BOLT BERANEK AND NEWMAN INC

CONSULTING • DEVELOPMENT • RESEARCH

AFCRL-70-0505

NATURAL COMMUNICATION WITH COMPUTERS III

Daniel G. Bobrow

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

Contract No. F19628-70-C-0013

Project No. 8568

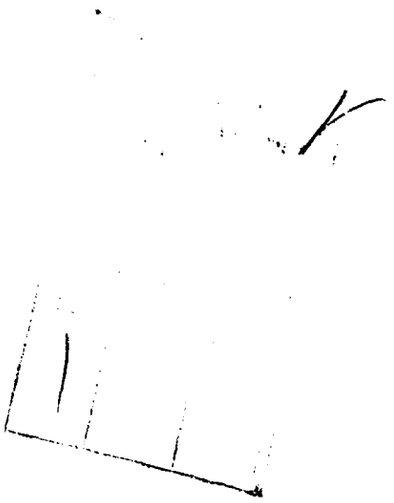
Final Report

Period Covered: 31 August 1969 through 30 September 1970

30 September 1970

Contract Monitor: Hans Zschirnt
Data Sciences Laboratory

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 627
Monitored by
AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS 01730



	1969 1970
Program Code No.	9D30 & 0D30
Effective Date of Contract	1 August 1969
Contract Expiration Date	30 September 1970
Principal Investigator and Phone No.	Dr. Daniel G. Bobrow 617-491-1850
Project Scientist or Engineer and Phone No.	Dr. Hans H. Zschirnt 617-861-3671

Qualified requestors may obtain additional copies from the Defense Documentation Center. All others should apply to the Clearinghouse for Federal Scientific and Technical Information.

AFCRL-70-0505

NATURAL COMMUNICATION WITH COMPUTERS III

Daniel G. Bobrow

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

Contract No. F19628-70-C-0013

Project No. 8668

Final Report

Period Covered: 31 August 1969 through 30 September 1970

30 September 1970

Contract Monitor: Hans Zschirnt
Data Sciences Laboratory

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 627
Monitored by
AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
BEDFORD, MASSACHUSETTS 01730

ABSTRACT

Bolt Beranek and Newman has been engaged in a continuing research program whose goal is to develop techniques to facilitate natural communication with computers, people, other computers, and real-time devices. This work has been supported under two separate contracts both funded by the Advanced Research Projects Agency. The work has been carried on as an integrated research program; work on semantic automatic language processing and automatic programming aids was performed under contract F19628-70-C-0013 through Hanscom Field Air Force Cambridge Research Laboratories and work on the development of time-sharing systems was done under contract XC-3169(62-7036)7OR.

Our work on natural communication with computers has focused on many levels of the man-machine interface. An important aspect of the work has been the development of tools to allow English language interaction between men and machines. Another is the development of a programming environment and aids to facilitate construction of the complex programs necessary for English language research. At another level we have developed a time-sharing system based on the DEC PDP-10, which allows maximum flexibility and ease of interaction in the development of the programming environment in which we wish to work.

In accordance with our responsibility to report the results of our investigation we have prepared a number of documents, some of which have been distributed as scientific reports under the contract, some of which have been published as technical articles in professional journals, and a few of which are internal memoranda which we have made available to interested outside parties. Following a brief summary of our research we include abstracts of these documents for further information.

TABLE OF CONTENTS

SECTION I

INTRODUCTION

Bolt Beranek and Newman has been engaged in a continuing research program whose goal is to develop techniques to facilitate natural communication with computers, people, other computers, and real-time devices. This work has been supported under two separate contracts both funded by the Advanced Research Projects Agency. The work has been carried on as an integrated research program; work on semantic automatic language processing and automatic programming aids was performed under contract F19628-70-C-0013 through Hanscom Field Air Force Cambridge Research Laboratories and work on the development of time-sharing systems was done under contract XC-3169(62-7036)7OR.

Our work on natural communication with computers has focused on many levels of the man-machine interface. An important aspect of the work has been the development of tools to allow English language interaction between men and machines. Another is the development of a programming environment and aids to facilitate construction of the complex programs necessary for English language research. At another level we have developed a time-sharing system based on the DEC PDP-10, which allows maximum flexibility and ease of interaction in the development of the programming environment in which we wish to work.

Our research on English language communication between man and machines has focused on the two general problems - syntax and semantics - and one major application. Our work in syntax centers on the development of a natural and efficient syntactic analyzer for English that allows easy interfacing to a semantic analysis

procedure. In semantics we have been working to develop computer representations of the meaning of English language statements. These representations, and procedures to manipulate them, are used, for example, to test whether paired situations are analogous to each other and to understand paraphrased versions of previously known information. We have applied the results of these efforts to the development of a program which can conduct a dialogue with a user to review material he has already learned. This program can also provide instruction where a user wishes to learn only part of a subject, wants the computer to provide interesting information, and wishes to guide the course of the conversation. Unlike conventional frame oriented teaching programs this mixed-initiative diaglogue system is based on an information structure similar to the one used for semantic representation.

In the design and development of complex programs such as those described above, the programmer needs an environment which is rich and flexible but which is also forgiving of errors and helpful. We have been continuing work in the development of a LISP system which, over and above its capability as a programming language, provides a set of tools to aid the user in the development of complex programs. These range from a syntax directed editor to automatic error correcting facilities which can correct errors in user programs at run time, and allow computation to continue. The design philosophy of these tools is to allow the user to concentrate on the problem level he wants to work on, only descending to a primitive implementation level when desired.

To facilitate use of this programming environment, Bolt Beranek and Newman has designed and implemented a time-sharing system, TENEX, whose human engineering allows easy communication between

man and machine. TENEX also allows each user a very large data base, and large complex program structures. In general, TENEX is designed to provide the extended capabilities demanded by very hard jobs and to adapt at different times to requirements ranging from real time computation to very large data base manipulation. Special processors attached to this machine provide added facilities, such as a high speed dynamic display and an intelligent "hybrid" I/O processor.

In accordance with our responsibility to report the results of our investigation we have prepared a number of documents, some of which have been distributed as scientific reports under the contract, some of which have been published as technical articles in professional journals, and a few of which are internal memoranda which we have made available to interested outside parties. Following a brief summary of our research we include abstracts of these documents for further information.

SECTION II

LANGUAGE PROCESSING RESEARCH

A. Syntactic Analysis System

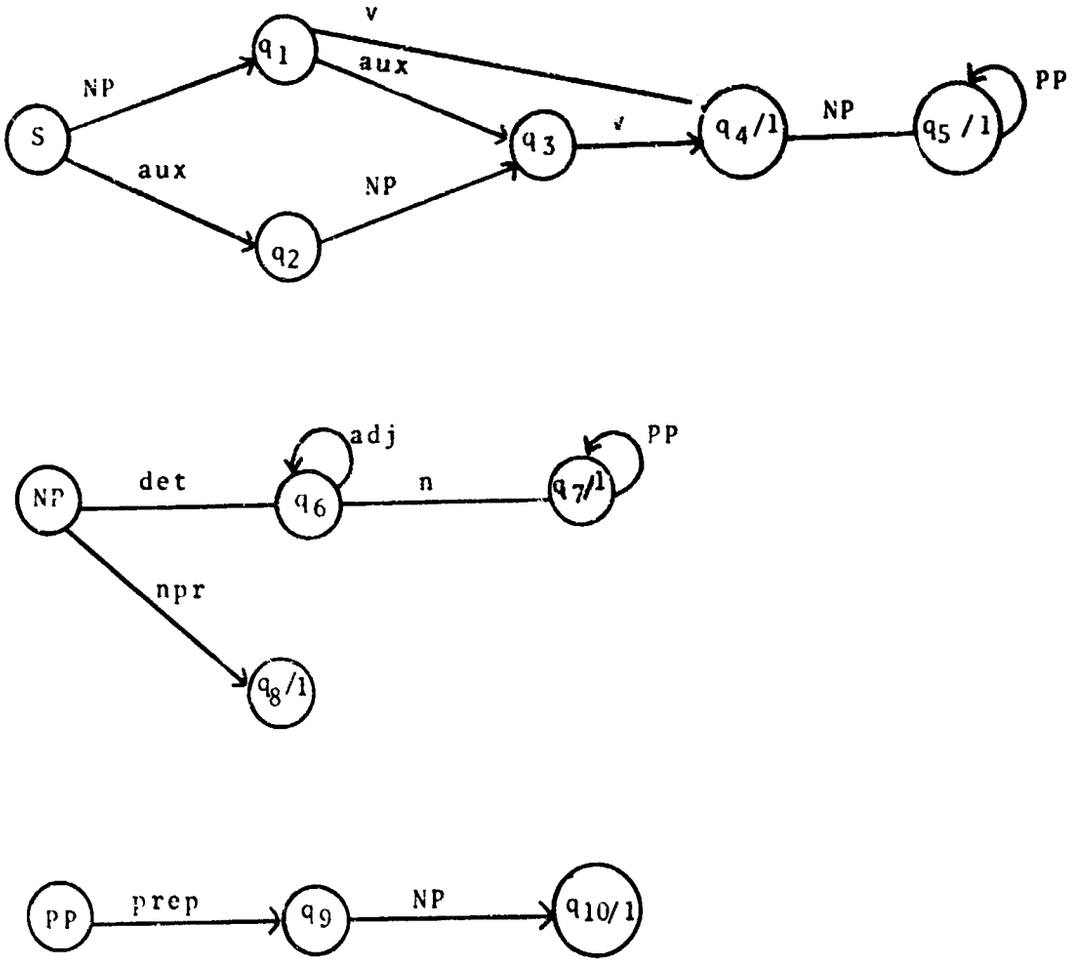
We have focused our attention on an analysis system that utilizes a representation of grammar, superficially similar to a finite state transition network, but augmented to give it the power of a transformational grammar. This representation, based on an augmented recursive transition network, is integrated with a semantic analysis procedure designed to produce interpretations which can express very general queries of the data base. The procedure has been used previously only with formally defined fixed structure data bases, but we plan to integrate it with the semantic memory processor described below.

One component of the syntactic analysis system, the parser, simulates the action of a non-deterministic, recursive, augmented transition network which represents the grammar for a language. The network is a labeled directed graph whose nodes represent transitions from one state to another. Each arc of the network is labeled with an English word, a set of words, a lexical category name, or the name of a state in the diagram. An arc from state X to state Y labeled Z represents the fact that the parsing algorithm in state X may make a transition to state Y if the "next thing" in the input string "satisfies" the label Z. E.g., if the label Z is a word, the transition is permitted if the next word in the string is equal to Z.

The network is non-deterministic because it is possible for several arcs leaving a node to be equally applicable at a given

point in a computation. The parser must provide for the possibility of following any or all of these arcs. The network is recursive because an arc labeled with the name of a state in the network causes a pushdown to a lower level computation. This computation will begin at the indicated state and continue until it reaches a final state, at which time it will return to the previous level and enter the state at the end of the arc which caused the pushdown. Thus, for example, an arc labeled NP in Fig. 1 causes a recursive call to the transition network to recognize a noun phrase. The effect of such an arc is to permit the transition if a noun phrase is the "next thing" in the input string.

Figure 1 is an example of a recursive transition network for a small subset of English. It accepts such sentences as "John washed the car," "Did the red barn collapse?", etc. It is easy to visualize the range of acceptable sentences from inspection of the transition network. To recognize the sentence, "Did the red barn collapse," the network is started in state S. The first transition is the aux transition to state q_2 permitted by the auxiliary "did". From state q_2 we see that we can get to state q_3 if the next "thing" in the input string is a NP. To ascertain if this is the case, we call the state NP. From state NP we can follow the arc labeled DET to state q_6 because of the determiner "the". From here, the adjective "red" causes a loop which returns to state q_6, and the subsequent noun "barn" causes a transition to state q_7. Since state q_7 is a final state, it is possible to "pop up" from the NP computation and continue the computation of the top level S beginning in state q_3 which is at the end of the NP arc. From q_3 the verb "collapse" permits a transition to the state q_4, and since this state is final and "collapse" is the last word, the string is accepted as a sentence.



S is the start state

q_4, q_5, q_7, q_8, and q_{10} are the final states

Figure 1: A Sample Transition Network

The transition network is augmented because each arc contains in addition to its label an arbitrary condition and action. The condition on an arc prevents the transition unless the condition is satisfied, and the actions are used for building the structural description if the arc is followed. The conditions on the arcs can allow complex syntactic tests, e.g. test if this S is embedded in a higher S; if so, forbid acceptance of a sentence adverbial. More generally, the conditions can be procedure c·'ls which test semantic features or context information to be used to restrict the actions of the parser. They also allow for the special recognition of certain regular classes of "words" such as integers and times of day which are easily recognizable but cannot reasonably be represented in the dictionary (e.g., because the class is indefinite).

B. Semantic Network Representation

The problem of natural language interaction with a computer depends heavily on the development of adequate data bases to represent the meanings of English sentences. We have been developing a theoretical model which can represent such information, and processes which can manipulate information in this data base. The data base we use is a semantic network, a hierarchically structured memory in which a unit (representing a concept usually associated with an English word) is represented by a set of properties which it satisfies. Within the definition are pointers to other units representing concepts which are explicitly required in the definition, with relation labels defining the structure of the concept in terms of these ingredients. Paths through the network from one unit to another represent implicit relationships between these two nodes. This information network provides a base for our semantic processing.

At the present time we have on hand a number of diverse program components and capabilities. There are either completed routines, or at least more or less precise ideas about how to:

1. Structure a data base
2. Answer questions with such a base
3. Generate English from such a base
4. Recognize relationships between items in such a data base
5. Parse English sentences

We are continuing to develop these capabilities and others closely related, and are attempting to integrate them into a single system.

We will describe in some detail how we have applied some of these capabilities to the solution of verbal analogy problems given

as intelligence tests. Two examples shown here were solved by the current program:

1. WINDOW:PANE::DOOR(PANEL,JAMB,KNOB,KEY)
2. RAT:RODENT::WHALE:(MAMMAL,OCEAN,ANIMAL,GIGANTIC)

The problem is defined in terms of path comparison in the network memory. The first subproblem involves the choice of which paths to use in comparison, because, for any pair of nodes (words) a large number of paths is possible. Intelligent selection among these is necessary to compare one pair of nodes with another. We use a heuristic measure of similarity between pairs which counts the number of partially ordered common elements in the two paths to be compared. Those pairs with more common subsequences are liable to yield closer analogies. Given a pair of similar paths, an analogy is formed by matching the forms of the paths, i.e. the property names; and the corresponding nodes. For example, if one path makes a superset jump, it is important to search the comparison path for such a jump. The form comparison determines which nodes can be paired for semantic comparison. We are developing a general node comparison process which appears to have general import for a number of semantic network problems in addition to analogy solving. Its solution with strong constraint produces a synonym finder, and also is a basic part of a solution to the paraphrase problem of determining when one statement is like another.

A major question for a node comparison program is "Can it be determined if one node is like another without a context for the comparison?" For example, can we say if a table is like a horse? The answer to this, like the answer to any possible pair of nodes compared, is both yes and no. They are alike in some ways and different in others. Clearly a table and horse are different in

that one is animate, the other not. Yet they are the same in that each has four legs. One way of answering the question is to require that the context or at least the relevant dimension be specified. Along the line of physical properties the table and horse may be likened to each other.

Rather than specifying a dimension we can rephrase the question to be: is node A more like node B or node C? This is useful in the analogy program. It requires a metric on the complexity of the analogy. Simple metrics fail sometimes; for example, if A is a common superset of B and C the question is unanswerable unless there are contravening properties. There is no solution to the question "is a bird more like a wren or a sparrow?" But if the question is "is a bird more like a wren or a kiwi," the answer is wren since kiwi has the property of no wings, which contravenes a general property of birds. Other important considerations include the superset relationships and common property comparisons. Common superset membership determines that sparrow is more like hawk than bird since sparrow and hawk both belong to the same superset.

Property comparison is both semantic and "positioned"

1. Position. The position of a property on the property list indicates its importance. This importance is to a certain extent idiosyncratic to a particular memory in humans. Consider, "Is a cardinal more like a fire engine or a sparrow?" An adult might hesitate and then say sparrow since the superset bird is a most important property. A child's memory might store red as a more important property of cardinal than its birdness and thus might answer fire engine.

2. Semantic property comparison. Two properties are alike if their attributes and values are equivalent or superset equivalent, that is, if the attributes and values are equal or linearly related and not divergent. Two nodes are linear if one is on the superset chain of the other. They are divergent if they share a superset and are not linearly related.

Two properties are discriminating if either their attributes are equivalent and their value is not, or vice versa. Thus (color red) discriminates an object from (color blue), and (nibbles food) discriminates an object from (devours food) if all other properties are irrelevant.

When testing to see if node A is more like B or C, we look for comparable properties, that is, equivalent or discriminating ones which exist for all three cases. The comparison can be decided if a property of A is found to be equivalent to a property of B and discriminating for the node C. If a simple discrimination is not found after a certain effort limit, the program recurses on properties using the general comparison routine for attributes and values.

SECTION III

AN APPLICATION OF LANGUAGE PROCESSING TECHNIQUES

As an application of our language processing techniques we have developed a program, SCHOLAR which is capable of conducting a mixed-initiative dialgoue in English with a user, based upon information stored in a semantic memory. This work has been applied to a computer assisted instruction task conducted under other government sponsorship. We are combining for the first time semantic network information representation with generative techniques for originating questions and detailed program behavior. Generative techniques produce program behavior which has not been anticipated in full detail in advance but which still follows some general guidelines. For example, generative techniques can, produce computer questions related to answers (perhaps with errors) given by the student in response to prior questions. The reason for not having full anticipation is the practical impossibility of thinking of all the possible variations, especially when an important element in producing these variations is the behavior of the human and the interaction. Although the original subject matter chosen for the system was the geography of South America, we believe that the system is readily adaptable to other tasks. We plan to use it, for example, to build a program which can give instruction in the use of the LISP system we are now developing for the PDP-10 computer. We present here a description of the SCHOLAR program, illustrating its behavior with a sample protocol for a dialogue between a student and SCHOLAR.

Figure 2 illustrates the operation of the program. As indicated, it transfers information between two semantic information structures: that of the student, and a structure stored in computer

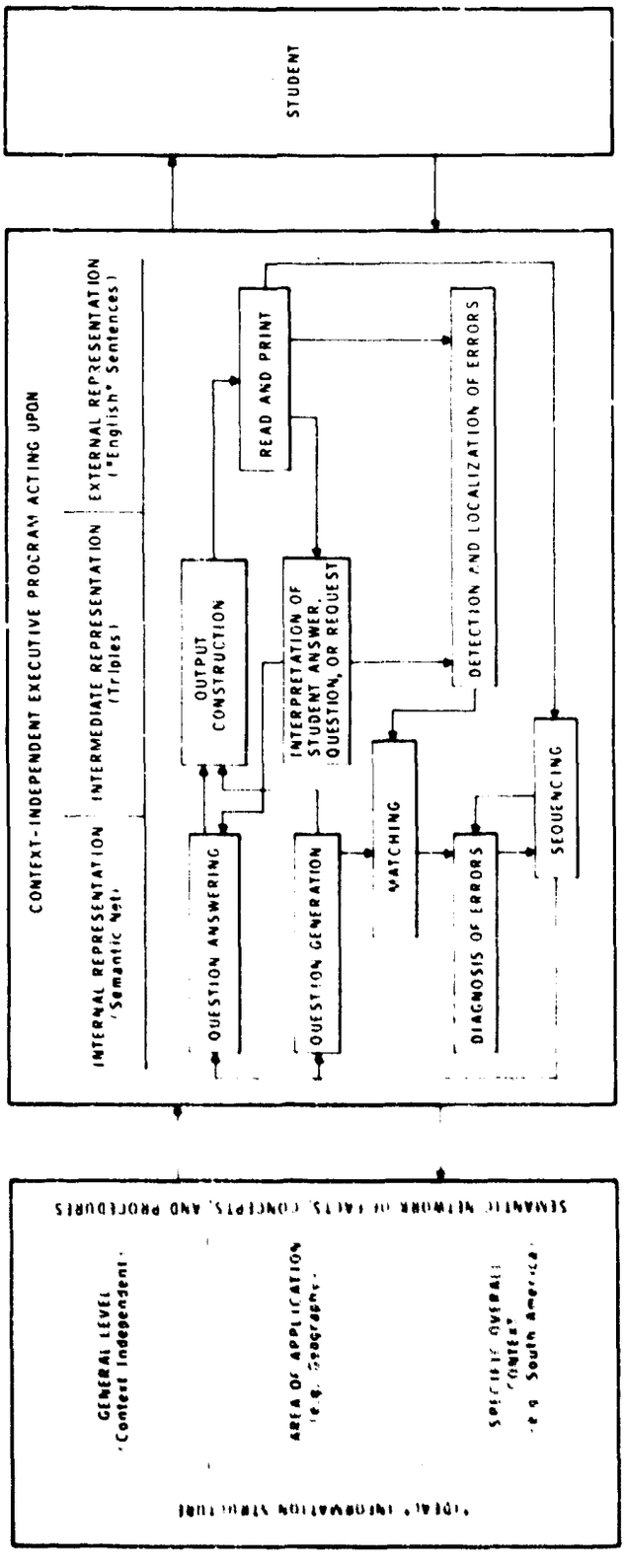


Figure 2. General Structure of SCHOLAR, a Program Capable of Generative, Mixed-Initiative Tutoring.

memory. The latter is a semantic network of concepts, facts and procedures. It is organized in a modular way, with a general top level which is context-independent (commonly used English words, and simple procedures), a level related to the area of application (e.g., geography), and a level with the specific context on which SCHOLAR is going to work (in our case, South America). It is expected that there will be many more pointers from units in a lower level to units in a higher one than vice versa. Therefore, we could replace the information on South America by information, on, say, New England, without substantially affecting general geographic information; or even replace geography by, say, history or meteorology, and still maintain the general level almost unchanged.

In Figure 2, the main functional packages of SCHOLAR are shown in the box labeled: "context-independent executive program." This executive acts on an internal representation (the semantic network), an intermediate representation mainly used in internal function calls, and a close-to-English input/output representation. The intermediate representation is based on the well-known object-attribute-value triples.

Program flow within the executive is highly recursive, since different function groups call each other in many ways. For example, the analysis of the student's answer to a question may trigger the presentation of some explanation to him, or the generation and formulation of a new question, and so on.

Figures 3a through 3c present a fragment of a protocol, taken on-line, which demonstrates some of the basic capabilities of SCHOLAR. In this protocol SCHOLAR starts typing after being called. The student's turn comes after the asterisk and can be a response to

```
 THE POPULATION IN CHILE IS APPROX 8500000. PEOPLE.
 TRUE OR FALSE?

*TRUE*

 YOU ARE CORRECT.


 USE ONE OF THE FOLLOWING:

 SUCRE
 FRENCH
 ARGENTINA
 SPANISH

 TO ANSWER THE QUESTION:

    WHAT IS THE LANGUAGE IN CHILE?

*YOU ARE TAKING TOO MUCH TIME...

*SPANICH*

 YOU SHOULD HAVE WRITTEN "SPANISH".

 VERY GOOD.


 THE CAPITAL IN CHILE IS SANTIAGO.
 CORRECT OR INCORRECT?

 *CORRECT*

 THAT'S RIGHT.

   APPROX WHAT IS THE AREA OF BRAZIL?

*2500000 SQUARE MILES*

 WRONG.
 PLEASE INDICATE IF THE FOLLOWING STATEMENT IS CORRECT
           OR INCORRECT:

 THE AREA OF PARAGUAY IS APPROX 47432 SQUARE MILES.

*WHAT IS THE AREA OF BRAZIL?*


   THE AREA OF BRAZIL IS APPROX 3600000. SQUARE MILES.

 NOW ANSWER THE QUESTION YOU DIDN'T ANSWER BEFORE.

*FALSE*

 REPHRASE YOUR STATEMENT. I CAN'T UNDERSTAND IT.

*INCORRECT*
```

Figure 3a. Fragment of an On-Line Protocol Using SCHOLAR

```
 VERY GOOD.

 THE AREA OF PARAGUAY IS APPROX 157000 SQUARE MILES.

 THE LANGUAGE IN BRAZIL IS FRENCH.
 RIGHT OR WRONG?

*WRONG*

 YOU ARE CORRECT.

 THE LANGUAGE IN BRAZIL IS PORTUGUESE.
 IS IT TRUE THAT THE CAPITAL IN BRAZIL IS BRAZILIA?

*YES*

 THAT'S FINE.

   APPROX WHAT IS THE POPULATION IN BRAZIL?

*60000000.*

 APPROXIMATELY TRUE, BUT A MORE EXACT VALUE IS 70000000.

 WHAT IS BRAZIL?

*TELL ME SOMETHING ABOUT PERU.*


 WAIT A MINUTE. I'M CHECKING MY FILES.

 LET'S PROCEED NOW.

     PERU IS A COUNTRY.
     IT IS IN SOUTH AMERICA.
     THE POPULATION IS APPROX 7000000. PEOPLE.
     THE CAPITAL IS LIMA.

 NOW ANSWER THE QUESTION YOU DIDN'T ANSWER BEFORE.

*Q/A*


 DO YOU WANT TO CHANGE THE GENERAL MODE OF THE INTERACTION
      TO Q/A?
 PLEASE ANSWER Y OR N.

*Y*

 OK, MR. JONES. LET'S GO AHEAD.


*PLEASE, TELL ME MORE ABOUT PERU.*

     THE AREA IS APPROX 480000.  SQUARE MILES.
     THE LANGUAGE IS SPANISH.
```

Figure 3b. Fragment of an On-Line Protocol Using SCHOLAR

a question by SCHOLAR, a question to SCHOLAR, or a command (like halting, or changing the mode of the interaction to either Q/A - for question-answering - or mixinit - for mixed-initiative - or test - for testing). The student returns control to SCHOLAR by typing another asterisk and a carriage return.

SCHOLAR can also generate questions, and evaluate the student's answers, deciding when these are correct, wrong, or only approximately or partially correct, and then take some conditional actions. It keeps track of content, and changes it on the basis of relevance and time considerations. SCHOLAR does all this without faithfully following specific and detailed directions, but rather by applying general criteria and procedures to a body of "knowledge" about the subject being discussed.

SECTION IV

AUTOMATIC PROGRAMMING AIDS

Our work in this area grows out of techniques we have developed to facilitate on-line interaction with users within the context of BBN-LISP. The ultimate goal is to produce programs that write other programs given very general specifications, while allowing the user to communicate at whatever level is natural at the moment. An important aspect of this problem is the development of a data base which describes programs, and of processes which can manipulate this data base. If we consider how two programmers communicate with one another about a program we can see that there is a great deal that is unsaid or appeals to induction. There are obvious statements that are suppressed etc. We are currently concentrating on making that kind of implicit information available to a program which is aiding the programmer.

In order to provide a powerful base on which to build such automatic programming aids we have implemented a new BBN-LISP for the DEC PDP-10 containing a number of new features. These include a facility for allowing a user to define new data types based on techniques developed in the LISP II project. By defining tracing, marking and relocation functions and special print functions which are accessed through a data type number the user can create new data types and have the LISP system take care of problems of storage allocation and maintenance. In this LISP we have developed an improved representation of the environment using a parameter push down stack. Also, we have extended the types of operations which can be performed in an arbitrary environment by using capabilities for evaluating arbitrary forms in earlier environments

and then returning to these earlier environments with values for form evaluation.

Recognizing the fact that a user does not work in a static context but rather in one in which he remembers the types of things he has done recently we have adopted the concept of saving for the user information about things he has done in a recent "time slice". As one result of this, if the user enters a complex expression for immediate evaluation but mistypes part of it, he is able to retrieve the mistyped portion (which will have caused an error), edit it and have it reevaluated. Similarly, in editing a function, when the user has deleted a significant portion of his program there is sufficient information saved so that he can undo his last operation. Within the LISP environment we continue to embed error correction features in many of the programming tools. These tools are all designed with the philosophy that they should be active, have "do what I mean" (DWIM) features, which make intelligent guesses as to what the user meant if there are errors, and generally monitor and participate in the interactive process of program development.

All the DWIM and error-correcting features in this system have been designed to meet the following criteria, which we feel are important for such tools in any system:

1. The user must be able to disable the entire program.
2. The user must be able to interrupt and/or abort any attempted correction.
3. The program must have a measure of how certain it is about the nature and correction of a mistake, and use this measure in determining the amount of interaction with the user.

4. The program should be able to distinguish between significant and trivial corrections, and be more cautious, i.e., more interactive, about correcting the former.
5. The user should be able to specify to the program his degree of confidence in its corrections, as reflected by the amount of interaction he desires.

The protocol shown in Fig. 4 represents the kind of corrections the program will handle and indicates the flavor of what we are trying to achieve. User input is preceded by an arrow (←).

In this example, the user first defines a function FACT whose value is to be N factorial, where N is its argument. The function contains several errors: TIMES and FACT have been misspelled. The 9 in N9 was intended to be a right parentheses but the teletype shift key was not depressed. Similarly, the 8 in 8SUB1 was intended to be a left parentheses. Finally, there are two left parentheses in front of the T that begins the second clause in the conditional, instead of the one required.

```
←DEFINEQ((FACT (LAMBDA (N)
(COND ((ZEROP N9 1) ((T (TIMESS N (FACCT 8SUB1 N]
(FACT)
←PRETTYPRNT((FACCT]
=PRETTYPRINT
=FACT

(FACT
  [LAMBDA (N)
    (COND
      ((ZEROP N9 1)
        ((T (TIMESS N (FACCT 8SUB1 N]))
NIL
←FACT(3)
N9(IN FACT) >>---> N)
(IN FACT) (COND -- ((T --))) >>---> (COND -- (T --))
TIMESS(IN FACT)->TIMES
FACCT(IN FACT)->FACT
8SUB1(IN FACT) >>--->  (SUB1
6
←PRETTYPRINT((FACT]

(FACT
  [LAMBDA (N)
   (COND
      ((ZEROP N)
        1)
      (T (TIMES N (FACT (SUB1 N]))
NIL
←
```

Figure 4. Sample of Operation of DWIM

SECTION V

TENEX DEVELOPMENT

During the past two years we have completed the design and implementation of a DEC PDP-10 time-sharing system to support our work in natural communication with computers. Development of the system, which we have called "TENEX", has been jointly supported by Bolt Beranek and Newman and ARPA. The system is currently operational, providing extended facilities necessary for our research, including the ability to handle large data bases and large programs, and real-time computation facilities. Its human engineering and program structure facilitate the development of the complex artificial intelligence programs necessary to do English language interaction and development of automatic programming aids. Its utility has become apparent to a number of other members of the ARPA community, and we are cooperating to help them obtain the use of a TENEX system.

TENEX is a system which utilizes paged core memory. In contrast to the DEC 10/40 or 10/50 monitor, TENEX allows users to write their programs as if they had a large, (virtual) memory at their disposal, while at the same time reducing swapping time, since only the "working set" pages of a user's program need to be in core for his program to run. BBN designed and built the necessary paging hardware, which is now being made available to outside research PDP-10 users.

TENEX has a powerful Executive language, which constitutes the user's handle on the Time Sharing System. The language is easy to use; it is based on highly natural, mnemonic commands which allow

command recognition, line editing of commands and multiple input formats to be freely intermingled.

TENEX has a flexible file system. Files are designated by device, directory name, file name, extension, and version. Names and extensions may be up to 39 characters long. A carefully planned set of default values makes it easy to reference commonly used files. Users can have several directories, and an elaborate system for file sharing and protection has been developed. The file system allows both random access and sequential files. Files can be very large, up to 128 million words.

TENEX allows users to run hierarchically dependent "parallel processes" that share memory among themselves. A software interrupt system facilitates interprocess communication.

TENEX is compatible with the standard DEC monitors; most standard user programs and CUSPS that run under a 10/50 operating system will also run under TENEX.

TENEX was officially put into service on June 15, 1970 with capabilities to serve LISP and machine language programmers. TENEX was scheduled to be on the air May 1, 1970. This six week slippage was the only unanticipated delay in a very tight schedule held since November, 1969. While the crash rate was very high in the first week of operation, system's personnel were able to find and correct many bugs which were provoked by having "real users" on the system. Within a week, the crash rate was 3 or 4 a day, and a number of subsystems became available on TENEX, among them FORTRAN IV and TELCOMP III. By the end of the month the crash rate was further reduced. Reliability has continued to improve, with only 3 crashes occurring during the period July 1 through 9.

The preesent TENEX is running slower than we would like, but we expect this to be improved. The reasons for the slowness are:

1. There are only two hardware associative registers in the current pager.
2. Some software "tune-up" still needs to be performed.

We anticipate that more associative registers will be available soon. To aid software "tune-up", system performance measurements are being made. When TENEX first became operational, it was discovered that the I/O routines were quite inefficient. These have been speeded up by a factor of 5 and have made a noticable improvement in system's response. Another aspect of slow response time was traced to the scheduling algorithm, which is now being modified.

User reaction to TENEX has been good. In spite of the system's present slowness, it compares favorably with the 10/50 system, and the niceties and power of the EXEC language, coupled with the flexibility of the file system and the large "virtual" core, have been enough to rapidly win over our research time sharing system users as well as some outside users who have sampled TENEX. Users have been impressed by the excellent reliability that has been achieved in a very short time.

SECTION VI

LIST OF PUBLICATIONS

Bobrow, R.J., and Bobrow, D.G., "EDMS-An Experimental Data Management System", Proceedings of the Third Hawaii Conference on Systems Sciences, January 1970.

Collins, Allan M. and Quillian, M. Ross, "Facilitating Retrieval from Semantic Memory: The Effect of Repeating Part of an Inference", Acta Psychologica 33 Attention and Performance III (A.F. Sanders, ed.) pp. 304-314, North Holland Publishing Co., Amsterdam, 1970.

Collins, Allan M. and Quillian, M. Ross, "Retrieval Time from Semantic Memory", Journal of Verbal Learning and Verbal Behavior 8, 240-247, 1969.

TENEX Design Memos, BBN, January 15, 1970.

The TENEX Executive Manual, BBN, July 1, 1970.

TENEX Monitor Calls Manual, BBN, May 14, 1970.

Woods, William A., "Context-Sensitive Parsing", Communications of the ACM, Vol. 13, (7), July 1970.

Woods, W.A., "Transition Network Grammars for Natural Language Analysis", Communications of the ACM, October 1970.

ABSTRACTS OF PUBLICATIONS

EDMS - AN EXPERIMENTAL DATA MANAGEMENT SYSTEM

Daniel G. Bobrow
Robert J. Bobrow

This paper describes an experimental data-management system (EDMS) implemented within BBN-LISP, a high-level, time-shared, interactive list-processing system which allows associative data structures to be conveniently manipulated in a large virtual memory. Our purpose in constructing this system was to obtain both a useful tool for in-house problems such as contract management, and a flexible instrument for experimentation. The BBN-LISP system was selected because of its unique and powerful capabilities for handling data structures and because of the convenience of the BBN-LISP environment. The purpose of this paper is two-fold: to describe EDMS, and in particular those aspects of its data structure which represent extensions of the structures of previously available systems and the aspects of its design which make it a convenient tool for experimental investigation of various problems in the design of data management systems; and to discuss the advantages and disadvantages of constructing such a system within a higher level language and interactive operating system such as BBN-LISP, both from the point of view of ease of construction and modification of an experimental system, and from the point of view of efficiency of the end product as an operational data management system.

We have implemented in BBN-LISP a data management system which is both a useful tool and an experimental system; the flexibility introduced through the BBN-LISP environment has made the study and investigation of the following subjects extremely convenient:

1. Languages for querying hierarchically and recursively structured data bases.

2. New data structures, including items whose description allows arbitrary recursive hierarchial nesting of subitems (recursive groups) as well as arbitrary repetition of subitems on the same level of the hierarchy (Repeating groups as in SDC/TDMS).

3. New types of elementary data items, (e.g., functions embedded within the data file) allowing improvement in the flexibility and generality of data storage, access and deduction.

4. Search and indexing techniques which make efficient use of the known hierarchial and recursive structure of the data base.

5. Flexible mechanisms for file security - permitting independent access keys for different items and classes of data items, and for the hierarchial relations between items as well.

6. Techniques for system design to permit the final user to tailor the system to his own needs in a flexible manner, without causing system performance to deteriorate drastically.

The needs of users and prospective users have suggested several of the new and experimental features, and feedback from the users on the performance of the system is providing a useful evaluation of such features.

FACILITATING RETRIEVAL FROM SEMANTIC MEMORY: THE EFFECT OF REPEATING PART OF AN INFERENCE

Allan M. Collins
M. Ross Quillian

In Collins and Quillian (1969) we found evidence that people decide whether simple sentences are true or false by using inferences. For instance, a sentence like 'A canary can fly' apparently was confirmed by inference from the two facts that a canary is a bird and that birds can fly. If so, then this has a possible implication for reaction time (RT) to such sentences presented in succession. Prior exposure to one sentence should reduce RT to a second sentence whenever the same fact is involved in confirming both sentences. For example, prior exposure to 'A canary is a bird' should reduce RT to 'A canary can fly' more than to 'A canary can sing', since we assume that no inference is used to confirm the latter sentence. In total eight RT difference predictions were made for various kinds of sentence pairs, and all eight of these predictions held. Two possible models could explain these results.

RETRIEVAL TIME FROM SEMANTIC MEMORY

Allan M. Collins
M. Ross Quillian

To ascertain the truth of a sentence such as "A canary can fly," people utilize long-term memory. Consider two possible organizations of this memory. First, people might store with each kind of bird that flies (e.g. canary) the fact that it can fly. Then they could retrieve this fact directly to decide the sentence is true. An alternative organization would be to store only the generalization that birds can fly, and to infer that "A canary can fly" from the stored information that a canary is a bird and birds can fly. The latter organization is much more economical in terms of storage space but should require longer retrieval times when such inferences are necessary. The results of a true-false reaction-time task were found to support the latter hypothesis about memory organization.

TENEX DESIGN MEMOS

These TENEX memos contain the detailed design specifications of the TENEX system. These memos are both for communicating the TENEX design goals to PDP-10 users, and provide a coherent design structure used by the TENEX systems programmers.

The documents first present an overview of TENEX with emphasis on the need for a time sharing system with extended capabilities on the PDP-10 computer system. These extended capabilities include large programs and data bases; large numbers of users, a diverse user community, real-time capabilities, high system reliability, etc. Then the coding style for TENEX systems code is specified. Emphasis is placed on a specified consistent style and modular code. TENEX is designed to be compatible with existing PDP 10/50 user programs and CUSPs and is implemented in a sufficiently modular fashion to permit exportability to other PDP-10 installations. The rest of the memos deal with the structure of TENEX internally and as it appears to the time sharing user.

In TENEX there exists an entitity called a job which is an hierarchical structure of one or more processes. A process in an entity which has a 256K address space and is capable of being run by a processor. Processes may share memory with one or more other processes. Interprocess communication is made possible by a software interrupt system. A "mailbox file" is also available for inter job communication. All I/O in TENEX is done through an elaborate file system which has been designed for rapid sequential and random access to files.

A great deal of emphasis was placed on human-engineering at all levels of the system design. The well human engineered time sharing EXECUTIVE command language, file name recognition logic, and terminal service routine features typify the results of this effort.

THE TENEX EXECUTIVE MANUAL

In TENEX the Executive is the primary vehicle through which users communicate and work with the system. The Executive grants users access to TENEX and allows them to access and control the subsystems, virtual computer, and file system that comprise the major TENEX facilities.

Communication with the Executive takes the form of a dialogue in which the user gives a command, the Executive performs the desired action and then awaits a new command. The collection of available commands, together with special characters and conventions make up what is known as the Executive Language. Executive commands are entered through TENEX terminals; in future versions of TENEX it will also be possible to control operations through Executive commands stored in Files.

Executive commands are made up of English language words and abbreviations and have a simple but rich structure. Each command begins with a "keyword" that identifies its main function. Following this, there may be arguments, such as file names or user names, additional keywords to specify options, and/or "noise words" (typed back by TENEX) to make the command more understandable. With certain commands, selection among many options is accomplished by sub commands, having the same structure as suggested above, and inferred immediately following the main command.

The Executive Language features a flexible abbreviation/recognition scheme. For any command (or part of it) TENEX will accept any initial substring of characters, from the minimum necessary to identify the input up to the full input. Furthermore

the user can, if desired, request that TENEX "recognize" any abbreviated input by typing back the omitted characters. Full, abbreviated or recognized input can be mixed freely in a session or even within a command.

The Executive Language includes group of commands for access and use charge accounting, file manipulation, subsystem control, and program control and debugging. In addition there are a number of queries that provide information about the state of the system and the user's job, and a group of commands for informing TENEX about terminal characteristics.

Executive Command arguments specify things such as user names, account numbers, core locations, and file names. File names, in particular, possess some unique and powerful features. A file name can select any file or input/output device (TENEX treats these as files) in the system. File names contain five identifiers that select device, directory name, file name, extension, and version. Through device and directory name, the user can select any input/output device or disc file directory. File name further narrows the selection to pinpoint a particular file. Extension allows the user to name files containing transformed versions of the same data, while version helps keep track of successive changes to a single file. In practice, it usually takes much less than the five designators to specify a file. Often, some designators are irrelevant and can be left out. The Executive will perform "recognition" on designators such as directory and file name. Still other designators can be omitted, in which case, TENEX supplies "default values. For example, version numbering can, if desired, be handled entirely automatically by TENEX.

TENEX MONITOR CALLS MANUAL

The TENEX JSYS Manual contains the detailed specifications of TENEX monitor calls. JSYS (which stands for Jump to SYStem) is an instruction designed and implemented by BBN for rapid dispatch to monitor subroutines from user programs. (JSYS may also be used to call user subroutines from user programs). This manual is for specifying the function of each monitor call to machine language TENEX users. It also provides a detailed design specification for the implementation of each monitor call to TENEX systems programmers.

The set of JSYS monitor calls may be thought of as augmenting the basic instruction set of the PDP-10 for both user programs and monitor software, JSYS's may be called within the monitor by other JSYS's to a limited depth. The 10/50 compatibility software essentially translates 10/50 monitor calls to TENEX JSYS's.

Certain JSYS's require privileged process or user capabilities for their successful execution. Some JSYS's have options which require some special capabilities.

JSYS monitor calls exist for logging jobs off/on the system, accounting functions, file accesses, obtaining information about the system, transmitting and enabling/disabling special capabilities, controlling forks, saving and getting address spaces, performing I/O conversion, creating directory name entries etc.

CONTEXT-SENSITIVE PARSING

William A. Woods

This paper presents a canonical form for context-sensicive derivations and a parsing algorithm which finds each context sensitive analysis once and only once. The amount of memory required by the algorithm is essentially no more than that required to store a single complete derivation. In addition, a modified version of the basic algorithm is presented which blocks infinite analyses for grammars which contain loops. The algorithm is also compared with several previous parsers for context-sensitive grammars and general rewriting systems, and the difference between the two types of analyses is discussed, The algorithm appears to be complementary to an algorithm by S. Kuno in several respects, including the space-time trade-off and the degree of context dependence involved.

TRANSITION NETWORK GRAMMARS FOR NATURAL LANGUAGE ANALYSIS

William A. Woods

The use of augmented transition network grammars for the analysis of natural language sentences is described. Structure-building actions associated with the arcs of the grammar network allow for the reordering, restructuring, and copying of constituents necessary to produce deep-structure representations of the type normally obtained from a transformational analysis, and conditions on the arcs allow for a powerful selectivity which can rule out meaningless analyses and take advantage of semantic information to guide the parsing. The advantages of this model for natural language analysis are discussed in detail and illustrated by examples. An implementation of an experimental parsing system for transition network grammars is briefly described.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

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

1. ORIGINATING ACTIVITY (Corporate author)
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
NATURAL COMMUNICATION WITH COMPUTERS III

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Scientific. Final. 31 August 1969 through 30 September 1970. Approved 1 October 1970

5. AUTHOR(S) (First name, middle initial, last name)
Daniel G. Bobrow

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
30 September 1970	40	8

8a. CONTRACT OR GRANT NO
F19628-70-C-0013
ARPA Order No. 627

b. PROJECT NO
8668

c. DoD Element 61101D

d. DoD Subelement n/a

9a. ORIGINATOR'S REPORT NUMBER(S)
BBN Report No. 2015

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
AFCRL-70-0505

10. DISTRIBUTION STATEMENT
1-This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES
This research was supported by the Advanced Research Projects Agency

12. SPONSORING MILITARY ACTIVITY
Air Force Cambridge Research Laboratories (CRB)
L.G. Hanscom Field
Bedford, Massachusetts 01730

13. ABSTRACT

Bolt Beranek and Newman has been engaged in a continuing research program whose goal is to develop techniques to facilitate natural communication with computers, people, other computers, and real-time devices. This work has been supported under two separate contracts both funded by the Advanced Research Projects Agency. The work has been carried on as an integrated research program; work on semantic automatic language processing and automatic programming aids was performed under contract F19628-70-C-0013 through Hanscom Field Air Force Cambridge Research Laboratories and work on the development of time-sharing systems was done, under contract XC-3169(62-7036)70R.

In accordance with our responsibility to report the results of our investigation we have prepared a number of documents, some of which have been distributed as scientific reports under the contract, some of which have been published as technical articles in professional journals and a few of which are internal memoranda which we have made available to interested outside parties. Following a brief summary of our research we include abstracts of these documents for further information.

DD FORM 1473

Unclassified

Security Classification

A-31409

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
computer systems						
time-sharing						
man-computer interaction						
linguistic rule testers						
linguistics						
LISP						
list processing						
format directed list processing						
on-line systems						
semantics						
syntax						
linguistic theory						
human engineering						
man-machine systems						
human factors						
natural language						
language comprehension						
computational linguistics						
modeling of computer systems						
simulation						
semantic network						
English language comprehension						
memory models						
real time systems						