A STUDENT-ORIENTED NATURAL LANGUAGE ENVIRONMENT FOR LEARNING LISP

Richard O. Hart
University of New Hampshire
Durham, New Hampshire

Elliot B. Koffman
Temple University
Philadelphia, Pennsylvania

Abstract

A computer-based instructional environment for students learning LISP is described. Its design includes a LISP Interpreter with an extended error-handling capability for evaluating functions written by students. Students can ask questions in natural language about LISP in general and concerning specific LISP functions which they have defined. A fuzzy parser interprets the student requests and builds a LISP function which operates on a semantic network to determine its response.

Introduction

As an application area of Artificial Intelligence, computer-aided instruction (CAI), has received considerable interest (1). Carbonell's SCHOLAR (2) is perhaps the best known example of a "mixed-initiative" CAI system. Through SCHOLAR, students can ask questions about concepts relevant to the geography of South America.

Koffman and Blount (3) have developed a system to teach machine-language programming. This system can generate programming problems, and monitor and assist students in coding their solution programs. This system concatenates a small set of problem primitives and associated solution routines in order to create and solve meaningful programming problems.

Natural language understanding is becoming more important In CAI system. Many powerful parsers have been developed for very extensive sub-Bets of English including Wood's parser (A) and Vlnograd's PROGRAMMER (5). These methods are beginning to be of use for question answering subsystems and for interpreting student responses. Of particular Interest is the work done by Brown et al (6). It is his semantlcally driven fuzzy grammar that has Influenced this research.

Brown analyzed student scenarios Involving electronic circuit trouble-shooting and discovered that the concepts and items discussed fell Into small numbers of semantic groupings or categories.

These included: measurements, control settings, circuit elements, and circuit locations relative to the electronic circuit. The parse was then accomplished by anticipating and looking for phrases which matched these semantic categories where they might be appropriate. The parser's fuzzlness stems from the fact that words that are not understood can be skipped over.

Teaching a programming language embraces two major tasks. One is presenting new material In an orderly controlled fashion. The other involves making available a friendly, informative environment that allows experimentation, and that can recognize errors and misconceptions and point them out. With ISLE (Interactive Student LISP Environment) we have concentrated on this second item by developing a computer-based environment for LISP. The system knows about the structure and semantics of the LISP language and can answer questions about LISP data structures, concepts and functions; both those functions defined by the student and the primitives internal to the system.

As an example consider the following dialogue:

```
STUDENT:  Is Atom a function?
MACHINE:  Yes, Atom is subr.
STUDENT:  How many arguments does it take?
MACHINE:  Atom is a subr and takes 1 argument.
```

Notice that questions are answered With a complete statement to avoid any ambiguity, also note the use of the pronoun it by the student. Some limited anaphoric reference Is permitted by the system to allow a more natural dialogue.

```
STUDENT:  Describe it.
MACHINE:  Atom is a subr. and a predicate func-
          tion. It returns true (the Atom T)
          if its argument is an atom, nil other-
          wise.

STUDENT:  What is an Atom?
MACHINE:  An Atom is the most elementary type
          of s-expression. It may be either a
          number or a literal atom.
```

Some words like atom can refer to more than one thing. The correct reference is found by looking at the context of the word and making a decision based on plural markers, determiners, or qualifying adjectives.

```
STUDENT:  Are property lists kinds of lists?
MACHINE:  A property or P list is an alternating
          list of literal atom property names
          and corresponding property values.
          It can be searched using the function
          GET.
```

Subset-superset and part-part of relationships are used extensively by the system for answering questions of this type. A yes answer is implied here by giving the student a definition of the subset item.

```
STUDENT:  DEFINE ( ( (FACT (LAMBDA (X) (COND
                 ((ZEROO X) 1)
            (T (TIMES X (FACT (SUB1 X))))))))))
```

```
  MACHINE:  Arguments for Evalquote:
            DEFINE
 (((FACT (LAMBDA (X) (COND((ZEROO X) L) (T(TIMES X
 (FACT (SUB1 X))))))))
 VALUE IS:
 (FACT)
```

Any student input that has the form of a function followed by an argument list is assumed to be input for the evaluator. Here the student defines a function to compute factorials.

```
  STUDENT:  FACT  (3))
  MACHINE:  Arguments for Evalquote:
            (3)
            VALUE IS:
            Undefined function found.
            It is ZEROO found in the function fact.
```

In defining FACT the student mistyped ZEROP. The error is found during evaluation and returned to the student. Notice that terminating symbols are necessary at the end of each student input line. They may be periods, question marks or an extra right parenthesis.

```
  STUDENT:  PRINT FACT
  MACHINE:  (LAMBDA  (X)  (COND & &))
```

When printing function definitions, only the highest elements of the definition appear; lower level structures are replaced with &.

```
  STUDENT:  3 2 P.
  MACHINE:  ((ZEROO X) 1)
```

Now using commands to the editor, the student looks at the lower level where he remembers ZEROP should go.

```
  STUDENT:  1 (1 ZEROP) P.
  MACHINE:  (ZEROP X)
```

The error is now corrected and this can be checked by evaluation or by asking another question.

```
  STUDENT:  Does FACT call any undefined func-
            tions?
  MACHINE:  FACT calls no undefined functions.

  STUDENT:  FACT (3))
  MACHINE:  Arguments for Evalquote.
            FACT
            (3)
            VALUE IS:
            6
```

System Organization

The system is built around several subnodules as shown In figure 1. Each stud ent command or request is received by the monitor where it is classified as input for either the editor, the evaluator, or the parser. Any input which is not either a list of edit commands or a function followed by an argument list is assumed to be a request for information and so is passed to the parser. Anything the parser cannot handle is rejected and the student is asked to rephrase or break up the question.

The parser and the semantic rountines work together to answer factual questions about the LISP language, its data structures and functions, and about the functions the student has defined. The parse mechanism has a semantic basis in that specific words or phrases are looked for that refer to things the system knows about, i.e., functions, definitions, or entries in the semantic network. These items are divided into groups or categories that are semantically similar.

Semantically similar items are those that might fit in a given slot in a sentence or question, and that fall into a superset classification such as data structures or function names. The result of the parse is an executable LISP function whose evaluation causes a response to be generated for the student.

The evaluator evaluates student functions when called upon, accepting nearly any LISP 1.5 constructions. When student errors are found, it reports the type of error and in what function it occured to the student. Editor commands can be used to look around inside of function definitions and to insert, delete, and change parts of the definition.

Semantic Categories

The ISLF- system uses four semantic categories. These categories allow the parser to be somewhat selective in choosing what to look for next while processing the student's input. The semantic categories are the following: (1) Functions (2) function types (3) data structures (A) other general concepts.

Each semantic category is represented in the implementation by a LISP function which will check, beginning at the head of the current sentence, for a phrase that matches an element in that category. Depending on the fuzziness set by the grammar at that moment, one or more of the initial words in the current sentence may be skipped.

The first semantic category is that of function names. It is represented in the grammar by FNNAME. This category consists of all functions that are currently defined in the system; that is, all SUBRS and FSUBRS plus any functions that may have been defined as EXPRS or FEXPRS by the user. FNNAME and the other three semantic category matching routines all operate in the same manner. They operate like any other grammar rule function in that they accept some tail of the input sentence. They can return NIL if they find no match at some point In that tail. If they do find a match, they return the next tall (consisting of the original tall minus the matched portion).

They also set the atom RESULT with a global (CSET) value that specifies what semantic elements were matched along with their semantic category. As an example, consider the tall 'THF FUKCTION FACT CALL ANY SUBRS' passed as an argument to FNNAME. (Perhaps as part of the question: "Can the function FACT call any subrs?") FNNAME would match the three words 'THE FUNCTION FACT. It would return the tail, 'CALL ANY SUBRS' and set the value of RESULT to the list (FN FACT). FACT is the semantic entity matched, while FN represents the category, function names. Table 1 shows some phrases that might be matched by FNNAME and the corresponding value of RESULT generated.

Table 1
Translations by the function FNNAME

Input Phrase	Value of RESULT
'CAR'	(FN CAR)
'CAR and CDR'	(FN CDR CAR)
'THE TWO FUNCTIONS CAR AND CDR'	(FN CDR CAR)
'THE 3 FUNCTIONS X, Y, AND Z'	(FN Z Y X)

Table 2
Translations by the function FN/TYPE

Input Phrase	Value of RESULT
'FSUBRS AND SUBRS'	(FTYPE SUBR FSUBR)
'ARITHMETIC AND BOOLEAN FUNCTION'	(FTYPE BOOLEAN ARITHMETIC)
'A CONDITIONAL FUNCTION'	(FTYPE CONDITIONAL)
'FUNCTIONS'	(FTYPE FUNCTION)

The semantic category consisting of function types is represented in the grammar by FN/TYPE. The operation of this function is very similar to that of FNNAME. In this case, a word or phrase that somehow describes or classifies a group of functions is looked for. This includes function types like expr, subr, fsubr, and fexpr as well as things like arithmetic functions and conditional functions. Table 2 shows the value of RESULT that will be produced for a few input phrases.

Perhaps the largest semantic category is that consisting of LISP data structures. This is represented in the grammar by STRUCTURES and in the implementation by a function of the same name. Table 3 shows some of the phrases accepted in this category by the function STRUCTURES.

The fourth and last semantic category serves as a catch-all for anything not included in the other three. This category is represented by CONCEPTS in the grammar and the implementation. A few phrases recognized by the CONCEPTS function are shown in Table 4.

Table 3
Translations by Structures

Input Phrase	Value of RESULT
'ATOM'	(STRUCTURE ATOM)
'A and P LISTS'	(STRUCTURE A-LIST P-LIST)
'A LAMBDA EXPRESSION'	(STRUCTURE LAMBDA-EXPRESSION)
'DOTTED PAIRS'	(STRUCTURE DOTTED-PAIR)

Table 4
Translations by Concepts

Input Phrase	Value of RESULT
'GARBAGE COLLECTIONS'	(CONCEPT GARBAGE-COLLECTOR)
'LISP'	(CONCEPT LISP)
'A FREE VARIABLE'	(CONCEPT FREE-VARIABLE)

The Grammar And Its Implementation

The heart of the English understanding component of the system is a BNF grammar. After a line has been read in, an interpretation of it is attempted. In the SOPHIE system (6), every nonterminal is considered a semantic entity to be searched for when necessary. In the ISLE system, however, only a few of the rules are actually concerned with semantic entitles or categories. These semantic entities are defined as only those things which have entry in the semantic network. The rules which embody certain semantic groups have already been described. The rest of the grammer rules are used to identify requests for certain relationships or properties of the semantic entitles.

Most programs which make use of a grammar use some kind of parser or grammar interpreter. This parser (a program) then uses a table or array in which the grammar rules are stored (data). Special control structures must be set up to control backing up when an incorrect parse is begun. In ISLE, this grammar is implemented directly in LISP. For each rule (nonterminal) in the grammar, there is a corresponding LISP function with the same name implementing that rule. The LISP control structures make this implementation relatively easy due to the recursive definition of LISP functions in general and the use of the special built-in functions; COND, AND, and OR. Backup is automatic as each rule-function can let its calling rule-functions know of its failure on return. All pointers and variable values will again be those originally set in the calling function. There is nothing to undo or redo as the LISP control structure handles this automatically.

The Semantic Routines

The parsing operation, if it is successful, will produce another LISP function to be evaluated. Some of these functions and the sentences that produced them are given in Table 5. Each is a call to a predefined semantic routine. The functions FN, FTYPE, CONCEPT, and STRUCTURE retrieve the desired semantic information for their arguments. In this way words such aa ATOM are disambiguated. For example, (FN ATOM) will retrieve information relevant to the function ATOM, while (STRUCTURE ATOM) will retrieve the Information for pronouns which it does by matching its arguments against the semantic categories of previously mentioned items.

Table 5
Sentences and their Translation into LISP Functions

```
IS ATOM A FUNCTION?
    (RELATE (FN ATOM) (FTYPE FUNCTION)

HOW MANY ARGUMENTS DOES IT TAKE?
    (ARGCOUNT  (LIST (PREF FN)))

DESCRIBE IT.
    (DESCRIBE (LIST (PREF FN FTYPE CONCEPT
     STRUCTURE)))

WHAT IS AN ATOM?
    (DESCRIBE (LIST (STRUCTURE ATOM )))
```

ISLE's semantic routines are all specialists for answering their own type6 of questions. Some take information directly from the network to be ftiven to the student or to be used in comparison or relationship tests. DESCRIBE, for example, gives the student a pre-defined definition or description if it exists. In the case of student defined functions, it tells the student the type of function it is. RELATE reports on 'superset', 'subset^1, and 'part-of relationships between Its argument, or in the case of student defined functions—the actual function definition, to tell how many arguments a particular function has.

The permanent semantic Information used by these functions is set up as association lists of relationships and values for each semantic entity. Table 6 shows this Information for the structural Item atom. The value of the relationship TEST is the name of a predicate function which teste for the associated semantic entity. In this case, the function ATOM tests for the structure which Is an atom. TYPE and TYPE OF indicate subset and superset relationships, and DESCRIPTION indicates a literal definition of the item.

Table 6

Semantic Information for the Structure Atom

```
(( TEST .  ATOM)
(TYPE OF .  (S-EXPRESSION INDICATOR))
( TYPE .  (LITERAL NUMBER))
( PART OF .  (S-EXPRESSION DOTTED-PAIR LIST))
( DESCRIPTION .  ((AN ATOM IS THE MOST ELEMENTARY
  TYPE OF S-EXPRESSION (DOT))
  (IT MAY BE EITHER A NUMBER OR A LITERAL ATOM
     (DOT)))))
```

Other, temporary Information that might be used by the semantic routines can be created and changed in various ways. When a student defines a function, the function is analyzed and lists of the variables It binds or uses and the functions It calls are created. This information is used by the routines which handle questions about the student's functions and Is updated whenever a function is edited or redefined. The editor and the evaluator also store information that could be used by the question-answering system. This Is done whenever errors occur and includes Information about the current state of the evaluator or editor (e.g., the association list) and the cause of the error. This would allow the student to obtain more information about the source of the error and what the evaluator (or editor) was doing before the error occured.

Knowing About Student Defined Functions

Knowledge concerning functions defined by the student falls into two distinct categories. The first is the category that might be called semantics; that is what the function does (or should do or perhaps what the student thinks It should do). This involves why the function does what it does, when it does it, and how it does it. The other category is that of syntax or function structure. The function structure consists of only the information contained in Its definition—the functions It calls, the variables It binds, sets, or uses, its arguments and any dlscernable relationships between these basic components. We have concentrated on this second category and ignored the first one. This system handles knowledge of user's or student's functions in two places. First the functions must each be scanned as they are defined. Information concerning the use of variables and function calls is recorded to be used by the semantic routines when needed. This is a sort of preprocessing to collect information for the semantic routines.

During the scan, the following properties are representative of the ones set up on the function name's property list:

VFNS- variable functions; those functions that the given function calls which are not defined.
EFNS- EXPR'S- those EXPRS that the given function calls.
SFNS- SUBR's-those SUBR'S that the function calls.
SETVRS- those variables which the function sets.
BINDS- those variables which the function binds, those that are found in PROG and lambda expression variable lists.

These properties are then used by the semantic routines which answer questions about the user defined functions. These include questions like:

WHAT VARIABLES DOES FOO SET?
DOES FIE CALL FUM?
DOES FOO CALL ANY SUBRS?
WHAT VARIABLES ARE BOUND BY FUM THAT FOO SETS?

Once the call to these semantic routines has been generated the answer to the question Is easily determined through the use of the above property values for the desired functions.

Problems Encountered

The problems encountered in dealing with natural language seem to fall Into four separate areas. It is difficult to measure the extent of each of these problems at present. There are viable approaches to the solution of each which should be examined.

a) The use of adjective and modifying phrases.

Students can be expected to try to describe various things concerning their programs in many different ways. At present, modifiers are looked for by the grammar at certain points and then ignored or, if necessary, used in writing the semantic interpretation functions. Problems can occur when one of the basic semantic entities is used as a modifier, for example In phrases like LIST VARIABLES and VARIABLE LISTS. This problem would be solved by adding more information to the semantic network. The parser could then use this Information to decide how to handle the semantic entitles when they are used as either modifiers or as nouns. Certain other modifiers that are not classified as semantic entities should be Included in the network to allow variations of meaning to be understood that are now ignored.

b) Cause and effect relationships.

Often during the writing and debugging of a program, students will want to ask 'What happens if ...' or 'Why' questions. To answer questions of this type it Is usually necessary to compare desired with actual results. It Is also necessary to know about various side-effects that might occur such as setting global or free variables. The information necessary to handle such questions could be obtained by tracing the evaluation of the student's functions. This could then be checked for validity in some way (i.e., compared against some desired result) or reported back to the student. It would also be useful to allow the system to simulate the evaluation of certain forms or lisp functions. This might take care of a large class of 'what happen if ...' type questions. Others might be handled by adding more information to the network, (i.e., that an error will occur when CAR is called with an atom as its argument).

At the present, none of these cause and effect type questions can be asked because of a lack *of* the necessary semantic information. If it were available, the addition of a few more rules to the grammar would allow many such question to be asked.

c) Verbs and action relationships.

The system now can only handle simple binary relationships between the semantic entities known to it. These relations can only describe static situations such as subset-superset (an EXPR is a FUNCTION), part-part of (a VARIABLE LIST is part of a LAMBDA-EXPRESSION) and function structure (function X can call function Y). When talking about writing and using LISP functions, students often need to use verbs specifying actions or relationships involving two or more things. These verbs include the following among others—DO, MAKE, EDIT, CHANGE, INSERT, STOP, and REMOVE. A taxonomy of the ways students use these verbs would be very helpful. The number of such verbs that might be needed may be small enough to keep the system relatively efficient. The best place to try this would be to attempt some additions to the grammar that would allow the student to edit LISP functions using English sentences instead of edit commands. Most edition operations involve actions on various object and subject nouns.

d) Pronouns, anaphoric deletion, and ellipsis.

Students often use pronouns or leave out words completely that might seem obvious. We have attempted to allow the use of pronouns, but in some cases the results are not the desired ones. For example, the following dialogue:

```
STUDENT:  What are atoms?
MACHINE:  An atom is the most elementary type
          of s-expression
            It may be either a number or a lit-
          eral atom.

STUDENT:  How many arguments does PLUS need?
MACHINE:  PLUS is a FSUBR and can take any
          number of arguments.

STUDENT:  Describe them.
```

At this point the system would not have recognized the relationship between 'PLUS' and 'arguments* due to Its inability at present to recognize the relationship between modifier and modifled entity. A very minimal history of the previous dialogue is kept so that, although the system remembers that PLUS was discussed previously, it does not remember that it was its arguments that were in question. Therefore, the system thinks that 'them' refers to 'atoms' and not the arguments of PLUS, and describes an atom. This particular problem could be solved by handling modifiers in a better, more consistant manner and by keeping a more complete history of the previous dialogue.

The handling of anaphoric deletion and ellipsis is, however, even more difficult. The following, seemingly natural dialogue, would not be accepted:

```
STUDENT:  IS X AN ATOM?
MACHINE:  YES, X IS AN ATOM.
STUDENT:  AND Y?
```

The student has deleted most of the question "IS Y AN ATOM?' assuming that Y is to be substituted for X in the previous question. There are many similar deletions to be found in nearly any natural dialogue. Verbs, subjects, object, modifiers, or any combination of these might be deleted. A study and classification of the various types of ellipsis that might occur in a student-teacher dialogue would be very helpful here.

The most important lesson learned concerns the Interaction of semantic and syntactic information in dealing with and understanding natural language. Heavy use Is made *of* semantic information during the parsing operation. A parse will in fact fail If it does not make sense based on what the system knows, even though it may be syntactically correct.

The shortcomings here involve the structuring and use of the semantic information. There is a large literature describing how to form and use grammars and other syntactic structures; however, similar studies for semantic structures are not yet available.

There are many models under study such as semantic networks and conceptual models (7). However, as yet, there are no definitive measures of the capabilities and limitations for these various techniques. Also, there are no studies comparing various means of Implementation and the storage requirements of these model**. Hopefully, research along these lines would aid those attempting to use these models for CAI In other application areas.

Conclusions

This system is undergoing continued development. The question-answerer is being expanded to allow the student to get more of the Information he or she might want and to perform more edit functions in English.

ISLE Is implemented in LISP which runs interactively on an IBM 360/65. This Interactive LISP is an improved version of the Waterloo LISP which uses a cathode-ray display as the active user terminal. The system runs in 250k bytes of memory. Each question is processed in one second or less.

Preliminary indications are that the system will serve as a useful tool for familiarizing a student with LISP concepts. The question answering capability allows a student to inquire about the semantics of LISP; he can use the LISP student evaluator to test his knowledge of LISP syntax and to help him correct his errors. The expanded diagnostic information presented should help hire clear-up initial misconceptions and ease his transition from ISLE to the standard LISP evaluator

This approach appears to be general in that one could present any material of a factual nature in a similar manner. SOPHIE (6) is an example of a similar system for teaching electronic-circuit analysis and trouble-shooting. Other programming languages, logic circuit design, and basic algebra and calculus might possibly be taught using a similar computer environment.

The important lesson learned concerns the interaction of semantic and syntactic information in dealing with and understanding *natural* language. Heavy use is made of semantic information during the parsing operation. A parse will in fact fail if it does not make sense based on what the system knows, even though it may be syntactically correct.

The shortcomings here involve the structuring and use of the semantic information. There is a large literature describing how to form and use grammars and other syntactic structures; however, similar studies for semantic structures are not yet available.

There are many models under study such as semantic networks and conceptual models (7). However, as yet there are not definitive measures of the capabilities and limitations of these various techniques. Also, there are no studies comparing various means of implementation and the storage requirements of these models. Hopefully, research along these lines would aid those attempting to use these models for CAI or other application areas.

References

(1) E. B. Koffman, Generative computer assisted instruction: An Application of Artificial Intelligence to CAI, Proceedings of the 1st USA-Japan Computer Conference, Tokyo, 1972.

(2) J. R. Carbonell, AI in CAI: An Artificial Intelligence Approach to Computer-Assisted Instruction, IEEE Transactions on Man-Machine Systems, Vol. MMS-11, No. 4, December, 1970.

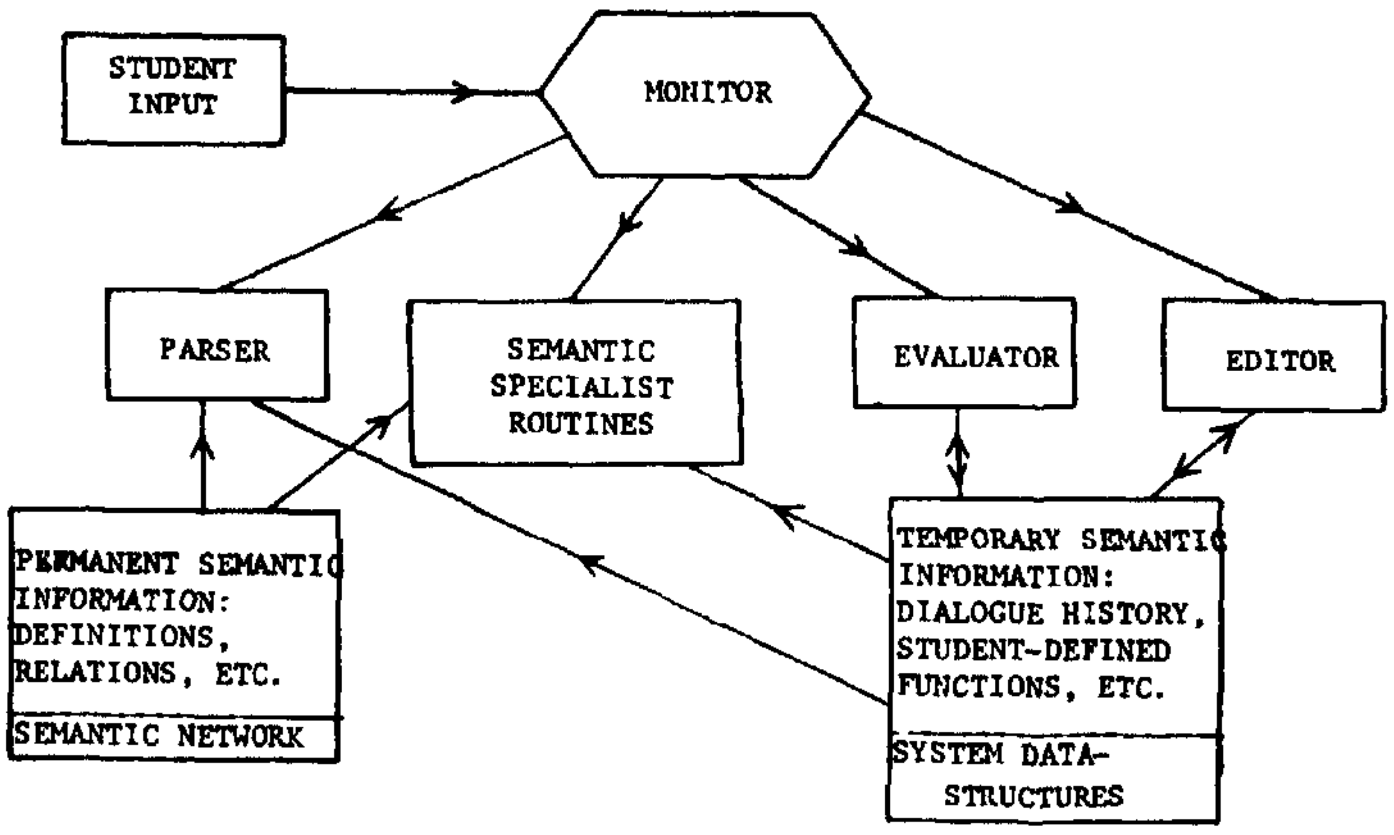
(3) E. B. Koffman and S. E. Blount, Artificial Intelligence and Automatic Programming In CAI, Proceedings of the 2nd International Joint Conference on Artificial Intelligence, 1973.

(4) W. A. Woods, Transition Network Grammars For National Language Analysis, Communications of The ACM, Vol. 13, No. 10, October, 1974.

(5) T. Winograd, Understanding National Language, Academic Press, New York, 1973.

(6) J. S. Brown, R. R. Burton, and A. G. Bell, SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Trouble-shooting, Bolt, Beranek, and Newman Report No. 2790, Cambridge, Massachusetts, March, 1974.

(7) R. C. Schank, Identification of Conceptualizations Underlying National Language, in Computer Models of Thought and Language, edited by R. C. Schank and K. M. Colby, W. H. Freeman & Co., San Francisco, 1973.



STUDENT INPUT
MONITOR
PARSER
SEMANTIC SPECIALIST ROUTINES
EVALUATOR
EDITOR
PERMANENT SEMANTIC INFORMATION: DEFINITIONS, RELATIONS, ETC.
SEMANTIC NETWORK
TEMPORARY SEMANTIC INFORMATION: DIALOGUE HISTORY, STUDENT-DEFINED FUNCTIONS, ETC.
SYSTEM DATA-STRUCTURES

Fig. 1. System Organization