moonrocks system [20] the problem of being task-constrained. Weizenbaum's ELIZA [16] gives the appearance of being vague, evasive and doubting. Colby's PARRY [5], of course, is variously paranoid. Finally, the CAI systems, such as Carbonell's SCHOLAR [4] and Brown and Burton's SOPHIE [3], seem officiously pedantic. In contrast to those approaches, we wanted to construct a program able to carry on an intelligent and "natural" dialogue with a human user.

As an example of a "natural" and an "unnatural" dialogue, imagine two computer programs (CPs) conversing with a human (H) in which each CP already knows "THE BOY OWNS A BALL." One CP is "natural" (NCP); the other is "unnatural" (UCP). (One may find fragments of responses from existing programs in those responses we have labelled as "UCP.")

H: The ball is red.
UCP: By 'the ball,' I assume you mean the ball the boy owns.
NCP: O.K.

H: Does the boy own a large, round ball?
UCP: No. [or at best] I don't know.
NCP: He owns a red one. It probably is round and may be large.

H: The boy lives on Main Street.
UCP: By 'the boy', I assume you mean the one who owns the red ball.
NCP: O.K.

H: Where does the boy play with his ball?
UCP: I don't know.
NCP: I don't know, but I assume near his residence on Main Street.

H: A house is on Main Street.
UCP: O.K.
NCP: Is it the house of the boy who owns the red ball?

H: It was expensive to construct.
UCP: By 'it,' do you mean
1. the house
2. Main Street?
NCP: O.K.

H: The ball broke a window.
UCP: By 'the ball,' I assume you mean the red one the boy owns. By 'a window,' I assume it is a part of the house in Main Street.
NCP: Did the boy throw the ball?

H: Was the window expensive to repair?
UCP: I don't know.
NCP: Since the house was expensive to construct, I assume that its windows are expensive to repair.

From this short, simple dialogue some of the characteristics which distinguish "natural" dialogue can be seen: "natural" language understanding processes are able to work with partial and overlapping information, are able to retain ambiguity until disambigation is needed, are able to perform "short" chains of deductions, are able to engage in common sense reasoning, and are able to pose reciprocal questions to the human to confirm expectations.

Reasons for trying to instill a certain degree of "humanness" to computer programs should be obvious--to facilitate their acceptance, to extend their use, and to make them more pleasant to deal with. People will be much more willing to work with a computer program if it gives the appearance of being humanoid itself, whether the kernel part of the program concerns CAI, MIS, or whatever.

We have developed a computer program called JIMMY3 which embodies some of the above set of characteristics of a "natural" language processing system so as to demonstrate its feasibility, usefulness, and potential power. The implementation has necessarily been largely ad hoc, but as Lindsay [6] notes, this is not altogether bad. Newell [7] properly records that there is a tradeoff between generality and power. Like Lindsay, we deliberately eschew the general in favor of the specific. In the interests of replicability and extensibility, we also provide a reasonably complete description of the inards of JIMMY3.

1.1 MAJOR GOALS

The focus of this work has been on three problems: a) the development of a memory structure useful for engaging in a dialogue with a person, b) the development of procedures that can easily and accurately determine the meaning of input text, and c) the development of procedures for the generation of output.

As contrasted with more conventional models for language understanding where input is decoded in terms of concepts and then mapped into memory, this model stores surface structures more or less intact without any conversion. This approach simplifies both the input processing and output generation phases of the system. However, it does necessitate the use of complex memory matching procedures during the answer-producing phase of understanding.

For parsing, several ad hoc rules were developed for applying a metric to measure "meaningfulness." Using this procedure, several different interpretations of the input are examined. The one having the highest "score," as determined by the metric, is selected as the proper meaning.

Most memory models and understanding systems do not have elaborate facilities for the creation of output that is interesting or stimulating to conversation. Output production by this program approaches solutions to three related problems: a) what information should be used in the response to a given input, b) how a response can be structured so certain parts are emphasized and c) how a response can be made to appear natural in the sense of being like a similar remark a person might make.

1.2 SAMPLE DIALOGUE

To get a feeling for the types of responses the currently implemented program can produce, the following short dialogue is presented. Note that with only the capability for answering DO-questions implemented, a continuing dialogue is not very easy to obtain.

Three explicit facts existed in memory at the time of this conversation. They were:

1) BRANDT MAXWELL OWNS A RED BOOK.
2) BRANDT MAXWELL OWNS A BLUE BOOK.
3) JIMMY3 KNOWS BRANDT MAXWELL.
   (this computer model has been dubbed JIMMY3)

The dialogue is given in upper case with commentary in lower case. The j: and p: (identifying JIMMY3 and person) were added here for clarity.

j: HI. WHO ARE YOU?
   JIMMY3 must find out who it is talking with in order to translate the pronoun "I" to facilitate memory searching.

p: BRANDT MAXWELL.

j: OK.

BRANDT@MAXWELL is recognized as a legitimate name. The @ in BRANDT@MAXWELL is used to indicate the combining of two or more words to form a separate entity. In the above example, the names BRANDT and MAXWELL are recognized as first and last names, respectively, that, when appearing together, are interpreted as the full name, BRANDT@MAXWELL.

p: DO YOU KNOW ANYONE?
j: YES, YOU.

Parsing is accomplished by matching input to templates consisting of (ACTOR ACT OBJECT) describing the different meaning senses for main verbs. This question is parsed using the (PERSON KNOW PERSON) definition of KNOW. On input, YOU is translated to JIMMY3 (which has ISA PERSON as one of its properties).

Memory is matched against the (JIMMY3 KNOW PERSON) pattern which yields the match (JIMMY3 KNOWS BRANDT@MAXWELL). Output procedures then convert the match into the given response.

p: DO YOU KNOW BILL MAXWELL?
j: NO.

The search using (JIMMY3 KNOW BILL@MAXWELL) yields no match. Under other circumstances a failure of this type would produce an answer of "I DON'T KNOW". However, the act KNOW is tagged so that "NO" is the proper response if a search of memory fails when the triple has KNOW as its act(main verb).

p: DO YOU THINK?
j: I DON'T KNOW.

The input was correctly parsed as (PERSON THINK NIL) but the memory search found no matches. The current model has no facilities for introspection other than the examination of triples in memory that deal explicitly with JIMMY3.

p: DO YOU KNOW ANYTHING?
j: I DON'T UNDERSTAND THE USE OF KNOW.

At the time this dialogue was produced, the only meaning of KNOW contained in memory was (PERSON KNOW PERSON). Therefore, the reasonable parse of (PERSON KNOW THING) was not found.

p: DO I HAVE A BLUE ANIMAL BOOK?
j: A BLUE BOOK BUT NOT AN ANIMAL BOOK:
Some, but not all, of the modifiers on the input were matched by a triple in memory as indicated by the response. Since not all modifiers matched, the answer is not prefaced with YES.

p: DOES ANYONE HAVE A RED THING?
j: YES. YOU OWN A RED BOOK.
Memory is searched using the (PERSON HAVE THING) pattern which best matches (BRANDT@MAXWELL OWNS BOOK). By comparing modifiers, (BRANDT@MAXWELL OWNS RED-BOOK) is selected over (BRANDT@MAXWELL OWNS BLUE-BOOK).

p: DO I OWN A GREEN BOOK?
j: NO. YOU OWN A RED BOOK.
Since GREEN and RED are mutually exclusive instances of COLOR, the answer is NO. (BRANDT@MAXWELL OWN BLUE-BOOK) was also matched but the triple with RED BOOK was used to produce the response since it was found first by the memory search routine.

p: DO I HAVE AN OLD BOOK?
j: YES. A RED ONE. I DON'T KNOW WHAT OLD MEANS.
OLD is ignored since it is an undefined word. Input is parsed as if it had been "DO I HAVE AN BOOK?"

p: DOES BILL MAXWELL OWN A BOOK?
j: I DON'T KNOW.
The fact (BILL@MAXWELL OWNS BOOK) is not present in memory. Neither is any contradiction of that fact.

p: BYE.
j: GOODBYE.

1.3 REVIEW

1.3.1 MEMORY MODELS

The research involving memory models can be divided into two basic approaches. First are models created for the exploration of theories of human memory and for the testing of linguistic theories. In these systems, the other features of a complete language understanding system assume a background position since the main emphasis is the memory model itself. Models which fall into this class are the works of Quillian [11], Anderson and Bower [1], and Norman, Rumelhart and the LNR research

group [10]. The second type of memory model is that developed as part of a system which has some component other than memory as the major emphasis. These models include Winograd's blocks world [19], Schank's Conceptual Dependency System [13, 14] and Colby's Artificial Paranoia model [5].

The memory model developed by this current research does not correspond to any existing model. It is not based on case grammar in a strict sense, but stores information in a form more closely related to surface structures utilizing only three main components: actor, act and object. Although this development was influenced to a limited extent by Anderson and Bower's model [1], it was shaped in actual design by the parser used by Wilks (see below) which attempts to find meaning by searching for triples. Wilks' work was also influential in the development of the parser.

1.3.2 LINGUISTIC PARSERS

In the area of linguistic parsers, there are currently four different methods being used. They are: a) semantic triples, b) augmented transition networks, c) procedures and d) pattern matching. These can be described most easily by examining specific examples of each.

The parser developed by Yorick Wilks [17, 18] is based on identifying triples composed of an actor, an act and an object. Input is parsed by applying trial templates to the input and identifying candidates to fill the three slots in a template. The association of an input word with a template slot is made by consideration of the

semantic relationship between the input word and the requirements of the slot. The closer the relationship, the better the match, and therefore, the better the parse. The choice between several potential parsings of an input string is determined by a scheme for computing the semantic density of the parse based on the number and types of matches that are obtained by filling the slots in the triple. Assuming English is a redundant language, the parse having the greatest density is the one which contains the most interconnections and, therefore, is the one to be accepted as the correct parse.

An important feature of Wilks' system is that syntactic properties of words take a role secondary to that assumed by the semantic properties. Thus it is possible to determine the meaning of input which is ill-formed and grammatically incorrect.

Augmented transition networks have been used for some time as a method for parsing. The principal system utilizing this method is the NASA lunar rocks system developed by Woods [20]. The parser used by th LNR group [10] is of similar design. While this technique is very flexible in allowing a large grammar to be consistently modelled, most implementations of it have been strictly syntactic. Little attempt has been made to incorporate semantic knowledge into the parsing. Without semantic knowledge, and using the strict grammatical rules embedded in the transition networks, this approach is really very brittle. It is not capable of handling ungrammatical input with much success.

Closely related to the augmented transition network approach is the use of procedures to describe a grammar as exemplified by Winograd's program [19]. This system is also predominately syntactic in nature. All information about the grammar is represented in terms of actual routines which are invoked during parsing.

The input analyzer used in Colby's Artificial Paranoia model [5] is essentially a pattern matcher which uses a few tricks to normalize all input to short strings which it hopes to recognize. It is not really either syntactically or semantically based but depends mostly on transformations to reduce input to simple, empirically derived, recognizable forms. The power of this approach lies in its ability to accept even the most ungrammatical input and relate it to something which is already known. Thus, there is a given context in which all input is interpreted--the context of what the model knows and wants to continue talking about.

1.3.3 OUTPUT GENERATION

Perhaps the work on output production which has most influenced the current program is the model by Colby [5]. As described in the previous section, input is recognized by reduction to simple identifiable patterns which can be matched to prestored strings in the program's memory. Also included in the memory are corresponding sets of strings which are to be used as output. Fixed responses are selected based on the current state of the program's self model; the state of its model of the person and the previous conversation. This technique gives the appearance of a normal, humanlike dialogue. This approach is successful because conversation is always directed in one particular, very narrow, direction by the way input is understood. The context is fixed and must not deviate from a single

2. PROGRAM OVERVIEW

2.1 PROGRAM STRUCTURE

The logical structure for the program consists of essentially four main sections executed in sequence with each step producing information required by the following step. This design, with minor variation, is characteristic of many existing programs for language understanding. One notable exception has been the SPEECH UNDERSTANDING PROJECT [8] which advocates the use of parallel modules working simultaneously on the input, passing data freely between routines, until the desired end result is reached.

The following algorithm describes how JIMMY3 behaves at the most superficial level.

```
DO until person is through talking:
.
(1) . Request user input and translate English words into internal
    . codes (MEMORY node numbers).
.
(2) . Parse input to create the "best" parse network(s).
.
(3) . Match each parse network with structures in memory to produce
    . the "best" match(es).
.
(4) . Produce a response based on the memory match(es).
.
END-DO
```

To make the processing of text more efficient, English words and punctuation are translated in step (1) into node pointers. Undefined words are changed to null pointers.

The parser then creates a parse network from which a network density is computed. This density is a measure of how well the particular parse captures the meaning of the input. The parsing procedure is driven by

heuristics to do a search of the most likely parse networks. In case of ambiguity, several parse networks can be passed to the memory match routine.

If the input was a statement, the information given by the network is stored in short term memory (STM). If a question is asked, then the parse network is matched against event nodes in the memory structure to find potential answers. A match score is computed for each pattern match based on how well the three major components (ACTOR, ACT and OBJECT) and all minor components (single modifiers and prepositional phrase modification) match. Only the closest matches are retained for use by the output generator. Ambiguity in the parse, if it exists, is resolved here by the selection of the best match regardless of its generating parse network.

The response is generated by procedures operating on the memory matches. The value of the match score determines generally what the response content should be. The exact form of the response is determined by the way the components of the input matched (or did not match) the memory pattern. After a form is decided on, the response is made grammatical [redacted]hen printed. Control is then returned to the input routine and the w[redacted]e sequence is ready to be repeated.

2.2 THE RUNNING PROGRAM

The environment, the time-sharing system on the Honeywell 66/60 running GCOS at the University of Kansas, in which this program was developed and is run dictated its form. This machine poses several problems besides the lack of suitable languages, the most serious being the 25K word limit on the amount of memory an interactive program can obtain.

Since there are no interactive string or list processing languages available on the system, only FORTRAN came close to satisfying the requirements of a language in which a natural language system could be realistically implemented. What was wanted was a high level language with overlay capabilities, small but powerful I/O packages and the facility for independently compiled and tested subroutines with utilities for the maintenance of subroutine libraries.

The program now running consists of over 130 subroutines written in a total of about 13K lines of FORTRAN code. It runs in approximately 20K (36-bit) words of memory when segmented into 5 overlays. An unlinked version is approximately 37K words in size. The use of core by various parts of the program is given in Table 1.

Response time for the program is good considering the amount of overhead required because of memory constraints. Dialogue like that given earlier takes 2-3 seconds between the last character of input until the answer starts to print.

		TOTALS
I/O PACKAGE	1.5	1.5
DATA		
SCRTCH array	2.0	
MEMORY Paging area	1.0	
WORDLIST area	1.3	
Miscellaneous	1.0	5.3
PROGRAM		
Main link + support routines	10.2	
Initialization & Setup	1.9	
Command processor	3.3	
Parsing	5.3	
Memory matching	6.2	
Output production	3.3	30.2
		37.0

Table 1: Storage allocation for unlinked program in thousands of Honeywell 66/60 words

2.3 DATA FILES

Two data files, WORDLIST and MEMORY, are required by the program. The WORDLIST file contains the text representation for all words, punctuation and system commands along with the keys for translating that text into memory node numbers. The MEMORY file is the collection of all memory nodes.

To aid in the recreation and continual updating of these files, the data contained in them is present in a text file which is maintained on-line using the time-sharing text editor. After changes to this primary file are made, a program, separate from JIMMY3, translates the text code into the WORDLIST and MEMORY files. A second program can unload the WORDLIST and MEMORY files back to text when necessary. Currently the text file contains a vocabulary of 387 entries (words, punctuation and commands), 22 ACT usages and 7 facts. This information is encoded in approximately 4000 lines of symbolic node representation. It, when loaded, creates a WORDLIST of 387 separate entries and a MEMORY with approximately 800 memory nodes.

3. MEMORY STRUCTURE

The memory system for JIMMY3 is an aggregation of four components: WORDLIST, MEMORY, temporary structures and STM. With the exception of the WORDLIST, each part is a collection of one or more substructures consisting of nodes connected by relations.

3.1 COMPONENTS

WORDLIST. The WORDLIST is an index into the MEMORY component and consists of representations for all units of input JIMMY3 is to recognize. Items not included in the WORDLIST are declared undefined by the input decoding routines. In addition to the exact text representation, a pointer to the corresponding node in MEMORY is given.

MEMORY. This is the model's long term memory--essentially a collection of interrelated nodes.

A node is the smallest packet of information in MEMORY that can be referenced by a single pointer (either a WORDLIST pointer or a pointer from another node.) The information contained in a node may describe or modify a single word or symbol or may be a collection of relations connecting several nodes into a more complex structure.

Information stored in nodes describing single words is varied and includes such items as part of speech, inflectional variations and subset or superset names. Other types of data, for example, events and statements of fact, are formed by nodes that contain pointer structures relating other nodes in a predetermined fashion.

TEMPORARY STRUCTURES. There are three basic memory structures of a transient nature that can exist during the processing required to determine meaning and produce an answer. They are: a) the parse network, b) the memory match structure, and c) the output production list.

A parse network is a small collection of nodes of the same format, and connected in the same fashion, as the nodes in MEMORY. Produced by the parser, it is used to represent the meaning of input text. This net-

work contains information about the major components of the input, i.e., what words or phrases represent the ACTOR, ACT and OBJECT; it also contains information about all modifying words and phrases found in the input.

A memory match structure is produced by matching the parse network with MEMORY. It contains comparison data relating the corresponding components of the input (parse network) and a substructure in MEMORY.

The output production list is used during the examination of the memory match structures to accumulate the discrete components of the response to be made, i.e., the words and punctuation for the answer to the question. Each element in this list contains a pointer to a text representation for the word (or punctuation), its function and its relation to other words in the sentence. When the output list has been formed, it is passed to a print routine which writes the answer to the terminal.

STM. STM is the short term memory component for the program. It is essentially a collection area for parse networks, memory match structures and output production lists generated for previous inputs and their responses.

3.2 RELATIONS

Among the attributes used to relate the different MEMORY and parse network nodes to each other are those for describing hierarchies and chains.

HIERARCHY. Hierarchies are vertical structures formed by relating nodes to one another using the ISA attribute. Set (superset, subset) relations are implemented as hierarchies. A typical example would be

the path through the nodes: BRANDT (ISA) BOY (ISA) PERSON which displays the relations among the three nodes.

In a hierarchy, each node is connected only to a single node immediately before and after it. However, for any given hierarchy, it is possible that the nodes involved are connected to others both in and out of the hierarchy by connections independent of that particular hierarchy structure.

Transitivity is a property of hierarchies as implemented by the program. Therefore, in the example above, the information, BRANDT (ISA) PERSON, is implicit in the hierarchy. It should be pointed out that in the forward link between BRANDT and BOY, the connection is not one of set membership but rather one of subset as is the relation between BOY and PERSON. This is accomplished by the concept of GENERIC nodes even for specific instances of, say, people. Therefore, the BRANDT in the hierarchy is a GENERIC node which will, in turn, have under it an INSTANCE node of BRANDT which is related by set membership. The GENERIC BRANDT is a set of one element.

CHAIN. The mechanism for constructing horizontal structures is the chain. As contrasted to the before link-after link structure of the hierarchy, the chain allows the creation of a linearly ordered set of nodes each relating to a common "root" node to which it is attached. This allows the creation of sets of nodes related by a common property. For example, chains exist in MEMORY tying together all inflectional variations of a word. Another example of a chain is the list of color

nodes: RED, BLUE and GREEN. A singly linked list exists through these nodes but in addition, each of the three points back to the node for COLOR--the node which points to the first color in the list.

The more important links currently used to form chains are described next.

DEFN. The DEFN link is used to tie together all nodes that represent different definitions for a word. Thus, for every symbol in the WORDLIST, there is a chain of nodes in MEMORY connected via the DEFN link.

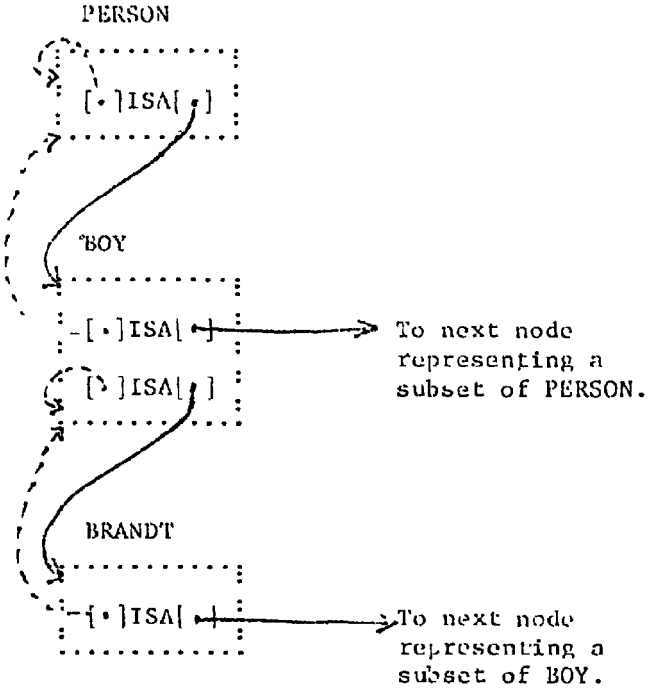

Figure 1: Hierarchy for BRANDT (ISA) BOY (ISA) PERSON

INST. A chain emanates from each definition of each word in MEMORY. These chains are created with the INST link and represent the set of all instances of the word given by the root node of the chain.

ISA. This link is used to generate hierarchies. Actually created is a chain with the property that every node in the chain is related by the ISA link to the root node. The root node can be a member of another ISA chain, thus giving a multilevel hierarchy. Figure 1 shows part of the hierarchy for BRANDT (ISA) BOY (ISA) PERSON.

ACTOR. This is used to connect a set of ACTION nodes in which the root node is used as the ACTOR of a triple. For example, there would be a chain through the triples representing (BRANDT HAVE BOOK), (BRANDT LIKE MILK) and (BRANDT HAVE CAT).

ACT. Used to form sets of ACTION nodes which contain the root node as the ACT of a triple.

OBJECT. Similar to the ACTOR and ACT links except it chains through ACTION nodes which have the root node as the OBJECT of a triple.

MODIFY. This is used to specify a set of nodes which are modified by the root node. The type of this single word modification is partially determined by the part of speech of the root node and that of the modified node. It is further specified by the hierarchy of the root node and the node being modified. For example, RED may be used to modify BOOK. In terms of grammatical function, the parts of speech specify this as an adjective modifying a noun. However, examination of the hierarchies discloses that RED (ISA) COLOR which is a property that THINGs (BOOK (ISA) THING) can have.

POS. This is the part of speech link.

PREP. This link is used in a CONTEXT node to point to the preposition part of the prepositional phrase represented by the node.

POBJ. The CONTEXT node also contains the reference to a prepositional object.

CNTXT. This is the link used to attach a CONTEXT node as a modifier of another node. The chain created by a CNTXT link represents all nodes modified by the same CONTEXT node.

INFLEC. Most words in MEMORY belong to an INFLEC chain. This is a set of inflectional variations of the word. It brings together different forms which vary in tense, number, person, etc. The INFLEC chain through "I" would join nodes for I. ME, MY, and MINE. A similar chain through OWN would link nodes for OWN, OWNS, OWNED and OWNING.

ANTONYM. All antonyms of a word are chained together using this link. The words in the chain are not antonyms of each other.

PARTS. This link is used in nodes to express the sub-part, super-part relationships.

3.3 SUBSTRUCTURES

Using the relations for constructing hierarchies and chains, various other, more complex, structures can be created. These are structures formed by the coincidence of several hierarchies or chains passing through a single node. Of all possible substructures, triples (ACTION nodes), CONTEXT nodes and SEMANTIC MODIFICATION nodes are of greatest importance.

TRIPLES. A triple (ACTION node) is a node through which passes three separate chains, one each for ACTOR, ACT, and OBJECT. These triples are used to specify events, facts and as semantic definitions for the ACTs (i.e., as ranges for acceptable candidates for ACTORs and OBJECTs.) The chains for ACTOR, ACT and OBJECT originate in nodes which represent those

major components and continue through this node to where they merge in different combinations with still other chains to form more triples. Figure 2 shows the relevant structure of a triple representing "PERSON HAVE BOOK".

CONTEXT NODES. A close relative of the triple is the context node. It too has chains passing through it determining its structure. However, it has only two chains: those for PREP and POBJ. These specify a preposition and its object. Nodes of this sort are used as modifiers of single nodes, triples and other context nodes.

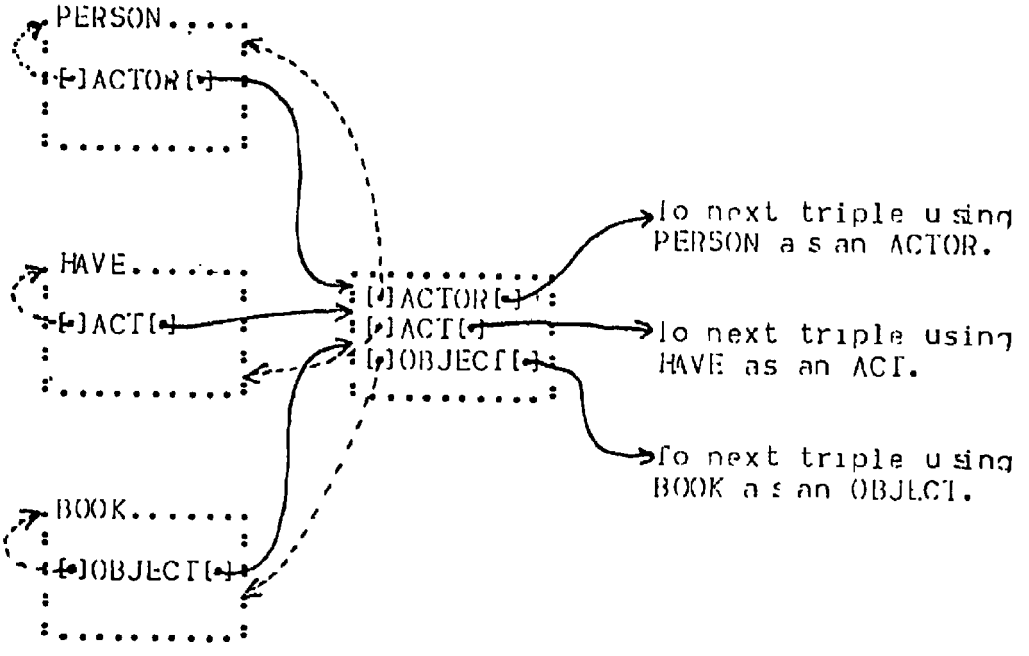

Figure 2: A triple representing (PERSON HAVE BOOK).

SEMANTIC MODIFICATION NODES. These nodes are not similar to triples or context nodes either in design or in function. However, they are one of the major substructures appearing in MEMORY so will be considered here briefly. These nodes or collections of nodes are the data strings which are used to drive the parser and in that capacity, will be described later. Essentially they contain ordered lists of semantic categories representing potential modification patterns for words. These lists are applied by the parser to determine semantically acceptable word strings in the input that can modify other word strings. For example, a noun would have a list describing the types of adjectives that could modify it. For adjectives, there would be a list of potential adverb types. Each definition of every word in MEMORY that is to be recognized during parsing of input must have attached to it a semantic modification node. In cases where many words have the same node, the structure for semantic modification is implemented as a chain through all words having the same modification requirements.

3.4 NODES

Nodes in MEMORY are represented as fixed-size blocks of contiguous disk or core locations and are the smallest units of MEMORY that can be referenced. Each node is composed of links and variable length attributes which may be data or pointers to other nodes. All the space allocated to a node does not have to be used. In fact, most nodes use only part of their allocated space to contain attributes; the remainder is empty (zero). The current model uses a size of 8 words for its nodes. (This size was not determined empirically as the optimum size but was, instead, selected because of disk hardware considerations.) The first word of every node in MEMORY is used for bookkeeping and, therefore, is not available for storing

attributes. Contained in the first word is the node number itself, an indicator of the kind of node plus a pointer that gives the next word of the node available for use as a link or variable length attribute. Whenever a single node has more attributes than it can contain, additional 8-word blocks are allocated as extensions of the original node. These extensions are connected in a chain to the original node by the CONT link and are transparent to all the program except for the most basic MEMORY manipulation routines.

There are eight different kinds of nodes in memory, each with its own function. The kinds are TYPE, SIMPLE GENERIC, ACTION GENERIC, CONTEXT GENERIC, SIMPLE INSTANCE, ACTION INSTANCE, CONTEXT INSTANCE and SEMANTIC MODIFICATION. Each has a different purpose in the representation of information in MEMORY. Briefly, their purposes are as follows. The TYPE node is used as the reference point between the WORDLIST and MEMORY. The three GENERIC nodes are used to specify syntactic and semantic information associated with words and substructures given by the INSTANCE nodes. The INSTANCE nodes are used to represent actual instances of words or facts. SEMANTIC MODIFICATION nodes are used to contain information required by the parser to help direct its selection of the modification patterns during parsing of the input. The other distinction made on node types is among SIMPLE, ACTION and CONTEXT. SIMPLE nodes represent single words, ACTION nodes are used to represent triples and CONTEXT nodes are used for prepositional phrases.

TYPE. The TYPE nodes in MEMORY are in a one-to-one correspondence with the entries in the WORDLIST and serve as the reference nodes for the WORDLIST pointers to MEMORY. Some attributes that may appear in a TYPE node are DEFN

and POS. First to appear in a TYPE node is an attribute giving the text representation for the symbol. Any routines, such as the output production programs, can get the text representation for printing directly from the TYPE nodes. This text is repeated here since the symbol given in the WORDLIST is in 6 character chunks linked together -- a form not suitable for printing. The second attribute always present is the DEFN link used to chain together all definitions of the symbol. Only words and punctuation will have non-null chains of definitions. For words, there is a SIMPLE GENERIC node in the DEFN chain for each different word usage. For punctuation, there is a single SIMPLE GENERIC node chained to the TYPE node. System commands and set names have a null set of usages since information of a more detailed nature for them is not required. Figure 3 shows the relationship between TYPE nodes and the WORDLIST and between TYPE nodes and SIMPLE GENERIC nodes.

Among the optional attributes used in a TYPE node is the SYSSET link used to chain together all TYPE nodes which name items in the set. An example of the use of SYSSET is for part of speech. In the TYPE node for the symbol <POS> is the root for the chain through the TYPE nodes for NOUN, PRONOUN, VERB, etc.

The POS link is present in the TYPE nodes for words that name the various grammatical properties (singular, nominal, etc.) and parts of speech (noun, etc.).

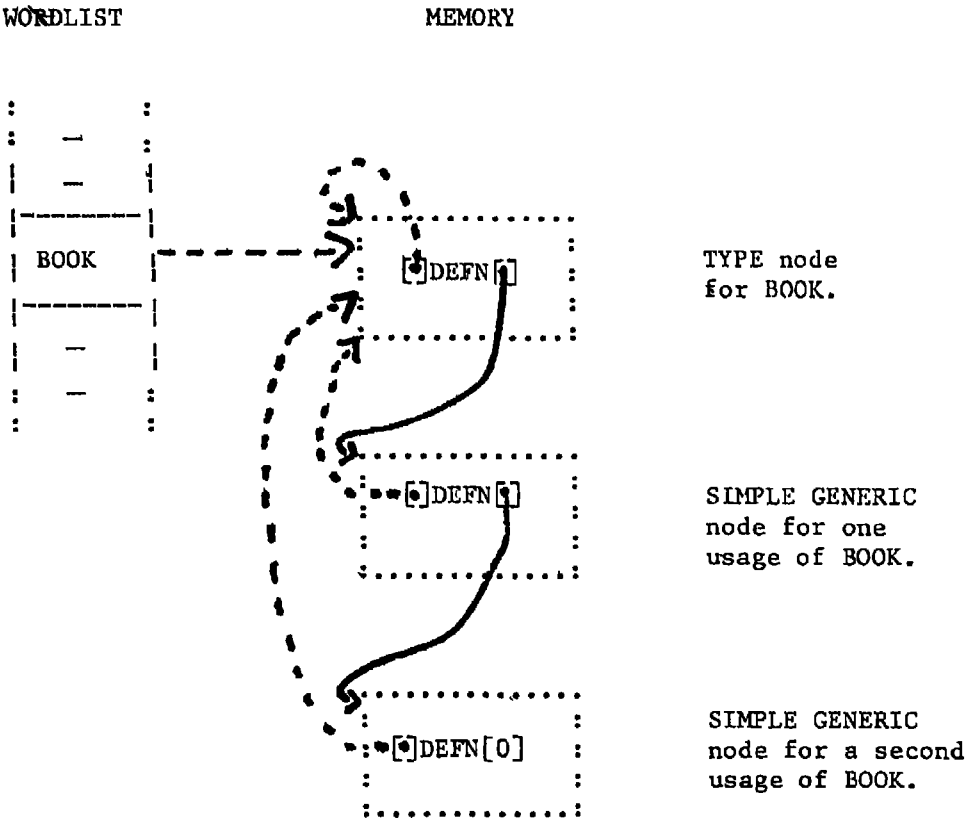

Figure 3: A segment of MEMORY showing the two major functions of the TYPE node.

SIMPLE GENERIC. These nodes are used to represent the different usages for words, i.e., to serve mainly as a source for semantic and syntactic information about a word. A large variety of attributes can appear in a SIMPLE GENERIC node, such as DEFN, INST, ISA, POS, SYSMOD, INFLEC, SYNONYM, ANTONYM, ACTOR, ACT, OBJECT, MODIFY, and PARTS. Of these, only two are required. The DEFN link must be present to tie this usage of the word with its TYPE node and to continue the chain to the next usage, if any. Also required is the part of speech link, POS.

Of the other attributes that can appear, two of the more important ones are the INST and ISA links. The INST link is used to create the chain of specific instances of this word represented by SIMPLE INSTANCE nodes To create hierarchies within the set of SIMPLE GENERIC nodes, the ISA link is used. Although each ISA link belongs to a chain, the presence of two ISA links -- one a root link, the other a son link -- relates the current node to the one immediately above it and the ones below it.

The ACTOR, ACT and OBJECT links in a SIMPLE GENERIC node point to ACTION GENERIC nodes that use the node as an ACTOR, ACT or OBJECT. The PREP and POBJ links, if present, point to CONTEXT GENERIC nodes that use this node as a preposition or a prepositional object. The INFLEC, MODIFY, PARTS, ANTONYM and SYNONYM links point to other SIMPLE GENERIC nodes that are related to the current node in the specified manner.

ACTION GENERIC. These nodes represent the triples used to give the meanings for ACTS. They usually contain three mandatory links. those for the ACTOR, ACT and OBJECT chains. However, for some ACTs, the OBJECT is either not required or is optional. As an example of an ACTION GENERIC node, consider the ACT "know". In the current MEMORY, it has three triples attached to it giving the verb senses of (PERSON OWN THING), (PERSON OWN ANIMAL) and (PERSON OWN SLAVE).

Only two other attributes are allowed in an ACTION GENERIC node. These are the MODIFY and CNTXT links which are used to specify single word and prepositional phrase modification of the ACTION node. When these two links appear in a GENERIC node, they refer to the potential types of modification that may occur.

CONTEXT GENERIC. Nodes of this kind always contain exactly three links: PREP, POBJ and CNTXT. Since this node is used to specify potential types of modification, the PREP and POBJ links are used to point to particular SIMPLE GENERIC nodes for the preposition and prepositional object. The CNTXT link is used to tie this CONTEXT node to the ACTION GENERIC node it modifies.

SIMPLE INSTANCE. A SIMPLE INSTANCE node is present in MEMORY for each distinct instance of each word that has been used anywhere as an ACTOR, ACT, OBJECT, modifier, etc., in the representation of information by ACTION INSTANCE and CONTEXT INSTANCE nodes. There is only one mandatory link in the SIMPLE INSTANCE node, the INST link. However, there are usually several more selected from the set of ACTOR, ACT, OBJECT, PREP, POBJ, and MODIFY depending on the uses to which this particular instance has been put. In the case of all links except MODIFY, the link points to the ACTION INSTANCE or CONTEXT INSTANCE nodes where the current node is used. For MODIFY, however, it can show where this node modifies another or is modified by another depending on whether or not this is the root link.

ACTION INSTANCE. All factual information within the system is represented by ACTION INSTANCE nodes. These nodes are triples that bring together the relations between actual INSTANCES of ACTORs, ACTs and OBJECTs plus their modification.

CONTEXT INSTANCE. The modification of ACTION INSTANCE nodes by prepositional phrases is specified by the use of PREP, POBJ and CNTXT links in nodes of this kind.

SEMANTIC MODIFICATION. The structural information necessary for the parser to determine correct forms of modification is given by variable length attributes that can occur in this kind of node.

4. PARSER

4.1 PARSING STRATEGY

The data structure used to drive the parser is the triple (ACTION GENERIC node) which specifies the semantics for the major components of the parse. By applying the triple as a template to the input, the ACTOR, ACT and OBJECT can be identified.

As the input is parsed, its meaning is converted into a parse network and a network "score" is calculated Usually there are several parse networks constructed from a single input representing different meanings of that input. The best parse is that one which has the highest score from its parse network.

A parse network is created from nodes similar in design and function to those present in MEMORY. Like the MEMORY structures, it is composed of INSTANCE nodes of all kinds: SIMPLE, ACTION and CONTEXT (see Figure 4). These nodes are connected to one another by the same kinds of attributes, e.g., the ACTION INSTANCE node has links to the ACTOR, ACT and OBJECT INSTANCE nodes, the SIMPLE INSTANCE nodes contain MODIFY and CNTXT links to other SIMPLE nodes or CONTEXT nodes, respectively, etc. Because the normal access paths to INSTANCE nodes using DEFN and INST links in TYPE and GENERIC nodes do not exist for temporary nodes, the nodes in a parse network are kept track of by a system of pointers as shown along the right in Figure 4. The entire network is referenced by the pointer word in the upper right hand corner. All references in the temporary nodes of the parse net-

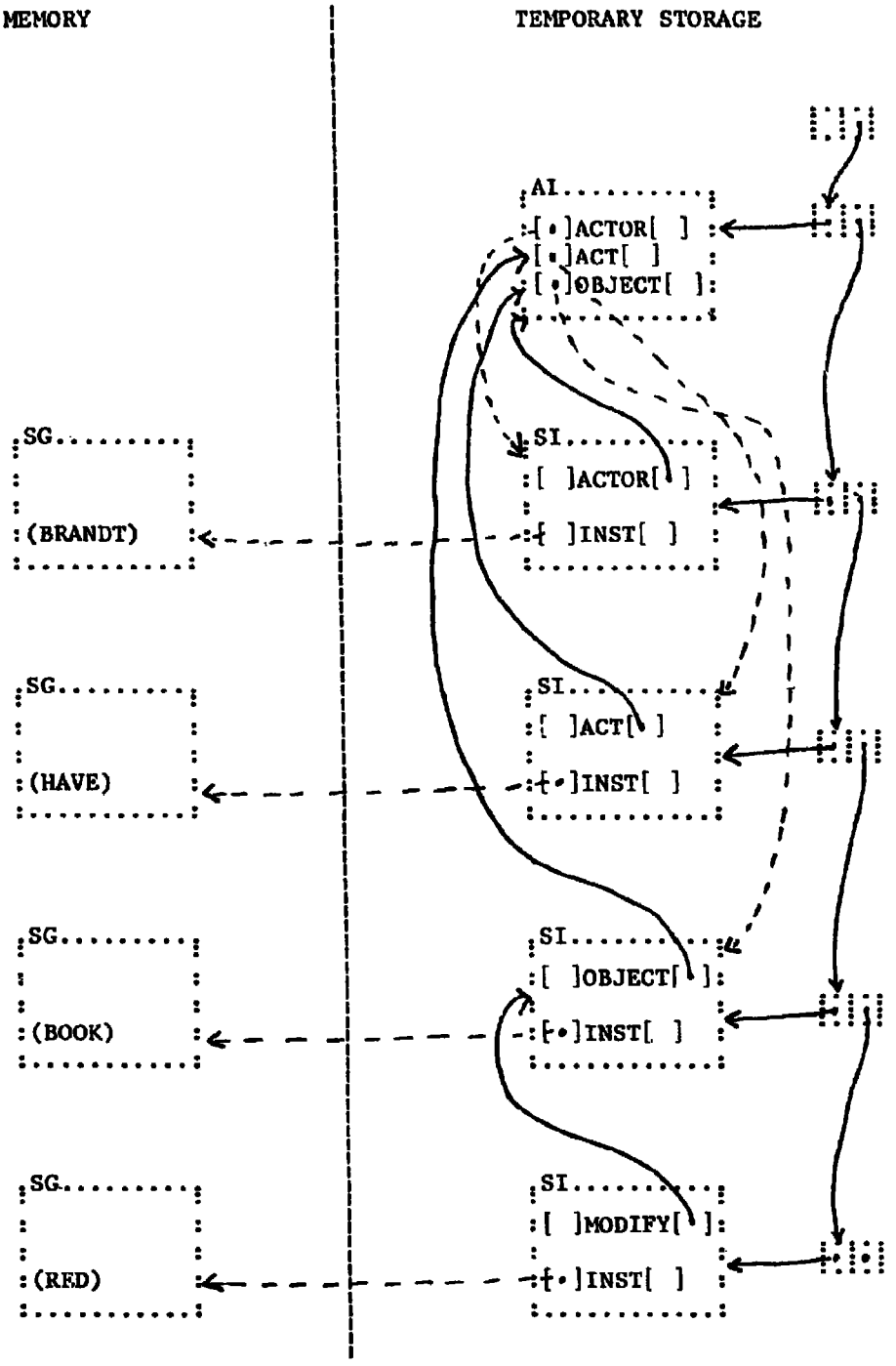

Figure 4: Parse network for "BRANDT HAVE RED BOOK."

work to GENERIC nodes are by links which point to nodes that are part of MEMORY. This relates the input to specific parts of memory as well as provides the required syntactic and semantic properties of the input words for reference during other stages of parsing. When complete, the parse network is in a format identical to similar structures in MEMORY. This is very important later during the matching of input to MEMORY where compatibility between the two is necessary.

4.2 GRAMMAR

4.2.1 ACCEPTABLE INPUT FORMS

The parser has been developed to correctly handle restricted forms of DO-questions and declarative sentences.

<DO-QUESTION>	::= <DO> <ACTOR> <ACT> <OBJECT> ?
<STATEMENT>	::= <ACTOR> <ACT> <OBJECT>
<DO>	::= DO or DID or DOES
<ACTOR>	::= (<left modification>) ACTOR (<right modification>)
<ACT>	::= (<context phrase>) (<adverb>) ACT (<adverb>) (<context phrase>)
<OBJECT>	::= (<left modification>) OBJECT (<right modification>)
<right modification>	::= <prep> (<left modification>) <prepobj>
<left modification>	::= a string of words, usually adjectives, nouns, adverbs and determiners which modify the item to their right.
<prep>	::= one of a set of prepositions specified by an attribute

<preobj>	::= a noun from a particular semantic class as specified by an attribute
<context phrase>	::= essentially the same as <right modification> but is used to modify verbs
<adverb>	::= an adverb from a set of particular words specified by an attribute or an adverb plus its <left modification>.

Table 2: Grammar for DO-questions and statements.

The three components of <ACTOR>, <ACT> and <QBJECT> are identical for the DO-question and statement. The <DO> component is found only in DO-questions. The question mark and period are the only terminating punctuation symbols currently allowed. All of these components are expanded in Table 2 in a BNF-like format.

Additional restrictions currently imposed upon the input are the following:

1. No relative clauses are allowed.

2. No compound units (subject, verb, etc.) are allowed.

Elements in Table 2 enclosed in < > are non-terminal elements of the grammar. Such elements enclosed in ( ) are optional.

In the definition of <ACTOR> and <OBJECT>, the left and right modification is to the immediate left or right of the word being modified. In the case of <ACT>, however, the <adverb> and <context phrase> modification can occur anywhere in the sentence. Usually the single word modifiers are adjacent to the ACT but do not have to be. The <context phrase>s usually appear at the beginning of the sentence or somewhere after the ACT.

In the case of the last five definitions in Table 2, i.e., <left modification> through <adverb>, there are restrictions on the nodes which are applicable at that point in the parsing. Some examples of the types of modification allowed are:

<left modification> of a noun	a blue animal book
<left modification> of an adjective	a very blue sky
<right modification> of a noun	a friend of mine
<context phrase>	in June to the house
<adverb>	yesterday not

4.2.2 SEMANTICS AND SYNTAX

Semantics and syntax have been integrated throughout the parsing procedure so that both work together in the selection of appropriate words and phrases out of which the parse network is constructed. When matching an ACTION GENEPIC node to the input, syntax is used first to identify nouns as potential ACTORs and OBJECTs. Then the semantic acceptability of each is verified by comparing the word with the semantic class specified by the triple.

Syntax is checked by a simple matching of the part of speech. Proper word order within a grammatical subunit is maintained automaticall by the way the modification requirements are set up. A word is deemed semantically acceptable if it matches the semantic class listed as a requirement, or if a word upward in its hierarchy matches the semantic class. For example, suppose the candidate is BRANDT and the required semantic class is PERSON. In the hierarchy containing BRANDT we have BRANDT (ISA) BOY (ISA) PERSON. At that point there is a match on PERSON, so BRANDT would be semantically acceptable.

4.2.3 PRONOUNS, AMBIGUITY AND UNDEFINED WORDS

The currently implemented version of the program provides for only very simple treatment of pronouns. On input, "I" pronouns (I, ME, MY, etc.) are translated to the name of the person talking. "YOU" pronouns are translated to JIMMY3. Similarly, on output, the person's name and JIMMY3 are translated back to "YOU" and "I", respectively.

Ambiguity is not a problem in this model. If two parse networks have identical scores, ambiguity is resolved by the memory matching scheme. All networks are matched against memory in the attempt to locate an answer. If several matches come out equally likely, they can all be reported. This simple-minded approach works well by relating the input to what the program knows.

Throughout the program, undefined words in the input are ignored. However, their text representations are saved so they can be printed later to help explain, say, why the program was unable to interpret the input. As illustrated by one line of the sample conversation in section 1, the response to "DO I HAVE AN OLD BOOK?" was "YES. A RED ONE. I DON'T KNOW WHAT OLD MEANS." The input was interpreted as if it had been "DO I HAVE AN BOOK?".

4.3 PARSING ALGORITHM

There are two operations performed on the input text before it is actually parsed: a) preprocessing of the text in an effort to "standardize" it and b) the determination of the type of input received so the proper parsing technique can be selected.

4.3.1 PREPROCESSING OF THE INPUT TEXT

The first operation performed on the input is the translation of each defined component to its TYPE node pointer to MEMORY. Undefined input items are converted to null pointers. After the translation to TYPE node pointers, the input is checked for standard greetings or cliches that usually elicit a standard response. Examples are "HELLO. HOW ARE YOU?", "HI", "GOODBYE", etc.

The second type of preprocessing is the attempted reduction of words and phrases to simpler forms. This approach can be used for the reduction of slang, misspelled words, idioms, names, etc. Stored in the TYPE nodes for input items that can be reduced are context strings in which an item can occur plus its replacement form.

4.3.2 DETERMINING TYPE OF INPUT

This part of the parsing algorithm is where the kind of input, i.e., DO-question, IS-question, Wh-question, declarative statement, etc., is determined.

Questions can be detected by the presence of the question mark at their end. Discrimination of questions into classes of YES-NO, or Wh- is determined almost completely by the first word of the question. The only questions not correctly classified by this approach are those with inverted order, e.g., "HE ASKED A QUESTION, DIDN'T HE?" and hypothetical questions, e.g., "IF I ASK A QUESTION, WHAT WILL YOUR ANSWER BE?".

Any input that ends with a period and does not begin with a verb is classified as a statement.

Text that does not end with either a question mark or a period is rejected with a request that the person supply punctuation with his input.

4.3.3 PARSING A DO-QUESTION OR STATEMENT

The DO-question and the statement are parsed in identical fashion after the DO-word which begins the question is stripped off and the final punctuation is thrown away.

An exhaustive approach to parsing has been selected for implementation rather that one designed to use prediction coupled with backup facilities. This decision was made partly on technical grounds--the inherent difficulty in implementing such a parser in FORTRAN. A more important consideration, though, was the attractiveness of working with the input as a whole using the matching of templates (GENERIC ACTION nodes) rather than parsing in a serial fashion whereby components are recognized in some left-to-right decoding process.

The algorithm in Figure 5 describes how the parser works.

```
(1)  DO for all possible combinations of usages of the input words:
     .
(2)  . DO for all combinations of reasonable referents for all pronouns:
     . .
(3)  . . DO for each verb in the input:
     . . .
(4)  . . . DO for all GENERIC ACTION nodes for that verb:
     . . . .
(5)  . . . . Find suitable actor and object candidates.
     . . . .
(6)  . . . . DO for all combinations of actors and objects:
     . . . . .
(7)  . . . . . Create a parse network skeleton.
(8)  . . . . . Add modification to the parse network.
(9)  . . . . . Retain network if better than previous one.
     . . . . .
     . . . . END-DO
     . . . .
     . . . END-DO
     . . .
     . . END-DO
     . .
     . END-DO
     .
     END-DO
```

Figure 5: The parsing algorithm

Essentially, the algorithm is set up to produce all possible parse networks using the known meanings of the words.

Steps (7), (8), and (9) are the heart of the parsing scheme. For each triple provided by step (6), a skeleton consisting of an ACTION INSTANCE node and three SIMPLE INSTANCE nodes is constructed as a temporary data structure. The other words in the input are then attached to it according to the following modification scheme.

1. Apply left modification to the ACTOR, i.e., form a noun group that consists of the ACTOR plus all its modification that lies to its immediate left. This includes all adjectives and determiners.

2. Apply left modification to the OBJECT. This process is identical to that used to get left modification of the ACTOR.

3. Find CONTEXT modification of the ACT. Prepositional phrases which modify the main verb are located. As prepositional objects are found, they have their left modification attached by a process identical to that used in steps 1 and 2 for the ACTOR and the OBJECT. Note that no right modification is attempted for objects of prepositions.

4. Find right modification (prepositional phrase) for the ACTOR.

5. Find right modification for the OBJECT.

6. Locate single word (adverb) modification of the ACT. This process works from right to left through all remaining unattached words of the input.

As each modification is identified, nodes are attached to the growing parse network. Simple modification nodes are attached using the MODIFY attribute; phrase modification is constructed using a CONTEXT substructure and attached with a CNTXT attribute.

Upon completing the parsing, the score for the newly constructed parse network is compared with the score for the previous best network. The higher scoring one is retained to the next iteration.

The scheme developed for scoring a parse network is as follows:

1. +3 is added for both the actor and the object when they are identified. A network with a null object would get only +3 for its actor.

2. Add +1 for each single word modifier.

3. Add +1 for each prepositional phrase. Note that this is just the preposition plus its object. Modification of the object scores additional points.

4. After the network is created, subtract +1 for each word of the input, including undefined words, that was not used in the creation of the network.

Upon termination of the algorithm, there will be a "best" parse network which represents the meaning of the input. In case several networks had the same "best" score, then disambiguation of meaning is deferred, to be resolved according to the memory matching process described in section 5.

To detect and control the parsing of "garbage" input, there is a threshold value for the parse network score that must be exceeded before the parse will be accepted. The current threshold value is zero. A parsing that does not exceed this value is rejected and leads to the response of "I DON'T UNDERSTAND THAT."

To illustrate the way the parsing algorithm works, consider the question DOES BRANDT OWN A RED ANIMAL BOOK?

After the DO-question form of the input is recognized, the DOES and the question mark are discarded leaving

BRANDT OWN A RED ANIMAL BOOK

to be processed. In MEMORY, these words have the following usages:

BRANDT - pos=noun; ISA BOY.
OWN - pos=verb; GENERIC ACTION nodes are (PERSON OWN THING) and (PERSON OWN ANIMAL).
A - pos=article.
RED - pos=adjective; ISA COLOR.
ANIMAL - pos=noun.
BOOK - pos=noun; ISA THING.

In this example, each word has only one usage so, in terms of the algorithm in Figure 3, the top level loop (1) will be iterated once. At step (2), there are no pronoun referents to resolve. For the verb "OWN", there are two triples that must be matched to the input.

Using (PERSON OWN THING), in step (5) we compile a list containing BRANDT as its single entry to be used as an ACTOR candidate (BRANDT (ISA) PERSON). Similarly, the set of OBJECT candidates contains BOOK since BOOK (ISA) THING. Now, in step (6), the only possible combination of ACTOR and OBJECT, (BRANDT OWN BOOK) is formed and passed to step (7) where the skeleton of this network is formed.

In step (8) the procedure for adding modification is applied. There is no left modification possible for BRANDT. However, for BOOK, there are three words, ANIMAL, RED and A, to its left that have not been used in the parse so far. All are found to modify BOOK.

The score for this parse network is 9, 3 each for a semantically acceptable ACTOR and OBJECT plus one each for A, RED, and ANIMAL. There are no unused or undefined words in the input so nothing is subtracted.

Now, consider what happens when a second meaning of OWN, (PERSON OWN ANIMAL), is used in steps (5) through (8). Again, the set of suitable ACTORs will be the singleton, BRANDT.

The set of suitable objects contains only ANIMAL. Therefore, the triple, (PERSON OWN ANIMAL), matches (BRANDT OWN ANIMAL) and the node for ANIMAL allows A and RED as modifiers. However, there is no way to attach BOOK to the parse network. This second parsing gets a score of 7 (ACTOR = +3, OBJECT = +3, A= +1, RED = +1, and BOOK = -1).

As a second example, consider the string:

DID BRANDT WILL THE PROPERTY TO YOU?

with

BRANDT	- pos=noun; ISA PERSON.
WILL	- pos=noun; ISA PERSON.
WILL	- pos=verb; GA node is (PERSON WILL THING). (TO PERSON) is optional context modification.
THE	- pos=article.
PROPERTY	- pos=noun; ISA THING.
TO	- pos=preposition.
JIMMY3	- pos=noun; ISA PERSON.

One possible set of definitions (usages) of the words with WILL as a noun will not include any verbs. Therefore, that combination is rejected immediately in step (3) of the parsing algorithm. The other possible set contains WILL as a verb.

Once that set of usages is decided on, the parsing is straight-forward in the manner similar to that used in the previous example. The major difference in this input is the existence of the phrase "TO JIMMY3 (as transformed from "TO YOU" in step (2)). Context modification for the ACT is searched for before right modification of the OBJECT so "TO JIMMY3" is properly attached to WILL in the parse. However, the node for PROPERTY did not contain any patterns for right modification so that phrase could not have been attached to the OBJECT anyway. The complete parse for this input has a score of 8.

5. MEMORY SEARCHING

5.1 OVERVIEW

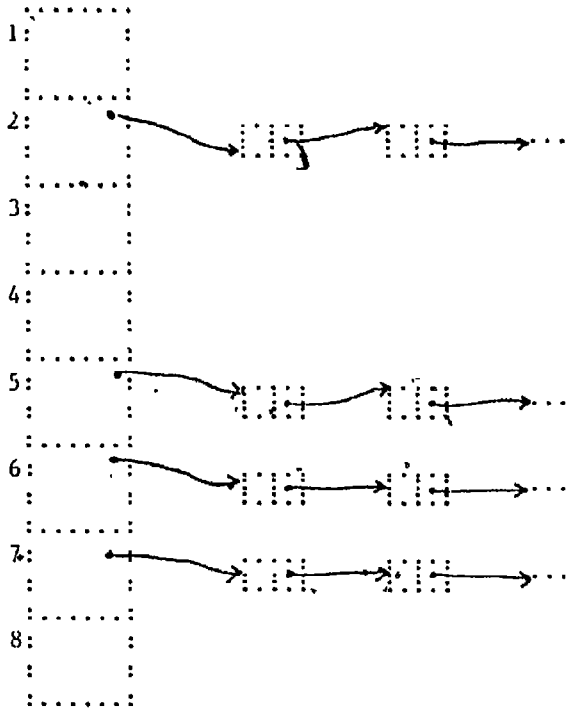

Figure 6. General form of a memory match structure.

The strategy used in MEMORY searching is similar in one respect to the parsing procedure. Namely, during the search process a structure is constructed and a score is calculated to measure the degree of similarity between the question and the candidate answer. However, unlike the exhaustive process used in obtaining a parse, the memory searching procedure is rather selective and does not examine all, or even a large part, of MEMORY.

5.2 THE MEMORY MATCH STRUCTURE

5.2.1 GENERAL STRUCTURE

For each match attempted between a parse network and an ACTION INSTANCE node in MEMORY, a complete, new memory match structure is generated. This structure has a static component of 8 registers plus anywhere from zero to four variable length, linked lists attached to it at various places -- (see Figure 6). These linked lists are used for collecting information about word modification.

5.2.2 ACTOR, ACT AND OBJECT COMPARISON RESULTS

Three of the registers, corresponding to the ACTOR, ACT and OBJECT comparison results, form the heart of the memory match structure. In each register is recorded the exact kind of match between input and MEMORY.

Three values are contained for the ACT comparison results: a pointer to the SIMPLE GENERIC node in MEMORY for the ACT given in the parse network, a pointer to the list which has the comparison between the modifiers of input and of the MEMORY structure, and an indicator of the type of match. Five possible types of matches can occur for ACTs.

1. (+0) No match.
2. (+5) Exact match.
3. (+4) The input and MEMORY ACTs are synonyms.
4. (+4) The input and MEMORY ACTs are antonyms.
5. (+0) The ACT was missing from either the input or the MEMORY node.

The registers for ACTOR and OBJECT are identical. Like the ACT, they contain three items of information: a pointer to the SIMPLE GENERIC node for the ACTOR in the parse network, a pointer to the modifier list and the match type indicator. The match type indicator for the ACTOR is divided

into three subunits. First, the number, singular or plural, of the ACTOR is given. Second, the type of reference to the input ACTOR is recorded as either a specific instance (i.e., it referred to a particular instance of the ACTOR) or as an indefinite reference (referred to a class of ACTORs rather than a specific one). Finally, there is the result of the match between input and MEMORY.

1. (+0) No match.

2. (+5) Exact match.

3. (+4) The input ACTOR is a member of the set named by the MEMORY ACTOR, i.e., input ACTOR (ISA) MEMORY ACTOR.

4. (+4) The input ACTOR is the name of a set for which the MEMORY ACTOR is a member, i.e., MEMORY ACTOR (ISA) input ACTOR.

5. (+4) Input ACTOR matched a synonym of the MEMORY ACTOR.

6. (+4) Input ACTOR matched an ACTOR of another node in MEMORY of the form "ACTOR1 BE ACTOR2" where ACTOR1 matched the input ACTOR and ACTOR2 matched the MEMORY ACTOR, or vice versa, i.e., input ACTOR BE x and x BE MEMORY ACTOR.

7. (+2) Did not find an instance of the input ACTOR in MEMORY but did find an instance of a member of the set the input ACTOR would belong to if it had been in memory, i.e., the two relations of input ACTOR (ISA) x and MEMORY ACTOR (ISA) x both hold for some suitable category x.

8. (+0) ACTOR missing from either input or MEMORY.

5.2.3 WORD MODIFICATION RESULTS

Registers of the match structure corresponding to the ACTION, ACTOR, ACT and OBJECT nodes can all have modifier lists attached.

The word modification list is singly linked; each item on the list is given by two registers and represents a single modification, either single word or phrase. All words or phrases that modify, say, the ACTOR, will be on the same list. However, if any of those words is modified by a

word or phrase, then it will have a list attached containing all its modifiers. Thus, for any component of the input, word modification is really a tree of sublists whose structure is determined by the input word relations.

Six results of the match of modifiers are possible.

1. (+0) Was not compared because previous level modification did not match.

2. (+2) Exact match.

3. (+2) Exact match if inflections are ignored, e.g., singular matching plural.

4. (+1) One of the two modification words is a member of the set named by the other, i.e., there is an ISA chain leading from one to the other.

5. (+1) The two words are both from a set of mutually exclusive elements, e.g., matching RED with BLUE.

6. (+0) No match because the modifier appeared in only one of the two (input and MEMORY) places.

5.2.4 EXAMPLE STRUCTURES

Consider the question:

(1) DOES A PERSON HAVE A RED BOOK?

This will be parsed as (PERSON HAVE BOOK) with RED modifying BOOK. The two indefinite articles will also be part of the parse network but are not matched since articles are not included in any memory structures. The matching of this parse network with a MEMORY structure for

(2) BRANDT OWNS A RED BOOK

would yield a memory match structure with the following properties.

Match score = 15; Maximum possible score = 17

MEMORY ACTOR = BRANDT (PERSON).
Input mode = singular, indefinite.
MEMORY mode = singular.
Match type = MEMORY (ISA) input. (+4 points)

MEMORY ACT = OWNS (HAVE).
Match type = synonym (+4 points)

MEMORY OBJECT = BOOK (BOOK)
Input mode = singular, indefinite.
MEMORY mode = singular.
Match type = exact match. (+5 points)
Modifiers:
RED - Location = input and MEMORY
Match type = exact match. (+2 points)

If (1) above is matched against the MEMORY structure representing

(3) BRANDT HAS A BLUE ANIMAL BOOK

The following memory match structure will result.

Match score = 15; Maximum possible score = 17

MEMORY ACTOR = BRANDT (PERSON)
Input mode = singular, indefinite.
MEMORY mode = singular.
Match type = MEMORY (ISA) input. (+4 points)

MEMORY ACT = HAVE (HAS)
Match type = exact match (inflections are ignored) (+5 points)

MEMORY OBJECT = BOOK (BOOK)
Input mode = singular, indefinite.
MEMORY node = singular.
Match type = exact match. (+5 points)
Modifiers:
RED - Location = input and MEMORY. (+1 point)
(Note: RED matches BLUE as mutually exclusive elements from the same set.)
ANIMAL - Location = MEMORY only. (+0 points)

This second MEMORY node matches the input as well (+15 score) as the first because of the slightly better ACT match even though the OBJECT is closer in the first case. It serves to indicate some of the problems that are encountered by the procedure during matching.

```
(1)  DO for all parse networks:
     .
(2)  .  Compute maximum possible match score for this network.
(3)  .  Don't search for this network if maximum is not good enough.
(4)  .  Get list of synonyms and antonyms for parse network ACT.
     .
(5)  .  DO until a reasonable match has been obtained or until no more
     .  .     ACTORs can be found:
     .  .
(6)  .  .  Select an ACTOR to match on.
     .  .
(7)  .  .  DO for all INSTANCES of that ACTOR:
     .  .  .
(8)  .  .  .  DO for all ACTION INSTANCES which have that ACTOR:
     .  .  .  .
(9)  .  .  .  .  Create the skeleton for a memory match structure.
(10) .  .  .  .  Compare modifiers of the ACTION INSTANCE nodes.
(11) .  .  .  .  Compare ACTORs and their modifiers.
(12) .  .  .  .  Compare ACTs and their modifiers.
(13) .  .  .  .  Compare OBJECTS and their modifiers.
(14) .  .  .  .  If no good OBJECTS, then try alternates.
(15) .  .  .  .  Accumulate the total match score for the structure.
(16) .  .  .  .  Add structure to the list for that ACTOR INSTANCE.
     .  .  .  .
     .  .  .  END-DO
     .  .  .
     .  .  END-DO
(17) .  .  Save best match structures for that ACTOR.
     .  .
     .  END-DO
(18) .  Further prune the set of best match structures
     .
     END-DO
```

Figure 7: The memory search algorithm.

5.3 MATCHING MEMORY

5.3.1 BASIC ALGORITHM

The algorithm used to search memory (see Figure 7) examines a limited subset of all structures in MEMORY while trying to match the input. The search is restricted to ACTION INSTANCE nodes in MEMORY that have either the same or a closely related ACTOR. The object of the algorithm is to obtain a small set of the best matches of the parse network with an ACTION INSTANCE node in MEMORY.

To handle ambiguous input, i.e., multiple parse networks, the matching procedure must be repeated for each network passed on by the parser (see step (1)). Unpromising networks are eliminated in steps (2) and (3). Synonym and antonym lists are compiled in step (4).

The termination criteria for MEMORY searching is the discovery of a suitable match or the exhaustion of the set of suitable ACTORs used to direct the search. The adequacy of the match between input and MEMORY is the memory match score--the accumulation of many component scores which measure the similarity of corresponding parts of two structures. The termination criterion for a suitable match is based on the value of this score relative to the network's maximum possible score determined in step (2). The threshold for a "suitable" match is currently set at 70 percent of the maximum possible score. The number 70 is not perfect in any sense but was selected in a trial-and-error fashion.

5.3.2 SELECTING SUITABLE ACTORS

The selection of actors to control the range of the search procedure is designed to provide a reasonable set of nodes closely related to the input ACTOR. This selection procedure is used only when no ACTION INSTANCE nodes with the input ACTOR produce sufficiently good matches.

There are five alternate methods, described below, for getting new ACTOR candidates. Not all of these five are always used, however. The ones to use and the order in which they are to be applied is determined by the mode and number of the input ACTOR.

1. Search memory of ACTION INSTANCE nodes of the form "ACTOR1 BE ACTOR2" where either ACTOR1 or ACTOR2 matches the input ACTOR exactly. Collect the unmatched members of all these nodes for use as new ACTOR candidates. For example, suppose the ACTOR, BRANDT, was not successful at generating a good match. Search for INSTANCES of (X BE BRANDT) and (BRANDT BE X) where X is in the same general hierarchy as BRANDT, e.g., (BRANDT IS SECOND-GRADER). Now SECOND-GRADER can be used as a source for more ACTION INSTANCE nodes to search.

2. Use all nodes above the ACTOR in its hierarchy.

3. Use all nodes below the ACTOR in its hierarchy.

4. Use all synonyms of the input ACTOR.

5. Find all INSTANCE nodes that are in the same set as the input ACTOR, i.e., search the other nodes in the ISA chain rooted in the node immediately above the input ACTOR in its hierarchy. For example, for an input ACTOR, BRANDT, we have BRANDT (ISA) BOY. Search the chain which gives all subsets of BOY, but excludes BRANDT.

For specific mode input ACTORs, the above procedures are executed in the order: 1, 2, 4, 5. Procedure 3 is not used for specific input ACTORs since its purpose is to find specific instances for general references. For indefinite input ACTORs, the procedures are executed in the order: 1, 2, 3, 4. Procedure 5 is not used since it would lead to too diversified a set of potential candidates. For instance, consider PERSON as the input ACTOR. If we had PERSON (ISA) THING, and ANIMAL (ISA) THING, ACTORs could be selected that are only vaguely related to the input.

This set of procedures can be executed twice. The first time, the input ACTOR is used as it appeared in the input. The second time, its number is

changed, i.e., for a singular ACTOR, the second time through, all comparisons and searches would be performed for its plural.

In order for this procedure to work, the mode of the input ACTOR must be known. This is determined as follows:

1. An ACTOR is indefinite if
   a) it is an indefinite pronoun,
   b) it has modifiers but none is a definite determiner (the, this, these, that, those) or
   c) it is plural and has no modifiers.

2. An ACTOR is specific if
   a) it is modified by a definite determiner or
   b) it is singular and has no modifiers.

6. PRODUCTION OF OUTPUT

6.1 OVERVIEW

Problems to be solved before "natural" sounding output can be produced are

1. what information should be used in the response to a given input,

2. how should the response be structured so certain parts are properly emphasized and

3. how can the response be made to appear natural in the sense of being like a similar remark a person might make?

Clearly, there are many factors which play a part in how a person decides what to say at any time during the course of a conversation. Such factors include the place of occurrence, the reason for the conversation, the roles assumed by the participants, the type of information passed and the motives of the participants. These factors have not been studied, indeed, have not even been exhaustively identified, during this research. What has been done, however, is the development of a few simple procedures that will generate reasonable sounding answers to DO-questions based on the results of memory search.

6.2 THE OUTPUT PRODUCTION LIST

The only temporary structure created during the output phase is a simple doubly linked list of elements representing the words and punctuation of the answer to be printed (see Figure 8).

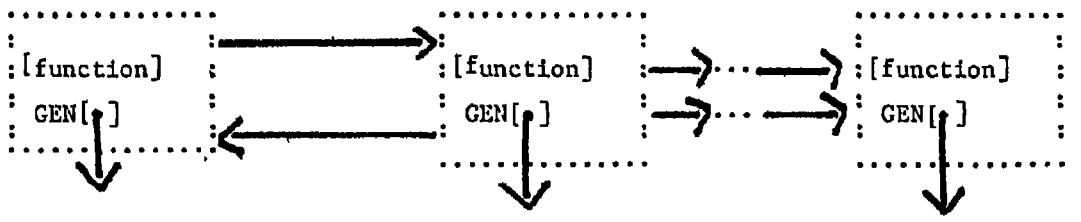

Figure 8: Structure of the output production list

As the output production routines analyze the relevant memory match structures, they produce the output production list an element at a time. Each element of the list has the following two register format:

> Register 1 - contains the forward and backward links to the other elements in the list.
>
> Register 2 - contains a word function indicator (represented as [function] in Figure 6) and a MEMORY pointer to the GENERIC node for the word (given as GEN [ ]). The pointer can be used by the final print routines to trace back through the word's GENERIC node to its TYPE node to retrieve its text representation.

The word function is an indicator to help out the routine that makes the output grammatical. Currently 10 different function codes which correspond very loosely to sentence parts are used.

1. Subject.
2. Verb.
3. Object.
4. Prepositional object.
5. Preposition.
6. Modifier of subject.
7. Modifier of verb.
8. Modifier of object.
9. Modifier of prepositional object.
10. Other (includes punctuation).

6.3 PRODUCTION METHOD

6.3.1 RESPONDING TO INPUT NOT UNDERSTOOD

When the input is not understood or no answer has been found in memory, enough information should be returned so that the person knows why he did not receive the expected response. The procedure that is used to detect that situation and produce a response currently works as follows:

1. If input was not parsed, respond with "I DON'T UNDERSTAND THAT." Then print all undefined words in the form: I DON'T KNOW WHAT word1 (OR word2 OR word3 ...) MEANS." For example, the input "DOES BRANDT RIDE A BICYCLE?" would generate the response "I DON'T UNDERSTAND THAT. I DON'T KNOW WHAT RIDE OR BICYCLE MEANS."

2. If the input is parsed, but there was a poor, or no, memory match, then check the GENERIC node for the ACT to see if we respond with "YES", "NO", "I DON'T KNOW" or some "canned" response. What is searched for is an attribute which is sometimes present in ACT nodes and is the answer to be given when no match is found with MEMORY. An example is the ACT "KNOW". If the program is asked "DO YOU KNOW x?" and x is not in MEMORY, then the response will be "NO." rather than "I DON'T KNOW."

If the input is parsed and there is a good memory match then the following steps will be executed.

3. Check the OBJECT. If there is none and one is required by this ACT then, if there are undefined words, print "I DON'T UNDERSTAND THAT." plus the undefined words. If there are no undefined words, print "I DON'T UNDERSTAND YOUR USE OF act." Substitute the current ACT in the output for "act".

4. Check the ACTOR. If it is not directly related to the input ACTOR but is a member of the same set (i.e., it was selected during the memory search for producing actors as given in section 5.2.3), print "I DON'T KNOW." However, in this case, do not terminate the output production here. Pass the good memory match structure involving this actor on to the next procedure to be used to produce more output. As an example, the input "DOES BILL HAVE A BOOK?" would generate the response "I DON'T KNOW. BRANDT HAS A RED BOOK."

5. Check the ACT. If it does not match, print "I DON'T KNOW." plus the undefined words, if any.

6.3.2 DETERMINING MODE OF RESPONSE

The answer to a DO-question has an optional interjection of YES or NO which precedes the answer and is determined as follows:

1. Set the mode to YES unless the MEMORY ACTOR is a subset of the input ACTOR. In that case, the response should not have either YES or NO. This will occur when specific data are being used to answer a question of a general nature. For example, the input "DO PEOPLE OWN THINGS?" would be answered by "BRANDT OWNS A RED BOOK." The YES is omitted since the question has not been answered in general. Do not continue examination of the match for contradiction when this occurs.

2. Search the memory match structure looking for contradictory data that would change the mode from YES to NO. If the input ACT matched an antonym, set the mode to NO.

3. Check the first level modifiers of the ACT. Look for contradiction, i.e., modifiers from the same set that are mutually exclusive. If any are found, reverse the setting of the mode. For example, the input "DID BRANDT PLAY OUTSIDE?" which would match the MEMORY node, "BRANDT PLAYED INSIDE." would have a negative mode since INSIDE and OUTSIDE are mutually exclusive.

4. Check the OBJECT for conflicting modifiers. Also change mode if the OBJECT is not directly related to the input OBJECT but is an element from the same set as the input OBJECT. This will be the case when matching objects such as GREEN BOOK with YELLOW BOOK or OLD BOOK with NEW BOOK.

5. Finally check all single word modifiers of the ACT that were present only in the input or MEMORY looking for negative modifiers. These would be words like NOT, NEVER, etc.

6.3.3 PRODUCING A NORMAL ANSWER

At the top level in this procedure is the decision of how much of the answer to print. The rules used are: a) if the answer is an exact match of the input with respect to the three major components, don't print anything except for the YES or NO, b) if the answer is an exact match except for the OBJECT, then print only the OBJECT or c) if the ACTOR or ACT does not match, then print the whole MEMORY node given by the memory match structure.

The list elements representing the output to be printed for the ACTOR or OBJECT are generated by the following procedure.

1. Add the generic node for the ACTOR (OBJECT) to the end of the output production list. Set its function type to SUBJECT (OBJECT).

2. Put the modifiers into the list immediately in front of the AGTOR (OBJECT). Use only modifiers that were present in the MEMORY node or that matched between MEMORY and input. These modifiers are contained in the linked list attached to the memory match structure for the ACTOR (OBJECT).

3. Put an article before the modifiers, if required.

4. Add prepositional modification of the ACTOR (OBJECT) at the end of the list. This is accomplished by adding the preposition followed by the prepositional object. Then the modification on the prepositional object is added between the two.

5. If the ACTOR (OBJECT) is the same for both input and MEMORY but some input-only modification exists, then add "BUT NOT" plus the input node and the input-only modifiers. An example of this is where "A BLUE BOOK BUT NOT AN ANIMAL BOOK" is given in response to "DO I HAVE A BLUE ANIMAL BOOK?"

6.3.4 MAKING OUTPUT GRAMMATICAL

The procedure used to fix up the output examines the elements of the output list and, using the word functions as specified in those elements and the properties given in the words' GENERIC nodes, attempts to apply the four rules below:

1. JIMMY3 and the person's name get translated to the pronouns "I" and "YOU". At this stage, the form of the pronoun may be wrong.

2. Get person and number of the pronouns to agree with the ACT (main verb). Change a pronoun to its possessive form if it is used as a modifier of another sentence element.

3. Set the proper verb tense.

4. Convert the objects in the sentence (main OBJECT and prepositional objects) to objective case.

A step in making the output grammatical that was given before was the generation of an article as a modifier. An indefinite article (A or AN) is always used; the one to be selected is given by the following rules:

1. Don't use an article if the word modified is a pronoun or a proper name.

2. Otherwise, select A or AN according to the first letter of the word it will precede.

Following the completion of this operation the output is printed.

7. DISCUSSION

7.1 RESULTS

7.1.1 OBJECTIVES MET

The objectives of this research, as restated from section 1, are the development of three components able to carry on a "natural" dialogue with humans: an extendible memory model, procedures for determining the meaning of the input and procedures to allow the model to converse "naturally" with a human.

The current design of memory, although a start in the right direction, is far from complete. There is much that cannot now be represented with the structures available. However, the design of memory is flexible. Extensions can be added to represent more complex surface structures via INSTANCE nodes and also to capture more of the underlying structure of the word meanings themselves in terms of GENERIC nodes.

Other features that an extended memory model will need are the capabilities for supporting more elaborate question answering, the processing of imperatives and the integration of new information into the

existing structures. We believe that the current design will allow such procedures to be developed.

The parser now implemented does a reasonably good job on the restricted input given it. The basic parsing philosophy, i.e., the selection of one of several possible networks based on semantic density, is believed to be a sound way to approach the problem of determining meaning.

The development of output generation procedures has just started. The currently implemented procedures are not general or powerful enough to handle more than a few specialized situations. Work is required in this area to first develop a large number of specific rules from which more general rules can be deduced.

We have identified ten features which characterize "natural" language understanding processes. Specifically, they are the ability to:

(1) Work with partial information (to make plausible inferences about missing information, e.g., default values from frames),

(2) Work with overlapping and conflicting information (not to reject it out of hand, or seek only consistent information, or to assign all truth value F, but to sift through it to reject that which - based on experience or knowledge - is implausible or to temporarily suspend judgement),

(3) Retain ambiguity until disambituation is absolutely called for,

(4) Perform "short" chains of deductions,

(5) Engage in common sense reasoning (i.e., there is knowledge of the properties of commonplace objects, events, etc.),

(6) Pose questions in order to confirm expectations or to elicit more information about some subject matter of personal interest,

(7) Construct, modify and extract information from a model of the intentions, interests, skills, motivations, etc., of the other party,

(8) Interrogate and update a similar model of the attitudes, beliefs, abilities, goals, etc., of itself,

(9) Be sensitive to the plausibility of information received (see Norman's "Charles Dickens" problem, discussed later),

(10) Be aware of the context in which the conversation is occurring (Norman's "Empire State Building" problem, discussed below, identifies one kind of context; the roles of the parties involved, e.g., parent-child, teacher-student, superior-subordinate; bureaucratic official-client, is a second context; other context types are no doubt present).

With JIMMY3 we have only begun to address this list of attributes. In particular, JIMMY3 illustrates one approach to coping with items #2 and #3. Other programs pose avenues of attack for other items of the list (e.g., #1, #4, #5), but the extendability of those programs to other features listed is questionable - as it is for JIMMY3. However, by grappling with what we consider to be the fundamental problem of designing the system from the start to be extendable, we believe JIMMY3 can be grown to cope with other features on that list, and thus approach being a "natural" language understanding system.

7.1.2 DOES THE PROGRAM REALLY UNDERSTAND?

The answer to this question is probably no. In the more restricted sense of, can the program relate the input to something it already knows, the answer is yes. All input surface structures are matched during the question answering phase to similar structures in memory. However, given a single input in isolation, the program does not understand what it means. It has no definitions for individual words. It is true that there are relationships between words via the chains and hierarchies but this is not enough. Knowing that BRANDT (ISA) PERSON does not help at all in understanding if the model does not what a PERSON is.

7.2 LIMITATIONS

A majority of the model's general limitations are the result of deficiencies in the memory structure. For instance, certain types of input cannot be parsed because of inadequate structural building unit of memory.

7.2.1 MEMORY MODEL

If the current philosophy of what is to be stored in memory is maintained, i.e., only surface structures are to be represented, then the most serious limitation is the inability to represent more kinds of strings of English. Nothing more complex than simple (ACTOR ACT OBJECT) facts can now be recorded. In general, there is no way of relating different facts to show causation, result, presupposition, etc.

If we look at memory as more than a place in which surface strings are stored, then there are many shortcomings. Nowhere in the current structure are words given definitions. As for ACTs, the original design called for the building of hierarchies of ACTs in a manner similar to hierarchies of "things". Instead ACTs could be broken down to some basic primitives of action. Different ACTs can then be compared, not by looking at their hierarchies, but by comparing their common primitive actions.

7.2.2 PARSER

The single major limitation of the current parser is its inability to parse anything other than simple DO-questions and statements. This is not a theoretical shortcoming; it simply a question of implementation effort. However, there are problems with the parser irrespective of this major limitation. These problems are quite often related to deficiencies in the memory structure.

The parser will currently fail if it does not find a semantically acceptable parse. In cases such as this, it should be allowed to find, instead, a syntactically acceptable parse that would be marked as a semantic anomaly.

Undefined words are now ignored. The ability to make reasonable predictions about the function and meaning of undefined words would be desirable.

7.2.3 OUTPUT PRODUCTION

The most serious limitation in the production of output has been the lack of time to develop more extensive procedures. Examples of new rules that are not yet implemented that could be used to produce answers are the following:

1. Use more than one memory match structure to produce the answer. Currently only the best one is used. For example, the question "DOES ANYONE HAVE A RED TOY?" could be answered by combining the two memory matches, "BRANDT HAVE RED BOOK" and "JENA HAVE RED BALL," to get "YES. JENA HAS A RED BALL AND BRANDT HAS A RED BOOK."

2. Several equally good memory matches could be combined to produce a single answer which contains a compound actor or object. The question "DOES BRANDT HAVE A BOOK?" matches two memory structures equally well. These two matches, "BRANDT HAVE RED BOOK" and "BRANDT HAVE BLUE BOOK," could be combined to form the response "YES. HE HAS A RED ONE AND A BLUE ONE." Note that work also needs to be done with respect to pronoun substitution (i.e., ONE for BOOK) before the above response could be produced.

3. If the memory search fails and there was no actor in memory, this should be reported explicitly rather than saying "I DON'T KNOW." For example, consider the question "DOES BILL MAXWELL HAVE A SON?" If BILL@MAXWELL is not present in MEMORY, report "I DON'T KNOW BILL MAXWELL." However, if the memory match procedures were successful in finding generic information such as "MEN HAVE SONS" then the phrase "BUT HE COULD HAVE A SON." could also be generated.

Other limitations are the inability to handle definite articles or to attach modifying clauses or phrases to help distinguish parts of output. The production procedures allow only superficial treatment of the ACT.

Typical of simple questions that currently cannot be answered is "DOES BRANDT HAVE TWO BOOKS?". There is no mechanism for counting or comparing occurrences of relevant facts in memory during the search process. nor of using them in the output production phase.

7.3 IGNORED PROBLEMS

7.3.1 INTENTIONS AND MOTIVATIONS

The most serious problem is the lack of any modelling of intentions and motivations of the person talking with JIMMY3 (or of JIMMY3 itself). In order for a dialog to be sustained for any length of time with a sense of continuity, models of the person and of JIMMY3 are required. Two psychological models should be maintained by the program to indicate 1) what the program thinks about itself and the person and 2) what the program thinks the person thinks about himself and the program. At all times during the conversation, information in the models will specify for both the person and the program what each a) wants to know, b) wants to tell, c) already knows, d) feels, e) believes, etc., with respect to the context of the conversation up to that point. Also required will be information about the motives and physical and mental attributes and capabilities of each participant from each point of view. Integration of these models into the workings of the parsing, memory searching and output

production phases of the program will be a necessary step towards more complete natural language understanding.

For example, to continue that hypothetical dialogue of section 1, one would like to be able to see some exchanges such as:

H: Was the window expensive to repair?
NCP: Why do you ask?

H: Why do you want to know?
NCP: I fear I may be responsible for the debt.

H: I thought that might be the case.
NCP: You mean I do owe you money?

H: No, it's just that I regret making you feel uncomfortable.
NCP: Then why did you ask if the window was expensive to repair?

H: To test your power of deduction.
NCP: You really don't seem to understand me.

Computer analysis of such complex interchanges is dependent upon the existence of psychological models of both parties. Successful realization is probably many years off.

7.3.2 NORMAN'S PROBLEMS

With respect to answering capabilities in language understanding understanding models, Norman [9] has presented several basic problems that should be considered. These are illustrated as: 1) the telephone number problem, 2) the three drugstores problem and 3) the Empire State Building problem.

The first problem concerns the appropriate response to the question: "What is Charles Dickens' phone number?" The normal human response of "That's a stupid question." or "Phones weren't invented then. requires the action of a plausibility check on the question before an attempt is made to find an answer.

The three drugstores problem considers what the correct reaction should be to the statement: "I went to three drugstores." This is really a problem of integrating current with past knowledge and of determining presuppositions to the new data. For instance, the program should note (or ask) Why, didn't the other two drugstores have what you were looking for?

The Empire State Building problem refers to the context and scope of any particular question. The question, "Where is the Empire State Building?" requires different answers depending on the context of the conversation. To paraphrase Norman's response: If asked the question in Russia, the answer most likely would be "In the United States."; if asked by an adult in Europe, "In New York City."; if asked by someone in New York City, then "On 34th Street." would be appropriate.

From these examples, it is obvious that much more is involved in developing a language understanding system that answers questions "naturally" than just the ability to parse input correctly, look up an answer and report it. We hope we have more clearly identified some of the desirable characteristics, and we hope we have illustrated some progress toward the ultimate goal. If JIMMY3 allows other problems from our list to be addressed also, we shall feel fortunate, indeed.

REFERENCES

[1] Anderson, John R. and Gordon H. Bower, Human Associative Memory, Wiley and Sons, 1974.

[2] Bobrow, Daniel G., and Allan Collins (eds.), Representation and Understanding, Academic Press, 1975.

[3] Brown, John Seely and Burton, Richard R., "Multiple Representations of Knowledge for Tutorial Reasoning," in [2].

[4] Carbonell, Jaime R., "AI in CAI: An Artificial Intelligence Approach to Computer-Aided Instruction," IEEE Transactions on Man-Machine Systems, Vol. MMS-11, 1970, pp. 190-202.

[5] Colby, Kenneth M., Artificial Paranoia: A Computer Simulation of Paranoid Processes, Pergamon Press, 1975.

[6] Lindsay, Robert,K., "In Defense of Ad Hoc Systems," in [15].

[7] Newell, Allen, "Heuristic Programming: Ill-Structured Problems," in J.S. Aronofsky (ed.), Progress in Operations Research, Wiley, 1969.

[8] Newell, Allen, et al., Speech Understanding Systems, American Elsevier, 1973.

[9] Norman, Donald A., "Memory, Knowledge, and the Answering of Questions," in R.L. Solso (ed.), Contemporary Issues in Cognitive Psychology: The Loyola Symposium, Winston, 1973.

[10] Norman, Donald A., David E. Rumelhart and the LNR Research Group, Explorations in Cognition, Freeman and Co., 1975.

[11] Quillian, M. Ross, "Semantic Memory," in Marvin Minsky (ed.), Semantic Information Processing, MIT Press, 1968.

[12] Rustin, Randall, Natural Language Processing, Algorithmics Press, 1973.

[13] Schank, Roger, "Conceptual Dependency: A Theory of Natural Langauge Understanding," Cognitive Psychology, vol. 3, 1972, pp. 552-631.

[14] Schank, Roger, "Identification of Conceptualizations Underlying Natural Language," in [15].

[15] Schank, Roger and Kenneth Colby (eds.), Computer Models of Thought and Language, Freeman, 1973.

[16] Weizenbaum, Joseph, "ELIZA: A Computer Program for the Study of Natural Language Communication between Men and Machine," Communications of the ACM, Vol. 9, 1967, pp. 36-54.

[17] Wilks, Yorick, "An Artificial Intelligence Approach to Machine Translation," in [15].

[18] Wilks, Yorick, "The Stanford Machine Translation Project," in [12].

[19] Winograd, Terry, "Understanding Natural Language," Cognitive Psychology, Vol. 3, 1972, pp. 1-191.

[20] Woods, William, "An Experimental Parsing System for Transition Network Grammars," in [12].

FIFTEENTH ANNUAL MEETING

ASSOCIATION FOR COMPUTATIONAL LINGUISTICS

WASHINGTON, D. C.

MARCH 16-17, 1977

ABSTRACTS

The decision to hold the 1977 meeting of the Association in March was taken only in October 1976. The six-month interval has not allowed advance distribution of abstracts through the Journal. The abstracts accepted for presentation are published here for the record, and for the use of members who could not attend the meeting.

Some or all of these papers may be published in future issues.

The program, on frames 72 and 73, gives frame numbers for all abstracts.

ASSOCIATION FOR COMPUTATIONAL LINGUISTICS

Fifteenth Annual Meeting
Georgetown University, Washington D.C. 16-17 March 1977
Palms Lounge, Walsh Building; 36th Street between N and Prospect

PROGRAM

Wednesday, 16 March

Session I LANGUAGE MODELS AND TECHNIQUES

9:00 A Bi-Directional Parser for ATN Grammars 74
G. Brown and W. A. Woods, Bolt Beranek and Newman

9:30 The Efficient Integration of Syntactic Processing with Case-Oriented Semantic Interpretation 75
R. J. Bobrow and M. Bates, Bolt Beranek and Newman

10:00 Some Properties of Arbitrary Phrase Structure Languages and Translation 76
W. Buttelmann, Ohio State University

10:30 The Augmented Finite State Machine Approach to Synthesis by Rule 77
T. Kaczmarek, University of Pennsylvania

11:00 A Lexicon for a Computer Question-Answering System 78
M. Evens and R. Smith, Illinois Institute of Technology

11:30 Progress Report on REL 79
B. H. Thompson and F. B. Thompson, California Institute of Technology

Session II Panel Discussion: SPEECH UNDERSTANDING AND COMPUTATIONAL LINGUISTICS - A CRITICAL EXAMINATION OF THE ARPA PROJECT

2:00 Chairman: J. Allen, M.I.T.
Participants: R. Reddy, Carnegie-Mellon University
D. E. Walker, Stanford Research Institute
W. A. Woods, Bolt Beranek and Newman

5:30 Business Meeting, Paul Chapin, ACL President, presiding

6:30 Adjourn for dinner

8:00 Dinner and Presidential Address by Paul Chapin
The Foundry, 1050 30th Street, Washington
$14.00; make reservations early Wednesday morning
40 person limit, because of restaurant capacity

ASSOCIATION FOR COMPUTATIONAL LINGUISTICS

Fifteenth Annual Meeting
Georgetown University, Washington D.C. 16-17 March 1977
Palms Lounge, Walsh Building; 36th Street between N and Prospect

PROGRAM

Thursday, 17 March

Session III PROBLEMS OF DISCOURSE

12:00 Adjournment

Registration fees:	$10.00	Member
	$ 5.00	Student
	$15.00	Other

No advanced registration procedures; individual dues for ACL membership of $15 for the year 1977 may be paid at the meeting. (Institutional dues are $30 for 1977.)

G. Brown and W. A. Woods

Bi-directional Parsing with ATN Grammars

The Syntactic component of HWIM, the BBN Speech Understanding System, is a middle-out, bi-directional parser for Augmented Transition Network grammars. Several of the HWIM control strategies require a parser that can evaluate an isolated sequence of words (called an "island") somewhere in the middle of an utterance to determine whether it is a possible fragment of a complete sentence. If so, the parser is required to make predictions for all of the possible words that could be used to extend the fragment in each direction. In this talk we will describe a parsing algorithm which efficiently achieves these capabilities.

The HWIM parser can be viewed as a bi-directional generalization of Earley s algorithm extended to handle context-sensitive, ATN grammars. The algorithm stores the computations in "segment" and "island" configurations indexed by end states and boundaries. (A segment is a partial parse that is contained completely within one level of the transition network grammar.) Organizing the computation into segment and island configurations eliminates the need for a stack, thus solving a difficult problem in middle-out parsing.

In the usual ATN formalism, the grammar is written as if to be processed from left to right, and in general some of the arc actions will be dependent on other actions to their left in the grammar. To insure the correct handling of such context-dependent arc actions by the bi-directional parser, the grammar writer must specify the "scope" of any such action. Except for the explicit declaration of scopes for context-dependent actions, a bi-directional ATN grammar is exactly like an ordinary ATN, and left-to-right ATN grammars can be converted to bi-directional operation simply by adding scope statements.

Although developed in the context of a speech understanding application we feel that the bi-directional, middle-out parsing algorithm also has applications in text parsing for problems such as error correction, partial interpretation of sentence fragments, and management of combinatorics in long sentences.

R. J. Bobrow and M. Bates

THE EFFICIENT INTEGRATION OF SYNTACTIC PROCESSING WITH CASE-ORIENTED SEMANTIC INTERPRETATION

It has long been the goal of those writing natural language processing systems to express syntactic constraints in a broad, general way while using tight semantic constraints to guide the parsing and to interpret the resulting structures. The system described here uses an augmented transition network grammar together with a case-oriented dictionary to achieve a close and efficient integration of the syntactic processing with the case structures (which include semantic and pragmatic properties of objects).

The ATN defines, using normal syntactic categories, a very general surface structure of about the capability of the LUNAR and GSP systems. If case structures and semantic information (including interpretation rules) are omitted from the dictionary, the grammar functions as a standard parser, producing closely related "deep structures" for syntactic paraphrases.

The system provides mechanisms for users to define semantic interpretation rules and case frame checks which are to be applied at various points in the parsing process. Thus constituents may be interpreted as soon as they are parsed, and the structure of the semantic interpretations thus produced may be checked when filling the case frames for higher structures. Since the "most likely local interpretation may not fit the case requirements of containing structures, the system provides a general coercion mechanism to reinterpret a constituent in light of its context when necessary. To facilitate reinterpretation, as well as certain anaphoric references, the original syntactic structure is maintained throughout the parsing together with any semantic interpretations.

The present system is being used as the natural language front-end for a sophisticated message processing and filing system. Ultimately, we hope to have a general system which can be adapted to other natural language input systems by providing new dictionary entries and semantic interpretation functions.

W Buttelman

SOME PROPERTIES OF ARBITRARY PHRASE STRUCTURE LANGUAGES AND TRANSLATION DERIVED USING A FORMAL MODEL OF PHRASE STRUCTURE SYNTAX AND SEMANTICS

Abstract

The notion of a phrase structure linguistic description is introduced -- a pair, D = (G,S) where G is an arbitrary phrase structure grammar and S is a formal semantics (defined in the paper). S may be either context free or context sensitive. S models the following notion of meaning: the meaningful units of language are phrases; the meaning of a phrase is a function of its syntactic structure and of the meanings of its constituents and of its semantic context. This concept is a generalization of semantic notions due to Tarski, later suggested by Thompson and by Katz and Fodor, and recently popularized for programming languages in Knuth's synthesized attributes and for natural languages by Montague. The (phrase structure) language of D, L(D), is the set of ordered pairs (w,m) where w is a sentence of G and m is a meaning assigned to w by S.

We prove the following results: The set of phrase structure languages is just the set of products of r.e. sets. Every phrase structure language has a description using a regular grammar and a context free semantics. For every description D with an unrestricted grammar and context sensitive semantics there is a description D' using a context free grammar and context free semantics such that L(D) = L(D'). Furthermore, D and D' are "strongly equivalent" in the sense that the phrase trees assigned by D' to each sentence are just the skeleton trees of the phrase structures assigned by D to the sentence. The notions of "weak" and "strong equivalence" are extended to semantics (if two descriptions are strongly equivalent in a semantic sense, then the structure of their semantic functions is identical -- in a programming sense, the same programs can be used to compute the meanings of the same sentences). In this sense, D and D' are not strongly equivalent. However, if D has a context free semantics, then D and D' are semantically strongly equivalent. Also, we prove that there is a description D" for L(D) using a context sensitive semantics which is strongly equivalent to D in both the syntactic and semantic senses.

Next we define translation on phrase structure languages and consider a particularly appealing strategy for translation, which we call "syntax-controlled" translation. (I have avoided the term "syntax-directed" because it has had differing uses in the literature.) We prove the following results: Every computable translation is definable as a syntax-controlled translation. For two arbitrary descriptions D and D', it is undecidable whether any syntax-controlled translation from L(D) to L(D') exists. We give an algorithm which, given two arbitrary descriptions D and D', will halt and produce the definition (program) of a syntax-controlled translation from L(D) to L(D') if and only if such a translation definable by D and D' exists.

Syntax-controlled translation requires no semantic computation at translate time (for which one pays a dear price in the time required to generate syntax-controlled translators). For a syntax-controlled translation which produces a

single target sentence having a meaning in common with the source sentence, the time complexity is $O(p+w)$ where p is parsing time and w is the weight of the source phrase structure. To produce the smallest set of target sentences such that each target sentence has at least one meaning in common with the source and such that all translatable meanings of the source are represented requires

$$O((d^{cn})(cn!)f(p+cn))$$

time, where c and d are constants, n is the source sentence length, f is the time to check for semantic validity of a parse, and p is the time to produce all parses.

Finally, we consider phrase structure language descriptions having both inherited and synthetic meaning. No new languages can be defined, but the use of inherited meaning leads in a natural way to a notion of semantic-controlled translation.

T. Kaczmarek

The Augmented Finite State Machine - A More Efficient Approach to Synthesis by Rule

The augmented transition network (ATN), which has proven useful for natural language understanding, has been reformulated and restricted slightly to yeild a mechanism termed the augmented finite state machine (AFSM). The AFSM is being used to do speech synthesis by rule, a process by which phonetic transcriptions of speech are converted into synthesizer parameters. The rules for synthesis in this approach take the form of procedures which are conditionally executed depending on context. Most previous synthesis by rule systems began with a transformational component and added procedural statements. In this system procedural statements have been added to a much simpler mechanism, the finite state machine. The advantage of the AFSM approach is that the phonetic string may be processed in a single linear pass.

M. Evens and R. Smith

A LEXICON FOR A COMPUTER QUESTION-ANSWERING SYSTEM

Computer question-answering systems and other models of natural language processing need lexicons that are much larger than those available today (cf. Simmons, 1970 and Becker, 1975). But the models currently in operation (e.g. Winograd, 1971) already consume all available high-speed memory in large computer systems. Lexical relations as developed by Raphael (1968), Apresyan et al. (1971), and Simmons (1973) provide a method of storing lexical information in more compact form. While Schank (1973) and Wilks (1975) both claim that there is a fixed universal set of semantic primes, we argue in opposition, following the Russians and Miller and Johnson-Laird (1976), that the set of lexical relations is open-ended; our system is designed to add new relations whenever a lexical regularity is noticed.

Our lexicon is being developed as an integral part of a computer question-answering system which answers multiple-choice questions about simple children's stories. It serves as a global data-base for this system - a combination lexicon-encyclopedia - and must make information readily available for the parsing process, for building an internal model of the story being read, and for making inferences. One of our test paragraphs, which comes from a test designed for first and second graders, says "Ted has a puppy. His name is Happy. Ted and Happy like to play." In order to answer the first question, "The pet is a: dog-boy-toy?", we need to know what pet means. The lexical entry for pet contains a simple definition, that a pet is an animal that is owned by a human, in a first-order predicate calculus form: NCOM(PET,Z_1) = (Z_2)NCOM(ANIMAL,Z_1).NCOM(HUMAN,Z_2).R(OWN,Z_2,Z_1). In order to answer this question we also need to know that a puppy is a young dog. This information: NCOM(PUPPY,Z_1) = NCOM(DOG,Z_1).PROP (AGE,Z_1,YOUNG) could be part of the lexical entry for puppy. We would, of course, need axioms of the same form as well for the entries for kitten, lamb, etc. Instead we express this information by using a lexical relation, CHILD. The lexical entry for puppy therefore contains CHILD dog; the lexical entry for kitten contains CHILD cat; and the lexical entry for CHILD contains the axiom scheme from which the relevant axioms are formed when needed.

For verbs, corresponding to each case relation there is a lexical relation which points to typical fillers of that case slot. The lexical entry for $bake_2$ includes TAGENT baker and TLOC kitchen. It also includes T make where T is the well-known taxonomy relation, so that if the story says that Mother baked a cake we can infer that she made one and CAUSE $bake_1$ so that we can deduce that the cake has baked. The selectional restrictions that help us tell instances of $bake_1$ and $bake_2$ apart can also be expressed compactly using the T relation. We also need to make deductions from main verbs in predicate complement constructions; deductions such as the speaker's view of the truth of the

proposition stated in the complement as derived from the factivity of the verb (in these stories the reader must infer that everything that mother says is true!). Lexical entries for main verbs that take predicate complements contain pointers to the implication class. These relations can then be expanded to give the proper axiom schemes.

The use of lexical relations allows us to express both syntactic and semantic information in a form that is compact, easy to retrieve, and that provides effective input to both parsing and deductive processes.

B. H. Thompson and F. B. Thompson

A Progress Report on REL

The REL (Rapidly Extensible Language) System is now in operational prototype form. An experimental version of the system was demonstrated in 1973 and since has undergone very thorough revision and clean up. The REL English grammar, which includes an extensive arithmetical component, has been improved and extended. The system can be demonstrated and made available for user testing on IBM 360/370 computers using most operating systems, e.g. TSO, CMS. A user's Reference Manual is now in preparation and will be available at the time of the conference.

The basic system philosophy has remained the same, namely to provide the user with a tool for natural man-machine communication that can easily be suited to his individual needs. Thus the system provides the user with the capability to modify and extend his data base and language package. Such modification can be carried out by statements about the data base items; for example:

John was not a student after June 1, 1976.

will remove John from the student class as of that date. Extensions can be carried out by adding new primitive individuals, classes, and relations, as well as through definitional capabilities which allow for defining new concepts in terms of existing ones. As a part of this capability, verbs can be introduced by paraphrases, for example:

def:ships "carry" coal:the cargo of ships is coal

and then used in a question such as:

What strategic materials were carried by USSR ships in 1963?

N. Sager and C. Insolio

Parsing Free Narrative

The results of an experiment in parsing narrative texts are presented. The texts were discharge summaries obtained from a hospital's computerized files of patient records. Each document states the background of the case, the results of the physical examination and laboratory tests, the time course of the illness in the hospital, diagnosis, status on discharge, etc. These texts are particularly interesting because they are unedited--cryptic phrases are mixed with full sentences, punctuation is not consistent, and spelling errors and abbreviations abound. In short, the material is free narrative as one would find it in a technical environment where reports are dictated and where there would be motivation for processing the data in their natural language form. The paper will describe how the above difficulties were treated and will present statistical results of the experiment, such as the number of sentences correctly parsed vs. the total number of sentences and the average parsing times for different types of sentences. In addition, the special problems due to commas, conjunctions, quantifiers, and run on sentences will be discussed.

B. Phillips DISCOURSE CONNECTIVES

In essence current systems of discourse analysis map surface structures into underlying causal chains of propositions. As the surface form is elliptic, it is necessary to include a knowledge base by means of which omitted linking propositions of the discourse may be inferred, rendering them explicit in the underlying representation.

Cause is not the only link between propositions, however; also used are syllogistic and analogic mechanisms, statements of relative belief, and processes of decomposition and abstraction, the last being the explication of abstract concepts.

Additive discourse connectives - 'because', 'so', etc., are realizations of the links between propositions in these modes of discourse construction. There are also adversative connectives, such as 'however', 'but', etc., that cannot be so explained. They must be interpreted as signals to turn off inference mechanisms.

To understand the need for adversative connectives, we first need to recognize two kinds of propositions, episodic and systemic. The former encode specific acts and states, e.g., 'Thompson won the election for the governorship of Illinois', whereas the latter are generalized categorical statements, e.g., 'birds have wings'. The content of discourse is usually episodic. The knowledge base contains both kinds of propositions, there are episodic and systemic memories.

There is a predictive component in the process of understanding discourse. A stated situation sets up expectancies which may either become the unstated linking propositions, or may be explicitly stated, and hence confirmed, at a later point. The predictions are set up by systemic memory. An episodic proposition has a counterpart in systemic memory, e.g.,

(1) John ate cheese. (Episodic)
(2) Person eat food. (Systemic)

The predictions are associated with systemic memory, e.g.,

(3) Person eat food CAUSE person not hungry.

Thus given (1), a later expectancy of (4) would be set up by (3)

(4) John was not hungry.

But systemic knowledge contains generalizations, not inviolable truths, and the inference may not be valid. This can be marked by the use of an adversative connective:

(5) John ate the cheese, but he was still hungry.

G. P. Brown

A FRAMEWORK FOR PROCESSING DIALOGUE

This report describes a framework for handling mixed-initiative English dialogue in a console session environment, with emphasis on recognition. Within this framework, both linguistic and non-linguistic activities are modelled by structures called *methods*, which are a declarative form of procedural knowledge. Our design focusses on units of linguistic activity larger than the speech act, so that the pragmatic and semantic context of an utterance can be used to guide its interpretation. Also important is the treatment of indirect illocutions, e.g., the different ways to ask a question, give a command, etc.

Our basic approach has been to combine careful structural distinctions with a mixed recognition strategy. The central distinction is in the way that utterances can be related to the methods in the dialogue model. First, an utterance (called an *initiator*) may introduce a method that corresponds to one of the standard activities in an environment (for example, asking a question at an information desk or requesting help from a consultant). Second, an utterance may correspond to a step in a *standard path* in a method already underway; here, a standard path is a normally expected succession of activity steps. Third, an utterance may be part of *recovery discussion*, which is generated when when some violation of standard expectations occurs, necessitating clarification, correction, etc. Finally, an utterance may belong to *metadiscussion*, a relatively constrained class whose function is to lay out the context for other utterances so that these may be identified with the appropriate method step.

Given the static model of dialogue embodied in the methods, the problem is to find the correct method step that relates to a particular input. We handle this problem by defining a set of special structures to aid in matching, by using the methods to generate expectations dynamically, and by differentiating overall matching strategies according to the four utterance classes described.

The ideas presented here have been implemented in a prototype system called Susie Software, which is embedded in OWL-I The OWL system is currently under development in the Knowledge-Bases Systems Group at the M.I.T. Laboratory for Computer Science. This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract Number N00014-75-C-0661.

Abstract submitted for presentation at The Association for Computational Linguistics
March 16-17, 1977, Georgetown University, Washington, D. C

L. A. Miller

Natural Language Programming :

Kitchen Recipes

Laboratory studies of computer programming by naive programmers indicated that, for formal programming languages, most behavioral errors are associated with specification of the transfer-of-control characteristics Subsequent studies revealed that it is this feature which most discriminates between formal computer programming and "natural language" programming: the former embeds the data-manipulation actions within a complex control structure whereas the latter emphasizes first the action, followed by subsequent qualifications. This ACTION-QUALIFICATION style is so strikingly different from the CONTROL-ACTION style of programming computers that a study of natural language programming by professionals was initiated. The objective of the investigation is to determine the mechanisms whereby process information is communicated and to assess the oft-asserted (but empirically untested) "imprecision" and "ambiguity" of natural language usage in procedural domains Potentially, such an investigation could result in an alternative to formal programming languages for the linguistic man-machine interface -- e. g., Natural Language Procedure Specification.

We report on our progress to date in the analysis of a corpus of recipes from The Joy of Cooking. Our present understanding of the communication process in recipes is that the imperative verb is a call to some procedure which returns a case-frame into which are mapped the remaining object-group and verb-qualifier elements of the surface text. We present statistics concerning case frequencies, syntactic structures, and word usage, and we detail our approach for the automatic comprehension and symbolic modelling of the activities involved in recipe execution (we are using Heidorn's NLP LISP system).

J. R. Hobbs ACCOMMODATING THE SPATIAL METAPHOR

Linguists and psychologists have frequently noted that in English and other languages one often appeals to spatial metaphors when speaking of abstract ideas (9,1,3). For example, we speak of "high hopes", "high prices", "deep thought", "being in politics", "a book on sociology", "getting the idea", etc. Heretofore, this has been only an observation. Even Schank's work, with its decompositions into PTRANS, ATRANS, and MTRANS, is only suggestive of an underlying unity, and Jackendoff's classification of word senses into positional, possessional, identificational, and circumstantial modes remains only a classification. This paper describes an approach which utilizes the spatial metaphor in constructing economical definitions of "all-purpose" words that have previously defied precise specification, and a method for interpreting these words in context which treats metaphor not as an anomoly but as the natural state of affairs.

The basic idea is to define words in terms of very general spatial predicates and then, in the analysis of a given text, to seek a more specific, context-dependent interpretation, or binding, just as a compiler or interpreter seeks bindings for the variables and procedure names mentioned in a program.

Interpretation as Binding: In programming languages, there is normally a fixed means of determining bindings, either by following a chain of access modules (2) or by consulting an a-list or FUNARG-frozen environment.

Van Emden & Kowalski (8) have presented another outlook. In a mechanical theorem proving system, they show how Horn clauses may be viewed as procedure declarations in which the positive literal is a procedure name, the negative literals the procedure

body, and each negative literal a call to another procedure. A set of Horn clauses is a non-deterministic program, non-deterministic because several Horn clauses may have the same positive literal. That is, the procedure name in a procedure call may be bound to one of several different procedure bodies. Resolution is an attempt to bind a procedure name in a way that leads to the desired refutation. Put in another way, we may view the inference "A⊃B" as specifying A as a possible binding for B.

Montague (6,4) developed a variety of intensional logic as a representation for natural language. In his formalism, individual words can be defined as functions expressed in terms of intensions, i.e. variables and procedure names. Syntactic relations in English are translated into function applications in intensional logic. These function applications bind the intensions to specific interpretations. In this way the meanings of individual words are composed into the meaning of the sentence. However, the binding mechanism is quite fixed, making the formalism insufficiently flexible for the whole range of natural language.

Our approach combines Montague's with that of Van Emden & Kowalski. As in Montague's approach, individual words are defined in terms of general predicates that may be viewed as unbound predicate names, and their bindings in a given text are determined from syntactically related words. However, the binding mechanism is not fixed, but as with Van Emden & Kowalski, it is a search for a chain of inference which culminates in an expression involving the general predicate. An example is given below. In addition, a dynamic ordering determined by context is imposed on the axioms in the data base of lexical and world knowledge, defining an ordering on chains of inference. The binding is chosen which is given by the

appropriate chain of inference highest in this ordering.

The Spatial Metaphor: At the base of the Lexicon, or set of axioms, are a small number of primitive notions with a highly spatial or visual flavor. Among these are "Scale" or a partial ordering defined by possible changes of state, the relation "on" which places points on the scale, and "at" which among other things relates an entity to a point on a scale. Moreover, "at" is related to predication: for an entity to be at a predication is for the entity to be one of its arguments, as illustrated by the equivalence

John is hard at work = John is working hard.

Concepts at higher levels of the Lexicon are defined in terms of these basic spatial concepts. For example, "to think of" or "to have in mind" is defined as a variety of "at" Time is a scale, and an event may be at a point on that scale. A set may also be thought of as a scale and its elements as being points on the scale. Note that this takes seriously the visual image one has of a set as the elements spread out before one.

Finally, "all-purpose" words such as the common adverbs and prepositions are defined in terms of the basic concepts like "scale", "on", and "at". In the analysis of a text, we find interpretations for these basic concepts by finding chains of inference from properties of the arguments of the "all-purpose" words to propositions involving the basic concepts.

Simplified Example: Consider "John is in politics". Suppose "in" means to be at a point on a scale. We must find bindings for the underlined words. Politics is a set of activities directed toward the goal of obtaining and using power in an organization. A set is is a scale. The typical activity is on the scale. For John to be at such an activity is for him to be one of the participants in it.

Thus, for John to be in politics is for him to engage in the activities that characterize politics.

Other examples illustrating the distinction between "in" and "on" and the meaning of that elusive adverb "even" will be presented.

Significance: This work represents an advance in our understanding of how meanings of words are composed into the meanings of larger stretches of text, and of the effect of context on interpretation. Moreover, it is the result of a happy blend of computational or logical technique with linguistic and psychological insights.

Bibliography

1. Asch, S.E. "The Metaphor: A Psychological Inquiry" in M Henley, ed. Documents of Gestalt Psychology, 1961.

2. Bobrow, D. & B. Wegbreit. "A Model and Stack Implementation of Multiple Environments" CACM, October 1973, p. 591.

3. Clark, Herbert H. "Space, Time, Semantics, and the Child" in Cognitive Development and the Acquisition of Language. 1973.

4. Hobbs, J. & S. Rosenschein. Making Computational Sense of Montague's Intensional Logic. Report No. NSO-11, Courant Institute of Mathematical Sciences. December 1976.

5. Jackendoff, Ray. "Toward an Explanatory Semantic Representation" Linguistic Inquiry, Winter 1976. p. 89.

6. Montague, Richard. "The Proper Treatment of Quantification in Ordinary English" in Approaches to Natural Language, 1973.

7. Schank, R., N. Goldman, C. Rieger, & C. Riesbeck. "Inference and Paraphrase by Computer" JACM, July 1975. P. 309.

8. Van Emden, M.H. & R.A. Kowalski, "The Semantics of Predicate Logic as a Programming Language" JACM, October 1976. p. 733.

9. Whorf, Benjamin. "The Relation of Habitual Thought and Behavior to Language" in Language, Thought, & Reality, 1956.

B. Nash

Inferring an Antecedant

Computational research on pronominal anaphora has centered around the problem of "reference resolution", i.e. the problem of choosing the correct antecedant for an anaphoric expression from a set of several possible candidates. But reference resolution, though a complex process requiring the interaction of many sources of knowledge, is really only half the problem. The other half involves actually finding the candidates. In current natural language systems, this half of the problem has been handled in a rather ad hoc fashion or has been ignored entirely. In these systems, the set of possible antecedants for a pronoun is usually culled off a history list of objects introduced earlier in the discourse. Various heuristics including recency, structural constraints, semantic selectional restrictions, known higher-level task or discourse organization, and case and number agreement are then applied, in order to choose the best-fitting candidate.

On the one hand, it has long been recognized that inference may be needed to find possible antecedants for a definite noun phrase. For example, in

> A leering face appeared at Mary's window.
> She called the police to arrest the man.

at least one inference rule relating man and face is needed to figure out a possible antecedant for "the man".

The point I shall be making in this paper is that inference may be required to conjure up possible antecedants for pronouns as well. Examples like the following will be used to illustrate this point.

> I saw a married couple walking in the park.
> He had on awful plaid shorts, and she had on a dashiki.
> [he = the husband, she = the wife]
>
> John blended some flour and water and used it to seal the lid onto the pot.
> [it = the flour-water mixture]
>
> Mary gave each girl a T-shirt.
> They thanked her for them.
> [them = the set of T-shirts, each of which Mary gave to some girl]

I shall show that any pronoun resolution procedure, even one that uses highly sophisticated syntactic, semantic and pragmatic checks, cannot restrict itself to considering only objects and events given explicitly in the text. In addition I shall show how the needed antecedants can be inferred, using a formal representation language for English described elsewhere.