NATURAL LANGUAGE INPUT FOR A COMPUTER PROBLEM SOLVING SYSTEM

by

DANIEL G. BOBROW

B.S., Rensselaer Polytechnic Institute
(1957)

S.M., Harvard University
(1958)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF
PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF
TECHNOLOGY

September, 1964

Signature of Author...
Department of Mathematics, June 3, 1964

Certified by...
Thesis Supervisor

....
Chairman, Departmental Committee
on Graduate Students

Acknowledgements

The work reported herein was supported in part by the MIT Computation Center, and in part by Project MAC, an MIT research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government. Associated preliminary research was supported in part by Bolt, Beranek and Newman, Inc., Cambridge, Massachusetts, and in part by System Development Corporation, Santa Monica, California.

The author expresses his gratitude to Marvin Minsky for his supervision of this thesis; to Victor Yngve and Murray Eden for their critical reading of the manuscript; to Herbert Simon, Noam Chomsky and Seymour Papert for enlightening discussions of various aspects of the research; and to his wife Eileen for her patience, encouragement and unfailing confidence without which this thesis could not have been completed.

NATURAL LANGUAGE INPUT FOR A COMPUTER PROBLEM SOLVING SYSTEM

by

DANIEL G. BOBROW

Submitted to the Department of Mathematics on June 3, 1964 in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ABSTRACT

The STUDENT problem solving system, programmed in LISP, accepts as input a comfortable but restricted subset of English which can express a wide variety of algebra story problems. STUDENT finds the solution to a large class of these problems. STUDENT can utilize a store of global information not specific to any one problem, and may make assumptions about the interpretation of ambiguities in the wording of the problem being solved. If it uses such information, or makes any assumptions, STUDENT communicates this fact to the user.

The thesis includes a summary of other English language question-answering systems. All these systems, and STUDENT, are evaluated according to four standard criteria.

The linguistic analysis in STUDENT is a first approximation to the analytic portion of a semantic theory of discourse outlined in the thesis. STUDENT finds the set of kernel sentences which are the base of the input discourse, and transforms this sequence of kernel sentences into a set of simultaneous equations which form the semantic base of the STUDENT system. STUDENT then tries to solve this set of equations for the values of requested unknowns. If it is successful it gives the answers in English. If not, STUDENT asks the user for more information, and indicates the nature of the desired information. The STUDENT system is a first step toward natural language communication with computers. Further work on the semantic theory proposed should result in much more sophisticated systems.

Thesis Supervisor: Marvin L. Minsky
Title: Professor of Electrical Engineering

TABLE OF CONTENTS

CHAPTER I: INTRODUCTION

The aim of the research reported here was to discover how one could build a computer program which could communicate with people in a natural language within some restricted problem domain. In the course of this investigation, I wrote a set of computer programs, the STUDENT system, which accepts as input a comfortable but restricted subset of English which can be used to express a wide variety of algebra story problems. The problems shown in Figure 1 illustrate some of the communication and problem solving capabilities of this system.

In the following discussion, I shall use phrases such as "the computer understands English". In all such cases, the "English" is just the restricted subset of English which is allowable as input for the computer program under discussion. In addition, for purposes of this report I have adopted the following operational definition of understanding. A computer <u>understands</u> a subset of English if it accepts input sentences which are members of this subset, and answers questions based on information contained in the input. The STUDENT system understands English in this sense.

A. The Problem Context of the STUDENT System.

In constructing a question-answering system, many problems are greatly simplified if the problem context is restricted. The simplification resulting from the restrictions embodied in the STUDENT system, and the reasons these simplifications arise, will be discussed in detail in the body of this report.

The STUDENT system is designed to answer questions embedded

```
(THE PROBLEM TO BE SOLVED IS)
(THE DISTANCE FROM NEW YORK TO LOS ANGELES IS 3000 MILES .
IF THE AVERAGE SPEED OF A JET PLANE IS 600 MILES PER HOUR ,
FIND THE TIME IT TAKES TO TRAVEL FROM NEW YORK TO LOS ANGELES
BY JET .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02517 (TIME (IT / PRO) TAKES TO TRAVEL FROM NEW YORK
TO LOS ANGELES BY JET))

(EQUAL (AVERAGE SPEED OF JET PLANE) (QUOTIENT (TIMES 600 (MILES))
(TIMES 1 (HOURS))))

(EQUAL (DISTANCE FROM NEW YORK TO LOS ANGELES) (TIMES 3000
(MILES)))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(USING THE FOLLOWING KNOWN RELATIONSHIPS)
((EQUAL (DISTANCE) (TIMES (SPEED) (TIME))) (EQUAL (DISTANCE)
(TIMES (GAS CONSUMPTION) (NUMBER OF GALLONS OF GAS USED))))

(ASSUMING THAT)
((SPEED) IS EQUAL TO (AVERAGE SPEED OF JET PLANE))

(ASSUMING THAT)
((TIME) IS EQUAL TO (TIME (IT / PRO) TAKES TO TRAVEL FROM NEW
YORK TO LOS ANGELES BY JET))

(ASSUMING THAT)
((DISTANCE) IS EQUAL TO (DISTANCE FROM NEW YORK TO LOS ANGELES))

(THE TIME IT TAKES TO TRAVEL FROM NEW YORK TO LOS ANGELES BY
JET IS 5 HOURS)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE PRICE OF A RADIO IS 69.70 DOLLARS . IF THIS PRICE IS
15 PERCENT LESS THAN THE MARKED PRICE , FIND THE MARKED PRICE
.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02515 (MARKED PRICE))

(EQUAL (PRICE OF RADIO) (TIMES .8499 (MARKED PRICE)))

(EQUAL (PRICE OF RADIO) (TIMES 69.70 (DOLLARS)))

(THE MARKED PRICE IS 82 DOLLARS)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF TWO NUMBERS IS 111 . ONE OF THE NUMBERS IS CONSECUTIVE
TO THE OTHER NUMBER . FIND THE TWO NUMBERS .)

TRYING POSSIBLE IDIOMS

(THE PROBLEM WITH AN IDIOMATIC SUBSTUTION IS)
(THE SUM OF ONE OF THE NUMBERS AND THE OTHER NUMBER IS 111
. ONE OF THE NUMBERS IS CONSECUTIVE TO THE OTHER NUMBER . FIND
THE ONE OF THE NUMBERS AND THE OTHER NUMBER .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02522 (OTHER NUMBER))

(EQUAL G02521 (ONE OF NUMBERS))

(EQUAL (ONE OF NUMBERS) (PLUS 1 (OTHER NUMBER)))

(EQUAL (PLUS (ONE OF NUMBERS) (OTHER NUMBER)) 111)

(THE ONE OF THE NUMBERS IS 56)

(THE OTHER NUMBER IS 55)
```

```
(THE PROBLEM TO BE SOLVED IS)
(BILL S FATHER S UNCLE IS TWICE AS OLD AS BILL S FATHER . 2
YEARS FROM NOW BILL S FATHER WILL BE 3 TIMES AS OLD AS BILL
. THE SUM OF THEIR AGES IS 92 . FIND BILL S AGE .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02533 ((BILL / PERSON) S AGE))

(EQUAL (PLUS ((BILL / PERSON) S (FATHER / PERSON) S (UNCLE
/ PERSON) S AGE) (PLUS ((BILL / PERSON) S (FATHER / PERSON)
S AGE) ((BILL / PERSON) S AGE))) 92)

(EQUAL (PLUS ((BILL / PERSON) S (FATHER / PERSON) S AGE) 2)
(TIMES 3 (PLUS ((BILL / PERSON) S AGE) 2)))

(EQUAL ((BILL / PERSON) S (FATHER / PERSON) S (UNCLE / PERSON)
S AGE) (TIMES 2 ((BILL / PERSON) S (FATHER / PERSON) S AGE)))

(BILL S AGE IS 8)
```

Figure 1: Some Problems Solved by STUDENT

in English language statements of algebra story problems such as those shown in Figure 1. STUDENT does this by constructing from the English input a corresponding set of algebraic equations, and solving this set of equations for the requested unknowns. If needed, STUDENT has access to a store of "global" information, not specific to any particular problem, and can retrieve relevant facts and equations from this store of information. STUDENT comments on its progress in solving a problem, and can request the help of the questioner if it gets stuck.

There are a number of reasons why I chose the context of algebra story problems in which to develop techniques which would allow a computer problem solving system to accept natural language input. First, we know a good type of data structure in which to store information needed to answer questions in this context, namely, algebraic equations. There exist well known algorithms for deducing information implicit in the equations, that is, values for particular variables which satisfy the set of equations.

In addition, I felt that there was a manageable subset of English in which many types of algebra story problems were expressible. A large number of these story problems are available in first year high school text books, and I have transcribed some of them into STUDENT's input English. Since this question-answering task is one performed by humans, and since the entire process from input to solution of the equations was programmed, we can obtain a measure of comparison between the performance of STUDENT and of a human on the same problems. In fact, this program on an IBM 7094 answers most questions that it can handle as fast or faster than humans trying the same problem. In judging this comparison, one should remember the base speed of the IBM 7094, which can perform over one hundred thousand additions per second.

B. Reasons for Wanting Natural Language Input.

Why should one want to talk to a computer in English? There are many tongues the computer already understands - such as FORTRAN, COMIT, LISP, ALGOL, COBOL, to name just a few. These serve adequately as communication media with the computer for a large class of problems. A more pertinent question is really, when is English input to a computer desirable?

English input is desirable, for example, if it is necessary to use the computer for retrieval of information from a text in English. If a computer could accept English input, much information now recorded only in English would be available for computer use without need for human translation.

A computer which understood English would be more accessible to any speaker of English, whether or not he was trained in any "foreign" computer tongue. For a single shot at the computer with a question not likely to be repeated, it would not be worthwhile to train the user in a specialized language. For fact retrieval, rather than document retrieval, English is a good vehicle for stating queries. For a good description of the differences between fact and document retrieval, see Cooper (12).

Programming languages are process oriented. One cannot describe a problem, only a method for finding a solution to the problem. A natural language is a convenient vehicle for providing a description of the problem itself, leaving the choice of processing to the problem solver accepting the input. In an extreme case, one would like to talk to the computer about a problem, with appropriate questions and interjections by the computer on assumptions it finds necessary, until the computer claims that the problem is now well formed, and an attempt at solution can be made.

Finally, man's ability to use symbols and language is a prime factor in his intelligence, and if we can learn how to make a computer understand a natural language, we will have taken a big step toward creating an "artificially intelligent" computer (32).

C. Criteria for Evaluating Question-Answering Systems.

We have defined understanding in terms of an ability to answer questions in English. A number of question-answering systems have been built, and will be described in the next section. In this section, we shall give a number of criteria for evaluating question-answering systems.

In many systems there is a separation of data input and question input. For all systems under consideration, the input questions are in English. The input data may be either in English or in a prestructured format, e.g. a tree or hierarchy. The English data input may be used as a data base as is, or mapped into a structured information store. Simmons, in his competent survey of English question-answering systems (40), calls those systems using a structured information store "data base question-answerers", as opposed to "text-based question-answerers" which retrieve facts from the original text.

The extent of understanding of a question-answering system can be measured along three different dimensions, syntactic, semantic and deductive. Along the syntactic dimension one can measure the grammatical complexity allowable in input sentences. This may differ for the data input and question input. In the simplest case, one or some small number of fixed format sentences are allowable inputs. Less restricted inputs may allow any sentences which can be parsed by a fixed grammar. The nearer this grammar is to a grammar of all of English, the less restricted is the input. Because text-based question-answerers accept as input any string of words, without further processing, they have no syntactic limitation on input. However, the fact-retrieval program may only be able to abstract information from those portions of a text with less than some maximum syntactic complexity.

In data base question-answering systems, only certain relationships between words, or objects, may be representable in the information store. Other information may be discarded or ignored. This is a limitation in the semantic dimension of understanding.

In order to obtain answers to questions not explicitly given in the input, a question-answering system must have the power to perform some deductions. The structure of the information store may facilitate such deductive ability. The range of deductive ability is measured along the deductive dimension of understanding. The structure of the information store may also aid in selecting only relevant material for use in the deductive question-answering process, thus improving the efficiency of the system.

Another criteria closely related to the extent of understanding, is the facility with which the syntactic, semantic, or deductive abilities of a question-answering system can be extended. In the best case one could improve the system along any dimension by talking to it in English. Alternatively, one might have to add some new programs to the system, or at worst, any change might imply complete reprogramming of the entire system.

An important additional consideration for users of a question-answering system is the amount of knowledge of the internal structure of the system that is necessary to use it. At best one need not be aware of the information storage structure used at all. At worst, a thorough knowledge of the internal structure may be necessary to construct suitable input.

Another measure of the usefulness of a question-answering system is its ability to interact with the user. In the worst case, a question is asked and sometime later an answer or report of failure is given. When the question cannot be answered, no indication is given of the cause of failure, nor does the system allow the person to give any help. This is typical of the operation of a number of Air Force query systems (Jay Keyser, personal communication). In the best case, the system will ask the user for specific help and accept suggestions of appropriate courses of action.

In this section we have given four criteria for evaluating question-answering systems. They may be summarized as follows:

1) Extent of understanding (syntactic, semantic and deductive abilities)
2) Facility for extending abilities (syntactic, semantic, deductive)
3) Need by user for knowledge of internal structure of system
4) Extent of interaction with user

D. English Language Question-Answering Systems.

In this section, I shall give a critical summary of a number of English language question-answering systems, utilizing the criteria outlined in the previous section. This discussion will provide a context for the section of the concluding chapter which summarizes the capabilities of the STUDENT system. For a description of the different syntactic analysis schemes mentioned below, see the survey by Bobrow (4).

1) Phillips. One of the earliest question-answering systems was written in 1960 at MIT by Anthony Phillips (36). It is a data base system which accepts sentences which can be parsed by a very simple context-free phrase structure grammar, of the type defined by Chomsky (8). Additional syntactic restrictions require that each word must be in only one grammatical class, and that a sentence has exactly one parsing.

A parsed sentence is transformed into a list of five elements, the subject, verb, object, time phrase, and place phrase in the sentence. All other information in the sentence is disregarded. Questions are answered by matching the list from the transformed question against the list for each input sentence. When a match is found, the corresponding sentence is given as an answer.

Phillips' system has no deductive ability and adding new abilities would require reprogramming the system. A questioner must be aware that the system utilizes a matching process which does not recognize synonyms, and therefore the sentence "The teacher eats lunch at noon." will not be recognized as an answer to the question "What does the teacher do at twelve o'clock?" When Phillips' system cannot find an answer, it reports only "(THE ORACLE DOES NOT KNOW)". It provides for no further interaction with the user.

2) Green. Baseball is a question-answering system designed and programmed at Lincoln Laboratories by Green, Wolf, Chomsky and Laughery (19). It is a data base system, in which the data is placed in memory in a prestructured tree format. The data consists of the dates, location, opposing teams and scores of some American League baseball games. Only questions to the system can be given in English, not the data.

Questions must be simple sentences, with no relative clauses, logical or coordinate connectives. With these restrictions, the program will accept any question couched in words contained in a vocabulary list quite adequate for asking questions about baseball statistics. In addition, the parsing routine, based on techniques developed by Harris (21), must find a parsing for the question.

The questions must pertain to statistics about baseball games found in the information store. One cannot ask questions about extrema, such as "highest" score or "fewest" number of games won. The parsed question is transformed into a standard specification (or spec) list, and the question-answering routine utilizes this canonical form for the meaning of the question. For example, the question "Who beat the Yankees on July 4?" would be transformed into the "spec list":

Team (losing) = New York
Team (winning) = ?
Date = July 4

Because Baseball does not utilize English for data input, we cannot talk about deductions made from information implicit in several sentences. However, Baseball can perform operations such as counting (the number of games played by Boston, for example) and thus in the sense that it is utilizing several separate data units in its store, it is performing deductions.

Baseball's abilities can only be extended by extensive reprogramming, though the techniques utilized have some general applicability. Because the parsing program has a very complete grammar, and the vocabulary list is quite comprehensive for the problem domain, the user needs no knowledge of the internal structure of the Baseball program. No provision for interaction with the user was made.

3) Simmons. The SYNTHEX system is a text-based question-answering system designed and programmed at SDC by Simmons, Klein and McConologue (41). The entire contents of a children's encyclopedia has been transcribed to magnetic tape for use as the information store. An index has been prepared listing the location of all the content words in the text, i.e. including words like "worm," "eat," and "birds," while excluding function words like "and," "the," and "of." All the content words of a question are extracted, and information rich sections of the text are retrieved, i.e. sections that are locally dense in content words contained in the question. For example, if the question were "What do worms eat?", with content words "worms" and "eat", the two sentences "Birds eat worms on the grass." and "Most worms usually eat grass." might be retrieved. At this time, the program performs a syntactic analysis of the question and of the sentences that may contain the answer. A comparison of the dependency trees of the question and various sentences may eliminate some irrelevant sentences. In the example, "Birds eat worms on the grass" is eliminated because "worms" is the object of the verb "eats" instead of the subject as in the question. In the general case, the remaining sentences are given in some ranked order as possibly answering the question.

SYNTHEX is limited syntactically by its grammar to the extent that the syntactic analysis eliminates irrelevant statements. It makes no use of the meaning of any statements or words, and cannot deduce answers from information implicit in two or more sentences. Because the grammar is independent of the program, the syntactic ability of SYNTHEX can be extended relatively easily. However, before it can become a good question-answering system, some semantic abilities will have to be added.

SYNTHEX does not explicitly provide for interaction with the user, but because it is implemented in the SDC time-sharing system (9), a user may modify a previous question if the sentences retrieved were not suitable. The mechanism for selection of sentences must be kept in mind to get best results.

4) Lindsay. While at the Carnegie Institute of Technology, Robert Lindsay (28) programmed the SAD SAM question-answering system. The input to the system is a set of sentences in Basic English, a subset of English devised by C.K. Ogden (35), which has a vocabulary of about 1500 words and a simple subset of the full English grammar. The SAD part (Syntactic Appraiser and Diagrammer) of SAD SAM parses the sentence using a predictive analysis scheme. The Semantic Analyzing Machine (SAM) extracts from these parsed sentences information about the family relationships of people mentioned; it stores this information on a computer representation of the family tree, and ignores all other information in the sentence. For example, from the parsing of "Tom, Mary's brother, went to the store." Lindsay's program would extract the sibling relationship of Tom and Mary, place them on the family tree as descendants of the same mother and father, and ignore the information about where Tom went.

The information storage structure utilized by SAD SAM, namely, the family tree, facilitates deductions from information implicit in many sentences. Because a family relationship is defined in terms of the relative position (no pun intended) of two people in their family tree, computation of the relationship is independent of the number of sentences required to place in the tree the path between the individuals.

Extending the abilities of the SAD SAM system would require reprogramming. No provision is made for interaction with the user. No internal knowledge of the program structure is necessary if the user restricts his queries to questions of family relationships, and his language to Basic English.

5) Raphael. The SIR question-answering system (mnemonic for Semantic Information Retrieval) was designed by Bertram Raphael (38) at MIT. The SIR system accepts simple sentences in any of about 20 fixed formats useful for expressing certain relationships between objects. The semantic relationships extracted from these sentences are those of set membership, set inclusion, subpart, left-to-right position and ownership.

The information about the relationships between various objects is stored in a semantic network, where the nodes of the network are objects and the relationships are indicated by directed labeled links between nodes. For example, if the three sentences "John is a boy," "A boy is a person," and "Two hands are part of any person" were an input to SIR, four nodes labeled John, boy, person and hand would be created. Included in the network would be a link indicating set membership between John and boy, another with a label indicating set inclusion between boy and person, and a link indicating hand is a subpart of person, with the number of parts equal to 2.

Separate question-answering routines are used for questions involving different relationships. Each routine takes cognizance of the interaction of various relationships, and can deduce answers from the linked structure of the network, independent of the number of sentences which were necessary to set up these links. For example, by tracing the links from "John" to "hand," SIR would answer "YES" to the question "Is a hand part of John?"

The SIR system can interact with the user. For example, if told that "A finger is part of a hand" and asked "How many fingers does John have?" it would reply "How many fingers per hand?" Then if it is told "Every hand has five fingers," it would answer the question with "The answer is 10".

Any extensions of the SIR system necessitate additional programming effort, though it is considerably easier to add new syntactic forms than new semantic relationships. Within the input limits of the 20 fixed format statements, the user need not know anything of the internal structure of the information storage structure.

E. Other Related Work.

In addition to those question-answering systems described above, a number of programs have been written to translate English statements into a logical notation to check the consistency of a set of statements, and the validity of logical arguments. In the sense that, given a corpus transformed to some logical notation, and another statement, a logic-based system can answer the question "Is this statement (or its negation) implied by the corpus?", such logic-based systems are question-answering systems.

Cooper (12) and Darlington (14) both have programs which translate a subset of English into the propositional calculus. Darlington is also working on programs which can translate English into the first order and second order predicate calculi. A difficult problem being considered by Darlington, in trying to handle implications of English statements in terms of their logical translation, is the determination of the proper level of analysis for a particular problem - that is, whether to translate the input into second order predicate calculus where proofs are very difficult, or to try to use first order predicate or propositional calculus to prove the theorem, and perhaps find it logically insufficient.

At the National Bureau of Standards, Kirsch (22), Cohen (10) and Sillars (39) have designed a system in which pictures and English language statements are converted to expressions in the first order predicate calculus. One can then check to see if an English language statement is consistent with a given picture.

McCarthy's Advice-Taker (30), though not designed to accept English input, would make an excellent base for a question-answering system. Fischer Black (2) has programmed a system which can do all of McCarthy's Advice-Taker problems, and can be adapted to accept a very limited subset of English. The deductive system in Black's program is equivalent to the propositional calculus.

A number of people have done work bearing directly on the problem of solving algebra word problems stated in English. Sylvia Garfinkle (18) wrote a paper in which she described the heuristics she would use in programming a computer to solve algebra word problems, but never wrote the program. Most of the heuristics were too vague to really be used; e.g. just stating that one should identify two variables' names which are only slightly different, but giving no good criteria for a slight difference. The treatment of "this" was taken from Garfinkle's paper. So were a number of simplified statements of algebra story problems she transcribed and transformed from problems in a first year algebra text book.

Michael Coleman (11), at MIT, wrote a term paper describing a program of his which sets up the equations for some types of algebra story problems (also handled by STUDENT). Some of the special heuristics I use for "age problems" were inspired by techniques he invented.

In his thesis, David Kuck (24) describes his ideas on how to construct this type of program, but again did not implement these ideas. He suggests methods for transformation of English input to equations which would require much more information about words than is used in the STUDENT program, and therefore were not applicable in this work. The STUDENT program considers words as symbols, and makes do with as little knowledge about the meaning of words as is compatible with the goal of finding a solution to the particular problem.

CHAPTER II: SEMANTIC GENERATION AND ANALYSIS OF DISCOURSE

The purpose of this chapter is to put the techniques of analysis embedded in the STUDENT program into a wider context, and indicate how they would fit into a more general language processing system. We will describe in this chapter a theory of semantic generation and analysis of discourse. STUDENT can then be considered a first approximation to a computer implementation of the analytic portion of the theory, with certain restrictions on the interpretation of a discourse to be analyzed. It will be evident from the theory why analysis is so greatly simplified by the imposed restrictions.

A. Language as Communication.

Language is an encoding used for communication between a speaker and a listener (or writer and reader). To transmit an "idea", the speaker must first encode it in a message, as a string in the transmission language. In order to understand this message, a listener must decode it, and extract its meaning. The coding of a particular message, M, is a function of both its global context and local context. The global context of a message is the background knowledge of the speaker and the listener, including some knowledge of possible universes of discourse, and codings for some simple ideas.

The local context of a message, M, is the set of messages temporally adjacent to M. M may refer back to earlier messages. M may even be just a modification of a previous message, and only understandable in this context. For example, consider the second sentence of the following discourse: "How many chaplains are in the U.S. Army? How many are in the navy?"

In order for communication to take place, the information map of both the listener and the speaker must be approximately the same, at least for the universe of discourse; also the decoding process of the listener must be an approximate inverse of the encoding process of the speaker. Education in language is, in large part, an attempt to force the language processors of different people into a uniform mold to facilitate successful communication. We are not proposing that identity in detail is achieved, but as Quine so nicely put it (37):

> "Different persons growing up in the same language are like different bushes trimmed and trained to take the shape of identical elephants. The anatomical details of twigs and branches will fulfill the elephantine form differently from bush to bush, but the overall outward results are alike."

As a speaker transmits successive messages concerning some portion of his information map, the listener who understands the messages constructs a model of a "situation". The relation between the listener's model and the speaker's information map is that from each can be extracted the transmitted information relevant to the universe of discourse, including information deducible from the entire set of messages. The internal structure of the listener's model need bear no resemblance to that of the speaker, and may in general contain far less detail.

B. Theories of Language.

According to Morris' theory of signs (33), the encoding and decoding of language can be stratified into three levels. The first level is the syntactic which deals with the relationships of signs to other signs. A syntactic analysis, treating words as members of classes of words, can yield structurings of messages which indicate common processing features. The second level, semantic analysis, is concerned with the relationships of signs to the things they denote.

A third level, pragmatic analysis, is concerned with the relationships between signs and their interpretations in terms of actions required. Our theory will deal with all three levels of analysis, with a primary emphasis on the relation of the semantic aspect of language to the generation of discourse.

Many theories of syntax have been developed to describe the structure of English, and many of these have served as bases for computer programs which perform syntactic analysis. For a complete survey of such systems see the paper by Bobrow (4). Almost all of these theories ignore the concepts of meaning and semantics. Because they ignore such an important aspect of language, programs based on such theories often yield many possible structurings for a single sentence which is unambiguous to a person. With some use of meaning, many of the meaningless ambiguous interpretations could be eliminated. For a good discussion of why ambiguities arise in syntactic analysis see Kuno and Oettinger (25).

Based on some ideas described by Yngve (46), a number of programs have been written which generate syntactically correct English sentences. In most cases, the sentences generated are predominately meaningless nonsense. The coherent discourse generator of Klein (23) is the one exception I know. Klein utilizes an input text from which he extracts certain structural dependencies of the words in the input. He then generates sentences and before they are released for output, a postprocessor checks to see if the words in the generated sentence satisfy structural dependencies consistent with those found in the input text. However, even in Klein's program no attempt is made to use the denotive meaning of any word, except in so far as this meaning is reflected in its cooccurrences with other words in the input text.

Some theories which do consider the problem of semantics are being developed now. Pendegraft (27) states that the programs being developed at the Linguistic Research Center of the University of Texas are an explication of Morris' theory of signs. Though not yet implemented, the semantic analysis program will make use of a preliminary phrase structure syntactic analysis. A number of syntactic structures, with appropriate vocabulary items, will map onto single semantic constants, essentially indicating that these structures all have the same meaning. This gives a type of canonical form for structures in terms of their meanings, but does not utilize any explicit model of the world. No provision is made in the theory for deduction of information implicit in a set of sentences.

Lamb (26) also has proposed a stratificational theory of grammar, not yet implemented on a computer, in which successive levels of analysis are performed, with a final mapping of the input into structures in a "sememic" stratum of the language. In this sememic stratum are bundles of "sememes" or meanings, and indications of the relationships between different bundles. Different sentences which mean the same thing should map into the same structure in this sememic stratum. Sememic structures are thus canonical representations of meaning.

C. Definition of Coherent Discourse.

The theory of language generation and analysis which we shall describe below is designed to handle what we call coherent discourse. A discourse is a sequence of sentences such that the meaning of the discourse cannot be determined by interpreting each sentence independently, disregarding the other sentences in the discourse. The interpretation of each sentence may be dependent on the local context, in the sense defined previously. A discourse is coherent if it has a complete and consistent interpretation. Completeness implies that there is no substring within the discourse that does not have some interpretation in the model of the situation being built by the listener.

A listener's ability to build a model of a situation from a discourse is dependent on information available to him from his general store of knowledge. Therefore it is quite possible for a discourse to seem coherent to one listener and not another. A writer, reading his own writing, may feel that he has generated a coherent sequence of sentences, but in fact, it is incoherent to all other readers. This is, unfortunately, not a rare occurrence in the scientific literature. Conversely, a listener who is a psychiatrist, for example, may find coherence in a sequence of remarks which a patient thinks are entirely unrelated.

The STUDENT system utilizes an expandable store of general knowledge to build a model of a situation described in a member of a limited class of discourses. The form of this model of a situation built by STUDENT will be discussed in detail in a later section of this chapter. As far as I know, STUDENT is the only computer implementation of a theory of discourse analysis now extant that maps a discourse into some representation of its meaning. When the theories of Lamb and Pendegraft are implemented, they should also be able to analyze this class of discourse (and others). Harris also talks about "discourse analysis," (20) but in his use of this term he specifically excludes the use of meaning, stating:

> "The method [of discourse analysis] is formal, depending only on the occurrence of morphemes as distinguishable elements, and not upon the analyst's knowledge of the particular meaning of each morpheme."

D. The Use of Kernel Sentences in Our Theory.

A basic postulate of our theory of language analysis is that a listener understands a discourse by transforming it into an equivalent (in meaning) sequence of simpler kernel sentences. A kernel sentence is one which the listener can understand directly; that is, one for which he knows a transformation into his information store. Conversely, a speaker generates a set of kernel sentences from his information map, and utilizes a sequence of transformations on this set to yield his spoken discourse. This set of kernel sentences is not invariant from person to person, and even varies for a single individual as he learns.

The use of kernel sentences in this way is controversial. However, the theory is proposed as a good framework for understanding and implementing language processing on a computer, not necessarily as a model for human behaviour. The usefulness of this theory as a psychological model is an empirical question. Skinner (42) has given some psychological justification for assuming the existence of a set of base sentences, and Chomsky (7) has discussed the linguistic merits of the use of the concept of kernel sentences. Despite this common concept of kernel sentences, in practice, our use of kernel sentences is different than that of Skinner or Chomsky. Our use of kernel sentences as a basis of a language is analogous to the use of generators in defining a group.

Although we are not proposing our theory as a basis for a psychological model, it has been useful, to avoid circumlocutions, to describe the theory in terms of the properties and actions of a hypothetical speaker and listener. All statements about speakers and listeners should be interpreted as referring to computer programs which respectively, generate and analyze coherent discourse.

E. Generation of Coherent Discourse.

1) The Speaker's Model of the World. We assume that a speaker has some model of the world in his information store. We shall not be concerned here with how this model was built, or its exact form. Different forms for the model will be useful for different language tasks, but they must all have the properties described below.

The basic components of the model are a set of objects, $\{O_i\}$, a set of functions $\{F_i^n\}$, a set of relations $\{R_i^n\}$, a set of propositions $\{P_i\}$, and a set of semantic deductive rules. A function F_i^n is a mapping from ordered sets of n objects, called the arguments of F_i^n, into the set of objects. The mapping may be multivalued and is defined only if the arguments satisfy a set of conditions associated with F_i^n. A condition is essentially membership in a class of objects, but is defined more precisely below. A relation R_i^n is a special type of object in the model, and consists of a label (a unique identifier), and an ordered set of n conditions, called the argument conditions for the relation. Functions of relations are again relations.

An elementary proposition consists of a label associated with some relation, R_i^n, and an ordered set of n objects satisfying the argument conditions for this relation. One may think of these propositions as the beliefs of a speaker about what relationships between objects he has noticed are true in the world. Complex propositions are logical combinations (in the usual sense) of elementary propositions.

The semantic deductive rules give procedures for adding new propositons to the model based on the propositions now in the model. In addition to the ordinary rules of logic, these rules include axioms about the relationships of the relations in the model. The semantic deductive rules also include links to the senses of the speaker. For example, one such deductive rule for adding a propositon to the model might be (loosely speaking) "Look in the real world and see if it is true." These rules essentially determine how the model is to be expanded, and are the most complex part of a complete system. However, from our present point of view, we need only consider these rules as a black box which can extend the set of propositions in the model.

A <u>closed question</u> is a relational label for some R_i^n and an ordered set of <u>n</u> objects. The answer to this question is affirmative if the proposition, consisting of this label and the <u>n</u> objects, is in the model (or can be added to it). If the negation of this proposition is in the model (or can be added), the answer is negative. Otherwise the answer is undefined.

An <u>open question</u> consists of a relational label for an n-argument relation, R_i^n , and a set of objects corresponding to n-k of these arguments, where $n \geq k \geq 1$. An answer to an open question is an ordered set of k objects, such that if these objects are associated with the k unspecified arguments of R_i^n , the resulting proposition is in the model or can be added to it. An open question may have no answers, or may have one or more answers. A <u>condition</u> is an open question with k=1, and an object satisfies a condition if it is an answer to the question.

2) <u>Generation of Kernel Sentences.</u> We have described the logical properties of the speaker's model of the world. We shall now consider how strings in a language, words, phrases, and sentences, are associated with the model. Corresponding to the set of objects O_i there is a set N_{ij} of strings (in English in our case), called the <u>names</u> of the objects. There is a many-one mapping from

$\{N_{ij}\}$ onto $\{O_i\}$. It is many-one because one object may have more than one name, e.g. frankfurter and hot dog both map back into the same object in the model.

Recall that functions map n-tuples of objects into objects. Thus a function name and an n-tuple can specify an object. We can derive a name for this object from the function name and the names of its n arguments. Associated with each function is at least one linguistic form, a string of words with blanks in which names of arguments of the function must be inserted. Examples of linguistic forms associated with a model are "number of ________", "father of ________", and "the child of ______ and ________". There is a many-one mapping from the set of linguistic forms $\{L^n_{ij}\}$ onto the set of functions. Two examples of multiple linguistic forms for the same function are: "father of ______" and "______'s father"; and "______ plus ______" and "the sum of ______ and ______". Thus, if objects x and y have names "the first number" and "the second number" and associated with the function " * " is the linguistic form "the product of ______ and ______", then the name of the object produced by applying the function " * " to x and y is "the product of the first number and the second number". A parsing of a name thus must decompose it into the part which is the linguistic form, and the parts which are names of arguments of the corresponding function. We shall call objects defined in terms of a function and an n-tuple of objects a <u>functionally defined object</u>, and those which are not functionally defined we shall call <u>simple objects</u>. Simple objects have <u>simple names</u> and functionally defined objects have <u>composite names</u>.

In addition to linguistic forms associated with functions, there are linguistic forms associated with relations. For an <u>n</u> argument relation there are <u>n</u> blanks in the linguistic form. Examples of relational linguistic forms are: "_____ equals _____", "_____ gave _______ to _______" and " _______ speaks". It is this set of linguistic forms, corresponding to the relations in the model, that serve as frames for the kernel sentences.

In a manner similar to the way composite names are built, a kernel sentence corresponding to an elementary proposition is constructed by inserting names corresponding to each argument in the appropriate blank. Names may be ~~simple or composite~~. An example of a kernel sentence for a proposition built from such a relational linguistic form is "John's father gave .3 times the salary of Bill to Jack." which contains the simple names "John", ".3", "Bill", and "Jack". It contains the functional linguistic forms "_____'s father", "_____times _____" and "salary of _____" and the relational linguistic form " _____ gave _______ to ______".

A kernel sentence corresponding to a complex proposition is constructed recursively from the kernel sentences corresponding to its elementary propositional constituents by placing them in the corresponding places in the linguistic forms " _____ and _____", "______ or ______", "not ______" etc.

The kernel sentence corresponding to a closed question is constructed from the kernel of the corresponding proposition by placing it in the linguistic form "Is it true that _____?" For an open question, dummy objects are placed in the open argument positions to complete a propositional form. These dummy arguments have names "who", "what", "where", etc., and which dummy objects are used depends on the condition on that argument position. A question mark is placed at the end of the kernel sentence constructed in the usual way from the relational linguistic form and the names of the arguments.

In generating a coherent discourse, a speaker chooses a number of propositions in his model and/or some open or closed questions. He then uses linguistic information associated with the model to construct the set of kernel sentences corresponding to this set of chosen propositions. In the next section we will discuss how he generates his discourse from this set of kernels.

3) Transformations on Kernel Sentences. The set of kernel sentences is the base of the coherent discourse. The meaning of a kernel sentence is the proposition into which it maps, and similarly, the meaning of any name is the object which is its image under the mapping. To this set of kernels we apply a sequence of meaning preserving transformations to get the final discourse. We use the word "transformation" in its broad general sense, not in the narrow technical sense defined by Chomsky (7).

There are two distinct types of transformations, structural and definitional. A structural or syntactic transformation is only dependent on the structure of the kernel string(s) on which it operates. For example, one syntactic transformation takes a kernel in the active voice to one in the passive voice. Another combines two sentences into a single complex coordinate sentence.

One large class of syntactic transformations is used to substitute pronominal phrases for names. Pronominal phrases may be ordinary pronouns such as "he", "she", or "it". They may be referential phrases such as "the latter", "the former" or "this quantity". They may also be truncations of a full name such as "the distance" for "the distance between New York and Los Angeles". In cases where such pronominal reference is made, the coherence of the final discourse is dependent on the order in which the resultant strings appear.

The second type of transformation is definitional. It involves substitutions of linguistic strings and forms for ones appearing in the kernel sentences. For example, for any appearance of "2 times" we may substitute "twice", and for ".5 times" substitute "one half of". In addition to this string substitution, some transformations perform form substitution and rearrangement. For example, for a kernel sentence of the form "x is y more than z", where x, y, and z are any names, one definitional transformation can substitute "x exceeds z by y."

Some transformations are optional, and some may be mandatory if certain forms are present in the kernel set. Certain transformations are used by a speaker for stylistic purposes, for example, to emphasize certain objects; other syntactic transformations such as those which perform pronominal substitutions are used because they decrease the depth of a construction, in the sense defined by Yngve (44).

Let us review the steps in the generation of a coherent discourse. The speaker chooses a set of propositions, the "ideas" he wishes to transmit. He then encodes them as language strings called kernel sentences in the manner described above. He then chooses a sequence of structural and definitional transformations which are defined on this set of kernels or on the ordered set of sentences which result from applications of the first transformations. The resulting sequence of sentences will be a coherent discourse to a listener if he knows all the definitional transformations applied. In addition, for every pair of distinct names which the speaker maps back into the same object, the listener must also map into a single object.

In order to clarify this theory, we show, in Appendix E, a sample semantic generative grammar which will generate coherent discourse understandable by the STUDENT analysis program. The objects are numbers and the functions are the arithmetic operations of sum, difference, product and quotient. The only relation in the model is numerical equality. The transformations are described informally; further linguistic investigation is necessary before a formal notation for transformations can be decided upon. Parallel to the grammar is a sample problem generated by utilizing this grammar. This problem is solvable by the STUDENT system.

F. Analysis of Coherent Discourse.

Generation of coherent discourse consists of two distinguishable steps. From propositions in the speaker's model of the world, he generates an ordered set of kernel sentences. He then applies a sequence of transformations to this kernel set. The resulting discourse is a coded message which is to be analyzed and decoded by a listener. The listener's problem can be loosely characterized as an attempt to answer the question, "What would I have meant if I said that?"

To analyze a discourse the listener must find the set of kernel sentences from which it was generated; one way to do this is to find a set of inverse transformations which when applied to the input discourse yield a sequence of kernel sentences. The listener must then transform these kernel sentences to an appropriate representation in his information store. The appropriateness of a representation is a function of what later use the listener expects to make of the information contained in the discourse. The listener may simultaneously transform a given kernel sentence into a number of different representations in his information store. On a level of pragmatic analysis, statements require only storage of information. Questions and imperatives require appropriate responses from the listener. The difficulties in analysis dichotomize into those associated with finding the kernel sentences which are the base of the discourse, and those associated with transforming the kernel sentences into representations in the information store.

Mathews (29) has suggested that analysis can be performed by synthesis. A sequence of kernel sentences, and a sequence of transformations are chosen, and the transformations are applied to the kernel sentences. The resulting discourse is matched against the input. If they are the same, these kernel sentences and transformations give the required analysis of the input. If not, a change is made so that the resulting discourse becomes more like the input.

If the kernel sentences and transformations were chosen randomly, this method would obviously be too inefficient to work in any practical sense. However, by utilizing clues within the input discourse, the choice of kernels and transformations can be greatly restricted. This technique of sentence analysis is being implemented in a program being written at MITRE by Walker and Bartlett (43). This technique has the advantage that exactly the same grammar can be utilized for both analysis and generation of discourse.

A more direct analytical approach would utilize a set of inverse analytic transformations. If T_i is a transformation that may be used in generating a discourse, and $T_i(S) = \bar{S}$, where S and $\bar{S}$ are sets of sentences, then the analytic transformation T_i^{-1} is the inverse of T_i if and only if $T_i^{-1}(\bar{S}) = S$. The choice of which inverse transformations to apply and the order of their application may again be restricted by utilizing heuristics concerned with features of the input.

Once the base set of kernel sentences for a given discourse is determined, there remains the problem of entering representations of these sentences in the listener's information store. The major problem in accomplishing this step involves the separation of those words which are part of linguistic forms for relations, and those which are part of a name. This is difficult because the same word (lexicographic symbol) may have multiple uses in a language. Having separated the relational form from the names which represent the arguments of this relation, one can then analyze the name in terms of components which are functional linguistic forms and others which are simple names. From this parsing in terms of relational linguistic forms, functional linguistic forms and simple names, the discourse can be transformed into a canonical representation in the information store of the listener.

G. Limited Deductive Models.

A complete understanding of a discourse by a listener would imply that the representation of the discourse in his information store is essentially isomorphic to the speaker's model of the world, at least for the universe of discourse. The listener's representation must preserve all information implicit in the discourse.

If the listener is only interested in certain aspects of the discourse, he need only preserve information relevant to his interest, and discard the rest. Within his area of interest the listener's model is isomorphic to the speaker's model in the sense that all relevant deductions which can be made by the speaker on the basis of the discourse can also be made by the listener. Outside this area of interest, the listener will be unable to answer any questions. We call such restricted information stores limited deductive models.

The question-answering programs of Lindsay and Raphael, and the STUDENT system, all utilize limited deductive models. For the area of interest in each of these programs there was a "natural" representation for the information in the allowable input. These representations were natural in that they facilitated the deduction of implicit information. For example, Lindsay's family tree representation made it easy to compute the relationship of any two individuals in the tree, independent of the number of sentences necessary to build the tree.

Because the number of relations and functions expressible in the models in all three systems is very limited, there is a corresponding limitation on the number of linguistic forms that may appear in the input. This greatly simplifies the parsing problem discussed earlier, by restricting alternatives for words in the input text.

H. The STUDENT Deductive Model.

The STUDENT system is an implementation of the analytic portion of our theory. STUDENT performs certain inverse transformations to obtain a set of kernel sentences and then transforms these kernel sentences to expressions in a limited deductive model. Utilizing the power of this deductive model, within its limited domain of understanding, it is able to answer questions based on information implicit in the input information.

The analytic and transformational techniques utilized in STUDENT are described in detail in Chapter IV. We shall describe here the canonical representation of objects, relations and functions within the model. STUDENT is restricted to answering questions framed in the context of algebra story problems. Algebraic equations are a natural representation for information in the input.

The objects in the model are numbers, or numbers with an associated dimension. The only relation in the model is equality, and the only functions represented directly in the model are the arithmetic operations of addition, negation, multiplication, division and exponentiation. Other functions are defined in terms of these basic functions, by compostion, and/or substitution of constants for arguments of these functions. For example, the operation of squaring is defined as exponentiation with "2" as the second argument of the exponential function; subtraction is a composition of addition and negation.

Within the computer, a parenthesized prefix notation is used for a standard representation of the equations implicit in the English input. The arithmetic operation to be expressed is made the first element of a list, and the arguments of the function are succeeding list elements. The exact notation is given in Figure 2 below.

Operation	Infix Notation	Prefix Notation
Equality	A = B	(EQUAL A B)
Addition	A + B	(PLUS A B)
	A + B + C	(PLUS A B C)
Negation	- A	(MINUS A)
Subtraction	A - B	(PLUS A (MINUS B))
Multiplication	A * B	(TIMES A B)
	A * B * C	(TIMES A B C)
Division	A / B	(QUOTIENT A B)
Exponentiation	A^B	(EXPT A B)

Figure 2: Notation Within the STUDENT Deductive Model

In the figure, A, B, and C are any representations of objects in the model, either composite or simple names. The usual infix notation for these functional expressions is given for comparison. Because this is a fully parenthesized notation, no ambiguity of operational order arises, as it does, for example, for the unparenthesized infix notation expression A*B+C or its corresponding natural language expression "A times B plus C". Note also that in this prefix notation plus and times are not strictly binary operators. Indeed, in the model they may have any finite number of arguments, e.g. (TIMES A B C D) is a legitimate expression in the STUDENT model.

Representations of objects in the STUDENT deductive model are taken from the input. Any string of words not containing a linguistic form associated with the arithmetic functions expressible in the model are considered simple names for objects. Thus, "the age of the child of John and Jane" is considered a simple name because it contains no functional linguistic forms associated with functions represented in STUDENT's limited deductive model. In a more general model it would be considered a composite name, and the functional forms "age of ______" and "child of _____ and _____" would be mapped into their corresponding functions in the model.

Because such complex strings are considered simple names in the model, and objects are distinguished only by their names, it is important to determine when two distinct names actually refer to the same object. In fact, answers to questions in the STUDENT system are statements of the identity of the object referenced by two names. However, one of the names (the desired one) must satisfy certain lexical conditions. Most often this condition is just that the name be a numeral. For a more general model this restriction could be stated as requiring a simple name corresponding to some functionally defined name — because, for example, "number of ____" would be a functional linguistic form in the general model, and the only simple name for such an object would be the numeral corresponding to this number. An answer consists of a statement of identity e.g. "The number of customers Tom gets is 162."

The other lexical restriction on answers sometimes used in the STUDENT system is insistence that a certain unit (corresponding to a dimension associated with a number) appear in the desired answer. For example, <u>spans</u> is the unit specified by the question "How many spans equals 1 fathom?", and the answer given by STUDENT is "1 fathom is 8 spans".

The deductive model described here is useful for answering questions because we know how to extract implicit information from expressions in this model; that is, we know how to solve sets of algebraic equations to find numerical values which satisfy these equations. The solution process used in STUDENT is described in detail in Chapter VI. The transformation process, based on the theory described earlier, which STUDENT uses to go from an English input to this deductive model, is described in Chapter IV.

CHAPTER III: PROGRAMMING FORMALISMS AND LANGUAGE MANIPULATION

Almost any programming language is universal in the sense that with enough time, space, and work at the implementation, any computable function may be programmed. However, the task of programming can be made much easier by the proper choice of a higher level problem oriented programming language. The data to be manipulated by the STUDENT system is symbolic, and of indefinite length and complexity. For this reason, a list-processing language was the most appropriate type of programming for this task. There are a number of such languages available, each having its own set of advantages and disadvantages. For a description of the general properties of list-processing languages, with a detailed comparison of four of the better known list-processing languages, see Bobrow and Raphael (5). Mostly because I knew it so well, I chose LISP (31) as the basic language for the STUDENT system.

The LISP formalism is very convenient for programming recursive tasks such as the solving of a set of simultaneous equations. However, LISP does not provide any natural mechanisms for representing manipulation of strings of English words, another very important subtask in the STUDENT system. For this type of manipulation one would like to perform a sequence of steps involving operations such as recognizing a sentence format which fits a particular pattern, finding certain elements in a sentence by their context, rearranging a string of words, deleting, inserting, and duplicating parts of strings, and others.

The LISP formalism cannot easily express such string manipulations, though each could be individually programmed. However, a formalism for just this sort of manipulation is the basis of the COMIT (45) programming system. Rules in this formalism can easily express very complex string manipulations, and are easy to read and write. However, COMIT and LISP cannot be used simultaneously, and the problem context necessitates going back and forth between LISP-oriented tasks and COMIT-oriented tasks. Therefore, I adapted the COMIT rule notation for use in LISP, and constructed a LISP program called METEOR which would interpret string transformation rules in this notation.

In constructing the METEOR interpreter, I effectively extended the eloquence of the LISP programming language; that is, operations which could be done previously, but were awkward to invoke could now be expressed easily. An extended language embodying the best features of COMIT and LISP could have been built from scratch, but it is much more economical to achieve such extensions by embedding. The advantages and disadvantages of language extension by embedding are discussed in detail by Bobrow and Weizenbaum (6).

A. Specifying a Desired String Format.

METEOR has been described in detail elsewhere (3), but we include here a brief summary of its features. We do this because use of the notation makes later explication of the transformation process easier. In addition, if any ambiguity becomes apparent in the explanation of the operation of STUDENT, it may be resolved by consulting the listing of the STUDENT program in Appendix B. In this latter case, it may be necessary to consult the more complete specification of METEOR referenced above.

A METEOR program consists of a sequence of rules each specifying a string transformation and giving some control information. Let us first consider how a string transformation is specified. We shall call the string to be transformed the <u>workspace</u>. The workspace will be transformed by a rule only if it matches a pattern or format given in the "left half" of the rule. This left half is a list of elementary patterns which specifies a sequence of items that must be matched in the workspace. For example, if the left half were "(THE BOY)" then a match would be found only if the workspace contained a "THE" immediately followed by "BOY" . In addition to known constituents, one can match unknown constituents. The element $1 in a left half will match any one workspace constituent. The left half "(A $1 B $2 C)" will match a contiguous substring of the workspace which consists of an A followed by exactly one constituent (specified by the marker "$1") followed by a B followed by exactly 2 constituents (matching the "$2") followed by an occurrence of a C. Thus $1 will match an element of the workspace with a specified context. If a left half would match more than one substring in the workspace, the left-most such substring is the one found by the matching process.

We have discussed elementary patterns which match a fixed number of unknown constituents (e.g., "$3" matches 3 unknown constituents). **METEOR** also has an elementary pattern element "$" which matches an arbitrary number of unknown constituents. For example, the left half (THE $ BOY) will match a substring of the workspace which starts with an occurrence of "THE" followed by any number of constituents (including zero) followed by an occurrence of "BOY" . It would, for example, match a substring of the workspace "(GIVE THE GOOD BOY)" or of the workspace "(THE BOY HERE)" . If the left half ($ GLITCH $3) matches a substring of the workspace, then the elementary pattern "$" matches the substring from the beginning of the workspace up to but not including the first occurrence of "GLITCH"; the pattern "GLITCH" matches this occurrence of "GLITCH" in the workspace; and the elementary pattern "$3" matches the 3 elements or constituents of the workspace immediately following GLITCH.

Elements in the workspace may be tagged or subscripted to indicate special properties of this element; for example, one might have (HAVE/VERB) or (BOY/NOUN) as elements of the workspace. Such elements can be matched by name (using HAVE or BOY as pattern elements), or identified just by their subscripts (or by both). The elementary pattern ($1/VERB) will match any single constituent which is a verb; that is, one which has the subscript "VERB", even if this constituent has other subscripts. Thus the left half (ALFRED ($1/VERB) BOOKS) will match the substring (ALFRED (READS/VERB) BOOKS) in the workspace (NOW ALFRED (READS/VERB) BOOKS IN THE LIBRARY).

Other elementary pattern elements are provided, and new pattern elements can be defined and easily used within the METEOR system.

B. <u>Specifying a Transformed Workspace.</u>

We have discussed how a desired format can be specified through a prototype pattern, called a left half. If we try to match the workspace to a left half, but it is not in the format specified, we say the match has <u>failed</u>. If a substring of the workspace is in the specified format, the match is <u>successful</u>. When there is a successful match, we may wish to transform or manipulate the substring matched, or place in a temporary storage location, called a shelf, copies of segments of the matching substring. We shall now discuss the notation used for specifying such transformations, and storage of material.

A left half is a sequence of elementary patterns, and we associate with each elementary pattern a number indicating its position in this left-half sequence. For example, in the left half ($2 D $ E), the first elementary pattern, $2, would be associated with the number 1, the second, D, with 2, $ with 3, and E with 4. If a match is successful, each elementary pattern element in the left half matches a part of the substring of the workspace matched by this left half. The part matched by an elementary pattern can then be referenced by the number associated with this elementary pattern. For the left half given above, and the workspace (A B C D B A E G), the left-half match succeeds, and the substring (B C) may then be referenced with the number 1, the substring (D) by 2, (B A) by 3, and (E) by 4.

The transformed workspace is specified by the "right half" of a METEOR rule. This right half may be just the numeral 0, in which case the matched portion of the workspace is deleted. Otherwise this right half must be a list of elements specifying a replacement for the matched substring. Any numbers in this right-half list reference (specify) the appropriate part of the matched substring. Other items in the list may reference themselves, or strings in temporary storage, or functions of any referenceable substrings. In the example discussed above, if the right half were (3 2 M 2 H), then the matched portion of the workspace would be replaced by (B A D M D H), and the workspace would become (A B A D M D H G). Note that 1 and 4 were not mentioned in this right half and were therefore deleted from the workspace. Also 3 and 2 were in reverse order, and thus these referenced parts were inserted in the workspace in an order opposite to that in which they had appeared. 2 is referenced twice in this right half and therefore two copies of this referenced substring, "(D)" appear in the workspace. The elements M and H in this right half reference only themselves, and are therefore inserted directly into the workspace.

Using the right-half elements described, that is, numbers referencing matched substrings and constants (elements referencing themselves), one can express transformations of the workspace in which elements have been added to, deleted from, duplicated in, and rearranged in the workspace. Elements to be added to the workspace thus far can only be constants. Let us consider some other possible right-half elements. They are all indicated by lists which start with special flags.

The contents of any shelf (temporary storage list) can be referenced by a two element list with first element either *A (for All) or *N (for Next), and a second element, the shelf name. For example, (*A EQT) references the entire contents of a shelf named EQT. If this element appeared in a right half, the entire contents of that shelf would be placed in the corresponding place in the workspace. The first element of a shelf named SENTENCES could be put into the workspace by using the element (*N SENTENCES) in a right half.

The flag FN as the first member of a list serving as a right-half element indicates that the next member of this list is a function name, and the following ones are the arguments of this function. The value of the function for this set of arguments is placed in the workspace. In this way, any LISP function can be used within a METEOR rule.

The flag *K indicates that the rest of the list following is to be evaluated as a right-half rule, and then is to be "compressed" into a list which will be a single element of the workspace. Thus, chunks which are longer, and have more complex structure than a single word can be treated as a single unit within the METEOR workspace string. The inverse operation is the expansion of a chunk so that all its components appear as individual constituents in the workspace. Expansion is indicated by a *E flag at the beginning of a right-half element list.

We have thus far discussed how the transformation of a string, called the workspace, can be expressed in terms of a left half which is a pattern for a desired input format, and a right half which is a pattern for the desired output format. There is no reason to limit to one the number of outputs from a single left half match. In fact, a third section of a METEOR rule, called the "routing section" (for historical reasons), allows the programmer to give any number of other right halves, and place these referenced lists at the beginning or end of any shelf (temporary storage list). The storage of such a "right half" is indicated in the routing section by a list starting with a *S or a *Q, followed by the shelf name, and followed by a right half pattern. The *S indicates that the referenced material is to be Stored on the beginning of the named shelf. *Q indicates that it should be Queued on the end of the shelf. Used with a *N for retrieval, a shelf built up by a *S is a pushdown list, (a last-in-first-out list), and a shelf built up by a *Q is a queue (first-in-first-out list).

The only other significant feature of a METEOR program that we have not yet touched on is the control structure in a set of rules. A METEOR rule has a name, and has a "go-to" section. Ordinarily, if the left-half match fails, control is automatically passed to the next rule in sequence. If the left-half match succeeds, the right half and routing sections are interpreted, and then control is passed to the rule named in the "go-to". However, by insertion of a "*" immediately after the rule name in the rule, the method of transfer of control is switched, and only on left-half failure will control pass to the rule named in the "go-to".

Routing control can also be changed by a list of the form "(*D name1 name2)" in the routing section of a rule. After this list is interpreted, any occurrence of name1 in a "go-to" will be interpreted as a "go-to" containing "name2". This latter feature allows easy return from subroutines. The use of left-half success or failure as a switch for the transfer of control makes it possible to write significant one rule loops.

A METEOR program is a sequence (list) of rules. Each rule is a list of up to six elements. The following is an example of a METEOR rule containing all six elements:

(NAME * ($ BOY) (2 1) (/ (*S S1 2 2) (*D P1 P2)) P1)

We shall briefly review the function of each of these six elements. The first element of a METEOR rule is a name, and must be present in any rule. If no name is needed, the dummy name "*" can be used. The second element is a "*" and is optional. When it is <u>present</u> it reverses the switch on flow of control, and transfer of control to the rule named in the "go-to" is made on left-half <u>failure</u>.

The third element is mandatory, and is a left-half pattern which is to be matched in the workspace. The fourth element is optional, and is a right-half pattern specifying the result in the workspace of the string transformation desired. The fifth (optional) element is called the routing section, and is a list flagged with a "/" as a first element. The remainder of the routing section is a sequence of lists which specify operations which place items on shelves or set "go-to" values. The final element is called the "go-to" and specifies where control is to be passed if a match succeeds (in the normal case). A "*" in this position specifies the next rule in sequence.

C. Summary.

In this chapter, we have briefly summarized the features of a language for string manipulation which has been embedded (by building the METEOR interpreter) in the general list-processing language LISP. The ability to describe easily in METEOR the string transformations needed to process English sentences, and also use, where appropriate, the functional notation of the general list-processing language, LISP, was a great advantage in the programming effort involved in this study.

As a final illustration of the power of the combined METEOR-LISP language, we include a program for Wang's algorithm for proving theorems in the propositional calculus. This algorithm is described on pages 44-45 of the LISP manual (31), and a LISP program for the algorithm appears on pages 48-50. Figure 3 below contains the complete METEOR program for the algorithm, including definitions of four small auxiliary LISP functions used within the METEOR program.

In addition, the figure contains a trace of the program as it proves the theorem given after the first line containing "(THEOREM)". The other lines give the theorems that are proven by the algorithm as steps in the proof of this theorem. This METEOR program compares quite favorably in both size and understandability to the one given in the LISP manual, and to the one COMIT program which I have seen which performs the Wang algorithm.

Definition of WANG in METEOR

```
DEFINE((
(WANG (LAMBDA (X) (METEOR (QUOTE (
 (TØP    ((*P THEØREM))                                   *)
(*       ($ $1 $ ARRØW $ 2 $)   ((*K *T*))              END)
(A2      (ARRØW $ (FN MAINCØN NØT))   ((FN ARGØNE (*K 3)) 1
             2)                                         TØP)
(B2      ((FN MAINCØN NØT) $ ARRØW)   (2 3 (FN ARGØNE (*K 1
            )))                                         TØP)
(A3      ($ ARRØW $ (FN MAINCØN AND) $)   ((FN AN2 (FN WANG
             (*K 1 2 3 (FN ARGØNE (*K 4)) 5)) (FN WANG (*K
             1 2 3 (FN ARGTWØ (*K 4)) 5))))             END)
(B3      ((FN MAINCØN AND) $ ARRØW)   ((FN ARGØNE (*K 1)) (
            FN ARGTWØ (*K 1)) 2 3)                      TØP)
(A4      (ARRØW $ (FN MAINCØN ØR))   (1 2 (FN ARGØNE (*K 3)
            ) (FN ARGTWØ (*K 3)))                       TØP)
(B4      ($ (FN MAINCØN ØR) $ ARRØW $)   ((FN AN2 (FN WANG
            (*K 1 (FN ARGØNE (*K 2)) 3 4 5)) (FN WANG (*K
            1 (FN ARGTWØ (*K 2)) 3 4 5))))              END)
(A5      (ARRØW $ (FN MAINCØN IMPLIES))   ((FN ARGØNE (*K 3
            )) 1 2 (FN ARGTWØ (*K 3)))                  TØP)
(B5      ($ (FN MAINCØN IMPLIES) $ ARRØW $)   ((FN AN2 (FN
            WANG (*K 1 (FN ARGTWØ (*K 2)) 3 4 5)) (FN WANG
             (*K 1 3 4 5 (FN ARGØNE (*K 2))))))         END)
(A6      ($ ARRØW $ (FN MAINCØN EQUIV) $)   ((FN AN2 (FN
            WANG (*K 1 (FN ARGØNE (*K 4)) 2 3 (FN ARGTWØ (
            *K 4)) 5)) (FN WANG (*K 1 (FN ARGTWØ (*K 4)) 2
             3 (FN ARGØNE (*K 4)) 5))))                 END)
(B6      ($ (FN MAINCØN EQUIV) $ ARRØW $)   ((FN AN2 (FN
            WANG (*K (FN ARGØNE (*K 2)) (FN ARGTWØ (*K 2))
             3 4 5)) (FN WANG (*K 1 3 4 5 (FN ARGØNE (*K 2
            )) (FN ARGTWØ (*K 2))))))                   END)
(FAILURE   ($)   ((*K))                                 END)
))  X)))
))
```

Auxiliary Functions for WANG

```
DEFINE((
(MAINCON (LAMBDA (WS CON)(COND
      ((EQ CON (CAAR WS))(CONS(LIST(CAR WS))(CDR WS)))
      (T NIL) )))
(AN2 (LAMBDA (X Y) (COND (X Y) (T NIL))))
(ARGONE (LAMBDA (X) (LIST(CADAR X))))
(ARGTWO (LAMBDA (X) (LIST(CADDAR X))))
))
```

Trace of a Proof by WANG

```
(THEOREM)
((OR A (NOT B)) ARROW (IMPLIES (AND P Q) (EQUIV P Q)))
(THEOREM)
(A ARROW (IMPLIES (AND P Q) (EQUIV P Q)))
(THEOREM)
(A (AND P Q) ARROW (EQUIV P Q))
(THEOREM)
(A  P Q ARROW (EQUIV P Q))
(THEOREM)
(A P Q P ARROW Q)
(THEOREM)
(A P B Q ARROW P)
(THEOREM)
((NOT B) ARROW (IMPLIES (AND P Q) (EQUIV P Q)))
(THEOREM)
(ARROW B (IMPLIES (AND P Q) (EQUIV P Q)))
(THEOREM)
((AND P Q) ARROW B (EQUIV P Q))
(THEOREM)
(P Q ARROW B (EQUIV P Q))
(THEOREM)
(P  Q P ARROW B Q)
(THEOREM)
(P Q Q ARROW B P)
VALUE
(*T*)
```

Figure 3: A METEOR Program for the Wang Algorithm

CHAPTER IV: TRANSFORMATION OF ENGLISH TO THE STUDENT DEDUCTIVE MODEL

The STUDENT system consists of two main subprograms, called STUDENT and REMEMBER. The program called REMEMBER accepts and processes statements which contain global information; that is, information which is not specific to any one story problem. We shall discuss the processing and information storage techniques used in REMEMBER in the next chapter. A listing of the global information given to the STUDENT system may be found in Appendix C.

In this chapter, we shall describe the techniques embedded in the STUDENT program which are used to transform an English statement of an algebra story problem to expressions in the STUDENT deductive model. By implication we are also defining the subset of English which is "understood" by the STUDENT program. A more explicit description of this input language is given at the end of the chapter.

A. Outline of the Operation of STUDENT.

To provide perspective by which to view the detailed heuristic techniques used in the STUDENT program, we shall first give an outline of the operation of the STUDENT program when given a problem to solve. This outline is a verbal description of the flow chart of the program found in Appendix A.

STUDENT is asked to solve a particular problem. We assume that all necessary global information has been stored previously. STUDENT will now transform the English input statement of this problem into expressions in its limited deductive model, and through appropriate deductive procedures attempt to find a solution. More specifically, STUDENT finds the kernel sentences of the input discourse, and transforms this sequence of kernels into a set of simultaneous equations, keeping a list of the answers required, a list of the units involved in the problem (e.g. dollars, pounds) and a list of all the variables (simple names) in the equations. Then STUDENT invokes the SOLVE program to solve this set of equations for the desired unknowns. If a solution is found, STUDENT prints the values of the unknowns requested in a fixed format, substituting in "(variable IS value)" the appropriate phrases for variable and value. If a solution cannot be found, various heuristics are used to identify two variables (i.e. find two slightly different phrases that refer to the same object in the model). If two variables, A and B, are identified, the equation A = B is added to the set of equations. In addition, the store of global information is searched to find any equations that may be useful in finding the solution to this problem. STUDENT prints out any assumptions it makes about the identity of two variables, and also any equations that it retrieves because it thinks they may be relevant. If the use of global equations or equations from identifications leads to a solution, the answers are printed out in the format described above.

If a solution was not found, and certain idioms are present in the problem (a result of a definitional transformation used in the generation of the problem), a substitution is made for each of these idioms in turn and the transformation and solution process is repeated. If the substitutions for these idioms do not enable the problem to be solved by STUDENT, then STUDENT requests additional information from the questioner, showing him the variables being used in the problem. If any information is given, STUDENT tries to solve the problem again. If none is given, it reports its inability to solve this problem and terminates. If the problem is ever solved, the solution is printed and the program terminates.

B. Categories of Words in a Transformation.

The words and phrases (strings of words) in the English input can be classified into three distinct categories on the basis of how they are handled in the transformation to the deductive model. The first category consists of strings of words which name objects in the model; I call such strings, variables. Variables are identified only by the string of words in them, and if two strings differ at all, they define distinct variables. One important problem considered below is how to determine when two distinct variables refer to the same object.

The second class of words and phrases are what I call "substitutors". Each substitutor may be replaced by another string. Some substitutions are mandatory; others are optional and are only made if the problem cannot be solved without such substitutions. An example of a mandatory substitution is "2 times" for the word "twice". "Twice" always means "2 times" in the context of the model, and therefore this substitution is mandatory. One optional "idiomatic" substitution is "twice the sum of the length and width of the rectangle" for "the perimeter of the rectangle". The use of these substitutions in the transformation process is discussed below. These substitutions are inverses of definitional transformations as defined in Chapter II.

Members of the third class of words indicate the presence of functional linguistic forms which represent functions in the deductive model. I call members of this third class "operators". Operators may indicate operations which are complex combinations of the basic functions of the deductive model. One simple operator is the word "plus", which indicates that the objects named by the two variables surrounding it are to be added. An example of a more complex operator is the phrase "percent less than", as in "10 percent less than the marked price", which indicates that the number immediately preceding the "percent" is to be subtracted from 100, this result divided by 100, and then this quotient multiplied by the variable following the "than".

Operators may be classified according to where their arguments are found. A prefix operator, such as "the square of....." precedes its argument. An operator like ".....percent" is a suffix operator, and follows its argument. Infix operators such as ".....plus....." or ".....less than....." appear between their two arguments. In a split prefix operator such as "difference between.....and.....", part of the operator precedes, and part appears between the two arguments. "The sum of.....and .....and....." is a split prefix operator with an indefinite number of arguments.

Some words may act as operators conditionally, depending on their context. For example, "of" is equivalent to "times" if there is a fraction immediately preceding it; e.g., ".5 of the profit" is equivalent to ".5 times the profit"; however, "Queen of England" does not imply a multiplicative relationship between the Queen and her country.

C. Transformational Procedures.

Let us now consider in detail the transformation procedure used by STUDENT, and see how these different categories of phrases interact. To make the process more concrete, let us consider the following example which has been solved by STUDENT.

(THE PROBLEM TO BE SOLVED IS)
(IF THE NUMBER OF CUSTOMERS TOM GETS IS TWICE THE SQUARE OF 20 PER CENT OF THE NUMBER OF ADVERTISEMENTS HE RUNS, AND THE NUMBER OF ADVERTISEMENTS HE RUNS IS 45, WHAT IS THE NUMBER OF CUSTOMERS TOM GETS Q.)

Shown below are copies of actual printout from the STUDENT program, illustrating stages in the transformation and the solution of the problem. The parentheses are an artifact of the LISP programming language, and "Q." is a replacement for the question mark not available on the key punch.

The first stage in the transformation is to perform all mandatory substitutions. In this problem only the three phrases underlined (by the author, not the program) are substitutors: "twice" becomes "2 times", "per cent" becomes the single word "percent", and "square of" is truncated to "square". Having made these substitutions, STUDENT prints:

```
(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(IF THE NUMBER OF CUSTOMERS TOM GETS IS 2 TIMES THE SQUARE
20 PERCENT OF THE NUMBER OF ADVERTISEMENTS HE RUNS, AND THE
NUMBER OF ADVERTISEMENTS HE RUNS IS 45, WHAT IS THE NUMBER
OF CUSTOMERS TOM GETS Q.)
```

From dictionary entries for each word, the words in the problem are tagged by their function in terms of the transformation process, and STUDENT prints:

```
(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
(IF THE NUMBER (OF / OP) CUSTOMERS TOM (GETS / VERB) IS
2 (TIMES / OP 1) THE (SQUARE / OP 1) 20 (PERCENT / OP 2)(OF/OP)
THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS, AND THE
NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS IS 45,
(WHAT / QWORD) IS THE NUMBER (OF / OP) CUSTOMERS
TOM (GETS / VERB) (QMARK / DLM))
```

If a word has a tag, or tags, the word followed by "/", followed by the tags, becomes a single unit, and is enclosed in parentheses. Some typical taggings are shown above. "(OF/OP)" indicates that "OF" is an operator and other taggings show that "GETS" is a verb, "TIMES" is an operator of level 1 (operator levels will be explained below), "SQUARE" is an operator of level 1, "PERCENT" is an operator of level 2, "HE" is a pronoun, "WHAT" is a question word, and "QMARK" (replacing Q.) is a delimiter of a sentence. These tagged words will play the principal role in the remaining transformation to the set of equations implicit in this problem statement.

The next stage in the transformation is to break the input sentences into "kernel sentences". As in the example, a problem may be stated using sentences of great grammatical complexity; however, the final stage of the transformation is only defined on a set of kernel sentences. The simplification to kernel sentences as done in STUDENT depends on the recursive use of format matching. If an input sentence is of the form "IF" followed by a substring, followed by a comma, a question word and a second substring (i.e. it matches the METEOR left half "(IF $, ($1/ QWORD($)") then the first substring (between the IF and the comma) is made an independent sentence, and everything following the comma is made into a second sentence. In the example, this means that the input is resolved into the following two sentences, (where tags are omitted for the sake of brevity):

> "The number of customers Tom gets is 2 times the square 20 percent of the number of advertisements he runs, and the number of advertisements he runs is 45." and "What is the number of customers Tom gets?"

This last procedure effectively resolves a problem into declarative assumptions and a question sentence. A second complexity resolved by STUDENT is illustrated in the first sentence of this pair. A coordinate sentence consisting of two sentences joined by a comma immediately followed by an "and" (i.e., any sentence matching the METEOR left half "($, AND $)") will be resolved into these two independent sentences. The first sentence above is therefore resolved into two simpler sentences.

Using these two inverse syntactic transformations, this problem statement is resolved into "simple kernel sentences." For the example, STUDENT prints

(THE SIMPLE SENTENCES ARE)

(THE NUMBER (OF/OP) CUSTOMERS TOM (GETS / VERB) IS 2 (TIMES /OP 1) THE (SQUARE / OP 1) 20 (PERCENT / OP 2) (OF / OP) THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS (PERIOD / DLM))

(THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS IS 45 (PERIOD / DLM))

((WHAT / QWORD) IS THE NUMBER (OF / OP) CUSTOMERS TOM (GETS / VERB) (QMARK / DLM))

Each simple sentence is a separate list, i.e., is enclosed in parentheses, and each ends with a delimiter (a period or question mark). Each of these sentences can now be transformed directly to its interpretation in the model.

D. From Kernel Sentences to Equations.

The transformation from the simple kernel sentences to equations uses three levels of precedence for operators. Operators of higher precedence level are used earlier in the transformation. Before utilizing the operators, STUDENT looks for linguistic forms associated with the equality relation. These forms include the copula "is" and transitive verbs in certain contexts. In the example we are considering, only the copula "is" is used to indicate equality. The use of transitive verbs as indicators of equality, that is, as relational linguistic forms, will be discussed in connection with another example. When the relational linguistic form is identified, the names which are the arguments of the form are broken down into variables and operators (functional linguistic forms). In the present problem, the two names are those on either side of the "is" in each sentence.

The word "is" may also be used meaningfully within algebra story problems as an auxiliary verb (<u>not</u> meaning equality) in such verbal phrases as "is multiplied by" or "is divided by". A special check is made for the occurrence of these phrases before proceeding on to the main transformation procedure. The transformation of sentences containing these special verbal phrases will be discussed later. If "is" does not appear as an auxiliary in such a verbal phrase, a sentence of the form "P1 is P2" is interpreted as indicating the equality of the objects named by phrases P1 and P2. No equality relation will be recognized within these phrases, even if an appropriate transitive verb occurs within either of them. If P1* and P2* represent the arithmetic transformations of P1 and P2, then "P1 is P2" is transformed into the equation

"(EQUAL P1* P2*)".

The transformation of P1 and P2 to give them an interpretation in the model is performed recursively using a program equivalent to the table in Figure 4. This table shows all the operators and formats currently recognized by the STUDENT program. New operators can easily be added to the program equivalent of this table.

In performing the transformation of a phrase P, a left to right search is made for an operator of level 2 (indicated by subscripts of "OP" and 2). If there is none, a left to right search is made for a level 1 operator (indicated by subscripts "OP" and 1), and finally another left to right search is made for an operator of level 0 (indicated by a subscript "OP" and no numerical subscript). The first operator found in this ordered search determines the first step in the transformation of the phrase. This operator and its context are transformed as indicated in column 4 in the table. If no operator is present, delimiters and articles (a, an and the) are deleted, and the phrase is treated as an indivisible entity, a variable.

In the example, the first simple sentence is

> (THE NUMBER (OF/OP) CUSTOMERS TOM (GETS/VERB) IS
> 2 (TIMES/OP 1) THE (SQUARE/OP 1) 20 (PERCENT/OP 2)
> (OF/OP) THE NUMBER (OF/OP) ADVERTISEMENTS
> (HE/PRO) RUNS (PERIOD/DLM))

This is of the form "P1 is P2", and is transformed to (EQUAL P1* P2*). P1 is "(THE NUMBER (OF/OP) CUSTOMERS TOM (GETS/VERB))". The occurrence of the verb "gets" is ignored because of the presence of the "is" in the sentence, meaning "equals". The only operator found is "(OF/OP)". From the table we see that if "OF" is immediately preceded by a number (<u>not</u> the word "number") it is treated as if it were the infix "TIMES". In this case, however, "OF" is not preceded by a number; the subscript OP, indicating that "OF" is an operator, is

Operator	Precedence Level	Context	Interpretation in the Model	
PLUS	2	P1 PLUS P2	(PLUS P1* P2*)	(a)
PLUSS	0	P1 PLUSS P2	(PLUS P1* P2*)	(b)
MINUS	2	P1 MINUS P2	(PLUS P1* (MINUS P2*))	(c)
		MINUS P2	(MINUS P2*)	
MINUSS	0	P1 MINUSS P2	(PLUS P1* (MINUS P2*))	(b)
TIMES	1	P1 TIMES P2	(TIMES P1* P2*)	
DIVBY	1	P1 DIVBY P2	(QUOTIENT P1* P2*)	
SQUARE	1	SQUARE P1	(EXPT P1* 2)	(d)
SQUARED	0	P1 SQUARED	(EXPT P1* 2)	
**	0	P1 ** P2	(EXPT P1* P2*)	
LESSTHAN	2	P1 LESSTHAN P2	(PLUS P2* (MINUS P1*))	
PER	0	P1 PER K P2	(QUOTIENT P1* (K P2)*)	(e) (f)
		P1 PER P2	(QUOTIENT P1* (1 P2)*)	
PERCENT	2	P1 K PERCENT P2	(P1 (K/100) P2)*	(f) (g)
PERLESS	2	P1 K PERLESS P2	(P1((100-K)/100) P2)*	(f) (g)
SUM	0	SUM P1 AND P2 AND P3	(PLUS P1* (SUM P2 AND P3)*)	
		SUM P1 AND P2	(PLUS P1* P2*)	
DIFFERENCE	0	DIFFERENCE BETWEEN P1 AND P2	(PLUS P1* (MINUS P2*))	
OF	0	K OF P2	(TIMES K P2*)	
		P1 OF P2	(P1 OF P2)*	

(a) If P1 is a phrase, P1* indicates its interpretation in the model.

(b) PLUSS and MINUSS are identical to PLUS AND MINUS except for precedence level.

(c) When two possible contexts are indicated, they are checked in the order shown.

(d) SQUARE P1 and SUM P1 are idiomatic shortenings of SQUARE OF P1 and SUM OF P1.

(e) * outside a parenthesized expression indicates that the enclosed phrase is to be transformed.

(f) K is a number.

(g) / and - imply that the indicated arithmetic operations are actually performed.

Figure 4: Operators Recognized by STUDENT

stripped away, and the transformation process is repeated on the phrase with "OF" no longer acting as an operator. In this repetition, no operators are found, and P1* is the variable

(NUMBER OF CUSTOMERS TOM (GETS/VERB))

To the right of "IS" in the sentence is P2:

(2 (TIMES/OP 1) THE (SQUARE/OP 1) 20 (PERCENT/OP 2) (OF/OP) THE NUMBER (OF/OP) ADVERTISEMENTS (HE/PRO) RUNS (PERIOD/DLM))

The first operator found in P2 is PERCENT, an operator of level 2. From the table in Figure 4, we see that this operator has the effect of dividing the number immediately preceding it by 100. The "PERCENT" is removed and the transformation is repeated on the remaining phrase. In the example, the "...20 (PERCENT/OP 2) (OF/OP)..." becomes "... .2000(OF/OP)....".

Continuing the transformation, the operators found are, in order, TIMES, SQUARE, OF and OF. Each is handled as indicated in the table. The "OF" in the context "... .2000 (OF/OP) THE ...." is treated as an infix TIMES, while at the other occurrence of "OF", the operator marking is removed. The resulting transformed expression for P2 is:

(TIME 2 (EXPT (TIMES .2 (NUMBER OF ADVERTISEMENTS (HE/PRO) RUNS)) 2))

The transformation of the second sentence of the example is done in a similar manner, and yields the equation:

(EQUAL (NUMBER OF ADVERTISEMENTS (HE/PRO) RUNS) 45)

The third sentence is of the form "What is P1?". It starts with a question word and is therefore treated specially. A unique variable, a single word consisting of an X of G followed by five integers, is created, and the equation (EQUAL Xnnnnn P1*) is stored. For this example, the variable X00001 was created, and this last simple sentence is transformed to the equation:

(EQUAL X00001 (NUMBER OF CUSTOMERS TOM (GETS/VERB))

In addition, the created variable is placed on the list of variables for which STUDENT is to find a value. Also, this variable is stored, paired with P1, the untransformed right side, for use in printing out the answer. If a value is found for this variable, STUDENT prints the sentence (P1 is value) with the appropriate substitution for value. Below we show the full set of equations, and the printed solution given by STUDENT for the example being considered. For ease in solution, the last equations created are put first in the list of equations.

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL X00001 (NUMBER OF CUSTOMERS TOM (GETS/VERB)))
(EQUAL (NUMBER OF ADVERTISEMENTS (HE/PRO) RUNS) 45)
(EQUAL (NUMBER OF CUSTOMERS TOM (GETS/VERB)) (TIMES 2 (EXPT (TIMES .2000 (NUMBER OF ADVERTISEMENTS (HE/PRO) RUNS)) 2)))

(THE NUMBER OF CUSTOMERS TOM GETS IS 162)

In the example just shown, the equality relation was indicated by the copula "is". In the problem shown below, solved by STUDENT, equality is indicated by the occurrence of a transitive verb in the proper context.

(THE PROBLEM TO BE SOLVED IS)

(TOM HAS TWICE AS MANY FISH AS MARY HAS GUPPIES. IF MARY HAS 3 GUPPIES, WHAT IS THE NUMBER OF FISH TOM HAS Q.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL X00001 (NUMBER OF FISH TOM (HAS/VERB)))

(EQUAL (NUMBER OF GUPPIES (MARY/PERSON) (HAS/VERB)) 3)

(EQUAL (NUMBER OF FISH TOM (HAS/VERB)) (TIMES 2 (NUMBER OF GUPPIES (MARY/PERSON) (HAS/VERB))))

(THE NUMBER OF FISH TOM HAS IS 6)

The verb in this case is "has". The simple sentence "Mary has 3 guppies" is transformed to the "equivalent" sentence "The number of guppies Mary has is 3" and the processing of this latter sentence is done as previously discussed.

The general format for this type of sentence, and the format of the intermediate sentence to which it is transformed is best expressed by the following METEOR rule:

(* ($($1/VERB)($1/NUMBER) $) (THE NUMBER OF 4 1 2 IS 3) *)

This rule may be read: anything (a subject) followed by a verb followed by a number followed by anything (the unit) is transformed to a sentence starting with "THE NUMBER OF" followed by the unit, followed by the subject and the verb, followed by "IS" and then the number. In "Mary has 3 guppies" the subject is "Mary", the verb "has", and the units "guppies". Similarly, the sentence "The witches of

Firth brew 3 magic potions" would be transformed to

"The number of magic potions the witches of Firth brew is 3."

In addition to a declaration of number, a single-object transitive verb may be used in a comparative structure, such as exhibited in the sentence "Tom has twice as many fish as Mary has guppies." The METEOR rule which gives the effective transformation for this type of sentence structure is:

```
(* ($ ($1/VERB) $ AS MANY $ AS $ ($1/VERB) $)
          (THE NUMBER OF 6 1 2 IS 3 THE NUMBER OF 10 8 9) *)
```

For the example, the transformed sentence is:

"The number of fish Tom has is twice the number of guppies Mary has."

Transformation of new sentence formats to formats previously "understood" by the program can be easily added to the program, thus extending the subset of English "understood" by STUDENT. In the processing that actually takes place within STUDENT the intermediate sentences shown never exist. It was easier to go directly to the model from the format, utilizing subroutines previously defined in terms of the semantics of the model.

The word "is" indicates equality only if it is not used as an auxiliary. The example below shows how verbal phrases containing "is", such as "is multiplied by", and "is increased by" are handled in the transformation.

(THE PROBLEM TO BE SOLVED IS)
(A NUMBER IS MULTIPLIED BY 6. THIS PRODUCT IS INCREASED BY 44.
THIS RESULT IS 68 . FIND THE NUMBER .)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL X00001 (NUMBER))
(EQUAL (PLUS (TIMES (NUMBER) 6) 44) 68)

(THE NUMBER IS 4)

The sentence "A number is multiplied by 6" only indicates that two objects in the model are related multiplicatively, and does not indicate explicitly any equality relation. The interpretation of this sentence in the model is the prefix notation product:

(TIMES (NUMBER) 6)

This latter phrase is stored in a temporary location for possible later reference. In this problem, it is referenced in the next sentence, with the phrase "THIS PRODUCT". The important word in this last phrase is "THIS" — STUDENT ignores all other words in a variable containing the key word "THIS". The last temporarily stored phrase is substituted for the phrase containing "THIS". Thus, the first three sentences in the problem shown above yield only one equation, after two substitutions for "this" phrases. The last sentence "Find the number." is transformed as if it were "What is the number Q.", and yields the first equation shown.

The word "this" may occur in a context where it is not referring to a previously stored phrase. Below is an example of such a context.

(THE PROBLEM TO BE SOLVED IS)
(THE PRICE OF A RADIO IS 69.70 DOLLARS . IF THIS PRICE IS
15 PERCENT LESS THAN THE MARKED PRICE, FIND THE MARKED PRICE.)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL X00001 (MARKED PRICE))
(EQUAL (PRICE OF RADIO) (TIMES .8499 (MARKED PRICE)))
(EQUAL (PRICE OF RADIO)(TIMES 69.70 (DOLLARS)))

(THE MARKED PRICE IS 82 DOLLARS)

In such contexts, the phrase containing "THIS" is replaced by the left half of the last equation created. In this example, STUDENT breaks the last sentence into two simple sentences, deleting the "IF". Then the phrase "THIS PRICE" is replaced by the variable "PRICE OF RADIO", which is the left half of the previous equation.

This problem illustrates two other features of the STUDENT program. The first is the action of the complex operator "percent less than". It causes the number immediately preceding it, i.e., 15, to be subtracted from 100, this result divided by 100, to give .85 (printed as .8499 due to a rounding error in floating point conversion). Then this operator becomes the infix operator "TIMES". This is indicated in the table in Figure 4 .

This problem also illustrates how units such as "dollars" are handled by the STUDENT program. Any word which immediately follows a number is labeled as a special type of variable called a <u>unit</u>. A number followed by a unit is treated in the equation as a product of the number and the unit, e.g.,"69.70 DOLLARS" becomes "(TIMES 69.70 (DOLLARS))". Units are treated as special variables in solving the set of equations; a unit may appear in the answer though other variables cannot. If the value for a variable found by the solver is the product of a number and a unit, STUDENT concatenates the number and the unit. For example, the solution for "(MARKED PRICE)" in the problem above was (TIMES 82 (DOLLARS)) and STUDENT printed out:

(THE MARKED PRICE IS 82 DOLLARS)

There is an exception to the fact that any unit may appear in the answer, as illustrated in the problem below.

(THE PROBLEM TO BE SOLVED IS)
(IF 1 SPAN EQUALS 9 INCHES, AND 1 FATHOM EQUALS 6 FEET, HOW MANY SPANS EQUALS 1 FATHOM Q.)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL X00001 (TIMES 1(FATHOMS)))
(EQUAL (TIMES 1(FATHOMS)) (TIMES 6 (FEET)))
(EQUAL (TIMES 1 (SPANS)) (TIMES 9 (INCHES)))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(USING THE FOLLOWING KNOWN RELATIONSHIPS)
((EQUAL (TIMES 1 (YARDS)) (TIMES 3 (FEET))) (EQUAL (TIMES 1 (FEET)) (TIMES 12 (INCHES))))

(1 FATHOM IS 8 SPANS)

If the unit of the answer is specified, in this problem by the phrase "how many <u>spans</u>" — then <u>only</u> that unit, in this problem "spans", may appear in the answer. Without this restriction, STUDENT would blithely answer this problem with "(1 FATHOM IS 1 FATHOM)".

In the transformation from the English statement of the problem to the equations, "9 INCHES" became (TIMES 9 (INCHES)). However, "1 FATHOM" became "(TIMES 1 (FATHOMS))". The plural form for fathom has been used instead of the singular form. STUDENT always uses the plural form if known, to ensure that all units appear in only one form. Since "fathom" and "fathoms" are different, if both were used STUDENT would treat them as distinct, unrelated units. The plural form is part of the global information that can be made available to STUDENT, and the plural form of a word is substituted for any singular form appearing after "1" in any phrase. The inverse operation is carried out for correct printout of the solution.

Notice that the information given in the problem was insufficient to allow solution of the set of equations to be solved. Therefore, STUDENT looked in its glossary for information concerning each of the units in this set of equations. It found the relationships "1 foot equals 12 inches." and "1 yard equals 3 feet." Using only the first fact, and the equation it implies, STUDENT is then able to solve the problem. Thus, in certain cases where a problem is not <u>analytic</u>, in the sense that it does not contain, explicitly stated, all the information needed for its solution, STUDENT is able to draw on a body of facts, picking out relevant ones, and use them to obtain a solution.

In certain problems, the transformation process does not yield a set of solvable equations. However, within this set of equations there exists a pair of variables (or more than one pair) such that the two variables are only "slightly different", and really name the same object in the model. When a set of equations is unsolvable, STUDENT searches for relevant global equations. In addition, it uses several heuristic techniques for identifying two "slightly different" variables in the equations. The problem below illustrates the identification of two variables where in one variable a pronoun has been substituted for a noun phrase in the other variable. This identification is made by checking all variables appearing before one containing the pronoun, and finding one which is identical to this pronoun phrase, with a substitution of a string of any length for the pronoun.

```
(THE PROBLEM TO BE SOLVED IS)
(THE NUMBER OF SOLDIERS THE RUSSIANS HAVE IS ONE HALF OF THE
NUMBER OF GUNS THEY HAVE . THE NUMBER OF GUNS THEY HAVE IS
7000 . WHAT IS THE NUMBER OF SOLDIERS THEY HAVE Q.)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL X00001 (NUMBER OF SOLDIERS (THEY/PRO) (HAVE/VERB)))
(EQUAL (NUMBER OF GUNS (THEY/PRO) (HAVE/VERB)) 7000)
(EQUAL (NUMBER OF SOLDIERS RUSSIANS (HAVE/VERB)) (TIMES .5000
(NUMBER OF GUNS (THEY/PRO) (HAVE/VERB))))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((NUMBER OF SOLDIERS (THEY/PRO) (HAVE/VERB)) IS EQUAL TO
(NUMBER OF SOLDIERS RUSSIANS (HAVE/VERB)))

(THE NUMBER OF SOLDIERS THEY HAVE IS 3500)
```

If two variables match in this fashion, **STUDENT** assumes the two variables are equal, prints out a statement of this assumption, as shown, and adds an equation expressing this equality to the set to be solved. The solution procedure is tried again, with this additional equation. In the example, the additional equation was sufficient to allow determination of the solution.

The example below is again a "non-analytic" problem. The first set of equations developed by STUDENT is unsolvable. Therefore, STUDENT tries to find some relevant equations in its store of global information.

```
(THE PROBLEM TO BE SOLVED IS)
(THE GAS CONSUMPTION OF MY CAR IS 15 MILES PER GALLON.
THE DISTANCE BETWEEN BOSTON AND NEW YORK IS 250 MILES.
WHAT IS THE NUMBER OF GALLONS OF GAS USED ON A TRIP
BETWEEN NEW YORK AND BOSTON Q.)


(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL X00001 (NUMBER OF GALLONS OF GAS USED ON TRIP
BETWEEN NEW YORK AND BOSTON))

(EQUAL (DISTANCE BETWEEN BOSTON AND NEW YORK) (TIMES
250(MILES)))

(EQUAL (GAS CONSUMPTION OF MY CAR) (QUOTIENT (TIMES
15(MILES)) (TIMES 1 (GALLONS))))


THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION


(USING THE FOLLOWING KNOWN RELATIONSHIPS)
((EQUAL (DISTANCE) (TIMES (SPEED) (TIME))) (EQUAL (DISTANCE)
(TIMES (GAS CONSUMPTION) (NUMBER OF GALLONS OF GAS USED))))

(ASSUMING THAT)
((DISTANCE) IS EQUAL TO (DISTANCE BETWEEN BOSTON AND NEW
YORK))

(ASSUMING THAT)
((GAS CONSUMPTION) IS EQUAL TO (GAS CONSUMPTION OF MY CAR))

(ASSUMING THAT)
((NUMBER OF GALLONS OF GAS USED) IS EQUAL TO (NUMBER OF
GALLONS OF GAS USED ON TRIP BETWEEN NEW YORK AND BOSTON))


(THE NUMBER OF GALLONS OF GAS USED ON A TRIP BETWEEN
NEW YORK AND BOSTON IS 16.66 GALLONS)
```

It uses the first word of each variable string as a key to its glossary. The one exception to this rule is that the words "number of" are ignored if they are the first two words of a variable string. Thus, in this problem, STUDENT retrieved equations which were stored under the key words <u>distance</u>, <u>gallons</u>, <u>gas</u>, and <u>miles</u>. Two facts about <u>distance</u> had been stored earlier; "distance equals speed times time" and "distance equals gas consumption times number of gallons of gas used". The equations implicit in these sentences were stored and retrieved now — as possibly useful for the solution of this problem. In fact, only the second is relevant.

Before any attempt is made to solve this augmented set of equations, the variables in the augmented set are matched, to identify "slightly different" variables which refer to the same object in the model. In this example "(DISTANCE)", "(GAS CONSUMPTION)" and "(NUMBER OF GALLONS OF GAS USED)", are all identified with "similar" variables. The following conditions must be satisfied for this type of identification of variables P1 and P2:

1) P1 must appear later in the problem than P2.
2) P1 is completely contained in P2 in the sense that P1 is a contiguous substring within P2.

This identification reflects a syntactic phenomenon where a truncated phrase, with one or more modifying phrases dropped, is often used in place of the original phrase. For example, if the phrase "the length of a rectangle" has occurred, the phrase "the length" may be used to mean the same thing. This type of identification is distinct from that made using pronoun substitution.

In the example above, a stored schema was used by identifying the variables in the schema with the variables that occur in the problem. This problem is solvable because the key phrases "distance", "gas consumption" and "number of gallons of gas used" occur as substrings of the variables in the problem. Since STUDENT identifies each generic key phrase of the schema with a particular variable of the problem, any schema can be used only once in a problem. Because STUDENT handles schema in this ad hoc fashion it cannot solve problems in which a relationship such as "distance equals speed times time" is needed for two different values of distance, speed, and time.

E. Possible Idiomatic Substitutions.

There are some phrases which have a dual character, depending on the context. In the example below, the phrase "perimeter of a rectangle" becomes a variable with no reference to its meaning, or definition, in terms of the length and width of the rectangle. This definition is unneeded for solution.

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF THE PERIMETER OF A RECTANGLE AND THE PERIMETER
OF A TRIANGLE IS 24 INCHES.  IF THE PERIMETER OF THE RECTANGLE
IS TWICE THE PERIMETER OF THE TRIANGLE, WHAT IS THE
PERIMETER OF THE TRIANGLE Q.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL X00001 (PERIMETER OF TRIANGLE))

(EQUAL (PERIMETER OF RECTANGLE) (TIMES 2 (PERIMETER OF
TRIANGLE)))

(EQUAL (PLUS (PERIMETER OF RECTANGLE) (PERIMETER OF TRIANGLE))
(TIMES 24 (INCHES)))

(THE PERIMETER OF THE TRIANGLE IS 8 INCHES)
```

However, the following problem is stated in terms of the perimeter, length and width of the rectangle. Transforming the English into

```
(THE PROBLEM TO BE SOLVED IS)
(THE LENGTH OF A RECTANGLE IS 8 INCHES MORE THAN THE WIDTH
OF THE RECTANGLE . ONE HALF OF THE PERIMETER OF THE RECTANGLE
IS 18 INCHES . FIND THE LENGTH AND THE WIDTH OF THE RECTANGLE
.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02516 (WIDTH OF RECTANGLE))

(EQUAL G02515 (LENGTH))

(EQUAL (TIMES  .5000 (PERIMETER OF RECTANGLE)) (TIMES 18 (INCHES)))

(EQUAL (LENGTH OF RECTANGLE) (PLUS (TIMES 8 (INCHES)) (WIDTH
OF RECTANGLE)))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(USING THE FOLLOWING KNOWN RELATIONSHIPS)
((EQUAL (TIMES 1 (FEET)) (TIMES 12 (INCHES))))

(ASSUMING THAT)
((LENGTH) IS EQUAL TO (LENGTH OF RECTANGLE))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

TRYING POSSIBLE IDIOMS

(THE PROBLEM WITH AN IDIOMATIC SUBSTUTION IS)
(THE LENGTH OF A RECTANGLE IS 8 INCHES MORE THAN THE WIDTH
OF THE RECTANGLE . ONE HALF OF TWICE THE SUM OF THE LENGTH
AND WIDTH OF THE RECTANGLE IS 18 INCHES . FIND THE LENGTH AND
THE WIDTH OF THE RECTANGLE .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02518 (WIDTH OF RECTANGLE))

(EQUAL G02517 (LENGTH))

(EQUAL (TIMES (TIMES  .5000 2) (PLUS (LENGTH) (WIDTH OF RECTANGLE)))
(TIMES 18 (INCHES)))

(EQUAL (LENGTH OF RECTANGLE) (PLUS (TIMES 8 (INCHES)) (WIDTH
OF RECTANGLE)))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(USING THE FOLLOWING KNOWN RELATIONSHIPS)
((EQUAL (TIMES 1 (FEET)) (TIMES 12 (INCHES))))

(ASSUMING THAT)
((LENGTH) IS EQUAL TO (LENGTH OF RECTANGLE))

(THE LENGTH IS  13 INCHES)

(THE WIDTH OF THE RECTANGLE IS  5 INCHES)
```

equations is not sufficient for solution. Neither retrieving and using an equation about "inches", the unit in the problem, nor identifying "length" with a longer phrase serve to make the problem solvable. Therefore, STUDENT looks in its dictionary of possible idioms, and finds one which it can try in the problem. STUDENT actually had two possible idiomatic substitutions which it could have made for "perimeter of a rectangle"; one was in terms of the length and width of the rectangle and the other was in terms of the shortest and longest sides of the rectangle. When there are two possible substitutions for a given phrase, one is tried first, namely the one STUDENT has been told about most recently. In this problem, the correct one was fortunately first. If the other had been first, the revised problem would not have been any more solvable than the original, and eventually the second (correct) substitution would have been made. Only one non-mandatory idiomatic substitution is ever made at one time, although the substitution is made for all occurrences of the phrase chosen.

In this problem, the idiomatic substitution made allows the problem to be solved, after identification of the variables "length" and "length of rectangle". The retrieved equation about inches was not needed. However, its presence in the set of equations to be solved did not sidetrack the solver in any way.

This use of possible, but non-mandatory idiomatic substitutions can also be used to give STUDENT a way to solve problems in which two phrases denoting one particular variable are quite different. For example, the phrase, "students who passed the admissions test" and "successful candidates" might be describing the same set of people. However, since STUDENT knows nothing of the "real world" and its value system for success, it would never identify these two phrases. However, if told that "successful candidates" sometime means "students who passed the admissions test", it would be able to solve a problem using these two phrases to identify the same variable. Thus, possible idiomatic substitutions serve the dual purpose of providing tentative substitutions of definitions, and identification of synonomous phrases.

F. Special Heuristics.

The methods thus far discussed have been applicable to the entire range of algebra problems. However, for special classes of problems, additional heuristics may be used which are needed for members of the class, but not applicable to other problems. An example is the class of age problems, as typified by the problem below.

```
(THE PROBLEM TO BE SOLVED IS)
(BILL S FATHER S UNCLE IS TWICE AS OLD AS BILL S FATHER. 2
YEARS FROM NOW BILL S FATHER WILL BE 3 TIMES AS OLD AS BILL.
THE SUM OF THEIR AGES IS 92 . FIND BILL S AGE .)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL X00001 ((BILL / PERSON) S AGE))
(EQUAL (PLUS ((BILL / PERSON ) S (FATHER / PERSON) S (UNCLE
/ PERSON) S AGE) (PLUS ((BILL / PERSON) S (FATHER / PERSON)
S AGE) ((BILL / PERSON) S AGE))) 92)
(EQUAL (PLUS ((BILL / PERSON) S (FATHER / PERSON) S AGE) 2)
(TIMES 3 (PLUS ((BILL / PERSON) S AGE)))

(BILL S AGE IS 8)
```

Before the age problem heuristics are used, a problem must be identified as belonging to that class of problems. STUDENT identifies age problems by any occurrence of one of the following phrases, "as old as", "years old" and "age". This identification is made immediately after all words are looked up in the dictionary and tagged by function.

After the special heuristics are used the modified problem is transformed to equations as described previously.

The need for special methods for age problems arises because of the conventions used for denoting the variables, all of which are ages. The word age is usually not used explicitly, but is implicit in such phrases as "as old as". People's names are used where their ages are really the implicit variables. In the example, for instance, the phrase "Bill's father's uncle" is used instead of the phrase "Bill's father's uncle's age".

STUDENT uses a special heuristic to make all these ages explicit. To do this, it must know which words are "person words" and therefore, may be associated with an age. For this problem STUDENT has been told that Bill, father, and uncle are person words. They can be seen tagged as such in the equations. The " " following a word is the STUDENT representation for possessive, used instead of "apostrophe - s" for programming convenience. STUDENT inserts a "S AGE" after every person word not followed by a "S" (because this "S" indicates that the person word is being used in a possessive sense, not as an independent age variable). Thus, as indicated, the phrase "BILL S FATHER S UNCLE" becomes "BILL S FATHER S UNCLE S AGE".

In addition to changing phrases naming people to ones naming ages, STUDENT makes certain special idiomatic substitutions. For the phrase "their ages", STUDENT substitutes a conjunction of all the age variables encountered in the problem. In the example, for "THEIR AGES" STUDENT substitutes "BILL S FATHER S UNCLE S AGE AND BILL S FATHER S AGE AND BILL S AGE". The phrases "as old as" and "years old" are then deleted as dummy phrases not having any meaning, and "will be" and "was" are changed to "is". There is no need to preserve the tense of the copula, since the sense of the future or past tense is preserved in such prefix phrases as "2 years from now", or "3 years ago".

The remaining special age problem heuristics are used to process the phrases "in 2 years", "5 years ago" and "now". The phrase "2 years from now" is transformed to "in 2 years" before processing. These three time phrases may occur immediately after the word "age", (e.g., "Bill's age 3 years ago") or at the beginning of the sentence. If a time phrase occurs at the beginning of the sentence, it implicitly modifies all ages mentioned in the sentence, <u>except</u> those followed by their own time phrase. For example, "In 2 years Bill's father's age will be 3 times Bill's age" is equivalent to "Bill's father's age in 2 years will be 3 times Bill's age in 2 years". However, "3 years ago Mary's age was 2 times Ann's age now" is equivalent to "Mary's age 3 years ago was 2 times Ann's age now". Thus prefix time phrases are handled by distributing them over all ages not modified by another time phrase.

After these prefix phrases have been distributed, each time phrase is translated appropriately. The phrase "in 5 years" causes 5 to be added to the age it follows, and "7 years ago" causes 7 to be subtracted from the age preceding this phrase. The word "now" is deleted.

Only the special heuristics described thus far were necessary to solve the first age problem. The second age problem, given below, requires one additional heuristic not previously mentioned. This is a substitution for the phrase "was when" which effectively decouples the two facts combined in the first sentence. For "was when", STUDENT substitutes "was K years ago . K years ago" where K is a new variable created for this purpose.

(THE PROBLEM TO BE SOLVED IS)
(MARY IS TWICE AS OLD AS ANN WAS WHEN MARY WAS AS OLD AS ANN IS NOW . IF MARY IS 24 YEARS OLD, HOW OLD IS ANN Q.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL X00008 ((ANN / PERSON) S AGE))

(EQUAL ((MARY / PERSON) S AGE) 24)

(EQUAL (PLUS ((MARY / PERSON) S AGE) (MINUS (X00007))) ((ANN / PERSON) S AGE))

(EQUAL ((MARY / PERSON) S AGE) (TIMES 2 (PLUS ((ANN / PERSON) S AGE) (MINUS (X00007)))))

(ANN S AGE IS 18)

In the example, the first sentence becomes the two sentences: "Mary is twice as old as Ann X00007 years ago. X00007 years ago Mary was as old as Ann is now." These two occurrences of time phrases are handled as discussed previously. Similarly the phrase "will be when" would be transformed to "in K years . In K years".

These decoupling heuristics are useful not only for the STUDENT program but for people trying to solve age problems. The classic age problem about Mary and Ann, given above, took an MIT graduate student over 5 minutes to solve because he did not know this heuristic. With the heuristic he was able to set up the appropriate equations much more rapidly. As a crude measure of STUDENT's relative speed, note that STUDENT took less than one minute to solve this problem.

G. When All Else Fails.

For all the problems discussed thus far, STUDENT was able to find a solution eventually. In some cases, however, necessary global information is missing from its store of information, or variables which name the same object cannot be identified by the heuristics of the program. Whenever STUDENT cannot find a solution for any reason, it turns to the questioner for help. As in the problem below, it prints out "(DO YOU KNOW ANY MORE RELATIONSHIPS BETWEEN THESE VARIABLES)" followed by a list of the variables in the problem. The questioner can answer "yes" or "no". If he says "yes", STUDENT says "TELL ME", and the questioner can append another sentence to the statement of the problem.

```
(THE PROBLEM TO BE SOLVED IS)
(THE GROSS WEIGHT OF A SHIP IS 20000 TONS . IF ITS NET
WEIGHT IS 15000 TONS , WHAT IS THE WEIGHT OF THE SHIPS
CARGO Q.)

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

TRYING POSSIBLE IDIOMS

(DO YOU KNOW ANY MORE RELATIONSHIPS AMONG THESE VARIABLES)
(GROSS WEIGHT OF SHIP)
(TONS)
(ITS NET WEIGHT)
(WEIGHT OF SHIPS CARGO)

yes
TELL ME

(the weight of a ships cargo is the difference between
the gross weight and the net weight)

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((NET WEIGHT) IS EQUAL TO (ITS NET WEIGHT))

(ASSUMING THAT)
((GROSS WEIGHT) IS EQUAL TO (GROSS WEIGHT OF SHIP))

(THE WEIGHT OF THE SHIPS  CARGO IS 5000 TONS)
```

In this problem, the additional information typed in (in lower case letters) was sufficient to solve the problem. If it was not, the question would be repeated until the questioner said "no", or provides sufficient information for solution of the problem.

In the problem below, the solution to the set of equations involves solving a quadratic equation, which is beyond the mathematical ability of the present STUDENT system. Note that in this case STUDENT reports that the equations were unsolvable, not simply insufficient for solution. STUDENT still requests additional information from the questioner. In the example, the questioner says "no", and STUDENT states that "I CANT SOLVE THIS PROBLEM" and terminates.

```
(THE PROBLEM TO BE SOLVED IS)
(THE SQUARE OF THE DIFFERENCE BETWEEN THE NUMBER OF
APPLES AND THE NUMBER OF ORANGES ON THE TABLE IS EQUAL
TO 9 . IF THE NUMBER OF APPLES IS 7 , FIND THE NUMBER
OF ORANGES ON THE TABLE .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02515 (NUMBER OF ORANGES ON TABLE))

(EQUAL (NUMBER OF APPLES) 7)

(EQUAL (EXPT (PLUS (NUMBER OF APPLES) (MINUS (NUMBER
OF ORANGES ON TABLE))) 2) 9)

UNABLE TO SOLVE THIS SET OF EQUATIONS

TRYING POSSIBLE IDIOMS

(DO YOU KNOW ANY MORE RELATIONSHIPS AMONG THESE
VARIABLES)

(NUMBER OF APPLES)
(NUMBER OF ORANGES ON TABLE)

no

I CANT SOLVE THIS PROBLEM
```

H. Summary of the STUDENT Subset of English.

The subset of English understandable by STUDENT is built around a core of sentence and phrase formats, which can be transformed into expressions in the STUDENT deductive model. On this basic core is built a larger set of formats. Each of these are first transformed into a string built on formats in this basic set and then this string is transformed into an expression in the deductive model. For example, the format ($ IS EQUAL TO $) is changed to the basic format ($ IS $), and the phrase "IS CONSECUTIVE TO" is changed to "IS 1 PLUS". The constructions discussed earlier involving single object transitive verbs could have been handled this way, though for programming convenience they were not.

The complete list of the basic formats accepted by the present STUDENT system can be determined by examining (in the program listing in the Appendix) the rules from the one labeled OPFORM to the one labeled QSET. The METEOR rules of the STUDENT program precisely specify the acceptable formats, and their translations to the model, but I shall try to summarize the basic and extended formats here. Implicitly assumed in the syntax is that any operator appears only within one of the contexts specified in the table given in Chapter II, and only the operators given in the table appear. The listing of STUDENT starting at the rule labeled IDIOMS gives translations of additonal operators to those in the table.

The basic linguistic form which is transformed into an equation is one containing "is" as a copula. The phrases "is equal to" and "equals" are both changed to the copula "is". The auxiliary verbal constructions "is multiplied by", "is divided by" and "is increased by" are also acceptable as principal verbs in a sentence. As discussed in detail earlier, a sentence with no occurrence of "is" can have as a main verb a transitive verb immediately followed by a number. This number must be an element of the phrase which is the direct object of the verb, as in "Mary has three guppies". This type of transitive verb can also have a comparative structure as direct object, e.g.,"Mary has twice as many guppies as Tom has fish".

This completes the repertoire of declarative sentence formats. Any number of declarative sentences may be conjoined, with ",and" between each pair, to form a new (complex) declarative sentence. A declarative sentence (even a complex declarative) can be made a presupposition for a question by preceding it with "IF" and following it with a comma and the question.

Questions, that is, requests for information from STUDENT, will be understood if they match any of the patterns:

(WHAT ARE $ AND $) (WHAT IS $)

(FIND $ AND $) (FIND $)

(HOW MANY $ DO $ HAVE) (HOW MANY $ DOES $ HAVE)

(HOW MANY $1 IS $)

This completes the summary of the set of input formats presently understood by STUDENT. This set can be enlarged in two distinct ways. One is to enlarge the set of basic formats, using standard subroutines to aid in defining, for each new basic format, its interpretation in the deductive model. The other method of extending the range of STUDENT input is to define transformations from new input formats to previously understood basic or extension formats. In the next chapter we discuss how this latter type of extension can be performed at run time, using the STUDENT global information storage facility. A combination of English and METEOR elementary pattern elements can be used to define the input format and transformation.

Even if a story problem is stated within the subset of English acceptable to STUDENT, this is not a guarantee that this problem can be solved by STUDENT (assuming it to be solvable). Two phrases describing the object must be at worst only "slightly different" by the criteria prescribed earlier. Appropriate global information must be available to STUDENT, and the algebra involved must not exceed the abilities of the solver. However, though most algebra story problems found in the standard texts cannot be solved by STUDENT exactly as written, the author has usually been able to find some paraphrase of almost all such problems, which is solvable by STUDENT. Appendix D contains a fair sample of the range of problems that can be handled by the STUDENT system.

I. Limitations of the STUDENT Subset of English.

The techniques presented in this chapter are general and can be used to enable a computer program to accept and understand a fairly extensive subset of English for a fixed semantic base. However, the current STUDENT system is experimental and has a number of limitations.

STUDENT's interpretation of the input is based on format matching. If each format is used to express the meaning understood by STUDENT, no misinterpretation will occur. However, these formats occur in English discourse even in algebra story problems, in semantic contexts not consistent with STUDENT's interpretation of these formats. For example, a sentence matching the format "($, AND $)" is always interpreted by STUDENT as the conjunction of two declarative statements. Therefore, the sentence "Tom has 2 apples, 3 bananas, and 4 pears." would be incorrectly divided into the two "sentences"

"Tom has 2 apples, 3 bananas." and "4 pears."

Each of the operator words shown in Figure 4 must be used as an operator in the context as shown or a misinterpretation will result. For example, the phrase "the number of times I went to the movies" which should be interpreted as a variable string will be interpreted incorrectly as the product of the two variables "number of" and "I went to the movies", because "times" is always considered to be an operator. Similarly, in the current implementation of STUDENT, "of" is considered to be an operator if it is preceded by any number. However, the phrase "2 of the boys who passed" will be misinterpreted as the product of "2" and "the boys who passed".

These examples obviously do not constitute a complete list of misinterpretations and errors STUDENT will make, but it should give the reader an idea of limitations on the STUDENT subset of English. In principle, all of these restrictions could be removed. However, removing some of them would require only minor changes to the program, while others would require techniques not used in the current system.

For example, to correct the error in interpreting "2 of the boys who passed", one can simply check to see if the number before the "of" is less than 1, and if so, only then interpret "of" as an operator "times". However, a much more sophisticated grammar and parsing program would be necessary to distinguish different occurrences of the format "($, AND $)", and correctly extract simpler sentences from complex coordinate and subordinate sentences.

Because of limitations of the sort described above, and the fact that the STUDENT system currently occupies almost all of the computer memory, STUDENT serves principally as a demonstration of the power of the techniques utilized in its construction. However, I believe that on a larger computer one could use these techniques to construct a system of practical value which would communicate well with people in English over the limited range of material understood by the program.

CHAPTER V: STORAGE OF GLOBAL INFORMATION

This algebra problem-solving system contains two programs which process English input. One is the problem thus far discussed, STUDENT, which accepts the statement of an algebra story problem and attempts to find the solution to the particular problem. STUDENT does not store any information, nor "remember" anything from problem to problem. The information obtained by STUDENT is the local context of the question.

The other program is called REMEMBER and it processes and stores facts not specific to any one problem. These facts make up STUDENT's store of "global information" as opposed to "local information" specific to the problem. This information is accepted in a subset of English which overlaps but is different from the subset of English accepted by STUDENT. REMEMBER accepts statements in certain fixed formats, and for each format the information is stored in a way that makes it convenient for retrieval and use within the STUDENT program. Some information is stored by actually adding METEOR rules to the STUDENT program, and other information is stored on property lists of individual words, which are unique atoms in the LISP system.

The following are the formats currently understood by REMEMBER, and the processing and information storage techniques used for each one:

1. Format: P1 <u>EQUALS</u> P2

Example: DISTANCE EQUALS SPEED TIMES TIME

Processing: The sentence is transformed into an equation in the same way it is done in STUDENT. This equation is stored on the property lists of the atoms which are the first words in each variable. In the example, the equation

"(EQUAL (DISTANCE) (TIMES (SPEED) (TIME)))"

is stored on the property lists of "DISTANCE", "SPEED" and "TIME". If any one of these words appears as the initial word of a variable in a problem, and global equations are needed to solve this problem, this equation will be retrieved.

2. Format: P1 <u>IS AN OPERATOR OF LEVEL</u> K

Example: TIMES IS AN OPERATOR OF LEVEL 1

Processing: A dictionary entry for P1 is created, with subscripts of OP and K. For TIMES, the dictionary entry (TIMES / OP 1) is created. The dictionary entry for any word is placed on the property list of that word (atom), and is retrieved and used in place of any occurrence of that word in a problem.

3. Format: P1 <u>IS AN OPERATOR</u>

Example: OF IS AN OPERATOR

Processing: A dictionary entry is created for P1 with the subscript OP. The entry for OF is (OF/OP).

4. Format: P1 <u>IS A</u> P2

Example: BILL IS A PERSON

Processing: A dictionary entry is created for P1 with subscript P2. The entry for BILL is (BILL/PERSON).

5. Format: P1 <u>IS THE PLURAL OF</u> P2

Example: FEET IS THE PLURAL OF FOOT

Processing: P2 is stored on the property list of P1, after the flag SING; the word P1 is stored on the property list of P2 after the flag PLURAL. Thus FEET is stored after PLURAL on the property list of the atom FOOT.

6. Format: P1 <u>SOMETIMES MEANS</u> P2

Example: TWO NUMBERS SOMETIMES MEANS ONE NUMBER AND THE OTHER NUMBER.

Processing: The STUDENT program is modified so that an idiomatic substitution of P2 for P1 will be made in a problem if it is otherwise unsolvable. All such "possible idiomatic substitutions" are tried when necessary, with the last one entered being the first one tried. The STUDENT program is modified by the addition of four new METEOR rules. Since P1 and P2 are inserted as left and right halves of a METEOR rule, they need not contain only words, but can use the METEOR elementary patterns to specify a format change instead of just a phrase change. For the example shown, the rules added to the STUDENT program, as listed in Appendix B, are the rule labeled C02510, the rule following that one, the rule labeled G02511 and the rule following it.

7. Format: P1 <u>ALWAYS MEANS</u> P2

Example: ONE HALF ALWAYS MEANS 0.5

Processing: The program STUDENT is modified so that if P1 occurs, a mandatory substitution of P2 for P1 will be made in any problem. The <u>last</u> sentence in this format processed by REMEMBER will be the <u>first</u> mandatory substitution made. Thus "one always means 1" <u>followed</u> by "one half always means 0.5" will cause the desired substitutions to be made; if these sentences were reversed no occurrence of "one half" would ever be found since it would have been changed to "1 half", by mandatory substitution of 1 for one.

For each sentence in this format processed by REMEMBER, a new METEOR rule is added to the STUDENT program, immediately following the rule named IDIOMS. The format of the METEOR rule added is (* (P1) (P2) IDIOMS) where P1 and P2 are the strings in the sentence processed. Thus by using a combination of English and METEOR elementary patterns and reference numbers in P1 and P2, one can add a new format of sentence to the STUDENT repertoire. For example, the following statement was processed by REMEMBER to allow STUDENT to "understand" (properly transform) a sentence in which the main verb was "exceeds":

($ EXCEEDS $ BY $ ALWAYS MEANS 1 IS 5 MORE THAN 3)

This permanently extended the STUDENT input subset of English, while avoiding the necessity of actually editing and changing the STUDENT program.

The global information stored for STUDENT ranged from equations to format changes to plural forms. Again, the compatible use of the METEOR prototype notation and the use of the general list processing operations in LISP facilitated programming of processing, storage and retrieval of this wide range of information. In Appendix C is a listing of the global information currently embodied in the STUDENT system.

CHAPTER VI: SOLUTION OF SIMULTANEOUS EQUATIONS

This chapter contains a description of the LISP program used by STUDENT to solve sets of simultaneous equations. The definitions of the three top level functions SOLVE, SOLVER and SOLVE1 are shown in the figure at the end of this chapter. This description of these functions is essentially independent of a detailed knowledge of LISP, although occasional parenthetical comments will be directed to the more knowledgeable.

The top level function, SOLVE, is a function of three arguments. One, labeled EQT in the definition of SOLVE, is the set of equations to be solved. The argument labeled WANTED in the definition is a list of variables whose values are wanted. The third argument, labeled TERMS, is another list of variables which is disjoint from WANTED. SOLVE will find the value of any variable which is wanted in terms of any or all of the variables on the list TERMS. In use, the list TERMS is a list of units, such as pounds, or feet, which may appear in the answer.

The output of SOLVE is dependent on whether the set of equations given can be solved for the variables wanted. If no solution can be found because the solution involves nonlinear processes, SOLVE returns with the value UNSOLVABLE. If no solution is found because not enough equations are given, SOLVE returns with the value INSUFFICIENT. If however, a solution is found, SOLVE returns with a list of pairs. The first element of each pair is a variable, either on the wanted list, or a variable whose value was found while solving for the desired unknowns. The second element of each pair is an arithmetic expression (in the prefix notation shown in Figure 2), which contains only numbers and variables on the list TERMS. Thus, the answer found by SOLVE is an "association list" of variables, and their values in the proper terms.

For example, let us consider the set of seven simultaneous equations shown below, and suppose SOLVE were asked to solve this set of equations for x and z. These are given in infix notation for ease of reading.

(1) $x + w = 9$	(5) $x + 2y = 4$
(2) $x^2 - C = D$	(6) $y^2 - 3y + 2 = z$
(3) $C + 3D = 6$	(7) $4x - y = 7$
(4) $2C - D = 5$	

The list TERMS is empty, and thus the values must all be numbers. In this case SOLVE would return with the list of pairs "((y, 1)(x, 2)(z, 0))," which indicates that the values x = 2 and z = 0 satisfy this set of equations (or those members of this set which were used to determine the values). The value y = 1 was found during the solving process.

Most of the work of SOLVE is done by the function SOLVER. SOLVE transmits to SOLVER the list of WANTED variables, the list of TERMS, and a null association list (called ALIS) which is recursively built up to give the answer. The value of SOLVER is this association list of pairs, with the first element of each pair being a variable whose value has been found. The second element of each is an arithmetic expression which may contain any variable on the list TERMS (as was the case for the ALIS of SOLVE). However, it may also contain variables which are first elements of pairs later on the association list. If values for variables given by later pairs are substituted into this arithmetic expression, one gets the arithmetic expression given by SOLVE containing only variables on the list TERMS. In the example, SOLVER would return with the association list ((y, (4x-7)) (x,2) (z,0)) which gives y in terms of x. SOLVE makes the substitutions and simplification on the association list returned by SOLVER.

SOLVER is a program which solves for a list of wanted variables. It does this by choosing one of these variables, adding the others to the list of terms and calling SOLVE1 to solve for this one variable in terms of the other wanted variables and the original TERMS. If SOLVE1 succeeds in solving for this variable, SOLVER pairs this one variable with the expression found, puts this pair on the end of the ALIS, and using this substitution in every equation it tries to solve, attempts to solve for the remaining wanted variables. If there are no more, SOLVER is finished and returns the association list built up.

SOLVE1 solves for a single wanted variable by finding an equation containing this variable, after all substitutions of values for variables listed on the ALIS have been made. It then makes a list of all the other variables in the equation, and checks to see if there are any not on the list TERMS. If so it calls SOLVER to solve for these new variables in terms of the wanted variable and the variables in TERMS. If SOLVER is unsuccessful, SOLVE1 tries to find another equation containing the wanted variable, and repeats the process. If there is none, SOLVE1 has the value INSUFFICIENT. If SOLVER is successful, and values for these new variables are found, or if there were no new variables, SOLVE1 finally calls SOLVEQ which attempts to solve this equation for the wanted variable. If the equation is linear in <u>this</u> variable, SOLVEQ will be successful and give a solution. SOLVE1 will add a pair consisting of the wanted variable and this value to the end of ALIS, and return with this augmented ALIS as its value. If SOLVEQ is unsuccessful, SOLVE1 tries another equation, but then if no solution can be found SOLVE1 returns the value UNSOLVABLE.

This description has been a rather long-winded attempt to explain the one page of LISP program at the end of this chapter. To make it more specific, let us consider what happens when SOLVER tries to solve the set of equations below (the same ones shown earlier);

(1) $x + w = 9$ (5) $x + 2y = 4$
(2) $x^2 - C = D$ (6) $y^2 - 3y + 2 = z$
(3) $C + 3D = 5$ (7) $4x - y = 7$
(4) $2C - D = 5$

SOLVER is asked to solve for x and z. It asks SOLVE1 to solve for x in terms of z. SOLVE1 picks equation (1), finds that a new variable, w, has appeared and asks SOLVER to solve for w in terms of x and z. Since there is no other occurrence of w in this set, SOLVER is unsuccessful and SOLVE1 abandons equation (1), and goes to equation (2). Here it calls SOLVER to solve for the two new variables C and D in terms of x and z. In this case SOLVER is successful, using equations (3) and (4), but when these values are substituted in equation (2), SOLVEQ cannot solve for x because the equation is not linear in x.

SOLVE1 now abandons equation (2) and the results it obtained as subgoals for solving (2). It finds an occurrence of x again in (3). Again it calls on SOLVER, to solve for the new variable y in terms of x and z. SOLVER tries to use (6), but SOLVEQ cannot solve this equation for y. Using (7) SOLVER returns with an ALIS of ((y,(4x - 7))). Using this ALIS, substituting this value for y into (5), SOLVE1 calls on SOLVEQ to solve this equation for <u>x</u>, which it does, and finally SOLVE1 returns to SOLVER the ALIS ((y, (4x - 7)), (x,2)) which does give the value of <u>x</u> in terms of <u>z</u>. Having found <u>x</u> in terms of <u>z</u>, SOLVER will now call SOLVE1 to find the value of <u>z</u>. SOLVE1 finds an occurrence of <u>z</u> in equation (6), and after substitution of terms on the ALIS, SOLVEQ is able to solve this equation for <u>z</u>, because it is linear in <u>z</u>. Adding the pair (z,0) to the ALIS, SOLVE1 returns it to SOLVER, which passes on this ALIS ((y, (4x - 7)), (x,2), (z,0)) to SOLVE. SOLVE, using the function SUBORD, which <u>sub</u>stitutes in <u>order</u> pairs on an ALIS into an expression and simplifies, finally returns the ALIS ((y,1)(x,2)(z,0)).

This example shows the rather tortuous recursions that these functions use to solve a set of equations. Why should we use this type of solving program instead of a more straightforward matrix method? The principal reason is that, as shown, nonlinear equations may appear in the set. In this case, if appropriate values can be found from other equations which when substituted into this non-linear equation make it linear in the variable for which we want to solve, then SOLVE will find the value of this variable.

The method of operation of SOLVER requires that if <u>n</u> variables appear in any equation, and that equation is used, then at least n-1 other independent equations containing these variables must be in the set of equations, or the actual mechanics of solving will not be started. This eliminates much work if there are extraneous equations in the set which contain one or two of the wanted variables. However, it precludes solving a set of equations which is homogeneous in one unwanted variable, and would therefore cancel out in the solution process. This is the principal reason why problems such as:

"Spigot A fills a tub in 1 hour, and spigot B in 2 hours. How long do they take together?"

cannot be solved by STUDENT.

This solving subroutine set is an independent package in the STUDENT program. Therefore, improvements can be made to it without disturbing the rest of the processing. The routine described here was designed to handle most of the problems that can be found in first year algebra texts.

```
(SOLVE
   (LAMBDA (WANTED EQT TERMS ALIS) (PROG (A B)
            (SETQ A (SOLVER WANTED TERMS ALIS))
      START (COND
               ((NULL A) (RETURN B))
               ((NULL (CDR A)) (RETURN (CONS (CAR A) B)))
               ((ATOM A) (RETURN A)))
            (SETQ B (CONS (CONS (CAAR A) (SUBORD (CDAR A) (
CDR A))) B))
            (SETQ A (CDR A))
            (GO START))))

(SOLVER
   (LAMBDA (WANTED TERMS ALIS) (PROG (A B C D E G H J)
            (SETQ A WANTED)
            (SETQ J (QUOTE INSUFFICIENT))
      START (COND
               ((NULL A) (RETURN J)))
            (SETQ B (CAR A))
            (SETQ C (CDR A))
            (SETQ E (SOLVE1 B (APPEND C (APPEND D TERMS)) ALIS
))
            (COND
               ((ATOM E) (GO ON)))
            (SETQ H (NCONC D C))
            (COND
               ((NULL H) (RETURN E)))
            (SETQ E (SOLVER H TERMS E))
            (COND
               ((NOT (ATOM E)) (RETURN E)))
      ON    (COND
               ((EQ E (QUOTE UNSOLVABLE)) (SETQ J E)))
            (SETQ D (CONS B D))
            (SETQ A C)
            (GO START))))

(SOLVE1
   (LAMBDA (X TERMS ALIS) (PROG (A B C D E G H J)
            (SETQ A EQT)
            (SETQ J (QUOTE INSUFFICIENT))
      START (COND
               ((NULL A) (RETURN J)))
            (SETQ B (CAR A))
            (SETQ C (SUBORD B ALIS))
            (SETQ D (VARTERMS C))
            (COND
               ((MEMBER X D) (GO ON)))
      B     (SETQ E (CONS B E))
            (SETQ A (CDR A))
            (GO START)
      ON    (SETQ G (CONS X TERMS))
            (SETQ H (LOGMINUS D G))
            (SETQ EQT (EFFACE B EQT))
            (COND
               ((NULL H) (GO SOLVEQ)))
            (SETQ G (SOLVER H G ALIS))
            (COND
               ((ATOM G) (GO D)))
            (SETQ ALIS G)
            (SETQ C (SUBORD B ALIS))
      SOLVEQ (SETQ G (SOLVEQ X C))
            (COND
               ((ATOM G) (GO B)))
            (RETURN (APPEND ALIS (LIST
               G)))
      D     (COND
               ((EQ G (QUOTE UNSOLVABLE)) (SETQ J G)))
            (SETQ EQT (APPEND E A))
            (GO B))))
```

Figure 5: The SOLVE Program in STUDENT

CHAPTER VII: CONCLUSION

A. Results.

The purpose of the research reported here was to develop techniques which facilitate natural language communication with a computer. A semantic theory of coherent discourse was proposed as a basis for the design and understanding of such man-machine systems. This theory was only outlined, and much additional work remains to be done. However, in its present rough form, the theory served as a guide for construction of the STUDENT system, which can communicate in a limited subset of English.

The language analysis in STUDENT is an implementation of the analytic portion of this theory. The STUDENT system has a very narrow semantic base. From the theory it is clear that by utilizing this knowledge of the limited range of meaning of the input discourse, the parsing problem becomes greatly simplified, since the number of linguistic forms that must be recognized is very small. If a parsing system were based on any small semantic base, this same simplification would occur. This suggests that in a general language processor, some time might be spent putting the input into a semantic context before going ahead with the syntactic analysis.

The semantic base of the STUDENT language analysis is delimited by the characteristics of the problem solving system embedded in it. STUDENT is a question-answering system which answers questions posed in the context of "algebra story problems." In the introduction, we used four criteria for evaluating several question-answering systems. Let us compare the STUDENT system to these others in the light of these criteria.

1) Extent of Understanding. All the other question-answering systems discussed analyze input sentence by sentence. Although a representation of the meaning of all input sentences may be placed in some common store, no syntactic connection is ever made between sentences.

In the STUDENT system, an acceptable input is a sequence of sentences, such that these sentences cannot be understood by just finding the meanings of the individual sentences, ignoring their local context. Inter-sentence dependencies must be determined, and inter-sentence syntactic relationships must be used in this case for solution of the problem given. This extension of the syntactic dimension of understanding is important because such inter-sentence dependencies (e.g., the use of pronouns) are very commonly used in natural language communication.

The semantic model in the STUDENT system is based on one relationship (equality) and five basic arithmetic functions. Composition of these functions yield other functions which are also expressed as individual linguistic forms in the input language. The input language is richer in expressing functions than Lindsay's or Raphael's system. The logical systems discussed may have more relationships (predicates) allowable in the input, but do not allow any composition of these predicates. The logical combinations of predicates used are only those expressed in the input as logical combinations (using and, or, etc.).

The deductive system in STUDENT, as in Lindsay's and Raphael's programs, is designed for the type of questions to be asked. It can only deduce answers of a certain type from the input information, that is, arithmetic values satisfying a set of equations. In performing its deductions it is reasonably sophisticated in avoiding irrelevant information, as are the other two mentioned. It lacks the general power of a logical system, but is much more efficient in obtaining its particular class of deductions than would be a general deductive system utilizing the axioms of arithmetic.

2) Facility for Extending Abilities. Extending the syntactic abilities of any of the other question-answering systems discussed would require reprogramming. In the STUDENT system new definitional transformations can be introduced at run time without any reprogramming. The information concerning these transformations can be input in English, or in a combination of English and METEOR, if that is more appropriate. New syntactic transformations must be added by extending the program.

The semantic base of the STUDENT system can be extended only by adding new program, as is true of the other question-answering systems discussed. However STUDENT is organized to facilitate such extensions, by minimizing the interactions of different parts of the program. The necessary information need only be added to the program equivalent of the table of operators in Figure 4, in Chapter IV.

Similarly, the deductive portion of STUDENT, which solves the derived set of equations, is an independent package. Therefore, a new extended solver can be added to the system by just replacing the package, and maintaining the input-output characteristics of this subroutine.

3) Knowledge of Internal Structure Needed by User. Very little if any internal knowledge of the workings of the STUDENT system need be known by the user. He must have a firm grasp of the

type of problem that STUDENT can solve, and a knowledge of the input grammar. For example, he must be aware that the same phrase must always be used to represent the same variable in a problem, within the limits of similarity defined earlier. He must realize that even within these limits STUDENT will not recognize more than one variation on a phrase. But if the user does forget any of these facts, he can still use the system, for the interaction discussed in the next section allows him to make amends for almost any mistake.

4) Interaction With the User. The STUDENT system is embedded in a time-sharing environment (the MIT Project MAC time-sharing system (13)), and this greatly facilitates interaction with the user. STUDENT differentiates between its failure to solve a problem because of its mathematical limitations and failure from lack of sufficient information. In case of failure it asks the user for additional information, and suggests the nature of the needed information (relationships among variables of the problem). It can go back to the user repeatedly for information until it has enough to solve the problem, or until the user gives up.

STUDENT also reports when it does not recognize the format of an input sentence. Using this information as a guide, the user is in a teaching-machine type situation, and can quickly learn to speak STUDENT's brand of input English. By monitoring the assumptions that STUDENT makes about the input, and the global information it uses, the user can stop the system and reword a problem to avoid an unwanted ambiguity, or add new general information to the global information store.

The crucial point in this user interaction is that STUDENT is embedded in an on-line time-sharing system, and can thus provide more interaction than any of the other systems mentioned.

B. Extensions.

The present STUDENT system has reached the maximum size allowable in the LISP system on a thirty-two thousand word IBM 7094. Therefore, very little can be added directly to the present system. All the programming extensions mentioned here are predicated on the existence of a much larger memory machine.

Without inventing any new techniques, I think that the STUDENT system could be made to understand most of the algebra story problems that appear in first year high school text books. If new operators, new combinations of arithmetic operations occur, they can easily be added to OPFORM, the subroutine which maps the kernel English sentences into equations. The number of formats recognizable in the system can be increased without reprogramming through the machinery available for storing global information (this was discussed in more detail in Chapter V). The problems it would not handle are those having excessive verbiage or implied information about the world not expressible in a single sentence.

As mentioned earlier, the system can now make use of any given schema only once in solving a problem. This is because the schema equation is added to the set of equations to be solved, and the variables in the schema only identified with one other set of variables appearing in the problem. For example, if "distance equals speed times time" were the schema, then "distance", as a variable in the schema might be set equal to "distance traveled by train" or "distance traveled by plane", but not both in the same problem. This problem could be resolved by not adding the schema equation directly to the set of equations to be solved, but by looking for consistent sets of variables to identify with the schema variables. Then STUDENT could add an instance of the schema equations, with the appropriate substitutions, for each consistent set of variables found which are "similar" to the schema variables.

At the moment the solving subroutine of STUDENT can only perform linear operations on literal equations, and substitutions of numbers in polynomials and exponentials. It would be relatively easy to add the facility for solving quadratic or even higher order solvable equations. One could even add, quite easily, sufficient mechanisms to allow the solver to perform the differentiation needed to do related rate problems in the differential calculus.

The semantic base of the STUDENT system could be expanded. In order to add the relations recognized by the SIR system of Raphael, for example, one would have to add on the lowest level of the STUDENT program the set of kernel sentences understood in SIR, their mapping to the SIR model, and the question-answering routine to retrieve facts. Then the apparatus of the STUDENT system would process much more complicated input statements for the SIR model. One serious problem which arises when the semantic base is extended is based on the fact that one kernel may have an interpretation in terms of two different semantic bases. For example, "Tom has 3 fish." can be interpreted in both SIR and the present STUDENT system. To resolve this semantic ambiguity, the program can check the context of the ambiguous statement to see if there has been one consistent model into which all the other statements have been processed. If the latter condition does not determine a single preferred interpretation for the statement, then both interpretations can be stored.

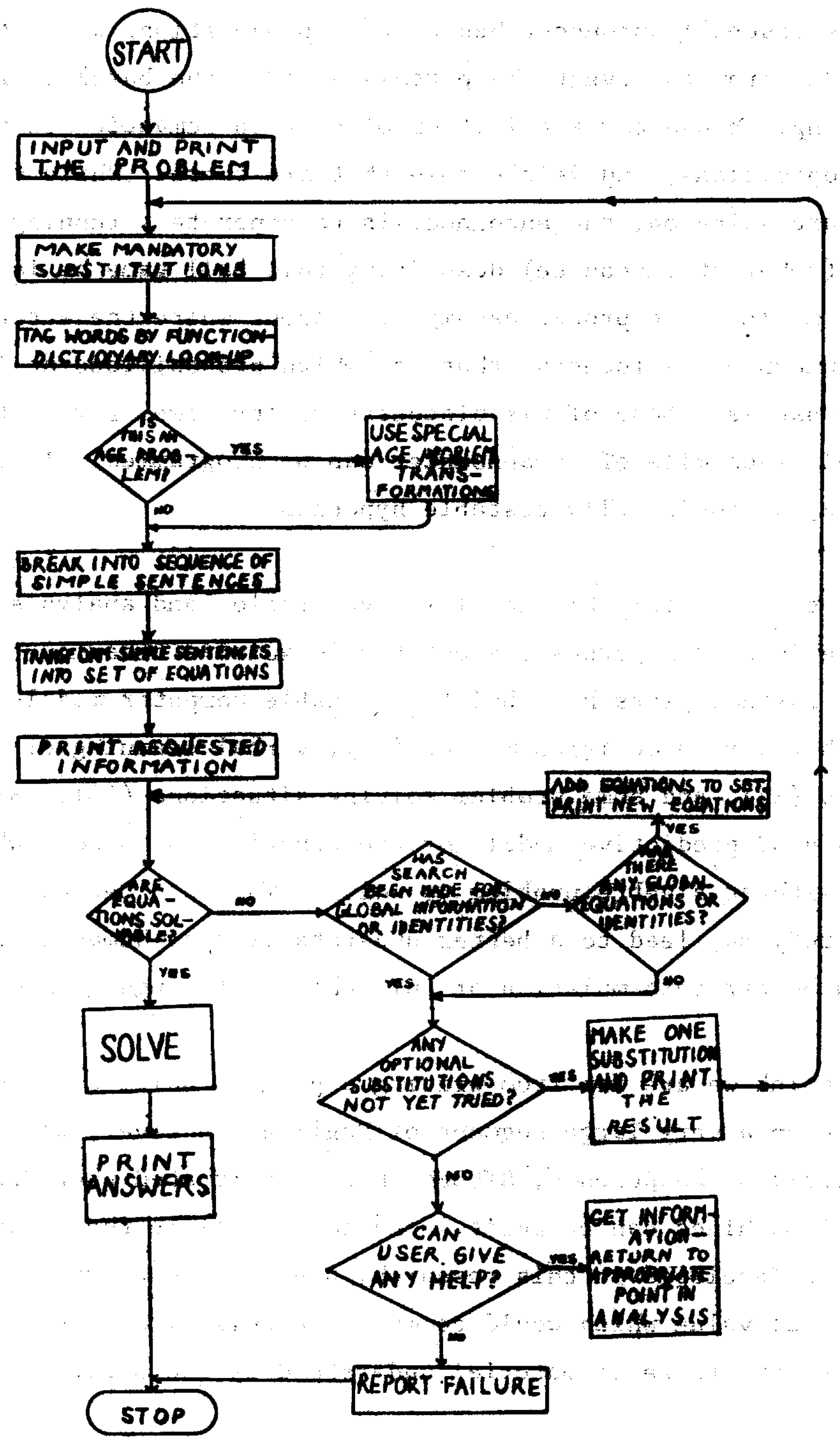
In addition to these immediate extensions of the STUDENT system, our semantic theory of discourse can be used as a basis for a much more general language processing system. As a start, one could implement the generative grammar described in Appendix E to produce coherent discourse—problems solvable by the STUDENT system.

Another more exciting possibility is to utilize this type of speaker's model of the world to attack Yngve's "baseball announcer" problem. The baseball announcer has certain propositions added to his world model from the events he perceives, i.e. the baseball game he is watching. Mandatory application of certain semantic rules add other propositions, and delete some that are there. While these changes are going on, the announcer is to generate a running commentary (coherent discourse) describing this ball game he is watching. By making the proper assumptions about where the attention of the announcer is focused, that is, which propositions he is going to use as a base of his discourse at any time, I feel that a reasonable facsimile of an announcer can be programmed. This is, of course, an empirically testable hypothesis.

Another use for this model for generation and analysis of discourse is as a hypothesis about the linguistic behaviour of people. Psychologists have built reasonable computer models for human behaviour in decision making (17), verbal learning of nonsense syllables (15), and some problem solving situations (34). STUDENT may be a good predictive model for the behaviour of people when confronted with an algebra problem to solve. This can be tested, and such a study may lead to a better understanding of human behaviour, and/or a better reformulation of this theory of language processing.

I think we are far from writing a program which can understand all, or even a very large segment of English. However, within its narrow field of competence, STUDENT has demonstrated that "understanding" machines can be built. Indeed, I believe that using the techniques developed in this research, one could construct a system of practical value which would communicate well with people in English over the range of material understood by the program.

APPENDIX A: FLOWCHART OF THE STUDENT PROGRAM



APPENDIX B: LISTING OF THE STUDENT PROGRAM

1) Definition of STUDENT

```
((STUDENT   ($)   (/ (*S ORGPRB 1))                              *)
(*      ($)   (1 (FN TERPRI) (FN TERPRI) (FN TERPRI))            *)
(*      ((*P THE PROBLEM TO BE SOLVED IS))                       *)
(IDIOMS ($)                                                      *)
(*      (HOW OLD)   (WHAT)                                  IDIOMS)
(*      (IS EQUAL TO)   (IS)                                IDIOMS)
(*      (YEARS YOUNGER THAN)   (LESS THAN)                  IDIOMS)
(*      (YEARS OLDER THAN)   (PLUS)                         IDIOMS)
(*      (PERCENT LESS THAN)   (PERLESS)                     IDIOMS)
(*      (LESS THAN)   (LESSTHAN)                            IDIOMS)
(*      (THESE)   (THE)                                     IDIOMS)
(*      (MORE THAN)   (PLUS)                                IDIOMS)
(*      (FIRST TWO NUMBERS)   (THE FIRST NUMBER AND THE
           SECOND NUMBER)                                   IDIOMS)
(*      (THREE NUMBERS)   (THE FIRST NUMBER AND THE SECOND
           NUMBER AND THE THIRD NUMBER)                     IDIOMS)
(*      (ONE HALF)   ( .5000)                               IDIOMS)
(*      (TWICE)   (2 TIMES)                                 IDIOMS)
(*      (TWO NUMBERS)                                       IDIOMS)
(*      ((* DOLLARS $1)   (2 DOLLARS)                       IDIOMS)
(*      (CONSECUTIVE TO)   ((QUOTE 1) PLUS)                 IDIOMS)
(*      (LARGER THAN)   (PLUS)                              IDIOMS)
(*      (PER CENT)   (PERCENT)                              IDIOMS)
(*      (HOW MANY)   (HOWM)                                 IDIOMS)
(*      (SQUARE OF)   (SQUARE)                              IDIOMS)
(*      (($.1) MULTIPLIED BY)   (TIMES)                     IDIOMS)
(*      (($.1) DIVIDED BY)   (DIVBY)                        IDIOMS)
(*      (THE SUM OF)   (SUM)                                IDIOMS)
(*      ($)   ((*S NONID 1))                                     *)
(WORDS  ($)   ((*Q SHELF (FN GETDCT 1 DICT)))
(*      ($)   WORDS)
(*      ($)   (SHELF))
(THE    (THE NUMBER)   (1)                                     THE)
(*      ($)   (/ (*S MARKWD 1))                                  *)
(*      (AS OLD AS)                                        AGEPROB)
(*      (AGE)                                              AGEPROB)
(*      (WILL BE)                                          AGEPROB)
(*      ($)   ((/ (*D RETURN SENTENCE))                    BRACKET)
(SENTENCE ($)                                                    *)
(*      ($1)   (/ (*S FIND (*E 1)) (*D RETURN SENTENCE
                                                           OPFORM)
(QUIET  ($)                                                      *)
(SUBSTITUTIONS  ($)   ((FN TERPRI) (*A NONID))                   *)
(*      ((*P WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS))
                                                                 *)
(TAGGING      ($)   ((FN TERPRI) (*A MARKWD))                    *)
(*      ((*P WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
        )                                                        *)
```

```
(BRACKETING ($)   ((FN TERPRI) (*A SIMSEN))                      *)
(*      ((FN PRINLIS ((THE SIMPLE SENTENCES ARE))))              *)
(EQUATIONS  ($)   ((FN TERPRI) (*A SHELF))                       *)
(*      ((FN PRINEIS ((THE EQUATIONS TO BE SOLVED ARE))))
                                                                 *)
(*      ($)   (/ (*S SHELF 1))                                   *)
(SOLUTION    ($)   ((FN TERPRI))                                 *)
(OPRN   ($)   ((FN REMDUP (*K (*A SHELF))))                      *)
(*      ($)   ((*A WANTED))   (/ (*S SHELF 1) (*S EQT 1))
(*      ($)   ((FN REMDUP (*K (*A UNITS))))   (/ (*S
           WANTED 1) (*S WANT 1))                                *)
(*      ($)   ((FN REMDUP (*K (*A AUNITS))))   (/ (*S
           UNITS 1) (*S UNIT 1))                                 *)
(*      ($)   (/ (*S AUNIT 1))                                   *)
(*      ($1 $)   ((FN SOLVE (*K (*A WANTED)) (*K (*A SHELF
           )) (*K 1 2)) (*K)))                                 OUT)
(*      ($1)   ((FN SOLVE (*K (*A WANTED)) (*K (*A SHELF))
           (*K (*A UNITS)) (*K)))                              OUT)
(OUT    ($)                                                      *)
(ANSWER (UNSOLVABLE)   ((*W UNABLE TO SOLVE THIS SET OF
              EQUATIONS $EOR$) (FN TERPRI))                 SIMVAR)
(*      (INSUFFICIENT)   ((*W THE EQUATIONS
              WERE INSUFFICIENT TO FIND A SOLUTION $EOR$)
           (FN TERPRI))                                     SIMVAR)
(*      ($1)   ((FN PRINTQ (*K (*A ANS)) (*K 1)))                *)
(*      (INCOMPLETE)   (UNSOLVABLE)                         ANSWER)
(*      ($1)   ((THE PROBLEM IS SOLVED))                       END)
(BRACKET ($1)   [illegible]                                    OUT)
(*      ($ $1 / DLM)   (/ (*Q PROBLEM (*K 1 2)))                 *)
(*      (2)   BRACKET)
(*      ($)   (0 (*Q PROBLEM (*K 1 PERIOD / DLM))))
(*      (NO END)                                                 *)
(BKT    ($)   ((/ (*S SHELF (*A PROBLEM)) (*D BKEND BKT)))
(BKT1   ($0)   ((FN SHELF)))                                     *)
(*      ($1)   ((*E 1))                                    PROBLEM)
(*      ($F) $ (* COMMA) $1 / QWORD) ($) [illegible] (*Q SHELF
           (*K 2 (PERIOD / DLM))) (*K 4 5)) (*D BKEND
           BKT))                                              BKT1)
(*      ($ (* COMMA) AND $))   (/ (*S SHELF (*K 1 ((
           PERIOD / DLM))) (*K 4)) (*D BKEND BKT))            BKT2)
(*      ($)   ((/ (*Q PROBLEM (*K 1)))                        BKT1)
(PBKT   ($)   ((*A PROBLEM))                                     *)
(*      ($)   (0 (/ (*S PROBLEM 1) (*S SIMSEN 1)))          RETURN
        )
```

```
(CRY     ($)   ((FN TERPRI) (*W I CANT SOLVE  THIS PROBLEM
               $EOR$))                                        END)
(GETEXP ($1)   0   (/ (*Q WORDS (FN GETDCT 1 DICT)))
               GETEXP)
(*       ($)    ((FN TERPRI) (*A WORDS))                        *)
(*       ($ IS $)   ((*K EQUAL (FN OPFORM (*K 1)) (FN
               OPFORM (*K 3))))   (/ (*D SIMVAR SIMVAR))
               EQTFN)
(*       ((*P I DONT UNDERSTAND THIS))                       CRY)
(IDTABLE ($)                                                    *)
(G02514 (THE PERIMETER OF $1 RECTANGLE)   (TWICE THE SUM
               OF THE LENGTH AND WIDTH OF THE RECTANGLE)
               G02514)
(*       ($)                                                PNEW)
(G02512 (TWO NUMBERS)   (ONE OF THE NUMBERS AND THE OTHER
               NUMBER)                                    G02512)
(*       ($)                                                PNEW)
(G02510 (TWO NUMBERS)   (ONE NUMBER AND THE OTHER NUMBER)
                 G02510)
(*       ($)                                                PNEW)
(PNEW    ((*P THE PROBLEM WITH AN IDIOMATIC SUBSTUTION IS))
               (2 1 2*)                                  IDIOMS)
(*       ($)    ((*K (*A ORGPRB)) 1 (FN NLSHLF) (FN TERPRI))
               *)                                        IDIOMS)
(*       ($1 $) (2)   (/ (*S ORGPRB (*E 1)))             IDIOMS)
(AGEPROB (AS OLD AS)   0                                AGEPROB)
(YEARSOLD (YEARS OLD)   0                              YEARSOLD)
(WHENFUT (WILL BE WHEN)   (WH* IN (FN GENSYM) YEARS
               PERIOD / DLM))                          WHENBOT)
(*  *    (WAS WHEN)   (WH* (FN GENSYM) YEARS AGO ((PERIOD /
               DLM)))                                  WHENBOT)
(WHENBOT    (WH* $ (PERIOD / DLM))   (2 3 2)           WHENFUT)
(WHENGO  ((WE / PERSON) WILL BE IN $1 YEARS)   (1 SHALL BE
               SECOND NUMBER)                           WHENGO)
(WAS     (WAS)                                             WAS)
(WILLBE  (WILL BE)   (IS)                               WILLBE)
(ISNOW   ((ONE / PERSON) IS NOW)   (1 S AGE NOW)         ISNOW)
(FROMNOW ($1 YEARS FROM NOW)   (IN 1 2)                FROMNOW)
(TOP     ($ ($1 / PERSON))   (MM)   (/ (*Q PROBLEM 1 2))     *)
(*       (MM $)    0   (/ (*Q PROBLEM 2))                  TOP)
(*       (MM)   0   (/ (*Q PROBLEM S AGE))                 TOP)
(*       ($)    ((*A PROBLEM) 1)                             *)
(*       (($1 / PERSON) $ S AGE)   (/ (*Q SUBJECT 1 2) (*Q
               SUBJECTS 1 2 3))                              *)
(SUBPRO  *   ($ ($1 / PRO) $)   ((*A SUBJECT))   (/ (*Q
               PROBLEM 1) (*Q REST 3))                    PPRO)
(*       ($)    ((*A REST))   (/ (*Q PROBLEM 1 S AGE) (*S
               SUBJECT 1))                              SUBPRO)
(PPRO    ($)    ((*A PROBLEM) 1)                             *)
(POSSPRO   *   ($ ($1 / POSSPRO) $)   ((*A SUBJECTS))   (/ (
               *Q PROBLEM 1) (*Q REST 3))               ENDPRO)
(*       ($)    ((*A REST))   (/ (*Q PROBLEM 1) (*S SUBJECTS
               1)))                                    POSSPRO)
(ENDPRO  ($)    ((*A PROBLEM) 1)                             *)
(GETAGE  ($ ($1 / PERSON) $ AGE)   0   (/ (*Q PROBLEM 1 2 3
               4) (*Q AGES (*K 2 3 4)))                 GETAGE)
(*       ($)    ((*A PROBLEM) 1)                             *)
(TAGE    ($ THEIR AGES $)   ((FN REMDUP (*K (*A AGES))))
               (/ (*Q LEFT 1) (*Q RIGHT 4))             THRAGE)
(*       ($)    (/ (*D RETURN AGESEN))                 BRACKET)
(AGESEN  ($)    ((*N PROBLEM))                               *)
(*  *    ($1)   (M* (*E 1))                            AGEDONE)
(*       (M* IN $1 YEARS)   0   (/ (*S AGEOP 2 3 4))   SETOP
               )
(*  *    (M* $1 YEARS AGO)   0   (/ (*S AGEOP 2 3 4))
               GETNEXT)
(SETOP   *   (A AGE)   (PP*)   (/ (*Q TEMP 1 2))         OPEND)
(*       (PP* IN $1 YEARS)   0   (/ (*Q TEMP 2 3 4))   SETOP
               )
(*       (PP* $1 YEARS AGO)   0   (/ (*Q TEMP 2 3 4))
               SETOP)
(*       (PP* NOW)   0                                   SETOP)
(*       (PP* $)   ((*A AGEOP))   (/ (*Q REST 2))            *)
(*       ($)    ((*A REST))   (/ (*Q TEMP 1) (*S AGEOP 1))
               SETOP)
(GETNEXT  (M*)   0                                       FUTOP)
(OPEND   ($)    ((*A TEMP) 1)   (/ (*S GARBAGE (*A AGEOP)))
               *)
(FUTOP   (IN $1 YEARS)   ((*K PLUSS / OP) 2)             FUTOP)
(PASTOP  ($1 YEARS AGO)   ((*K MINUSS / OP) 1)          PASTOP)
(*       (IS ($1 / DLM))   (2)                               *)
(*       ($)    0   (/ (*Q FIND (*K 1)))                AGESEN)
(AGEDONE    ($)   0   (/ (*S PROBLEM (*A FIND)))    SENTENCE
               )
(THRAGE  ($1)   0   (/ (*Q MIDDLE AND (*E 1)) (*Q AGES 1))
               THRAGE)
(*       ($)    ((*N MIDDLE))                                *)
(*       ($)    ((*A LEFT) (*A MIDDLE) (*A RIGHT))        TAGE)
)
```

```
(OPFORM ($)   ((*A FIND))                                     *)
(*   *   ($1)                                            RETURN)
(*       (WHAT ARE $ AND $ QMARK)   ((FN GENSYM) 3)     (/ (
             *S FIND 1 2 5 6))                             QSET)
(*       (WHAT ARE $ QMARK)   ((FN GENSYM) 3)              QSET)
(*       (WHAT IS $ QMARK)   ((FN GENSYM) 3)               QSET)
(*       (HOWM $1 IS $)   ((FN GENSYM) 4)   (/ (*S AUNITS (
             *K 2)))                                       QSET)
(*       (HOWM $ DO $ HAVE QMARK)   ((FN GENSYM) THE NUMBER
             OF 2 4 HAS)                                   QSET)
(*       ((HOW / QWORD) $ DOES $ HAVE QMARK)   ((FN GENSYM)
             THE NUMBER OF 2 4 HAVE)                       QSET)
(*       (FIND $ AND $)   ((FN GENSYM) 2)   (/ (*S FIND 1 4
             ))                                            QSET)
(*       (FIND $ ($1 / DLM))   ((FN GENSYM) 2)             QSET)
(*       ($)   ((FN REMART (*K 1)))                           *)
(*       ($ IS MULTIPLIED BY $)   0   (/ (*S REF (*K TIMES
             (FN OPFORM (*K 1)) (FN OPFORM (*K 5)))))
             RETURN)
(*       ($ IS DIVIDED BY $)   0   (/ (*S REF (*K QUOTIENT
             (FN OPFORM (*K 1)) (FN OPFORM (*K 5)))))
             RETURN)
(*       ($ IS INCREASED BY $)   0   (/ (*S REF (*K PLUS (
             FN OPFORM (*K 1)) (FN OPFORM (*K 5)))))
             RETURN)
(*       ($ IS $)   0   (/ (*S SHELF (*K EQUAL (FN OPFORM (
             *K 1)) (FN OPFORM (*K 3)))))                RETURN)
(*       ($ ($1 / VERB) $ AS MANY $ AS $ ($1 / VERB) $ ($1
             / DLM))   0   (/ (*S SHELF (*K EQUAL (FN
             OPFORM (*K THE NUMBER OF 5 1 2)) (FN OPFORM (
             *K 3 THE NUMBER OF 10 8 9 11)))))           RETURN)
(*       ($ ($1 / VERB) (FN NMTEST) $1 $ ($1 / DLM))   0
             (/ (*S SHELF (*K EQUAL (FN OPFORM (*K THE
             NUMBER OF 4 1 2)) (FN OPFORM (*K 3 5 6)))))
             RETURN)
(*       ((*P THIS SENTENCE FORM IS NOT RECOGNIZED))          *)
(*       ($)   0                                         RETURN)
(QSET    ($1 $)   0   (/ (*S SHELF (*K EQUAL 1 (FN OPFORM (
             *K (FN REMART (*K 2)))))) (*Q WANTED 1) (*Q
             ANS 1 (*K (FN OPREM (*K 2)))))              OPFORM)
(SIMVAR ($)   ((FN GETEQNS (*K (*A VAR))))                    *)
(*       ($1 $)   0   (/ (*S VBL (*E 1)) (*S EQT1 2))         *)
(*       ($)   ((FN READUP (*K (*A EQT1))))                   *)
(*   *   ($1 $)   (/ (*S EQT1 1 2))                          SP)
```

```
(*       ((*P USING THE FOLLOWING KNOWN RELATIONSHIPS))       *)
(*       ($)   ((FN TERPRI))                                  *)
(SP      ($)   ((*N VBL))   (/ (*D SIMVAR SIM))               *)
(SIMP   *   ($1)   ((*E 1))                              SIMFIN)
(*       ($ ($1 / PRO) $)   (1 (* DOLLAR) 3)   (/ (*S VAR (
             *K 1 2 3)))                                   VTST)
(*       ($)   ((* DOLLAR) 1 (* DOLLAR))   (/ (*S VAR (*K 1
             )))                                              *)
(VTST    ($)   ((*K (*K * (*K 1) (QUOTE 0) END) (QUOTE ((*
             ($) END)))))                                     *)
(VTOP    ($)   ((*N VBL))   (/ (*S TEST 1 1))                 *)
(*   *   ($1)   ((FN METRIX2 (*N TEST) 1))   (/ (*S VART 1)
                                                         TSTEND)
(*       ($1 $)   ((*N TEST))                              VTOP)
(*       ($)   (EQUAL (*N VAR) (*N VART))                     *)
(*       (EQUAL $1 $1)   (2 IS EQUAL TO 3)   (/ (*S VART 3)
             (*S EQT1 (*K 1 2 3)))                            *)
(*       ((*P ASSUMING THAT))                                 *)
(*       ($)   ((FN TERPRI))                                  *)
(TSTEND ($)   ((*N VART))                                     *)
(*       ($1)   0   (/ (*S VBL 1))                       TSTEND)
(*       ($)   ((*A TEST))                                    *)
(*       ($)   ((*N VBL))                                  SIMP)
(SIMFIN ($)   ((*A EQT1))                                     *)
(EQTIN   ($1 $)   1   (/ (*S SHELF 1 2 (*A EQT)) (*S UNITS
             (*A UNIT)) (*S AUNITS (*A AUNIT)) (*S WANTED (
             *A WANT)))                                    OPRN)
(SIM     ($)   ((FN TERPRI) (*A ORGPRB))                      *)
(*       ($)   (1 (FN TRYING POSSIBLE IDIOMS $EOR$) (FN
             TERPRI))   (/ (*S ORGPRB 1) (*D SIMVAR SIMVAR)
             (*D PO PO))                                 IDTEMP)
(IDTEMP ($)                                                   *)
(*       (THE PERIMETER OF $1 RECTANGLE)   (/ (*D IDTEMP
             G02513))                                    G02514)
(G02513 ($)                                                   *)
(*       (TWO NUMBERS)   (/ (*D IDTEMP G02511))          G02512)
(G02511 ($)                                                   *)
(*       (TWO NUMBERS)   (/ (*D IDTEMP G02509))          G02510)
(G02509 ($)                                                   *)
(IDFIN   ($)   ((*A VAR))                                     *)
(*       ((FN PRINLIS ((DO YOU KNOW ANY MORE RELATIONSHIPS
             AMONG THESE VARIABLES))))                        *)
(*       ($)   ((FN RDFLX))   (/ (*S VAR 1))                  *)
(*       (YES)   ((*W TELL ME $EOR$) (FN TERPRI) (FN RDFLX)
             )                                           GETEXP)
```

2) Definition of the Function OPFORM

```
((OPFORM    ($)                                                    *)
(*       ($ ($1 / OP 2) $)   ((FN CAR (*K 2)))    (/ (*S
              LEFT (*K 1)) (*S RIGHT (*K 3)))                 OPTST)
(*       ($ ($1 / OP 1) $)   ((FN CAR (*K 2)))    (/ (*S
              LEFT (*K 1)) (*S RIGHT (*K 3)))                 OPTST)
(*       ($ ($1 / OP) $)   ((FN CAR (*K 2)))    (/ (*S LEFT
              (*K 1)) (*S RIGHT (*K 3)))                      OPTST)
(*       ($ ($1 / DUM))   (1)                                      *)
(*       ($)   ((FN REMART (*K 1)))                                *)
(*       ($1)                                                  NMTST)
(ARGERR  *   ((*P ARGUMENT ERROR))                               END)
(NMTST   (($1) (FN NMTEST))                                   ARGERR)
(*       ((QUOTE 1) $1)   (1 (FN GETDCT 2 PLURAL))                 *)
(*       ((FN NMTEST) $1 $)   ((*K TIMES 1 (FN OPFORM (*K 2
              3)))   (/ (*S UNITS (*K 2)))                       END)
(*       ((FN NMTEST))                                           END)
(*   *   ($ THIS $)   ((FN REF) MM 1 2 3)                        OUT)
(*       ($1 MM $)   (1)                                         END)
(*       (MM)   ((FN SHELF) 1)                                     *)
(*   *   ($1 MM $)   ((*E 1))   (/ (*S SHELF 1))                  MM)
(*       (EQUAL $2 $)   (2)                                      END)
(*       ($)   (ERROR)                                           END)
(MM      (MM THIS $)   ((*K 3))   (/ (*S VAR (*K 3)))            END)
(OUT     ($2 $)   ((*K 1 2))   (/ (*S VAR (*K 1 2)))             END)
(*       (($.ATOM))   ((*K 1))   (/ (*S VAR (*K 1)))             END)
(*       ($1)   ((*E 2))                                           *)
(*       ($0 $1 / $)   ((*K (*K 2 3 4)))   (/ (*S VAR (*K (
              *K 2 3 4)))))                                      END)
(GTDCT   ($1)   ((*Q SDICT (FN GETDCT 1 DICT)))
              GTDCT)
(*       ($)   ((*A SDICT))                                   OPFORM)
(OPTST   ($1 $)   (2)                                              *)
(OF      ($)   ((*EN LEFT))                                        *)
(*       (($.NUMBER) $0)                                        OFOK)
(*       ($)   ($ OF (*EN RIGHT))                             OPFORM)
(OFOK    ($)   ((*K TIMES (FN OPFORM (*K 1)) (FN OPFORM (*N
              RIGHT))))                                          END)
(DIVBY   ($)   ((*K QUOTIENT (FN OPFORM (*N LEFT)) (FN
              OPFORM (*N RIGHT))))                               END)
(**      ($)   ((*K EXPT (FN OPFORM (*N LEFT)) (FN OPFORM (
              *N RIGHT))))                                       END)
(PER     ($)   (1 (*EN RIGHT))                                     *)
(*       (PER (FN NMTEST))   (2)                              PERNUM)
(*       (PER)   ((QUOTE 1))                                       *)
(PERNUM ($)   ((*K QUOTIENT (FN OPFORM (*N LEFT)) (FN
              OPFORM (*N 1))))                                   END)
(LESSTHAN    ($)   ((*K PLUS (FN OPFORM (*N RIGHT)) (*K
              MINUS (FN OPFORM (*N LEFT)))))                     END)
(MINUSS ($)                                                    MINUS)
(MINUS   ($)   ((*EN LEFT))                                        *)
(*       ($1 $)   ((*K PLUS (FN OPFORM (*K 1 2)) (*K MINUS
              (FN OPFORM (*N RIGHT)))))                          END)
(*       ($)   ((*K MINUS (FN OPFORM (*N RIGHT))))               END)
(TIMES   ($)   ((*EN LEFT))                                        *)
(*       ($1)                                                   OFOK)
(*   *   ((*P INCORRECT USE OF TIMES))                           END)
(PERLESS     ($)   ((*EN LEFT) 1)                                  *)
(*       (($.NUMBER) PERLESS)   ((FN QUOTIENT (FN
              DIFFERENCE 100 1)  100))                          OFOK)
(*       ($)                                                  PERERR)
(PERCENT     ($)   ((*EN LEFT) 1)                                  *)
(*       (($.NUMBER) PERCENT)   ((FN QUOTIENT 1  100) (
              *EN RIGHT))                                     OPFORM)
(PERERR  *   ((*P INCORRECT USE OF PERCENT))                     END)
(SQUARE ($)   ((*EN LEFT))                                         *)
(*       ($)   ((*K EXPT (FN OPFORM (*N RIGHT)) (QUOTE 2)))
              END)
(PLUSS   ($)                                                    PLUS)
(PLUS    ($)   ((*K PLUS (FN OPFORM (*N LEFT)) (FN OPFORM (
              *N RIGHT))))                                       END)
(DIFFERENCE ($)   ((*EN RIGHT))                                    *)
(*       (BETWEEN $ AND $)   ((*K PLUS (FN OPFORM (*K 2)) (
              *K MINUS (FN OPFORM (*K 4)))))                     END)
(*   *   ((*P INCORRECT USE OF DIFFERENCE))                      END)
(SQUARED     ($)   ((*EN RIGHT))                                   *)
(*       ($)   ((*K EXPT (FN OPFORM (*N LEFT)) 2))               END)
(SUM     ($)   ((*EN LEFT))                                        *)
(*       ($)   ((*EN RIGHT))                                       *)
(*       ($ AND $ AND $)   ((*K PLUS (FN OPFORM (*K 1)) (FN
              OPFORM (*K (*K SUM / OP) 3 4 5))))                 END)
(*       ($ AND $)   ((*K PLUS (FN OPFORM (*K 1)) (FN
              OPFORM (*K 3))))                                   END)
(*   *   ((*P SUM USED WRONG))                                   END)
)
```

APPENDIX C: GLOBAL INFORMATION IN STUDENT

```
REMEMBER((
(PEOPLE IS THE PLURAL OF PERSON)
(FEET IS THE PLURAL OF FOOT)
(YARDS IS THE PLURAL OF YARD)
(FATHOMS IS THE PLURAL OF FATHOM)
(INCHES IS THE PLURAL OF INCH)
(SPANS IS THE PLURAL OF SPAN)
(ONE HALF ALWAYS MEANS  0.5  )
(THREE NUMBERS ALWAYS MEANS THE FIRST NUMBER AND THE SECOND
NUMBER AND THE THIRD NUMBER)
(FIRST TWO NUMBERS ALWAYS MEANS
THE FIRST NUMBER AND THE SECOND NUMBER)
(MORE THAN ALWAYS MEANS PLUS)
(THESE ALWAYS MEANS THE)
(TWO NUMBERS SOMETIMES MEANS ONE NUMBER AND THE
     OTHER NUMBER)
(TWO NUMBERS SOMETIMES MEANS ONE OF THE
       NUMBERS AND THE OTHER NUMBER)
(HAS IS A VERB)
(GETS IS A VERB)
(HAVE IS A VERB)
(LESS THAN ALWAYS MEANS LESSTHAN)
(LESSTHAN IS AN OPERATOR OF LEVEL 2)
(PERCENT IS AN OPERATOR OF LEVEL 2)
(PERCENT LESS THAN ALWAYS MEANS PERLESS)
(PERLESS IS AN OPERATOR OF LEVEL 2)
(PLUS IS AN OPERATOR OF LEVEL  2)
(SUM IS AN OPERATOR)
(TIMES IS AN OPERATOR OF LEVEL 1)
(SQUARE IS AN OPERATOR OF LEVEL 1)
(DIVBY IS AN OPERATOR OF LEVEL 1)
(OF IS AN OPERATOR)
(DIFFERENCE IS AN OPERATOR)
(SQUARED IS AN OPERATOR)
(MINUS IS AN OPERATOR OF LEVEL 2)
(PER IS AN OPERATOR)
(SQUARED IS AN OPERATOR)
(YEARS OLDER THAN ALWAYS MEANS PLUS)
(YEARS YOUNGER THAN ALWAYS MEANS LESS THAN)
(IS EQUAL TO ALWAYS MEANS IS)
(PLUSS IS AN OPERATOR)
(MINUSS IS AN OPERATOR)
(HOW OLD ALWAYS MEANS WHAT)
(THE PERIMETER OF $1 RECTANGLE SOMETIMES MEANS
TWICE THE SUM OF THE LENGTH AND  WIDTH OF THE RECTANGLE)
(GALLONS IS THE PLURAL OF GALLON)
(HOURS IS THE PLURAL OF HOUR)
(MARY IS A PERSON)
(ANN IS A PERSON)
(BILL IS A PERSON)
(A FATHER IS A PERSON)
(AN UNCLE IS A PERSON)
(POUNDS IS THE PLURAL OF POUND)
(WEIGHS IS A VERB)
))
REMEMBER ((
(DISTANCE EQUALS SPEED TIMES TIME)
(DISTANCE EQUALS GAS CONSUMPTION TIMES
NUMBER OF GALLONS OF GAS USED)
(1 FOOT EQUALS 12 INCHES)
(1 YARD EQUALS 3 FEET)
))
```

```
(THE PROBLEM TO BE SOLVED IS)
(IF THE NUMBER OF CUSTOMERS TOM GETS IS TWICE THE SQUARE OF
20 PER CENT OF THE NUMBER OF ADVERTISEMENTS HE RUNS , AND THE
NUMBER OF ADVERTISEMENTS HE RUNS IS 45 , WHAT IS THE NUMBER
OF CUSTOMERS TOM GETS Q.)

(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(IF THE NUMBER OF CUSTOMERS TOM GETS IS 2 TIMES THE SQUARE
20 PERCENT OF THE NUMBER OF ADVERTISEMENTS HE RUNS , AND THE
NUMBER OF ADVERTISEMENTS HE RUNS IS 45 , WHAT IS THE NUMBER
OF CUSTOMERS TOM GETS Q.)

(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
(IF THE NUMBER (OF / OP) CUSTOMERS TOM (GETS / VERB) IS 2 (
TIMES / OP 1) THE (SQUARE / OP 1) 20 (PERCENT / OP 2) (OF /
OP) THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS , AND
THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS IS 45 ,
(WHAT / QWORD) IS THE NUMBER (OF / OP) CUSTOMERS TOM (GETS
/ VERB) (QMARK / DLM))

(THE SIMPLE SENTENCES ARE)

(THE NUMBER (OF / OP) CUSTOMERS TOM (GETS / VERB) IS 2 (TIMES
/ OP 1) THE (SQUARE / OP 1) 20 (PERCENT / OP 2) (OF / OP) THE
NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS (PERIOD / DLM))

(THE NUMBER (OF / OP) ADVERTISEMENTS (HE / PRO) RUNS IS 45
(PERIOD / DLM))

((WHAT / QWORD) IS THE NUMBER (OF / OP) CUSTOMERS TOM (GETS
/ VERB) (QMARK / DLM))

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02515 (NUMBER OF CUSTOMERS TOM (GETS / VERB)))

(EQUAL (NUMBER OF ADVERTISEMENTS (HE / PRO) RUNS) 45)

(EQUAL (NUMBER OF CUSTOMERS TOM (GETS / VERB)) (TIMES 2 (EXPT
(TIMES  .2000 (NUMBER OF ADVERTISEMENTS (HE / PRO) RUNS)) 2)))

(THE NUMBER OF CUSTOMERS TOM GETS IS  162)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF LOIS SHARE OF SOME MONEY AND BOB S SHARE IS $  4.500
. LOIS SHARE IS TWICE BOB S . FIND BOB S AND LOIS SHARE .)

(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(SUM LOIS SHARE OF SOME MONEY AND BOB S SHARE IS  4.500 DOLLARS
. LOIS SHARE IS 2 TIMES BOB S . FIND BOB S AND LOIS SHARE .)

(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
((SUM / OP) LOIS SHARE (OF / OP) SOME MONEY AND BOB S SHARE
IS  4.500 DOLLARS (PERIOD / DLM) LOIS SHARE IS 2 (TIMES / OP
1) BOB S (PERIOD / DLM) (FIND / QWORD) BOB S AND LOIS SHARE
(PERIOD / DLM))

(THE SIMPLE SENTENCES ARE)

((SUM / OP) LOIS SHARE (OF / OP) SOME MONEY AND BOB S SHARE
IS  4.500 DOLLARS (PERIOD / DLM))

(LOIS SHARE IS 2 (TIMES / OP 1) BOB S (PERIOD / DLM))

((FIND / QWORD) BOB S AND LOIS SHARE (PERIOD / DLM))

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02519 (LOIS SHARE))

(EQUAL G02518 (BOB S))

(EQUAL (LOIS SHARE) (TIMES 2 (BOB S)))

(EQUAL (PLUS (LOIS SHARE OF SOME MONEY) (BOB S SHARE)) (TIMES
4.500 (DOLLARS)))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((BOB S) IS EQUAL TO (BOB S SHARE))

(ASSUMING THAT)
((LOIS SHARE) IS EQUAL TO (LOIS SHARE OF SOME MONEY))

(BOB S IS  1.500 DOLLARS)

(LOIS SHARE IS  3 DOLLARS)
```

```
(THE PROBLEM TO BE SOLVED IS)
(MARY IS TWICE AS OLD AS ANN WAS WHEN MARY WAS AS OLD AS ANN
IS NOW . IF MARY IS 24 YEARS OLD , HOW OLD IS ANN Q.)

(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(MARY IS 2 TIMES AS OLD AS ANN WAS WHEN MARY WAS AS OLD AS
ANN IS NOW . IF MARY IS 24 YEARS OLD , WHAT IS ANN Q.)

(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
((MARY / PERSON) IS 2 (TIMES / OP 1) AS OLD AS (ANN / PERSON)
WAS WHEN (MARY / PERSON) WAS AS OLD AS (ANN / PERSON) IS NOW
(PERIOD / DLM) IF (MARY / PERSON) IS 24 YEARS OLD , (WHAT /
QWORD) IS (ANN / PERSON) (QMARK / DLM))

(THE SIMPLE SENTENCES ARE)

((MARY / PERSON) S AGE IS 2 (TIMES / OP 1) (ANN / PERSON) S
AGE G02521 YEARS AGO (PERIOD / DLM))

(G02521 YEARS AGO (MARY / PERSON) S AGE IS (ANN / PERSON) S
AGE NOW (PERIOD / DLM))

((MARY / PERSON) S AGE IS 24 (PERIOD / DLM))

((WHAT / QWORD) IS (ANN / PERSON) S AGE (QMARK / DLM))

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02522 ((ANN / PERSON) S AGE))

(EQUAL ((MARY / PERSON) S AGE) 24)

(EQUAL (PLUS ((MARY / PERSON) S AGE) (MINUS (G02521))) ((ANN
/ PERSON) S AGE))

(EQUAL ((MARY / PERSON) S AGE) (TIMES 2 (PLUS ((ANN / PERSON)
S AGE) (MINUS (G02521)))))

(ANN S AGE IS  18)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF THE PERIMETER OF A RECTANGLE AND THE PERIMETER
OF A TRIANGLE IS 24 INCHES . IF THE PERIMETER OF THE RECTANGLE
IS TWICE THE PERIMETER OF THE TRIANGLE , WHAT IS THE PERIMETER
OF THE TRIANGLE Q.)

(WITH MANDATORY SUBSTITUTIONS THE PROBLEM IS)
(SUM THE PERIMETER OF A RECTANGLE AND THE PERIMETER OF A TRIANGLE
IS 24 INCHES . IF THE PERIMETER OF THE RECTANGLE IS 2 TIMES
THE PERIMETER OF THE TRIANGLE , WHAT IS THE PERIMETER OF THE
TRIANGLE Q.)

(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)
((SUM / OP) THE PERIMETER (OF / OP) A RECTANGLE AND THE PERIMETER
(OF / OP) A TRIANGLE IS 24 INCHES (PERIOD / DLM) IF THE PERIMETER
(OF / OP) THE RECTANGLE IS 2 (TIMES / OP 1) THE PERIMETER (
OF / OP) THE TRIANGLE , (WHAT / QWORD) IS THE PERIMETER (OF
/ OP) THE TRIANGLE (QMARK / DLM))

(THE SIMPLE SENTENCES ARE)

((SUM / OP) THE PERIMETER (OF / OP) A RECTANGLE AND THE PERIMETER
(OF / OP) A TRIANGLE IS 24 INCHES (PERIOD / DLM))

(THE PERIMETER (OF / OP) THE RECTANGLE IS 2 (TIMES / OP 1)
THE PERIMETER (OF / OP) THE TRIANGLE (PERIOD / DLM))

((WHAT / QWORD) IS THE PERIMETER (OF / OP) THE TRIANGLE (QMARK
/ DLM))

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02517 (PERIMETER OF TRIANGLE))

(EQUAL (PERIMETER OF RECTANGLE) (TIMES 2 (PERIMETER OF TRIANGLE)))

(EQUAL (PLUS (PERIMETER OF RECTANGLE) (PERIMETER OF TRIANGLE))
(TIMES 24 (INCHES)))

(THE PERIMETER OF THE TRIANGLE IS  8 INCHES)
```

(THE PROBLEM TO BE SOLVED IS)
(BILL IS ONE HALF OF HIS FATHER S AGE 4 YEARS AGO . IN 20 YEARS HE WILL BE 2 YEARS OLDER THAN HIS FATHER IS NOW . HOW OLD ARE BILL AND HIS FATHER Q.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02550 ((BILL / PERSON) S (FATHER / PERSON) S AGE))

(EQUAL G02549 ((BILL / PERSON) S AGE))

(EQUAL (PLUS ((BILL / PERSON) S AGE) 20) (PLUS 2 ((BILL / PERSON) S (FATHER / PERSON) S AGE)))

(EQUAL ((BILL / PERSON) S AGE) (TIMES .5000 (PLUS ((BILL / PERSON) S (FATHER / PERSON) S AGE) (MINUS 4))))

(BILL S AGE IS 14)

(BILL S FATHER S AGE IS 32)

(THE PROBLEM TO BE SOLVED IS)
(BILL S FATHER S UNCLE IS TWICE AS OLD AS BILL S FATHER . 2 YEARS FROM NOW BILL S FATHER WILL BE 3 TIMES AS OLD AS BILL . THE SUM OF THEIR AGES IS 92 . FIND BILL S AGE .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02533 ((BILL / PERSON) S AGE))

(EQUAL (PLUS ((BILL / PERSON) S (FATHER / PERSON) S (UNCLE / PERSON) S AGE) (PLUS ((BILL / PERSON) S (FATHER / PERSON) S AGE) ((BILL / PERSON) S AGE))) 92)

(EQUAL (PLUS ((BILL / PERSON) S (FATHER / PERSON) S AGE) 2) (TIMES 3 (PLUS ((BILL / PERSON) S AGE) 2)))

(EQUAL ((BILL / PERSON) S (FATHER / PERSON) S (UNCLE / PERSON) S AGE) (TIMES 2 ((BILL / PERSON) S (FATHER / PERSON) S AGE)))

(BILL S AGE IS 8)

(THE PROBLEM TO BE SOLVED IS)
(A NUMBER IS MULTIPLIED BY 6 . THIS PRODUCT IS INCREASED BY 44 . THIS RESULT IS 68 . FIND THE NUMBER .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02588 (NUMBER))

(EQUAL (PLUS (TIMES (NUMBER) 6) 44) 68)

(THE NUMBER IS 4)

(THE PROBLEM TO BE SOLVED IS)
(THE PRICE OF A RADIO IS 69.70 DOLLARS . IF THIS PRICE IS 15 PERCENT LESS THAN THE MARKED PRICE , FIND THE MARKED PRICE .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02515 (MARKED PRICE))

(EQUAL (PRICE OF RADIO) (TIMES .8500 (MARKED PRICE)))

(EQUAL (PRICE OF RADIO) (TIMES 69.70 (DOLLARS)))

(THE MARKED PRICE IS 82 DOLLARS)

(THE PROBLEM TO BE SOLVED IS)
(TOM HAS TWICE AS MANY FISH AS MARY HAS GUPPIES . IF MARY HAS 3 GUPPIES , WHAT IS THE NUMBER OF FISH TOM HAS Q.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02520 (NUMBER OF FISH TOM (HAS / VERB)))

(EQUAL (NUMBER OF GUPPIES (MARY / PERSON) (HAS / VERB)) 3)

(EQUAL (NUMBER OF FISH TOM (HAS / VERB)) (TIMES 2 (NUMBER OF GUPPIES (MARY / PERSON) (HAS / VERB))))

(THE NUMBER OF FISH TOM HAS IS 6)

```
(THE PROBLEM TO BE SOLVED IS)
(IF 1 SPAN EQUALS 9 INCHES , AND 1 FATHOM EQUALS 6 FEET , HOW
MANY SPANS EQUALS 1 FATHOM Q.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02529 (TIMES 1 (FATHOMS)))

(EQUAL (TIMES 1 (FATHOMS)) (TIMES 6 (FEET)))

(EQUAL (TIMES 1 (SPANS)) (TIMES 9 (INCHES)))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(USING THE FOLLOWING KNOWN RELATIONSHIPS)
((EQUAL (TIMES 1 (YARDS)) (TIMES 3 (FEET))) (EQUAL (TIMES 1
(FEET)) (TIMES 12 (INCHES))))

(1 FATHOM IS 8 SPANS)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE NUMBER OF SOLDIERS THE RUSSIANS HAVE IS ONE HALF OF THE
NUMBER OF GUNS THEY HAVE . THE NUMBER OF GUNS THEY HAVE IS
7000 . WHAT IS THE NUMBER OF SOLDIERS THEY HAVE Q.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02519 (NUMBER OF SOLDIERS (THEY / PRO) (HAVE / VERB)))

(EQUAL (NUMBER OF GUNS (THEY / PRO) (HAVE / VERB)) 7000)

(EQUAL (NUMBER OF SOLDIERS RUSSIANS (HAVE / VERB)) (TIMES .5000
(NUMBER OF GUNS (THEY / PRO) (HAVE / VERB))))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((NUMBER OF SOLDIERS (THEY / PRO) (HAVE / VERB)) IS EQUAL TO
(NUMBER OF SOLDIERS RUSSIANS (HAVE / VERB)))

(THE NUMBER OF SOLDIERS THEY HAVE IS 3500)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE RUSSIAN ARMY HAS 6 TIMES AS MANY RESERVES IN A UNIT AS
IT HAS UNIFORMED SOLDIERS . THE PAY FOR RESERVES EACH MONTH
IS 50 DOLLARS TIMES THE NUMBER OF RESERVES IN THE UNIT . AND
THE AMOUNT SPENT ON THE REGULAR ARMY EACH MONTH IS $ 150 TIMES
THE NUMBER OF UNIFORMED SOLDIERS . THE SUM OF THIS LATTER AMOUNT
AND THE PAY FOR RESERVES EACH MONTH EQUALS $ 45000 . FIND THE
NUMBER OF RESERVES IN A UNIT THE RUSSIAN ARMY HAS AND THE NUMBER
OF UNIFORMED SOLDIERS IT HAS .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02532 (NUMBER OF UNIFORMED SOLDIERS (IT / PRO) (HAS
/ VERB)))

(EQUAL G02531 (NUMBER OF RESERVES IN UNIT RUSSIAN ARMY (HAS
/ VERB)))

(EQUAL (PLUS (AMOUNT SPENT ON REGULAR ARMY EACH MONTH) (PAY
FOR RESERVES EACH MONTH)) (TIMES 45000 (DOLLARS)))

(EQUAL (AMOUNT SPENT ON REGULAR ARMY EACH MONTH) (TIMES (TIMES
150 (DOLLARS)) (NUMBER OF UNIFORMED SOLDIERS)))

(EQUAL (PAY FOR RESERVES EACH MONTH) (TIMES (TIMES 50 (DOLLARS))
(NUMBER OF RESERVES IN UNIT)))

(EQUAL (NUMBER OF RESERVES IN UNIT RUSSIAN ARMY (HAS / VERB))
(TIMES 6 (NUMBER OF UNIFORMED SOLDIERS (IT / PRO) (HAS / VERB))))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((NUMBER OF UNIFORMED SOLDIERS) IS EQUAL TO (NUMBER OF UNIFORMED
SOLDIERS (IT / PRO) (HAS / VERB)))

(ASSUMING THAT)
((NUMBER OF RESERVES IN UNIT) IS EQUAL TO (NUMBER OF RESERVES
IN UNIT RUSSIAN ARMY (HAS / VERB)))

(THE NUMBER OF RESERVES IN A UNIT THE RUSSIAN ARMY HAS IS  600)

(THE NUMBER OF UNIFORMED SOLDIERS IT HAS IS  100)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE NUMBER OF STUDENTS WHO PASSED THE ADMISSIONS TEST IS 10
PERCENT OF THE TOTAL NUMBER OF STUDENTS IN THE HIGH SCHOOL
. IF THE NUMBER OF SUCCESSFUL CANDIDATES IS 72 , WHAT IS THE
NUMBER OF STUDENTS IN THE HIGH SCHOOL Q.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02553 (NUMBER OF STUDENTS IN HIGH SCHOOL))

(EQUAL (NUMBER OF SUCCESSFUL CANDIDATES) 72)

(EQUAL (NUMBER OF STUDENTS WHO PASSED ADMISSIONS TEST) (TIMES
 .1000 (TOTAL NUMBER OF STUDENTS IN HIGH SCHOOL)))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((NUMBER OF STUDENTS IN HIGH SCHOOL) IS EQUAL TO (TOTAL NUMBER
OF STUDENTS IN HIGH SCHOOL))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION
TRYING POSSIBLE IDIOMS
```

```
(THE PROBLEM WITH AN IDIOMATIC SUBSTUTION IS)
(THE NUMBER OF STUDENTS WHO PASSED THE ADMISSIONS TEST IS 10
PERCENT OF THE TOTAL NUMBER OF STUDENTS IN THE HIGH SCHOOL
. IF THE NUMBER OF STUDENTS WHO PASSED THE ADMISSIONS TEST
IS 72 , WHAT IS THE NUMBER OF STUDENTS IN THE HIGH SCHOOL Q.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02554 (NUMBER OF STUDENTS IN HIGH SCHOOL))

(EQUAL (NUMBER OF STUDENTS WHO PASSED ADMISSIONS TEST) 72)

(EQUAL (NUMBER OF STUDENTS WHO PASSED ADMISSIONS TEST) (TIMES
 .1000 (TOTAL NUMBER OF STUDENTS IN HIGH SCHOOL)))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((NUMBER OF STUDENTS IN HIGH SCHOOL) IS EQUAL TO (TOTAL NUMBER
OF STUDENTS IN HIGH SCHOOL))

(THE NUMBER OF STUDENTS IN THE HIGH SCHOOL IS 720)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE DISTANCE FROM NEW YORK TO LOS ANGELES IS 3000 MILES .
IF THE AVERAGE SPEED OF A JET PLANE IS 600 MILES PER HOUR ,
FIND THE TIME IT TAKES TO TRAVEL FROM NEW YORK TO LOS ANGELES
BY JET .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02517 (TIME (IT / PRO) TAKES TO TRAVEL FROM NEW YORK
TO LOS ANGELES BY JET))

(EQUAL (AVERAGE SPEED OF JET PLANE) (QUOTIENT (TIMES 600 (MILES))
(TIMES 1 (HOURS))))

(EQUAL (DISTANCE FROM NEW YORK TO LOS ANGELES) (TIMES 3000
(MILES)))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(USING THE FOLLOWING KNOWN RELATIONSHIPS)
((EQUAL (DISTANCE) (TIMES (SPEED) (TIME))) (EQUAL (DISTANCE)
(TIMES (GAS CONSUMPTION) (NUMBER OF GALLONS OF GAS USED))))

(ASSUMING THAT)
((SPEED) IS EQUAL TO (AVERAGE SPEED OF JET PLANE))

(ASSUMING THAT)
((TIME) IS EQUAL TO (TIME (IT / PRO) TAKES TO TRAVEL FROM NEW
YORK TO LOS ANGELES BY JET))

(ASSUMING THAT)
((DISTANCE) IS EQUAL TO (DISTANCE FROM NEW YORK TO LOS ANGELES))

(THE TIME IT TAKES TO TRAVEL FROM NEW YORK TO LOS ANGELES BY
JET IS 5 HOURS)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE COST OF A BOX OF MIXED NUTS IS THE SUM OF THE COST OF
THE ALMONDS IN THE BOX AND THE COST OF THE PECANS IN THE BOX
, FOR A LARGE BOX THIS COST IS $ 3.500 . THE WEIGHT , IN POUNDS
, OF A BOX OF MIXED NUTS IS THE SUM OF THE NUMBER OF POUNDS
OF ALMONDS IN THE BOX AND THE NUMBER OF POUNDS OF PECANS IN
THE BOX . THIS LARGE BOX WEIGHS 3 POUNDS . THE COST OF ALMONDS
PER POUND OF ALMONDS IS $ 1 . AND THE COST OF PECANS PER POUND
OF PECANS IS $ 1.500 . FIND THE COST OF THE ALMONDS IN THE
BOX AND THE COST OF THE PECANS IN THE BOX .)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL G02538 (COST OF PECANS IN BOX))

(EQUAL G02537 (COST OF ALMONDS IN BOX))

(EQUAL (QUOTIENT (COST OF PECANS) (TIMES 1 (POUNDS OF PECANS)))
(TIMES 1.500 (DOLLARS)))

(EQUAL (QUOTIENT (COST OF ALMONDS) (TIMES 1 (POUNDS OF ALMONDS)))
(TIMES 1 (DOLLARS)))
(EQUAL (WEIGHT , IN POUNDS , OF BOX OF MIXED NUTS) (TIMES 3 (POUNDS)))

(EQUAL (WEIGHT , IN POUNDS , OF BOX OF MIXED NUTS) (PLUS (NUMBER
OF POUNDS OF ALMONDS IN BOX) (NUMBER OF POUNDS OF PECANS IN
BOX)))

(EQUAL (COST OF BOX OF MIXED NUTS) (TIMES 3.500 (DOLLARS)))

(EQUAL (COST OF BOX OF MIXED NUTS) (PLUS (COST OF ALMONDS IN
BOX) (COST OF PECANS IN BOX)))

(THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((POUNDS OF PECANS) IS EQUAL TO (NUMBER OF POUNDS OF PECANS
IN BOX))
(ASSUMING THAT)
((COST OF PECANS) IS EQUAL TO (COST OF PECANS IN BOX))
(ASSUMING THAT)
((POUNDS OF ALMONDS) IS EQUAL TO (NUMBER OF POUNDS OF ALMONDS
IN BOX))
(ASSUMING THAT)
((COST OF ALMONDS) IS EQUAL TO (COST OF ALMONDS IN BOX))

(THE COST OF THE ALMONDS IN THE BOX IS 2 DOLLARS)

(THE COST OF THE PECANS IN THE BOX IS 1.500 DOLLARS)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE GAS CONSUMPTION OF MY CAR IS 15 MILES PER GALLON . THE
DISTANCE BETWEEN BOSTON AND NEW YORK IS 250 MILES . WHAT IS
THE NUMBER OF GALLONS OF GAS USED ON A TRIP BETWEEN NEW YORK
AND BOSTON Q.)

(THE EQUATIONS TO BE SOLVED ARE)
(EQUAL G02556 (NUMBER OF GALLONS OF GAS USED ON TRIP BETWEEN
NEW YORK AND BOSTON))
(EQUAL (DISTANCE BETWEEN BOSTON AND NEW YORK) (TIMES 250 (MILES)))
(EQUAL (GAS CONSUMPTION OF MY CAR) (QUOTIENT (TIMES 15 (MILES))
(TIMES 1 (GALLONS))))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION
(USING THE FOLLOWING KNOWN RELATIONSHIPS)
((EQUAL (DISTANCE) (TIMES (SPEED) (TIME))) (EQUAL (DISTANCE)
(TIMES (GAS CONSUMPTION) (NUMBER OF GALLONS OF GAS USED))))

(ASSUMING THAT)
((DISTANCE) IS EQUAL TO (DISTANCE BETWEEN BOSTON AND NEW YORK))

(ASSUMING THAT)
((GAS CONSUMPTION) IS EQUAL TO (GAS CONSUMPTION OF MY CAR))

(ASSUMING THAT)
((NUMBER OF GALLONS OF GAS USED) IS EQUAL TO (NUMBER OF GALLONS
OF GAS USED ON TRIP BETWEEN NEW YORK AND BOSTON))
(THE NUMBER OF GALLONS OF GAS USED ON A TRIP BETWEEN NEW YORK
AND BOSTON IS 16.66 GALLONS)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE DAILY COST OF LIVING FOR A GROUP IS THE OVERHEAD COST
PLUS THE RUNNING COST FOR EACH PERSON TIMES THE NUMBER OF PEOPLE
IN THE GROUP . THIS COST FOR ONE GROUP EQUALS $ 100 , AND THE
NUMBER OF PEOPLE IN THE GROUP IS 40 . IF THE OVERHEAD COST
IS 10 TIMES THE RUNNING COST , FIND THE OVERHEAD AND THE RUNNING
COST FOR EACH PERSON .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02521 (RUNNING COST FOR EACH PERSON))

(EQUAL G02520 (OVERHEAD))

(EQUAL (OVERHEAD COST) (TIMES 10 (RUNNING COST)))

(EQUAL (NUMBER OF PEOPLE IN GROUP) 40)

(EQUAL (DAILY COST OF LIVING FOR GROUP) (TIMES 100 (DOLLARS)))

(EQUAL (DAILY COST OF LIVING FOR GROUP) (PLUS (OVERHEAD COST)
(TIMES (RUNNING COST FOR EACH PERSON) (NUMBER OF PEOPLE IN
GROUP))))

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((OVERHEAD) IS EQUAL TO (OVERHEAD COST))

(ASSUMING THAT)
((RUNNING COST) IS EQUAL TO (RUNNING COST FOR EACH PERSON))

(THE OVERHEAD IS 20 DOLLARS)

(THE RUNNING COST FOR EACH PERSON IS 2 DOLLARS)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF TWO NUMBERS IS 96 , AND ONE NUMBER IS 16 LARGER
THAN THE OTHER NUMBER . FIND THE TWO NUMBERS .)

TRYING POSSIBLE IDIOMS

(THE PROBLEM WITH AN IDIOMATIC SUBSTUTION IS)
(THE SUM OF ONE OF THE NUMBERS AND THE OTHER NUMBER IS 96 ,
AND ONE NUMBER IS 16 LARGER THAN THE OTHER NUMBER . FIND THE
ONE OF THE NUMBERS AND THE OTHER NUMBER .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02516 (OTHER NUMBER))

(EQUAL G02517 (ONE OF NUMBERS))

(EQUAL (ONE NUMBER) (PLUS 16 (OTHER NUMBER)))

(EQUAL (PLUS (ONE OF NUMBERS) (OTHER NUMBER)) 96)

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

TRYING POSSIBLE IDIOMS

(THE PROBLEM WITH AN IDIOMATIC SUBSTUTION IS)
(THE SUM OF ONE NUMBER AND THE OTHER NUMBER IS 96 , AND ONE
NUMBER IS 16 LARGER THAN THE OTHER NUMBER . FIND THE ONE NUMBER
AND THE OTHER NUMBER .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02515 (OTHER NUMBER))

(EQUAL G02514 (ONE NUMBER))

(EQUAL (ONE NUMBER) (PLUS 16 (OTHER NUMBER)))

(EQUAL (PLUS (ONE NUMBER) (OTHER NUMBER)) 96)

(THE ONE NUMBER IS 56)

(THE OTHER NUMBER IS 40)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF TWO NUMBERS IS TWICE THE DIFFERENCE BETWEEN THE
TWO NUMBERS . THE FIRST NUMBER EXCEEDS THE SECOND NUMBER BY
5 . FIND THE TWO NUMBERS .)

TRYING POSSIBLE IDIOMS

(THE PROBLEM WITH AN IDIOMATIC SUBSTUTION IS)
(THE SUM OF FIRST NUMBER AND THE SECOND NUMBER IS TWICE THE
DIFFERENCE BETWEEN THE FIRST NUMBER AND THE SECOND NUMBER .
THE FIRST NUMBER EXCEEDS THE SECOND NUMBER BY 5 . FIND THE
FIRST NUMBER AND THE SECOND NUMBER .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02548 (SECOND NUMBER))

(EQUAL G02547 (FIRST NUMBER))

(EQUAL (FIRST NUMBER) (PLUS 5 (SECOND NUMBER)))

(EQUAL (PLUS (FIRST NUMBER) (SECOND NUMBER)) (TIMES 2 (PLUS
(FIRST NUMBER) (MINUS (SECOND NUMBER)))))

(THE FIRST NUMBER IS  7.500)

(THE SECOND NUMBER IS  2.500)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF TWO NUMBERS IS 111 . ONE OF THE NUMBERS IS CONSECUTIVE
TO THE OTHER NUMBER . FIND THE TWO NUMBERS .)

TRYING POSSIBLE IDIOMS

(THE PROBLEM WITH AN IDIOMATIC SUBSTUTION IS)
(THE SUM OF ONE OF THE NUMBERS AND THE OTHER NUMBER IS 111
. ONE OF THE NUMBERS IS CONSECUTIVE TO THE OTHER NUMBER . FIND
THE ONE OF THE NUMBERS AND THE OTHER NUMBER .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02516 (OTHER NUMBER))

(EQUAL G02515 (ONE OF NUMBERS))

(EQUAL (ONE OF NUMBERS) (PLUS 1 (OTHER NUMBER)))

(EQUAL (PLUS (ONE OF NUMBERS) (OTHER NUMBER)) 111)

(THE ONE OF THE NUMBERS IS  56)

(THE OTHER NUMBER IS  55)
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF THREE NUMBERS IS 9 . THE SECOND NUMBER IS 3 MORE
THAN 2 TIMES THE FIRST NUMBER . THE THIRD NUMBER EQUALS THE
SUM OF THE FIRST TWO NUMBERS . FIND THE THREE NUMBERS .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02527 (THIRD NUMBER))

(EQUAL G02526 (SECOND NUMBER))

(EQUAL G02525 (FIRST NUMBER))

(EQUAL (THIRD NUMBER) (PLUS (FIRST NUMBER) (SECOND NUMBER)))

(EQUAL (SECOND NUMBER) (PLUS 3 (TIMES 2 (FIRST NUMBER))))

(EQUAL (PLUS (FIRST NUMBER) (PLUS (SECOND NUMBER) (THIRD NUMBER)))
9)

(THE FIRST NUMBER IS  .500)

(THE SECOND NUMBER IS  4)

(THE THIRD NUMBER IS  4.500)
```

(THE PROBLEM TO BE SOLVED IS)
(THE SUM OF THREE NUMBERS IS 100 . THE THIRD NUMBER EQUALS THE SUM OF THE FIRST TWO NUMBERS . THE DIFFERENCE BETWEEN THE FIRST TWO NUMBERS IS 10 PER CENT OF THE THIRD NUMBER . FIND THE THREE NUMBERS .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02536 (THIRD NUMBER))

(EQUAL G02535 (SECOND NUMBER))

(EQUAL G02534 (FIRST NUMBER))

(EQUAL (PLUS (FIRST NUMBER) (MINUS (SECOND NUMBER))) (TIMES .1000 (THIRD NUMBER)))

(EQUAL (THIRD NUMBER) (PLUS (FIRST NUMBER) (SECOND NUMBER)))

(EQUAL (PLUS (FIRST NUMBER) (PLUS (SECOND NUMBER) (THIRD NUMBER))) 100)

(THE FIRST NUMBER IS 27.50)

(THE SECOND NUMBER IS 22.50)

(THE THIRD NUMBER IS 50)

(THE PROBLEM TO BE SOLVED IS)
(IF C EQUALS B TIMES D PLUS 1 , AND B PLUS D EQUALS 3 , AND B MINUS D EQUALS 1 , FIND C .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02539 (C))

(EQUAL (PLUS (B) (MINUS (D))) 1)

(EQUAL (PLUS (B) (D)) 3)

(EQUAL (C) (PLUS (TIMES (B) (D)) 1))

(C IS 3)

(THE PROBLEM TO BE SOLVED IS)
(3 * X + 4 * Y = 11 .
5 * X - 2 * Y = 1 .
FIND X AND Y .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02541 (Y))

(EQUAL G02540 (X))

(EQUAL (PLUS (TIMES 5 (X)) (MINUS (TIMES 2 (Y)))) 1)

(EQUAL (PLUS (TIMES 3 (X)) (TIMES 4 (Y))) 11)

(X IS 1)

(Y IS 2)

(THE PROBLEM TO BE SOLVED IS)
(X / 2 - (Y + 3) / 2 = 0 .
(X - 1) / 3 + 2 * (Y + 1) = 5 .
FIND X AND Y .)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02543 (Y))

(EQUAL G02542 (X))

(EQUAL (PLUS (QUOTIENT (PLUS (X) (MINUS 1)) 3) (TIMES 2 (PLUS (Y) 1))) 5)

(EQUAL (PLUS (QUOTIENT (X) 2) (MINUS (QUOTIENT (PLUS (Y) 3) 2))) 0)

(X IS 4)

(Y IS 1)

```
(THE PROBLEM TO BE SOLVED IS)
(THE SQUARE OF THE DIFFERENCE BETWEEN THE NUMBER OF APPLES
AND THE NUMBER OF ORANGES ON THE TABLE IS EQUAL TO 9 . IF THE
NUMBER OF APPLES IS 7 , FIND THE NUMBER OF ORANGES ON THE TABLE
.)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02515 (NUMBER OF ORANGES ON TABLE))

(EQUAL (NUMBER OF APPLES) 7)

(EQUAL (EXPT (PLUS (NUMBER OF APPLES) (MINUS (NUMBER OF ORANGES
ON TABLE))) 2) 9)

UNABLE TO SOLVE THIS SET OF EQUATIONS

TRYING POSSIBLE IDIOMS

(DO YOU KNOW ANY MORE RELATIONSHIPS AMONG THESE VARIABLES)

(NUMBER OF APPLES)

(NUMBER OF ORANGES ON TABLE)

no

I CANT SOLVE THIS PROBLEM
```

```
(THE PROBLEM TO BE SOLVED IS)
(THE GROSS WEIGHT OF A SHIP IS 20000 TONS . IF ITS NET WEIGHT
IS 15000 TONS , WHAT IS THE WEIGHT OF THE SHIPS CARGO Q.)

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

TRYING POSSIBLE IDIOMS

(DO YOU KNOW ANY MORE RELATIONSHIPS AMONG THESE VARIABLES)

(GROSS WEIGHT OF SHIP)

(TONS)

(ITS NET WEIGHT)

(WEIGHT OF SHIPS CARGO)

yes
TELL ME

(the weight of a ships cargo is the difference between
the gross weight and the net weight)

THE EQUATIONS WERE INSUFFICIENT TO FIND A SOLUTION

(ASSUMING THAT)
((NET WEIGHT) IS EQUAL TO (ITS NET WEIGHT))

(ASSUMING THAT)
((GROSS WEIGHT) IS EQUAL TO (GROSS WEIGHT OF SHIP))

(THE WEIGHT OF THE SHIPS CARGO IS 5000 TONS)
```

APPENDIX E: A SMALL SEMANTIC GENERATIVE GRAMMAR

The grammar outlined here will generate only word problems solvable by STUDENT, though not the set of all such problems.

RULES	EXAMPLES
Create a set of simultaneous equations which can be solved by strictly linear techniques, except that substitution of numerical values in higher order equations which reduce them to linear equations is allowed. These are the propositions of the speaker's model.	$2x + 3y = 7$ $y = 1/2x$ $y + z = x^2$
Choose unknowns for which STUDENT is to solve. This is the question.	$z = ?$
Choose unique names for variables without articles "a", "an", or "the". In the problem any of these articles may be used at any occurrence of a name. In a complete model these names would be associated with the objects in the chosen propositions.	x = first number y = second number Tom chose z = third number
Write one kernel sentence for each equation. Use any appropriate linguistic form given in the table below to	"2 times the first number plus three times the second number Tom chose is 7. The second number Tom chose

represent the arithmetic functions in the equation.

For each unknown whose value is to be found, use a kernel sentence of the form:

Find _____

What is _____

or Find _____ and _____

What are _____ and _____

for more than one such unknown.

If a name appears more than once in a problem, some (or all) occurrences after the first may be replaced by a "similar" name. Similar names are obtained by transformations which:

a) insert a pronoun for a noun phrase in the name.

b) delete initial and/ or terminal substrings of the name.

Only one such "similar" string can be used to replace an occurrence of a name, though any number of replacements can be made.

equals .5 of the first number.

The sum of the second number Tom chose and a third number is equal to the square of the first number. What is the third number?"

Similar names:

"first" for "first number"

"second number he chose" for "second number Tom chose"

If N_i occurs in S_j and S_{j+1}, and in S_j it is the entire substring to the left of "is", "equals" or "is equal to" (or the entire substring to the right) then in S_{j+1}, N_i may be replaced by any phrase containing the word "this".

Replace "the second number Tom chose" by "this second choice" in the third sentence.

Any phrase P_1 may be replaced by another phrase P_2 which means the same thing. This would mean that STUDENT had been told of this equivalence using REMEMBER and the sentence "P_2 always means P_1" or "P_2 sometimes means P_1" .

Replace "2 times" by "twice" and ".5" by "one half".

Two consecutive sentences may be connected by replacing the period after the first by ", and". A sentence can be connected to a question by preceding the sentence by "If" and replacing the period at the end of the sentence by ",".

Connect sentences 1 and 2, and sentence 3 and the final question to give:

"Twice the first number plus three times the second number Tom chose is 7, and the second number he chose is one half of the first. If the sum of this second choice and a third number is equal to the square of the first number, what is the third number?"

Summary of Linguistic Forms to Express Arithmetic Functions and the Equality Relation

x = y	x is y; x equals y; x is equal to y
x + y	x plus y; the sum of x and y; x more than y
x - y	x minus y; the difference between x and y; y less than x
x * y	x times y; x multiplied by y; x of y (if x is a number)
x / y	x divided by y; x per y

BIBLIOGRAPHY

(1) Berkeley, E.C. and D.G. Bobrow (eds.), The Programming Language LISP: Its Operation and Applications, Information International, Inc., Cambridge, Mass.; 1964.

(2) Black, F., "A Deductive Question-Answering System," Ph. D. Thesis, Division of Engineering and Applied Physics, Harvard University, Cambridge, Mass.; 1964.

(3) Bobrow, D.G., "METEOR: A LISP Interpreter for String Transformations," in (1).

(4) Bobrow, D.G., "Syntactic Analysis of English by Computer—A Survey," Proc. FJCC, Spartan Press, Baltimore, Md; 1963

(5) Bobrow, D.G. and B. Raphael, "A Comparison of List-Processing Computer Languages," Comm. ACM; April, 1964.

(6) Bobrow, D.G. and J. Weizenbaum, "List Processing and the Extension of Language Facility by Embedding," Trans. IEEE, PGEC; August, 1964.

(7) Chomsky, A.N., "On the Notion 'Rule of Grammar'," Proceedings of the Symposium in Applied Mathematics, vol. 12.

(8) Chomsky, A.N., Syntactic Structures, Mouton and Co., 'S-Gravenhage; 1957.

(9) Coffman, E.G., J.I. Schwartz and C. Weissman, "A General-Purpose Time-Sharing System," Proc. SJCC, Spartan Press, Baltimore, Md.; April, 1964.

(10) Cohen, D., "Picture Processing in a Picture Language Machine," NBS Report 7885, Dept. of Commerce, Wash., D.C.; April, 1963.

(11) Coleman, M., "A Program to Solve High School Algebra Story Problems," MIT Term Paper for 6.539; 1964.

(12) Cooper, W.S., "Fact Retrieval and Deductive Question Answering," JACM, vol. 11, no. 2; April, 1964.

(13) Corbato, F.J., et al., The Compatible Time-Sharing System, MIT Press, Cambridge, Mass.; 1963.

(14) Darlington, J., "Translating Ordinary Language into Symbolic Logic," Memo MAC-M-149, Project MAC, MIT; March, 1964.

(15) Feigenbaum, E., "The Simulation of Verbal Learning Behaviour," in (16).

(16) Feigenbaum, E. and J. Feldman (eds.), Computers and Thought, McGraw Hill, New York; 1963.

(17) Feldman, J., "Simulation of Behaviour in the Binary Choice Experiment," in (16).

(18) Garfinkle, S., "Heuristic Solution of First Year Algebra Problems," Working Paper Number 11, Management Science Group, University of California, Berkeley, Calif.; 1962.

(19) Green, B.F., A.K. Wolf, C. Chomsky and K. Laughery, "Baseball: An Automatic Question Answerer," Proc. WJCC; May, 1961.

(20) Harris, Z., "Discourse Analysis," Language, vol. 28, no. 1; Jan. - March, 1952.

(21) Harris, Z., String Analysis of Sentence Structure, Mouton and Co., The Hague; 1962.

(22) Kirsch, R.A. and B.K. Rankin III, "Modified Simple Phrase Structure Grammars for Grammatical Induction," NBS Report 7890, Dept. of Commerce, Wash., D.C.; 1963.

(23) Klein, S. and R.F. Simmons, "Syntactic Dependence and the Computer Generation of Coherent Discourse," Mechanical Translation; 1963.

(24) Kuck, D., "A Problem Solving System with Natural Language Input," Ph. D. Thesis, Technological Institute, Northwestern Univ., Evanston, Illinois; 1963.

(25) Kuno, S. and A. Oettinger, "Syntactic Structure and Ambiguity of English," Proc. FJCC, Spartan Press, Baltimore, Md.; Nov., 1963.

(26) Lamb, S.M., Outline of Stratificational Grammar, University of California, Berkeley, Calif.; 1962.

(27) Lehman, W.P. and E.D. Pendergraft, Machine Language Translation Study No. 16, Linguistic Research Center, University of Texas, Austin, Texas; June, 1963.

(28) Lindsay, R.K., "Inferential Memory as the Basis of Machines Which Understand Natural Language," in (16).

(29) Mathews, G.H., "Analysis by Synthesis of Sentences in a Natural Language," First International Conference on Machine Translation and Applied Language Analysis, HMSO, London; 1962.

(30) McCarthy, J., "Programs With Common Sense," Proc. of the Symposium on Mechanization of Thought Processes, HMSO, London; 1959.

(31) McCarthy, J., et al., LISP 1.5 Programmers Manual, MIT Press, Cambirdge, Mass.; 1963.

(32) Minsky, M., "Steps Toward Artificial Intelligence," in (16).

(33) Morris, C.W., "Foundations of the Theory of Signs," International Encyclopedia of Unified Science, vol. 1, no. 2, University of Chicago Press, Chicago; 1955.

(34) Newell, A. et al., "Report on a General Problem Solving System," Proc. International Conference on Information Processing, UNESCO House, Paris; 1959.

(35) Ogden, C.K., A System of Basic English, Harcourt-Brace, New York; 1934.

(36) Phillips, A.V., "A Question-Answering Routine," Masters Thesis, Mathematics Department, MIT, Cambridge, Mass.; 1960.

(37) Quine, W.V., Word and Object, MIT Press, Cambridge, Mass.; 1960.

(38) Raphael, B., "SIR: A Computer Program for Semantic Information Retrieval," Ph. D. Thesis, Mathematics Department, MIT, Cambridge, Mass.; 1964.

(39) Sillars, W., "An Algorithm for Representing English Sentences in a Formal Language," NBS Report 7884, Department of Commerce, Wash., D.C.; April, 1963.

(40) Simmons, R.F., "Answering English Questions by Computer—A Survey," SDC Report SP-1556, Santa Monica, Calif.; April, 1964.

(41) Simmons, R.F., S. Klein and K.L. McConologue, "Indexing and Dependency Logic for Answering English Questions," American Documentation, (in press).

(42) Skinner, B.F., Verbal Behaviour, Appleton Century Croft, New York; 1957.

(43) Walker, D.E. and J.M. Bartlett, "The Structure of Language for Men and Computers: Problems in Formalization," Proc. First Congress on the Information Sciences, Vista Press; 1963.

(44) Yngve, V., "A Model and an Hypothesis for Language Structure," Proceedings of the American Philosophical Society, vol. 104, no. 5, 1960.

(45) Yngve, V., COMIT Programmers Reference Manual, MIT Press, Cambridge, Mass.; 1961.

(46) Yngve, V., "Random Generation of English Sentences," Proc. 1961 International Conference on Machine Translation and Applied Language Analysis, Vol. I, HMSO, London; 1962.

BIOGRAPHICAL NOTE

Daniel G. Bobrow was born in New York City on November 29, 1935. He attended the Bronx High School of Science, received a B.S. degree in Physics from Rensselaer Polytechnic Institute in 1957, and received an S.M. in Applied Mathematics from Harvard University in 1958.

Mr. Bobrow held several scholarships at R.P.I. from 1953 to 1957, and was a Gordon McKay Fellow at Harvard in 1958. He was elected to Sigma Xi in 1957. At MIT, he has been a research assistant with the Research Laboratory of Electronics and with Project MAC.

Mr. Bobrow has been employed by the General Electric Company; Boeing Aircraft Company; RCA; Encyclopedia Britannica Film Corp.; Bolt, Beranek and Newman, Inc.; RAND Corporation; and the System Development Corporation. He has accepted a position as Assistant Professor of Electrical Engineering at MIT for the 1964-65 academic year.

His publications include:

Introductory Calculus, Britannica Press, Chicago; 1961.

Basic Mathematics, Britannica Press, Chicago; 1962.

The Programming Language LISP: Its Operation and Applications, (edited with E.C. Berkeley), Information International, Inc., Cambridge, Mass.; 1964

"METEOR: A LISP Interpreter for String Transformations," (in The Programming Language LISP: Its Operation and Applications).

"Syntactic Analysis of English by Computer — A Survey," Proc. FJCC, Spartan Press, Baltimore, Md.; 1963.

"A Comparison of List-Processing Computer Languages," (with B. Raphael), Comm.ACM; April, 1964.

"List Processing and the Extension of Language Facility by Embedding," (with J. Weizenbaum), Trans. IEEE, PGEC; August, 1964.

Mr. Bobrow is currently a member of the Association for Computing Machinery, the Association for Machine Translation and Computational Linguistics, and the American Mathematical Society.